TCP-ADaLR: TCP WITH ADAPTIVE DELAY AND LOSS RESPONSE FOR BROADBAND GEO SATELLITE NETWORKS

by

Modupe Omogbohun Omueti
B.Sc., Obafemi Awolowo University, 2001

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

In the
School
of
Engineering Science

© Modupe Omogbohun Omueti 2007

SIMON FRASER UNIVERSITY

2007

APPROVAL

Name:	**Modupe Omogbohun Omueti**
Degree:	**Master of Applied Science**
Title of Thesis:	**TCP-ADaLR: TCP with Adaptive Delay and Loss Response for Broadband GEO Satellite Networks**

Examining Committee:

Chair: **Dr. Jie Liang**
Assistant Professor of School of Engineering Science

Dr. Ljiljana Trajković
Senior Supervisor
Professor of School of Engineering Science

Dr. Rodney Vaughan
Supervisor
Professor of School of Engineering Science

Dr. Stephen Hardy
Internal Examiner
Professor of School of Engineering Science

Date Defended/Approved: June 26, 2007

ii

ABSTRACT

Transmission Control Protocol (TCP) performance degrades in broadband geostationary satellite networks due to long propagation delays and high bit error rates. In this thesis, we propose TCP with algorithm modifications for adaptive delay and loss response (TCP-ADaLR) to improve TCP performance. TCP-ADaLR incorporates delayed acknowledgement mechanism recommended for Internet hosts. We evaluate and compare the performance of TCP-ADaLR, TCP SACK, and TCP NewReno, with and without delayed acknowledgements. In the ideal channel case, TCP-ADaLR exhibits the lowest user-perceived latency for FTP and HTTP applications. In the presence of congestion, TCP-ADaLR shows comparable performance to TCP SACK and TCP NewReno. In the presence of error losses, TCP-ADaLR exhibits improvements up to 61% and 76% in throughput and utilization, respectively. In the presence of both congestion and error losses, TCP-ADaLR exhibits goodput and throughput improvements up to 43%. TCP-ADaLR exhibits better fairness and friendliness than TCP NewReno and maintains TCP end-to-end semantics.

Keywords: Transmission control protocol (TCP); delayed acknowledgement; high bit error rate; GEO satellite networks; performance evaluation

DEDICATION

To my fathers:

God Almighty from whom all blessings flow

Late Dr. J. O. Omueti

ACKNOWLEDGEMENTS

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

ABW	Available bandwidth
ACK	Acknowledgement
AIMD	Additive increase multiplicative decrease
ARQ	Automatic repeat request
ATM	Asynchronous Transfer Mode
BDP	Bandwidth delay product
BER	Bit error rate
BSS	Broadcast satellite service
BWE	Bandwidth estimate
CWND	Congestion window
ECN	Explicit congestion notification
ELN	Explicit loss notification
FEC	Forward error correction
FIN	Finish
FSM	Finite state machine
FSS	Fixed satellite service
FTP	File transfer protocol
ICMP	Internet control message protocol
IP	Internet protocol
IPSEC	Internet protocol security
ISL	Inter-satellite link
IW	Initial window
MSS	Mobile satellite service
GEO	Geostationary earth orbit
GPS	Global positioning system
HTTP	Hyper-text transfer protocol
LEO	Low earth orbit

LoS	Line of sight
MEO	Medium earth orbit
MTU	Maximum transmission unit
NACK	Negative acknowledgement
NGEO	Non-geostationary earth orbit
OSI	Open System Interconnection
PAWS	Protect against wraparound sequence numbers
PEP	Performance enhancing proxy
PER	Packet error rate
PPP	Point-to-point
RLOGIN	Remote login
RTO	Retransmission timeout
RTPD	Round trip propagation delay
RTT	Round trip time
RTTVAR	Round trip time variation
RWND	Receiver's advertised window
SACK	Selective acknowledgement
SEQNO	Sequence number
SMSS	Sender maximum segment size
SNACK	Selective negative acknowledgement
SCPS-TP	Space Communications Protocol Standards-Transport Protocol
SRTT	Smoothed round trip time
SSTHRESH	Slow start threshold
STD	State transition diagram
TCP	Transmission control protocol
UDP	User datagram protocol
WAN	Wide area network
WLAN	Wireless local area network
WWW	World Wide Web

CHAPTER 1: INTRODUCTION

Transmission control protocol (TCP) [1], [2] is a transport layer protocol that provides connection-oriented, in-order, window flow control, and reliable byte-stream delivery services for Internet application-level protocols such as remote login, telnet, hyper-text transfer protocol (HTTP), and file transfer protocol (FTP) [3]. Internet protocol (IP) [4] is a network layer protocol that provides addressing, routing, and internetworking functions for its packets known as datagrams. The TCP/IP suite specifies rules of communication between Internet hosts. TCP carries up to 90% of Internet traffic [5]–[7]. The Internet has witnessed an increasing demand for new high-speed multimedia and data applications over wireless networks such as satellite networks, wireless local area networks (WLANs), and cellular networks [8]. TCP was originally designed for wired networks characterized by negligible random bit error rates (BERs) and, hence, packet losses indicate congestion. TCP was enhanced by the congestion control algorithms [9], [10], in order to address packet losses caused by congestion, thus enabling TCP to perform well in wired networks.

Broadband geostationary earth orbit (GEO) satellite networks transmit and receive data using frequencies relayed by GEO satellites. They provide global Internet access to areas with limited or no terrestrial cable infrastructure available. Through high-bandwidth GEO satellite links, broadband GEO satellite networks offer data rates of the order of 1 Mb/s or higher.

1.1 Motivation

The next generation Internet needs to be pervasive, seamless, and ubiquitous [11]. High bandwidth GEO satellite links will play an important role by providing broadband Internet access and high-speed backbone network connectivity between remote networks through easily scalable architecture and multicast capabilities [12]. When providing the same continuous coverage, GEO satellite links are more attractive than non-GEO (NGEO) links because they have lower development risks compared to large number of satellite constellations required by NGEO satellites [13].

TCP increases its transmission rate when a sender receives acknowledgements (ACKs) of transmitted segments. Internet hosts are recommended to enable the delayed ACK option [14] in order to maximally utilize available network bandwidth by reducing the number of ACKs sent to a sender by a receiver. The delayed ACK option allows a TCP receiver to send an ACK for every two consecutive full-size packet received from a TCP sender. A full-size packet is equivalent to the sender maximum segment size (SMSS) packet. Many current TCP implementations in Internet hosts enable the delayed ACK option [15].

GEO satellite links are characterized by high bit error rate (BER), long propagation delay, and bandwidth asymmetry (different uplink and downlink bandwidths). TCP performs poorly in heterogeneous networks with GEO satellite links because of their characteristics. TCP misinterprets packet losses as an indication of congestion and reduces the transmission rate, thus leading to TCP throughput degradation. In the absence of error losses, long propagation delays result in large round trip times (RTTs) that coupled with limited TCP window size lead to poor utilization of

2

the available bandwidth of satellite links. Hence, TCP performance in broadband networks employing GEO satellite links needs improvements that will not violate the end-to-end semantics of TCP. These improvements will support TCP connections with delayed ACK and not violate the end-to-end semantics of TCP or negatively affect TCP connections without delayed ACK. It has been shown [16] that with the delayed ACK option enabled, TCP NewReno [17] and TCP selective ACK (SACK) [18] suffer higher throughput degradation when compared to older variants such as TCP Reno [9] and TCP Tahoe [2].

1.2 Contribution

In this thesis, we propose TCP with adaptive delay and loss response (TCP-ADaLR) algorithm for improving the end-to-end performance of TCP in broadband GEO satellite networks. TCP-ADaLR introduces division of congestion window *cwnd* increment phase into sub-phases in order to enable transmission of additional segments for better satellite link utilization in the absence of losses. It also adjusts transmission rate more adaptively in the presence of losses. It is designed for the case when the TCP delayed ACK option is enabled and does not affect TCP performance when the delayed ACK option is disabled.

We implement TCP-ADaLR algorithm as an extension to TCP SACK [18] in the OPNET network simulator [19]. The proposed algorithm is also applicable to TCP NewReno [17]. The TCP-ADaLR algorithm requires only sender-side modifications and, thus, maintains the end-to-end semantics of TCP. It is also is designed for the case when the TCP delayed ACK option is enabled. The proposed algorithm:

1. Improves performance in the absence of losses (ideal lossless GEO satellite link)
2. Has comparable performance to TCP SACK or TCP NewReno in the presence losses due to congestion
3. Improves performance in the presence of losses due to high BER only and both high BER and congestion
4. Has comparable or better TCP fairness than common TCP implementations (TCP SACK and TCP NewReno)
5. Exhibits TCP friendliness for connections using TCP SACK and/or TCP NewReno

We evaluate the performance of the TCP-ADaLR algorithm in the absence of congestion and error losses and in the presence of only congestion losses, only error losses, and both error losses and congestion losses for various bit error rates (BERs). We simulate TCP-ADaLR for FTP file download applications in order to evaluate its performance for bulk transfers. We also evaluate the performance of TCP-ADaLR for short-lived HTTP web transfers. More than 50% of Internet servers employ TCP SACK while the majority of others use TCP NewReno [20]. Hence, we present the performance comparison of TCP-ADaLR, TCP SACK, and TCP NewReno for both FTP file download and HTTP web transfer applications. We also evaluate the effect of the proposed algorithm modifications on TCP connections with the delayed ACK option disabled. We investigate the scalability of TCP-ADaLR with increased terrestrial propagation delays and increased GEO satellite link capacities. We also study the effect of varying file sizes (FTP file download application) on the TCP-ADaLR algorithm. Fairness of TCP-ADaLR in the presence of competing connections with short and long propagation delays is also considered. We also examine the friendliness of TCP-ADaLR with TCP NewReno in the absence of losses.

4

1.3 Thesis outline

This thesis is organized as follows. An overview of TCP, its congestion control algorithms, and options are given in Chapter 2. In Chapter 3, we present an overview of satellite networks, types of satellites, and GEO satellite link characteristics. A review of the impact of GEO satellite link characteristics and previous work on the performance of TCP in GEO satellite networks is presented in Chapter 4. In Chapter 5, we describe the TCP-ADaLR algorithm and its mechanisms for improving the end-to-end performance of TCP in broadband GEO satellite networks. The design and implementation details of the TCP-ADaLR algorithm in the OPNET network simulator is presented in Chapter 6. Simulation scenarios and results of the performance evaluation and comparison of TCP-ADaLR, TCP SACK, and TCP NewReno are presented in Chapter 7. We present conclusions and possible areas of future research work in Chapter 8.

CHAPTER 2: TRANSMISSION CONTROL PROTOCOL

TCP is an end-to-end connection-oriented Open Systems Interconnection (OSI) transport layer protocol [3], [21]. It provides reliable byte-stream delivery services independent of the underlying network architecture. TCP provides point-to-point full-duplex data transfer [22]. Hence, there is always a sender and a receiver for a TCP connection and data flow may be unidirectional or bidirectional. TCP employs mechanisms for connection management, flow control, and congestion control. TCP variants include TCP Tahoe [2], TCP Reno [9], and TCP NewReno [17]. In this Chapter, we present an overview of TCP and describe the congestion control algorithms, round-trip time (RTT) estimation, and the delayed acknowledgement (ACK) option.

2.1 TCP connection establishment and termination

Before TCP packet data units (segments) can be transmitted, a TCP connection is established between a sender and a receiver by a three-way handshake. The receiver (client) sends a SYN segment that specifies the TCP destination port number of the sender (server). TCP port numbers are located in the TCP segment header and they identify client and server applications at both ends of a connection. The sequence number, acknowledgement (ACK) number, receiver window size, header length, and TCP flags are also located in the TCP segment header. The TCP segment header fields, shown in Figure 2.1, hold information agreed upon by both sender and receiver for synchronisation and connection establishment. The length of a TCP segment header depends on the TCP options included. The sender acknowledges the client's request with

another SYN segment that must be acknowledged by the receiver to complete the connection establishment. The three-way handshake process is shown in Figure 2.2.

source port			destination port	
sequence number				
acknowledgment number				
hlen	reserved	flag bits	receiver window size	
checksum			urgent pointer	
options				padding
data				

0   4   10   16   24   31

←──────────────────── bits ────────────────────→
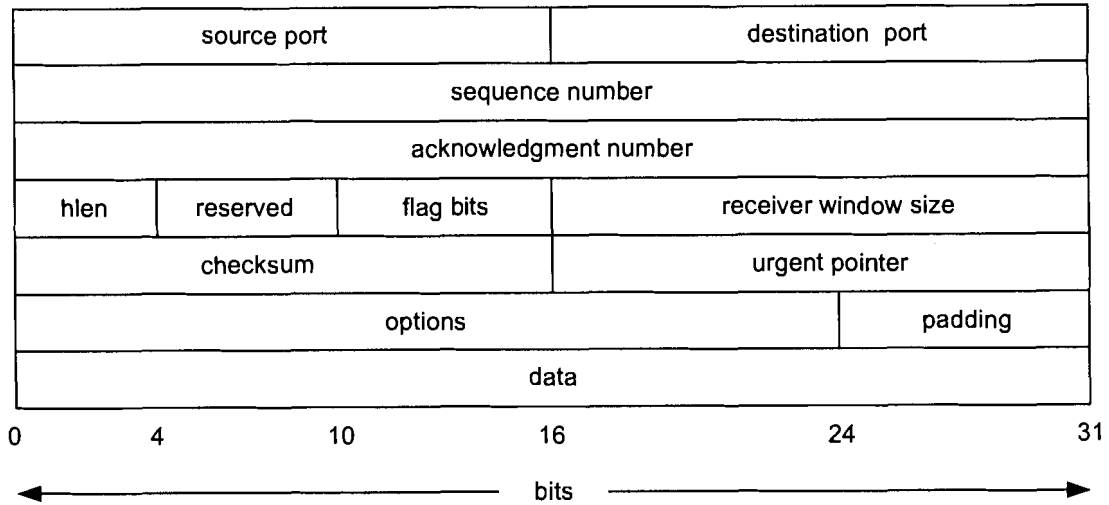
Figure 2.1. TCP header segment. The TCP header length (hlen) is 20 bytes if there are no TCP options selected.

Client   Server

Send SYN segment   t1   SYN, inital seqno = x
                                                    t2   Receive SYN segment
                                                         Send ACK of SYN segment
                            ACK of SYN, seqno = y
                                                    t3
                            ACK = x+1
Receive ACK of SYN segment t4
Send ACK segment   t5   ACK = y+1

                                                    t6   Receive ACK segment

                                                    seqno: sequence number
                                                    t: time
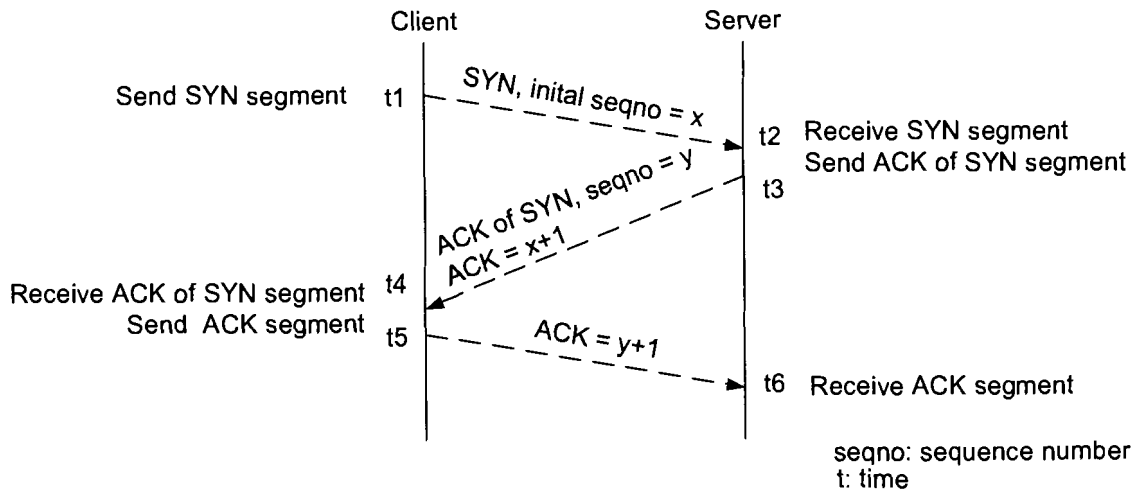
Figure 2.2. TCP three-way handshake. The dashed lines represent exchanged TCP segments. The initial sequence numbers are carried in the SYN segments. Time increases from the top to the bottom of the figure.

After data transfer is completed, a TCP connection is terminated. A full-duplex TCP connection requires four segments for termination. Either the sender or the receiver may initiate the termination process by sending a FIN segment. The receiving end

7

responds with an ACK of the FIN segment. The TCP full duplex connection is closed

when both sender and receiver have sent and acknowledged the receipt of a FIN segment,

as shown in Figure 2.3.
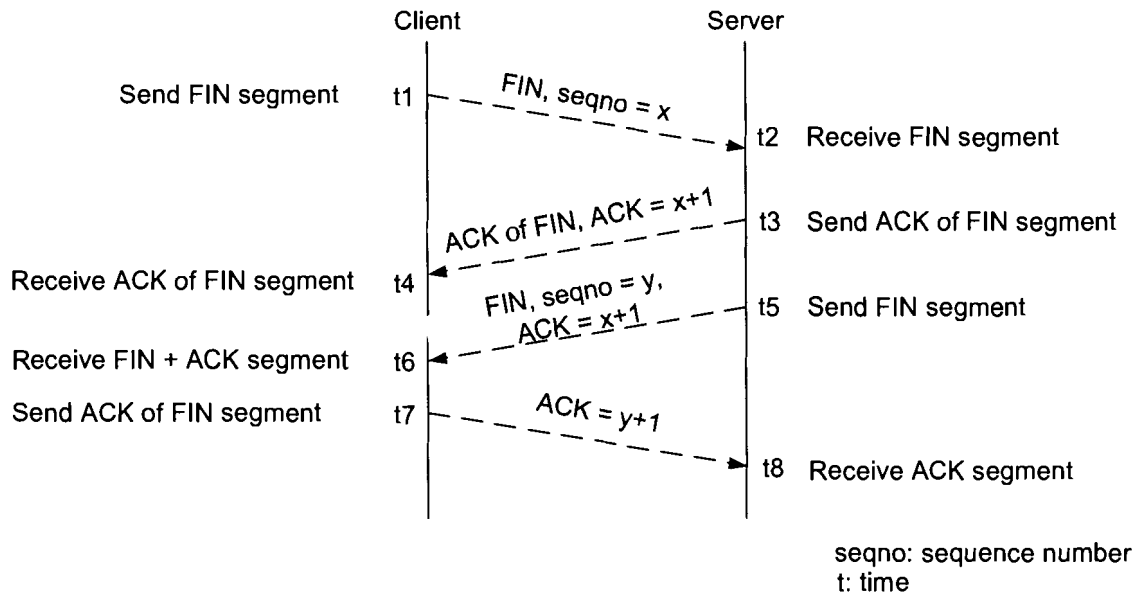
Figure 2.3. TCP connection termination initiated by the client. The dashed lines represent TCP segments exchanged. The initial sequence numbers are carried in the SYN segments. Time increases from the top to the bottom of the figure.

2.2 TCP flow control

After a connection is established, TCP utilizes flow control to manage the amount

and rate of data transmitted between a sender and receiver. TCP flow control is based on

the sliding window mechanism. At the start of transmission, a TCP sender has an initial

window (IW) that specifies the number of segments (window) that a TCP sender may

transmit without receiving an ACK from a receiver [9]. When a TCP sender receives

ACKs of transmitted segments, it slides (increases) the window size based on the number

of segments or bytes acknowledged and the window size advertised by a receiver.

2.3 TCP congestion control algorithms

TCP employs four congestion control algorithms [10]: slow start, congestion avoidance, fast retransmit, and fast recovery, as shown in Figure 2.4. The congestion control algorithms are based on additive increase multiplicative decrease (AIMD) [23] algorithms. The congestion control algorithms employ two TCP state variables, the congestion window size *cwnd* and the receiver's advertised window *rwnd*, to control the amount of data transmitted through the network. The *cwnd* is the maximum number of bytes the sender may send before receiving an ACK, while the *rwnd* is the maximum number of bytes the receiver may receive. The minimum of the two determines the amount of data transmitted through the network. The slow start threshold *ssthresh* determines the exit from the slow start phase and the onset of the congestion avoidance phase. The initial high value of *ssthresh* is adjusted when congestion occurs in a network [9].

2.3.1 Slow start

After the three-way handshake is completed, a TCP sender probes the network gradually during the slow start phase to determine its capacity. The initial value of the *cwnd* is equal to the initial window (IW) set to [24]

$$IW = \min (4 \times SMSS, \max (2 \times SMSS, 4380 \text{ bytes})). \tag{2.1}$$

The sender increments the *cwnd* exponentially during the slow start phase for each ACK received that acknowledges new data:

$$cwnd \mathrel{+}= SMSS. \tag{2.2}$$

A TCP sender remains in the slow start phase until the *cwnd* exceeds *ssthresh*.

9
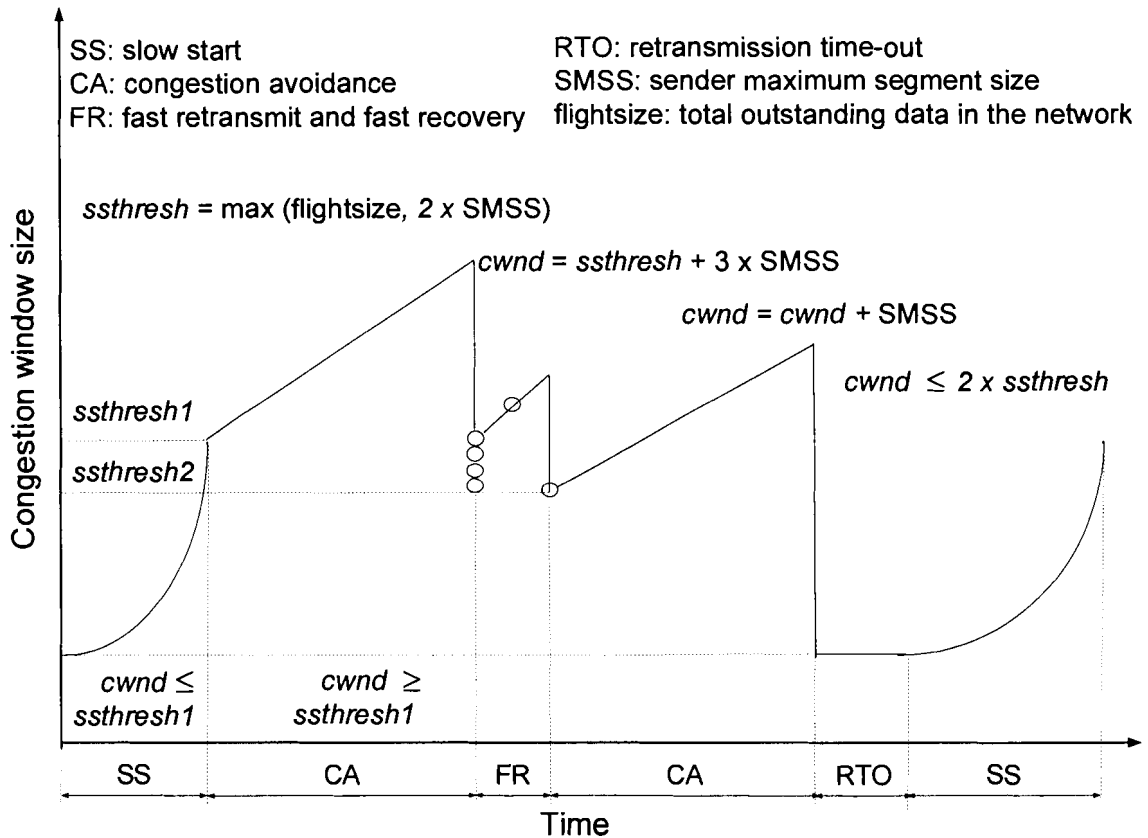
Figure 2.4. TCP congestion control algorithms. The congestion control algorithms and the mechanism used to indicate congestion determined the size of the congestion window.

2.3.2 Congestion avoidance

During the congestion avoidance phase, a TCP sender probes the network more slowly than in the slow start phase. The sender increments the *cwnd* linearly by one SMSS per RTT:

$$cwnd += \text{SMSS} \times \text{SMSS} / cwnd. \tag{2.3}$$

Congestion avoidance ends when congestion is detected (segment loss occurs) by the receipt of three duplicate ACKs or the expiration of the sender's retransmission time-out (RTO) timer.

2.3.3 Fast retransmit

A TCP sender enters the fast retransmit phase when a segment loss is detected by three duplicate ACKs. TCP assigns a unique sequence number *seqno* to each transmitted segment. When a segment loss occurs, TCP issues a duplicate ACK for any out-of-order segment received. Fast retransmit uses the arrival of three duplicate ACKs as an indication of segment loss. TCP retransmits the lost segments without waiting for the RTO timer to expire and sets the *ssthresh* value to half the current *cwnd*. TCP Tahoe implements slow start, congestion avoidance, and fast retransmit algorithms.

2.3.4 Fast recovery

In the fast recovery phase, a TCP sender adjusts the *cwnd* for all segments buffered by a TCP receiver:

$$cwnd = ssthresh + 3 \times \text{SMSS}. \tag{2.4}$$

For each additional duplicate ACK received, the *cwnd* is incremented by one SMSS to reflect a segment that has been transmitted. The new values of *cwnd* and *rwnd* may allow transmission of a new segment. When the next ACK of a newly transmitted segment arrives, the sender deflates the *cwnd* to *ssthresh* and resumes the congestion avoidance phase. TCP Reno implements slow start, congestion avoidance, fast retransmit, and the standard fast recovery algorithm. TCP NewReno implements a modification to the TCP Reno fast recovery algorithm to address the issue of partial ACKs. When loss is detected, ACKs that acknowledge only certain (but not all) outstanding segments are known as partial ACKs. In the TCP Reno implementation, TCP exits the fast recovery and resumes congestion avoidance when a partial ACK is received by a TCP sender. The TCP NewReno modification allows a TCP sender to remain in fast recovery even when a

partial ACK is received by retransmitting one lost segment per RTT until all lost segments from a single transmission window have been retransmitted. A TCP sender deflates the *cwnd* by the amount of data acknowledged and adds one SMSS to the *cwnd* for each partial ACK received during the fast recovery phase. TCP resumes congestion avoidance only after all lost segments have been retransmitted.

2.4 Retransmission time-out

When the network cannot trigger a threshold of three duplicate ACKs (*cwnd* < 4), expiration of the RTO timer is used as an indication of segment loss. Hence, TCP employs the RTO timer in order to avoid unnecessary delay before retransmitting a segment in the absence of an ACK. A TCP sender transmits a segment, sets the RTO timer, and expects an ACK of the transmitted segment. If the ACK is not received and the RTO timer expires, TCP retransmits the unacknowledged segment and resets the RTO timer. The value of *ssthresh* is set to

$$ssthresh = \max \left(flightsize / 2, 2 \times \text{SMSS}\right), \tag{2.5}$$

where *flightsize* is the amount of unacknowledged data in the network. A TCP sender then reduces the *cwnd* to the SMSS and reverts to the slow start phase. If the calculated value of RTO is too large, lost segments are not retransmitted quickly. If the calculated RTO is too small, the RTO timer expires prematurely resulting in unnecessary retransmissions. In both cases, network bandwidth is wasted.

A TCP sender maintains two variables used to compute the RTO value: smoothed RTT (*srtt*) (the moving average of RTT) and RTT variation (*rttvar*). The value of RTT is

estimated from RTT samples (*sampleRTT*) in data segments that are not retransmitted using the Karn's algorithm [25]. The values of *srtt* and *rttvar* are computed as

$$rttvar = (1 - \beta) \times rttvar + \beta \times |\, sampleRTT - srtt\,| \qquad (2.6)$$

$$srtt = (1 - \alpha) \times srtt + \alpha \times sampleRTT. \qquad (2.7)$$

Recommended parameter values $\alpha = 0.125$ and $\beta = 0.25$ enable a TCP sender to respond rapidly to changes in the RTT and to estimate the RTO accurately [10], [14]. The value of *srtt* (2.6) is its value before the update (2.7). Hence, these values are calculated in the given order (2.6) and (2.7) [26]. RTO is then computed as

$$RTO = srtt + \max(G, 4 \times rttvar), \qquad (2.8)$$

where G is clock (timer) granularity in seconds [26]. The RTO mechanism employs the exponential backoff algorithm where retransmission interval is doubled after each retransmission. The maximum RTO may not exceed 64 s.

The estimation of the RTT and RTO depends on the clock granularity [27]. Many current TCP implementations have coarse-grained TCP clocks that limit the timer granularity and, hence, affect the RTO estimation [27], [28]. However, the coarse timer granularity (500 ms) is employed as a low-pass filter to filter fluctuations in traffic [23], [29]. Hence, the resulting RTO estimation from the timer granularity prevents unnecessary retransmissions due to large spikes in RTT [28], [29]. Changing to finer timer granularity (100 ms or less) would reduce the impact of timer granularity on RTO estimation. It may, however, result in false retransmissions due to early RTO timer expiration when the RTT is large[15], [27]–[29].

2.5 Delayed acknowledgement

A TCP receiver may increase efficiency by not sending ACK for every data segment received [14]. This TCP option is known as delayed acknowledgement (ACK). By enabling the delayed ACK option, a TCP receiver increases network efficiency and maximizes bandwidth by acknowledging multiple segments and window updates with a single ACK. If a TCP receiver does not enable the delayed ACK option, separate ACKs are required to acknowledge data segments and to send window updates. If there is two-way data transfer, a delayed ACK may also be sent together with the data segment (ACK piggybacks with data) [3]. Hence, the delayed ACK option reduces protocol processing overhead [30]. It has been recommended that Internet hosts "should" implement the delayed ACK option [9], [14].

In WLANs, delayed ACK has been shown to reduce the number of collisions of data packets with ACK packets and, thus, increasing TCP throughput [31]. In asymmetric satellite IP networks, delayed ACK may reduce the number of ACKs sent on a slow speed terrestrial return link and improve TCP performance [32]. In the absence of losses caused by BER, TCP exhibits identical throughput for cases with and without delayed ACK [16]. However, various TCP variants suffer throughput performance degradation when losses occur because of satellite link BERs. Delayed ACK is also a source of wasted capacity during the slow start phase [33].

Many Internet TCP receivers implement the delayed ACK option [15], [16]. The default interval period before sending an ACK is 200 ms [14]. However, a TCP receiver may wait up to 500 ms within the arrival of the last unacknowledged segment before the delayed ACK timer expires. Most TCP implementations use the default delay interval of

200 ms [14]. Although TCP uses ACKs to ensure window flow control and reliability, generating more ACKs than necessary is not a desirable characteristic in wireless networks [34]. Hence, a TCP receiver may enable the delayed ACK option to generate the optimal number of ACKs required for reliable delivery of data and improved TCP performance [34]. An optimal number of ACKs reduces the effects of excessively delaying an ACK.

CHAPTER 3: SATELLITE NETWORKS

Satellite networks transmit and receive data using radio frequencies relayed by satellites. The frequency bands of operation are shown in Table 3.1. Satellites play an important role in providing global Internet services to areas where there is limited or no terrestrial communication infrastructure. They provide interconnectivity between geographically distant and heterogeneous networks. Satellites also provide Internet and communication services to aircrafts, ships, and individual users. Satellite services may be fixed, mobile, or broadcast, as shown in Table 3.1. Broadband satellite networks offer high data rates of 1 Mb/s and above [22] for multimedia and Internet applications through high bandwidth satellite links. In this Chapter, we briefly describe various types of satellites and satellite link characteristics that affect TCP performance in satellite networks.

Table 3.1. Satellite frequency bands of operation. Shown are the different services specified for each band. The Q and V frequency bands are considered experimental and have not been employed for satellite services [35].

Band	Frequency range (GHz)	Service
L	1.5 – 1.65	Mobile satellite service (MSS)
S	2.4 – 2.8	MSS
C	3.4 – 7.0	Fixed satellite service (FSS)
X	7.9 – 9.0	MSS, military, space research
Ku	10.7 – 15.0	FSS, broadcast satellite service (BSS)
Ka	18.0 – 31.0	FSS
Q	40.0 – 50.0	FSS
V	60.0 – 80.0	FSS

3.1 Types of satellites

Communication satellites may be classified by their orbit altitude to geostationary earth orbit (GEO), medium earth orbit (MEO), and low earth orbit (LEO), as shown in Figure 3.1. Orbit altitude classification determines power requirement, satellite lifetime, coverage, and antenna usage.
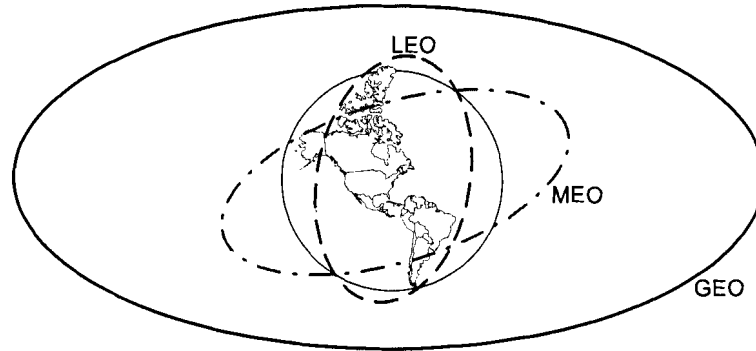
Figure 3.1. Types of satellite according to orbit altitude. Shown is only one orbit for each altitude. Depending on the type of satellite, additional orbits and satellites are required to provide continuous coverage.

3.1.1 Geostationary earth orbit satellite

GEO satellites are circular in shape and lie in the plane of the equator. They orbit at an altitude of ~36,000 km above the earth surface with a period of ~24 hours (earth rotation period) [35] (The earth rotates once a day about the polar axis for 23 hours 56 minutes 4 s and also completes 1/365.24 of the annual orbit of the sun). Hence, GEO satellites appear to be stationary to observers from the earth. A single GEO satellite has a large footprint (satellite signal coverage area of the earth surface) and provides coverage of an entire earth hemisphere except the polar regions [36]. Hence, receiving antennas positioned within the large footprint of the satellite require no tracking capabilities.

3.1.2 Non-geostationary earth orbit satellite

Non-GEO (NGEO) satellites orbit at altitudes lower than 36,000 km. They have shorter rotation periods, and, hence, their relative position to the earth changes. A single NGEO satellite has a smaller footprint than the GEO satellite. Hence, a satellite network employing NGEO satellites requires a constellation (a number of similar satellites with similar function, designed to be in complementary orbits and under shared control [37]) to provide simultaneous continuous coverage over the earth surface. A MEO satellite orbits at altitude range 5,000 – 12,000 km [35]. It has ~100 ms one-way single hop propagation delay and 5 – 10 hours orbit rotation period. A LEO orbits at altitude range 500 – 900 km [35]. It has ~50 ms one-way single hop propagation delay and 1.5 – 2 hours orbit rotation period.

3.2 Characteristics of GEO satellite links

GEO satellite links have characteristics that differ from terrestrial links [38]. These characteristics contribute to the degradation of TCP performance in satellite links.

3.2.1 Long propagation delay

GEO satellite links have one-way long propagation delays (~250 ms) due to high satellite altitudes. The RTT of a satellite link is at least 500 ms depending on the satellite inclination. Long propagation delays of GEO satellite links prevent TCP connections from rapidly achieving high transmission rates.

3.2.2 Large bandwidth delay product

The bandwidth delay product (BDP) defines the amount of data a protocol should have unacknowledged (in–flight) in order to fully utilize the available link capacity. For a satellite link, the BDP is the product of the satellite link capacity C and the RTT:

$$BDP = RTT \times C, \tag{3.1}$$

where C and RTT are measured in b/s and s, respectively. GEO satellite links have a large BDP due to long propagation delays and large bandwidth.

3.2.3 High bit error rates

Links employed for fixed satellite communication exhibit high bit error rates (BERs) $\sim 10^{-6}$ compared to terrestrial wired links. The high BERs are caused by propagation losses, noise, and interference from other services sharing the same frequency band. The BERs may become as large as 10^{-3} or 10^{-2} because of extreme weather conditions [39]. Propagation loss components and their loss contribution in various satellite frequency bands for GEO satellite links are shown in Table 3.2 [35]. Losses occur in GEO satellite links due to high BERs.

Table 3.2. Propagation mode losses in dB that affect GEO satellite links. Free space path loss is the dominating contributor.

Propagation mode loss	L band (1.6/1.5) GHz	C band (6/4) GHz	Ku band (14/12) GHz	Ka band (30/20) GHz
free space path	187	196	205	210
atmospheric	0.1	0.2	0.3	0.5
rain attenuation	0.1	0.5	2	6
refraction	6	3	2	1
diffraction	6 – 12			
ionospheric	3 – 6	1 – 3	< 1	

3.2.4 Bandwidth asymmetry

Transmission over a satellite link employs different frequencies for the uplink and downlink paths. The available bandwidth depends on the volume of incoming and outgoing traffic in the uplink and downlink, respectively. Hence, the satellite uplink and downlink capacities may differ [40]. Bandwidth asymmetry also occurs in hybrid terrestrial satellite networks that employ both a slow terrestrial link and a satellite link for outgoing traffic and incoming traffic, respectively.

CHAPTER 4: TCP PERFORMANCE IN SATELLITE NETWORKS

The connection management, flow control, and congestion control features of TCP make it robust in terrestrial networks that use packet loss as an indication of congestion. TCP carries over 90% of Internet traffic [12] that represents a dominant portion of the entire telecommunication network traffic [7], [8]. Broadband GEO satellite networks allow users to access Internet based applications and services regardless of the users' degree of mobility [41]. These networks support high data rates and multimedia services. Hence, TCP must be able to provide optimal performance in GEO satellite networks. In this Chapter, we discuss the issues of TCP in GEO satellite networks and present a survey of solutions that have been proposed to improve TCP performance.

4.1 Impact of GEO satellite link characteristics on TCP performance

4.1.1 Effect of long propagation delay on slow start and RTO mechanism

In a GEO satellite network, the RTT of a TCP segment exceeds 500 ms when combined with the terrestrial network delays. During the slow start phase, TCP needs to receive an ACK of a sent segment in order to increase the *cwnd*. TCP is unable to reach the maximum achievable throughput during the slow start phase due to the long propagation delays of GEO satellite links. The time required to reach a transmission rate B b/s is given as

$$T_{slow\ start} = RTT \times (1 + \log_2 (B \times RTT / l)), \qquad (4.1)$$

21

where l is the average TCP segment length in bits [42]. The times spent in the slow start phase for various transmission rates and satellite types for an average TCP segment length of 1 kB (a common size) are shown in Table 4.1.

Table 4.1. Duration of the slow start phase for various transmission rates for a GEO satellite link with RTT 500 ms. The higher the transmission rates, the longer the time spent in the slow start phase.

Transmission rate (Mb/s)	$T_{slow\ start}$ (s)
1	3.47
1.5	3.76
10	5.13
45	6.21
155	7.1

For the complete download 200 kB file using a 5 Mb/s GEO satellite link, a throughput <500 kb/s is achieved. If a 5 Mb/s LEO satellite link is employed for the complete download of the same file size, 1 Mb/s throughput is obtained [43]. Hence, the available satellite capacity is under-utilized.

4.1.2 Effect of large BDP on TCP window size

Large BDP values imply that large amount of unacknowledged data in flight should be available for TCP to maximally utilize the available network capacity. The BDP of a GEO satellite link depends on the maximum allowable unacknowledged data (*rwnd*) [44]. As shown in the TCP header segment in Figure 2.1, the maximum *rwnd* value is limited to 16 bits (64 KB). For an *rwnd* value of 64 KB, a GEO satellite link with standard E1 rate (2,048 kb/s) and RTT value of 500 ms may achieve only ~1,048 kb/s and, hence, making the available capacity underutilized.

4.1.3 Effect of high BERs on TCP congestion control algorithms

A major cause of the poor performance of TCP in heterogeneous networks characterized by high BERs is the assumption of segment loss as an indication of congestion. The congestion control algorithms respond to segment loss by deflating the *cwnd*, resulting in degraded throughput if losses are not due to congestion.

4.1.4 Effect of bandwidth asymmetry on TCP congestion control algorithms

A TCP sender increases its transmission rate based on the timely reception of ACKs of transmitted segments. A bandwidth-asymmetric satellite network is configured with high downlink (receiver to satellite) bandwidth and low uplink (satellite to receiver) bandwidth. Data segments are transmitted to the receiver through the downlink path and ACKs are forwarded to the sender through the uplink path. The low bandwidth uplink path may become easily congested leading to delay of ACKs, which causes a TCP sender to transmit data segments in bursts. Hence, bandwidth asymmetry in satellite networks results in traffic burstiness [45]. TCP throughput is also degraded when the delay or loss of ACKs in a low bandwidth uplink path results in the expiration of the RTO timer (misinterpreted as an indication of congestion). Hence, the data transmission rate is reduced accordingly.

4.2 Survey of proposed solutions for improving TCP performance

Solutions proposed for improving TCP performance in GEO satellite networks are classified as end-to-end, split-connection, link-layer, or non-TCP satellite-optimized protocols.

4.2.1 End-to-end TCP solutions

End-to-end solutions usually require modifications only at the TCP sender and/or receiver. They may also require that intermediate routers support priority mechanisms. End-to-end solutions maintain the end-to-end semantics of TCP. Hence, they preserve security of transmitted information when network layer security is employed. They also prevent termination of TCP connections when an intermediate node suffers an irrecoverable loss. End-to-end solutions include extensions to standard TCP mechanism or TCP variants and non-TCP satellite optimized transport protocols.

4.2.1.1 Extensions to standard TCP mechanisms

The TCP time stamps option [38], [46] allows a TCP sender to place a timestamp in each transmitted TCP segment using the TCP echo option. The receiver then returns the timestamp in the appropriate field of the corresponding ACKs using the TCP echo reply option. TCP window scale option [38], [46] defines a scale factor that expands to 32 bits the default 16-bit window size field in the TCP header. However, for an error-prone satellite link, enabling this option increases the probability of segment loss per window. The TCP window scale option may also lead to TCP *seqno* wraparound problem [47]. This problem arises when the same 32-bit *seqno* is reused within a single TCP connection. Protect Against Wraparound Sequence (PAWS) numbers mechanism [46] employs the TCP echo and echo reply time stamps options to enable a sender differentiate between segments having the same *seqno* by examining their time stamps. The path maximum transmission unit (MTU) discovery [38], [48] technique is used to probe network for the maximum segment size that could be supported along the end-to-end path without fragmentation. Path MTU increases the rate at which the *cwnd* opens.

However, delays up to multiple RTTs are incurred during the probing period [47]. Larger segment sizes are also more prone to loss and corruption.

The option of increasing the initial window (IW) [19], [20] allows the initial *cwnd* value to be larger than one SMSS and less than 4 × SMSS, and, thus reducing the time needed to reach a transmission rate during slow start by up to 3 RTTs [45]. Modification to the *ssthresh* and packet spacing during slow start [49] enable a TCP sender to enter congestion avoidance phase and reduce the effect of bursty traffic thus, preventing buffer overflow that may lead to losses in GEO satellite links. The selective acknowledgement (SACK) [18] option was proposed to improve TCP performance in the presence of multiple losses in an RTT. The SACK option allows a receiver to indicate only segments that were received. Hence, the sender may explicitly retransmit only the lost segments thus, reducing the number of unnecessary retransmissions.

4.2.1.2 TCP variants

For new connections starting after an idle period, TCP fast start [50] employs cached values of the most recent past TCP connection state variables (*cwnd, ssthresh, srtt,* and *rttvar*). However, it requires sender-side modifications and packet prioritization mechanism at intermediate routers to drop low priority segments when congestion occurs. The IPv6 datagram allows additional fields to be included in extension headers for specific purposes. The extension headers are placed before the encapsulated TCP payload. Bandwidth aware TCP (BA-TCP) [51] is an end-to-end solution that employs the IPv6 extension headers with fields for round-trip propagation delay (RTPD) and available bandwidth (ABW) to relay explicit network conditions to a TCP receiver. However, BA-TCP requires IPv6 hosts to be present in the network. Sharing TCP

(STCP) [52] was proposed to mitigate the effect of long propagation delays by sharing TCP state information (*ssthresh* and *cwnd*) among sequential and concurrent TCP connections. However, STCP requires an additional data structure in order to store the shared information.

TCP-Peach [53] and its variant TCP-Peach+ [54] introduce the sudden start and rapid recovery and the jump start and quick recovery algorithms based on the use of low priority segments (dummy and nil segments, respectively) to probe the network for available bandwidth. The *cwnd* is set based on the estimated available bandwidth. TCP-Peach and TCP-Peach+ require packet prioritization mechanism at every intermediate router along the data transmission path.

TCP Westwood (TCPW) [55] is an end-to-end sender side modification of TCP congestion control algorithms for estimating the available bandwidth in the computation of the *cwnd*. The value of the *cwnd* during congestion avoidance and after a packet loss is computed using the bandwidth estimate (BWE). Adaptive start (Astart) [56] is a satellite network modification to the slow start algorithm for adaptively resetting the *ssthresh* based on the BWE of TCPW. Astart prevents premature termination of the slow start phase and enable the *cwnd* to grow rapidly without incurring the risk of buffer overflow and multiple losses. It assumes that the *rwnd* is always large so that the sending rate depends only on the *cwnd*. TCP bulk repeat [57] improves TCPW performance in the presence of heavy losses due to link errors.

HighSpeed TCP [58] and Scalable TCP [59] are variants proposed for large BDP networks such as Gigabit Ethernet WANs. HighSpeed TCP adaptively increases or decreases the *cwnd* is as a function of the current *cwnd* when an ACK is received or when

a segment is lost, respectively. Scalable TCP adjusts the *cwnd* by a factor $\alpha = 0.01$ upon receipt of an ACK and by a factor $\beta = 0.125$ when segment loss is detected. Both HighSpeed TCP and Scalable TCP require sender-side modifications. However, the parameters used in adjusting the *cwnd* are not optimized for GEO satellite networks [60].

TCP-Swift [61] replaces the slow start and fast recovery algorithms with speedy start and speedy recovery algorithms. The speedy start algorithm enables the *cwnd* to open rapidly within two RTTs while the speedy recovery algorithm sends outstanding segments instead of dummy segments (used in TCP-Peach) to probe the network for available bandwidth. TCP priority-based congestion control strategy (TCP PBS) [62] introduces accelerative start and expeditious recovery algorithms. The accelerative start is similar to the speedy start but sets the IW to min (4 × SMSS, max (2 × SMSS, 4380 bytes)) [24] instead of one SMSS. The expeditious recovery employs explicit error notification (EEN) to distinguish congestion losses from error losses. Both TCP-Swift and TCP PBS require priority mechanisms at all intermediate routers in the data transmission path.

TCP-Star [63] implements the following three new mechanisms: congestion window setting (CWS), lift window control (LWC), and acknowledgment error notification (AEN). The CWS is employed to determine the cause of losses in order to adjust the *cwnd* accordingly. The LWC increments the *cwnd* during slow start and congestion avoidance phases based on available bandwidth estimation mechanism of TCP-Jersey [64]. The AEN prevents unnecessary retransmission of segments caused by ACK losses or delays. TCP Hybla [65] employs a time-scale modification algorithm to increment *cwnd* independent of RTTs during the slow start and congestion avoidance

phases. However, it assumes that the transmission rate does not depend on the *rwnd*. The TCP Hybla algorithm employs the SACK and timestamp options to recover multiple losses and prevent delays in RTO timer update, respectively. TCP New Vegas [66] is a variant of TCP Vegas [67] proposed for GEO satellite networks. It employs packet pacing and rapid window convergence to improve the performance of TCP during slow start. TCP New Vegas also implements packet pairing to reduce the negative impact of delayed ACK [16] on networks with large RTTs.

4.2.2   Split TCP solutions

In hybrid terrestrial satellite IP networks, split connections shield the satellite link characteristics from the terrestrial segment. Satellite-optimized transport protocols are utilized in the satellite segment. The TCP connections are split at intermediate nodes such as gateways [68]. The main disadvantage of split connection is the violation of the end-to-end semantics of TCP. TCP splitting, TCP spoofing, and web caching are common methods of split TCP solutions.

4.2.2.1 TCP splitting

In TCP splitting, the TCP connection between end-hosts is divided into two or more sections with each section representing a complete TCP connection [69]. TCP segments transmitted from one section to another require buffering at satellite gateways or intermediate nodes. Aeronautical transport control protocol (AeroTCP) [39] employs TCP splitting. Performance enhancing proxies (PEPs) [70] are examples of intermediate nodes at which TCP connections are split. SaTPEP [71], PEPsal [72], and Secure PEP (SPEP) [73] are examples of PEPs that propose the use of satellite optimized protocols in

the satellite section while the TCP variants such as TCP Reno or TCP NewReno may be used for the terrestrial wired section. SPEP requires an intermediate node to add and remove the SPEP header before data and ACK segments are forwarded to the destination and source, respectively. A preferential suppression (PS) scheme [74] with a PEP that employs TCP spoofing is proposed to increase efficiency and achieve fairness in resource sharing between satellite and terrestrial TCP connections.

4.2.2.2 TCP spoofing

TCP spoofing requires an intermediate host, usually a satellite gateway, to prematurely acknowledge TCP segments received from the terrestrial wired section. Unlike TCP splitting, a satellite-optimized TCP is not employed on the satellite section [70]. The intermediate host buffers all transmitted segments, suppresses the ACKs from a TCP receiver, and does not forward them to the sender. When a segment loss is detected, the intermediate host retransmits the lost TCP segment. Performance evaluation of TCP spoofing [75] indicates that TCP spoofing improves TCP throughput at a TCP sender for large file transfers. TCP spoofing is shown in Figure 4.1.
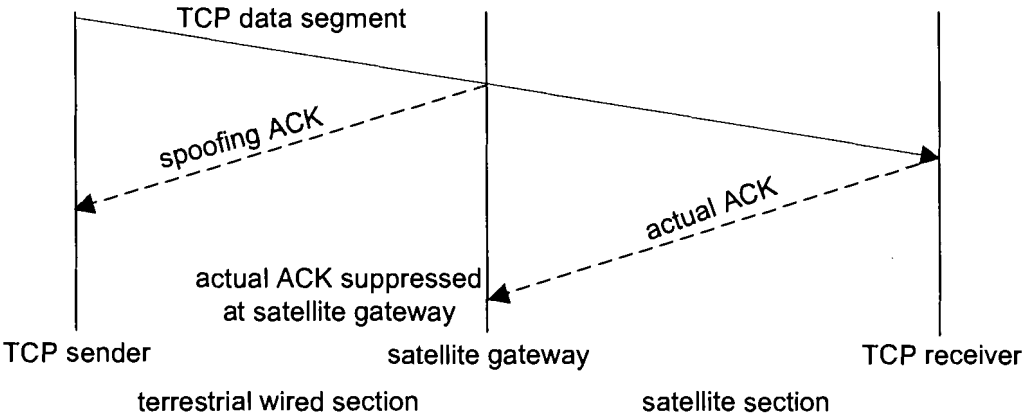
Figure 4.1. TCP spoofing. The spoofing ACK reaches a TCP sender quickly to reduce the estimated RTT at the sender and the actual ACK is suppressed at the satellite gateway.

4.2.2.3 Web caching

In a satellite IP network, web caches are employed to fill requests from remote web servers in the Internet. Most commonly requested information is stored locally in the web cache. When a request is subsequently received from a user, the locally cached information is sent to the user. Hence, the connection is effectively split at the web cache thereby reducing traffic to remote Internet web servers and also reducing user-perceived response time [68], [69]. Web caches require per-connection buffers for every active connection and large processing overhead to establish and release connections in order to accommodate a large number of users.

4.2.3 Link layer solutions

In order to improve TCP performance in the presence of high BERs, link layer solutions are proposed to provide reliability at the link layer. These are broadly classified as TCP-aware and TCP-unaware [76]. TCP-aware link layer solutions modify the TCP header information and are incompatible with applications that require IP security (IPSEC). TCP-unaware solutions employ forward error correction (FEC) and automatic repeat request (ARQ) techniques to detect and retransmit lost or corrupted packets at the link layer.

An example of a TCP-aware link layer solution is presented in [77]. Snoop protocol [78] is an example of a TCP-aware link layer protocol evaluated for GEO satellite link [79]. The snoop protocol is implemented at intermediate nodes such as gateways. It performs local retransmissions of unacknowledged segments stored in buffers at the gateways. The snoop protocol requires modifications at intermediates nodes as well as large buffers to store unacknowledged segments.

30

An example of a TCP-unaware link layer solution is presented in [80]. TCP packet control [81] is a link layer solution that addresses variable and sudden long propagation delays that a packet may experience in satellite networks. It requires modifications at intermediate routers in the network.

4.2.4 Non-TCP satellite-optimized transport protocols

The characteristics exhibited by satellite communication links have led to proposals of other transport layer protocols. These protocols may employ standard TCP or reliable non-TCP mechanisms to address congestion losses and additional extensions to address segment losses attributed to satellite link characteristics. The transport layer protocols may be employed in satellite sections of split TCP connections.

The Space Communications Protocol Standards-Transport Protocol (SCPS-TP) [82] employs extensions and enhancements to TCP such as selective negative ACK (SNACK) and explicit Internet control message protocol (ICMP) messages for corruption-induced losses and link outages. It also employs timestamps and modified delayed ACK options to compute ACK delays based on the estimated RTT. SCPS-TP with the TCP Vegas option for slow start improves throughput and is less sensitive to link delays than TCP [83]. Simulation results show that SCPS-TP has better performance in noisy asymmetric satellite networks than TCP [84].

Satellite transport protocol (STP) [85] is based on an Asynchronous Transfer Mode (ATM) link layer protocol known as service specific connection orientated protocol (SSCOP) [86]. STP does rely on timeout mechanisms. It employs an automatic repeat request (ARQ) mechanism that uses SNACK for retransmission. The ACK polling

cycle of STP is used for probing and early error detection [87] and, thus, it performs well in determining the cause of losses. The rate control scheme (RCS) [88] employs low priority dummy packets to probe the network for available resources and requires all routers to support priority schemes for discarding these packets when congestion occurs.

Explicit control protocol (XCP) [89] and its enhanced variant P-XCP [90] employ explicit feedback to determine network conditions and decouple utilization and fairness control. Simulation results show that an XCP PEP is able to utilize available bandwidth faster than TCP variants such as TCP Reno or TCP NewReno [91]. However, modifications are required at the sender, all intermediate routers, and the receiver. Stream Control Transmission Protocol (SCTP) [92], [93] employs the TCP congestion control algorithms, satellite extensions as defined in [38], [46], and other unique features such as multistreaming and multihoming.

CHAPTER 5:TCP WITH ADAPTIVE DELAY AND LOSS RESPONSE

5.1 Overview

In this thesis, we propose TCP with adaptive delay and loss response (TCP-ADaLR) algorithm for heterogeneous networks employing GEO satellite links [94]. It is designed to improve TCP performance in the presence of long propagation delays, high BERs, and delayed ACK. We implement TCP-ADaLR algorithm as an extension to TCP SACK. It may also be applied to TCP NewReno. Both TCP SACK and TCP NewReno are common Internet TCP implementations [20].

TCP-ADaLR algorithm maintains the TCP end-to-end semantics. It requires modifications only at the sender. TCP-ADaLR employs an initial window (IW) of $2 \times$ SMSS. The TCP-ADaLR algorithm modifications include a scaling component ρ and mechanisms for adaptive window (*cwnd* and *rwnd*) increase and loss recovery. The scaling component ρ depends on measurements taken from sample RTT segments (*sampleRTT*). A TCP sender computes RTO values continuously from the *sampleRTT* collected using the Karn's algorithm [25] that ensures that the *sampleRTT* is measured from a segment that is not retransmitted. We normalize the *sampleRTT* by 1 s (a common value of the minimum RTO in TCP implementations with a coarse grained timer [26]). The scaling component is calculated as

$$\rho = (sampleRTT\ s/1\ s) \times 60. \tag{5.1}$$

Hence, the scaling component depends on a variable that reflects the recent network condition (*sampleRTT*). The value of 60 is the minimum recommended value for the maximum RTO *rto_max* [26] normalized by 1 s. The *rto_max* is the upper limit on the retransmission interval that a TCP sender will wait before retransmission. The lower bound of p is set to 1 to ensure that TCP employs the standard algorithm for connections with extremely short RTTs. Conversely, we set an upper bound of p to be 60 to ensure that the value of p is not too large. The scaling component p mitigates the negative effect of the long propagation delays on achieving high transmission rates rapidly, as described in Sections 4.1.1 and 4.1.2. The default exponential TCP *cwnd* increments are increased by the scaling component p. Hence, with the reception of each ACK, the *cwnd* is increased to a larger value than the TCP default thereby allowing transmission of additional segments.

5.2 The proposed TCP-ADaLR algorithm

5.2.1 Adaptive *cwnd* increase mechanism

After the three-way handshake is completed, we divide the slow start phase into four sub-phases based on current *cwnd* and the *flightsize* (total outstanding unacknowledged data in the network). We select four sub-phases based on the ratio of the initial value of the *ssthresh* (64KB) and a large value (16KB) of the IW employed by TCP implementations. The *flightsize* is used to ensure that a TCP sender maintains default exponential TCP *cwnd* increments when the number of unacknowledged bytes in the network exceeds fractions of the *rwnd* during the four slow start sub-phases. In each sub-phase, the increment in *cwnd* depends on the value of p and the presence or absence of losses during transmission. To determine if losses have occurred, we initialize

34

snd_recover (the sequence number denoting the end of fast recovery for TCP NewReno) to zero at the beginning of the connection. If the value of *snd_recover* is nonzero, it implies that at least one segment loss has occurred during transmission. The sub-phases are described in Algorithm 5.1.

// *snd_max* = maximum send sequence number (the newest unacknowledged sequence number)

// *snd_una* = sequence number of the first unacknowledged segment (the oldest unacknowledged sequence number.)

// *snd_recover* = sequence number denoting the end of fast recovery (initialized to zero at the beginning of the connection)

// *acked_bytes* = number of bytes acknowledged by an ACK

flightsize = snd_max - snd_una;

// slow start phase

if (*cwnd* < *ssthresh*)

{

if ((*cwnd* $\leq$ *ssthresh*/4) && (*flightsize* < *rwnd*/4))
    set sub-phase = slow start sub-phase 1

if ((*cwnd* > *ssthresh*/4) && (*cwnd* $\leq$ *ssthresh*/2) && (*flightsize* < *rwnd*/4))
    set sub-phase = slow start sub-phase 2

if ((*cwnd* > *ssthresh*/4) && (*flightsize* $\geq$ *rwnd*/4) && (*flightsize* < *rwnd*/2))
    set sub-phase = slow start sub-phase 3

if ((*cwnd* > *ssthresh*/2) && (*flightsize* $\geq$ *rwnd*/4) && (*flightsize* < *rwnd*/2))
    set sub-phase = slow start sub-phase 4

}

Algorithm 5.1. Pseudo-code that describes the four sub-phases of the TCP slow start phase introduced by the adaptive *cwnd* increase mechanism.

In the default TCP implementation, the growth of the *cwnd* may not be exponential as desired when the delayed ACK option is enabled because the receiver may delay sending ACKs. It may send a single ACK for more than one data segment received. This delay exacerbates the long propagation delays of a GEO satellite link and the

number of acknowledged bytes may be larger than one SMSS. Hence, an increase of one

SMSS will not compensate for the case when the number of bytes acknowledged is larger

than one SMSS. We select a breakpoint value $\rho = 15$, corresponding to a *sampleRTT* of

250 ms. This value was selected based on measurements from simulation of a FTP file

download of a 50 MB file in an ideal (no congestion or satellite link error losses) satellite

link using TCP SACK. The simulation results are shown in Table 5.1. The download

response time begins to show significant increase after 250 ms. For values $\rho \geq 15$, we

increase the *cwnd* by integral multiples of the SMSS when no losses have occurred,

which allows better utilization of the network bandwidth.

Table 5.1. FTP download response time for 50 MB file used to determine the instance when propagation delay impacts the file download.

RTT (ms)	Download response time (s)
25	251.9
50	252.1
100	252.5
200	253.5
250	272.7
500	470.1

The *cwnd* is incremented exponentially for $\rho < 15$, as in default TCP slow start

phase. For $\rho \geq 15$, we increase the *cwnd* by ($\sqrt{\rho}/4$) × SMSS when no losses have

occurred. A delayed ACK of two outstanding segments will cause the transmission of 4

back-to-back segments that may result in micro-burstiness [95]. The value ($\sqrt{\rho}/4$) is

selected to accommodate the non-linear increase of the download response times shown

in Table 5.1 and micro-burstiness that may occur with delayed ACK enabled. Note that

($\sqrt{\rho}/4$) × SMSS lies in the range (1 − 2) × SMSS. A value of up to 2 × SMSS prevents

large line-rate bursts at the beginning of the connection when the TCP sender is probing the network and is recommended to accommodate the delayed ACK option enabled [95]. Based on the maximum value of ρ (60), increments of ($\sqrt{\rho}/4$) × SMSS also maintain a modest bursts size < 10 segments with a low probability of losing a segment [96].

If losses occur or the conditions of the four sub-phases are not met, the *cwnd* is increased as in the default TCP slow start phase. During the congestion avoidance phase, the *cwnd* is incremented linearly immediately after fast recovery or when the *flightsize* is larger than *rwnd/2*. When the *flightsize* is less than *rwnd/2*, the *cwnd* is incremented by ($\sqrt{\rho}/2$) × SMSS. Since increments during congestion avoidance phase are linear and more conservative, we double the *cwnd* increment to ($\sqrt{\rho}/2$). Note that if the value of ($\sqrt{\rho}/4$) or ($\sqrt{\rho}/2$) is less than one, it is rounded up to one, thus, being incremented by one SMSS as in the default TCP. Alternating the *cwnd* increments between one SMSS and ($\sqrt{\rho}/4$) × SMSS in the slow start sub-phases enables the TCP sender to smoothen out its transmission rate while utilizing the available bandwidth better than in default TCP. With the adaptive increase mechanism, the TCP sender can achieve higher transmission rates faster during the slow start phase especially when the RTT is large as in the case of GEO satellite links. The *cwnd* is incremented in slow start and congestion avoidance as shown in Algorithm 5.2.

```
// slow start phase
if (cwnd < ssthresh)
   {
   // slow start sub-phase 1
   if ((cwnd ≤ ssthresh/4) && (flightsize < rwnd/4))
      {
      if ((snd_recover = = 0) && (ρ ≥ 15))
         increment cwnd by (√ρ/4) × SMSS
      else
```

increment *cwnd* exponentially as in TCP Reno
        }
    // slow start sub-phase 2
    else if (((*cwnd* > *ssthresh*/4) && (*cwnd* ≤ *ssthresh*/2) && (*flightsize* < rwnd/4))
        {
        if ((*snd_recover* = = 0) && (ρ ≥ 15))
            increment *cwnd* by ($\sqrt{\rho}$/4) × SMSS
        else
            increment *cwnd* exponentially as in TCP Reno
        }
    // slow start sub-phase 3
    else if (((*cwnd* > *ssthresh*/4) && (*flightsize* ≥ *rwnd*/4) && (*flightsize* < *rwnd*/2))
        {
        if ((*snd_recover* = = 0) && (ρ ≥ 15))
            increment *cwnd* by ($\sqrt{\rho}$/4) × SMSS

        else
            increment *cwnd* exponentially as in TCP Reno
        }
    // slow start sub-phase 4
    else if (((*cwnd* > *ssthresh*/2) && (*flightsize* ≥ *rwnd*/4) && (*flightsize* < *rwnd*/2))
        {
        if ((*snd_recover* = = 0) && (ρ ≥ 15))
            increment *cwnd* by ($\sqrt{\rho}$/4) × SMSS

        else
            increment *cwnd* exponentially as in TCP Reno
        }
    }
else
    // congestion avoidance phase
    if (*cwnd* > *ssthresh*)
    {
    // fast recovery phase
        if (((*snd_una* ≤ *snd_recover*) && (ρ ≥ 15) && (*snd_recover* != 0)) ||
            (*flightsize*   ≥ *rwnd*/2))
            increment *cwnd* by SMSS × SMSS / *cwnd*

    // exiting the fast recovery phase
        else if (((*snd_una* ≥ *snd_recover*) && (ρ ≥ 15) && (*snd_recover* != 0)) ||
            (*flightsize* < *rwnd*/2))
            increment *cwnd* by ($\sqrt{\rho}$/2) × SMSS × SMSS / *cwnd*

    else
            increment *cwnd* by SMSS × SMSS / *cwnd*
    }

Algorithm 5.2. Pseudo-code that describes the adaptive *cwnd* increase mechanism. This mechanism allows the *cwnd* to increase more rapidly than default TCP and transmit additional segments during the slow start and congestion avoidance phases.

5.2.2 Adaptive *rwnd* increase mechanism

The TCP receiver-side limit imposed on the amount of transmitted data is defined by *rwnd*. The number of transmitted bytes is the minimum of either the *cwnd* or *rwnd*. The adaptive *rwnd* increase mechanism is implemented at the sender side. The *rwnd* increments depend on the value of p, *flightsize*, *cwnd* increment phase (slow start or congestion avoidance), and the presence or absence of losses. The adaptive *rwnd* increase mechanism compensates for the GEO satellite long propagation delays when no losses occur. It allows at least one additional segment to be transmitted when losses occur and a partial ACK is received. (A partial ACK acknowledges some of the lost segments in a window and allows the TCP sender to send the next unacknowledged segment while remaining in fast recovery phase.) The additional segment compensates for delayed ACK and allows the next two unacknowledged segments to be sent to recover from losses. This prevents serial retransmission timeouts from occurring if only one segment is recieved and the TCP receiver waits for the delayed ACK timer to expire before sending the ACK of the received segment. The adaptive *rwnd* increase mechanism modifies the *rwnd*, as shown in Algorithm 5.3.

rtt_dev_gain = RTT deviation gain

```
if (p ≥ 15)
    {
    if (flightsize > rwnd)
        {
        do nothing
        }
    // congestion avoidance phase
    else if (cwnd > ssthresh)
        {
        // no losses have occurred
        if (snd_recover = = 0)
            set rwnd to rwnd + rtt_dev_gain × p × SMSS
```

```
// losses have occurred
// fast recovery phase
else if (($snd\_una \leq snd\_recover$) && ($snd\_recover ! = 0$))
    set rwnd to rwnd + SMSS

    else
        do nothing
    }

// slow start phase
else if ($cwnd < ssthresh$)
    {
    // no losses have occurred
    if ($snd\_recover == 0$)
        set rwnd to rwnd + $rtt\_dev\_gain \times \rho \times$ SMSS

    // losses have occurred
    // fast recovery phase
    else if (($snd\_una \leq snd\_recover$) && ($snd\_recover ! = 0$))
        set rwnd to rwnd + SMSS

    else
        do nothing
    }

    else
        do nothing

}
```

Algorithm 5.3. Pseudo-code that describes the *rwnd* modification introduced by the adaptive *rwnd* increase mechanism.

5.2.3 Loss recovery mechanism

TCP-ADaLR modifies the fast recovery phase to compensate for delayed ACK and ensure quicker recovery from losses. TCP-ADaLR also modifies the exponential backoff after the first expiration of the RTO timer to prevent its premature expiration when delayed ACK is enabled. During the fast recovery phase, the minimum value of *cwnd* is set to 2 × SMSS instead of 1 × SMSS when the number of acknowledged bytes is greater than the current value of *cwnd*. This prevents the *cwnd* from shrinking to zero. By adjusting the value of *cwnd* at least two back-to-back segments may be transmitted during the fast recovery to ensure that the TCP sender is able to receive ACKs faster

when the delayed ACK option is enabled. If an RTO occurs, we add 200 ms to the current time to prevent premature expiration of the RTO timer that may lead to false retransmissions when the delayed ACK option is enabled. We also limit the number of retransmissions from the retransmission buffer to three segments per retransmission to prevent spurious or unnecessary retransmissions. The algorithm for setting *cwnd* during fast recovery phase is shown in shown in Algorithm 5.4.

```
// fast recovery phase
if (snd_una > snd_recover)
    {
    if (cwnd ≤ acked_bytes)
        set cwnd to 2 × SMSS

    else
        // deflate the congestion window by the number
        // of acknowledged bytes and add two SMSS
        set cwnd to cwnd - acked_bytes + (2 × SMSS)
    }
```

Algorithm 5.4. Pseudo-code that describes the loss recovery mechanism for setting the *cwnd* during the fast recovery phase.

CHAPTER 6: TCP-ADaLR IMPLEMENTATION IN THE OPNET MODELER NETWORK SIMULATOR

TCP-ADaLR is implemented in the Optimized Network Engineering Tools (OPNET) modeler [19], an object-oriented discrete event simulator for network simulation, modelling, performance evaluation, and analysis. In this Chapter, we describe the OPNET modeler and the OPNET implementation of TCP-ADaLR [97].

6.1 OPNET Modeler

OPNET modeler includes a library of standard network node models and Open Systems Interconnection (OSI) layer protocols' models implemented in Proto-C that is based on C/C++ programming language. OPNET modelling environment consists of a three-level hierarchical domain editors and additional specialized editors for specifying characteristics of a modelled system's behaviour at any of the three domain levels [98].

6.1.1 Project editor

A network model consists of communication nodes and links. The network domain is the highest modelling level in OPNET. The project editor is a graphical interface used in the network domain to define and edit the topology and architecture of a network model within a geographical or logical context. In a geographical context, physical positions of network nodes are specified using latitude/longitude of the world or regional maps for comparison with real-world network deployment. In cases where real-world comparison is not essential for modelling the network, network nodes may be

specified using logical dimensions and/or xy coordinates. An OPNET network model may consist of three basic objects: subnetworks, nodes, and links. OPNET supports three types of subnetworks: fixed, mobile, and satellite.

6.1.2 Node editor

Network nodes may represent any type of network device such as workstations, bridges, routers, or switches. A node model represents the architecture of a network device and the connectivity between its functional elements (modules defined by processes). The network editor is used to specify node parameters (attributes) for applications, protocols, and physical resources that define the node behaviour. Nodes may be fixed, mobile, or satellite. Fixed nodes remain at a specified location during simulation. Mobile nodes may have predefined trajectories that specify the path of motion during simulation. Satellite nodes may have predefined orbit attributes that describe their motion during simulation.

6.1.3 Process editor

The functionalities of modules in a network node are defined using the process editor in Proto-C language [98] and finite state machines (FSMs). Proto-C language supports protocol and algorithm development with a combination of state transition diagrams (STDs) and a library of high-level commands known as kernel procedures. Processes are represented by FSMs. FSMs are represented by STDs. A process model STD describes a set of states that a process may enter and conditions known as transitions required to exit and enter into another state. Each state has an enter executive and an exit executive. They are Proto-C code lines executed by a process when it enters or exits a

state, respectively. Processes may be hierarchical with a parent process invoking a new instance of a predefined process known as a child process. The OPNET editor modelling hierarchy is shown in Figure 5.1.

project editor

process editor   node editor
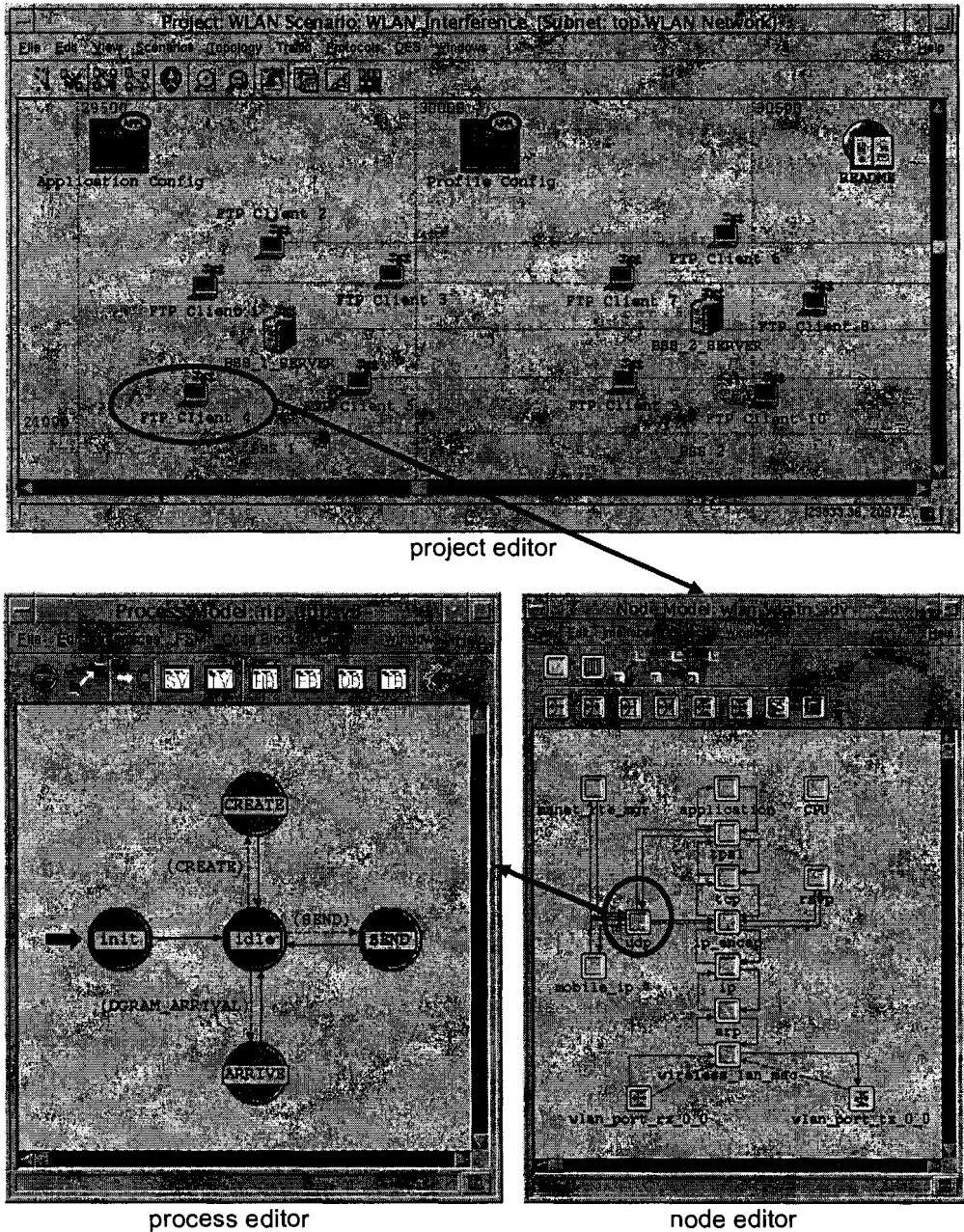
Figure 6.1. Hierarchy of OPNET modelling domain editors. The project editor is used to define network topology. The node and process editors are used to define node and process functions, respectively.

6.1.4 Specialized editors

Network model specification may require other specialized editors that are used to define various data models that are referenced by the project, node, or process editors during simulation. These specialized editors include link, antenna pattern, packet format, modulation curve, filter, probe, interface control information (ICI), and probability density function editors. An OPNET simulation scenario is a completely defined network model including all specifications by required editors.

6.2 OPNET implementation of TCP-ADaLR

We implemented TCP-ADaLR in the OPNET simulation tool [19]. We modified the TCP sender to implement TCP-ADaLR. The TCP sender is represented by the OPNET *Ethernet server advanced* model shown in Fig. 6.2. We modified the TCP process model associated with the server node model. The OPNET TCP process model implements standard TCP features [2], [9], [17]. It also includes additional features such as SACK [18], delayed ACK [14], explicit congestion notification (ECN) [28], and timestamp options [38], [46]. These features are defined in the TCP attributes of the OPNET *Ethernet server advanced* model.

The OPNET TCP implementation consists of a parent process *tcp_manager_v3* and a child process *tcp_conn_v3*. The *tcp_manager_v3* FSM, shown in Fig. 6.3, communicates with the session layer and network (IP) layers. The *tcp_conn_v3* process is invoked by the *tcp_manager_v3* process when a new TCP connection is established. A separate *tcp_conn_v3* process is invoked for each newly established TCP connection. The *tcp_conn_v3* process stores all individual TCP connection parameters in the transmission control block (TCB) and shares them with the *tcp_manager_v3* process. Connection

parameters stored in the TCB include the connection ID, local (sender) TCP port number, remote (receiver) TCP port number, local IP address, and remote IP address. We modified the OPEN state to invoke the modified *tcp_conn_v3* child process. The changes were made in the *tcp_manager_v3.pr.c* file.
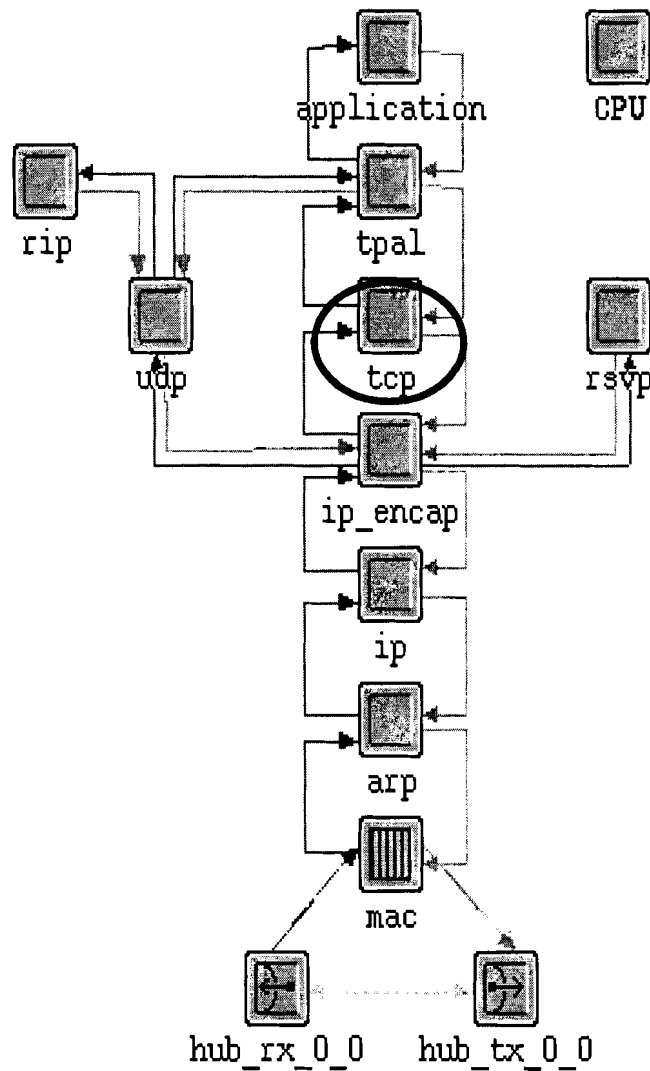
Figure 6.2. The OPNET *Ethernet server advanced* node model. The implementation of TCP-ADaLR requires modifications to the process models in the highlighted TCP module.
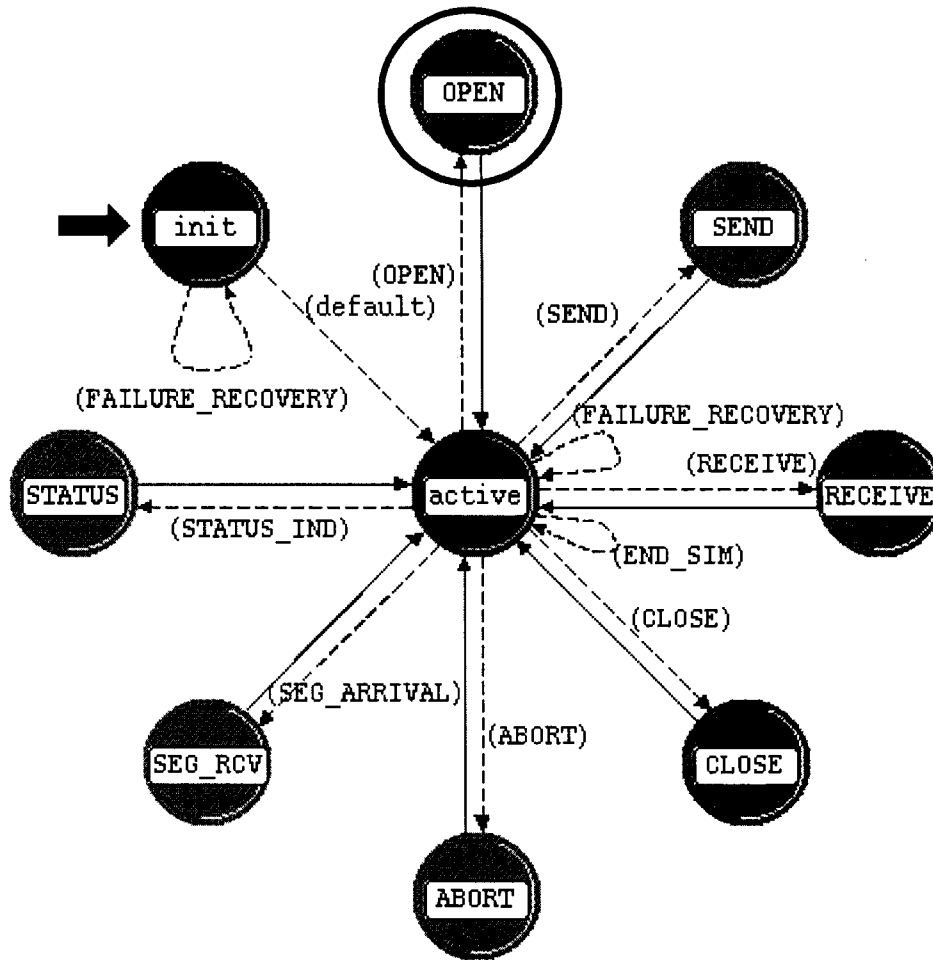
Figure 6.3. The *tcp_manager_v3* process model. The dashed lines represent conditions to transition from the state they originate from to the state they terminate. The solid lines represent transitions without conditions between the states they connect.

Additional modifications were made to functions defined in the *tcp_conn_v3* child process model. We defined the scaling component p as a global variable of type double accessible by all functions defined in the *tcp_conn_v3* process. The *srtt*, *rttvar*, and RTO values are computed based on the *sampleRTT*, estimated from the Karn's algorithm [25], using the *tcp_rtt_measurements_update* () function. We modified this function in order to compute the scaling component p that depends on the value of *sampleRTT* (5.1).

The *tcp_cwnd_update* () function is used to increment the *cwnd* during the slow start and congestion avoidance phases. We modified this function to implement the adaptive *cwnd* increase mechanism. The *tcp_cwnd_update* () function is also modified to implement the TCP-ADaLR loss recovery mechanism used to set the *cwnd* during the fast recovery phase.

The *tcp_snd_total_data_size* () and *tcp_snd_data_size* () functions are used to compute the number and the size of data segments to be sent after each ACK is received or when data segments are to be retransmitted. The number and size of data segments transmitted is computed from the current *cwnd* value (obtained from the *tcp_cwnd_update* () function) and the value of *rwnd* obtained from the receiver advertised window field in the most recent ACK received. These two functions were modified to implement the adaptive *rwnd* increase mechanism.

The *tcp_timeout_retrans* () function is used to retransmit segments when the RTO timer expires after no ACK has been received. It was modified to compute subsequent RTO timer expirations. We also modified the *tcp_una_buf_process* () function to avoid possible retransmission of unnecessarily large number of segments after a single RTO [17] by limiting the number of retransmissions to three segments when the function is called. The changes were made to the *tcp_conn_v3_pr.c* file. The tcp_*conn_v3* child process model is shown in Figure 6.4.
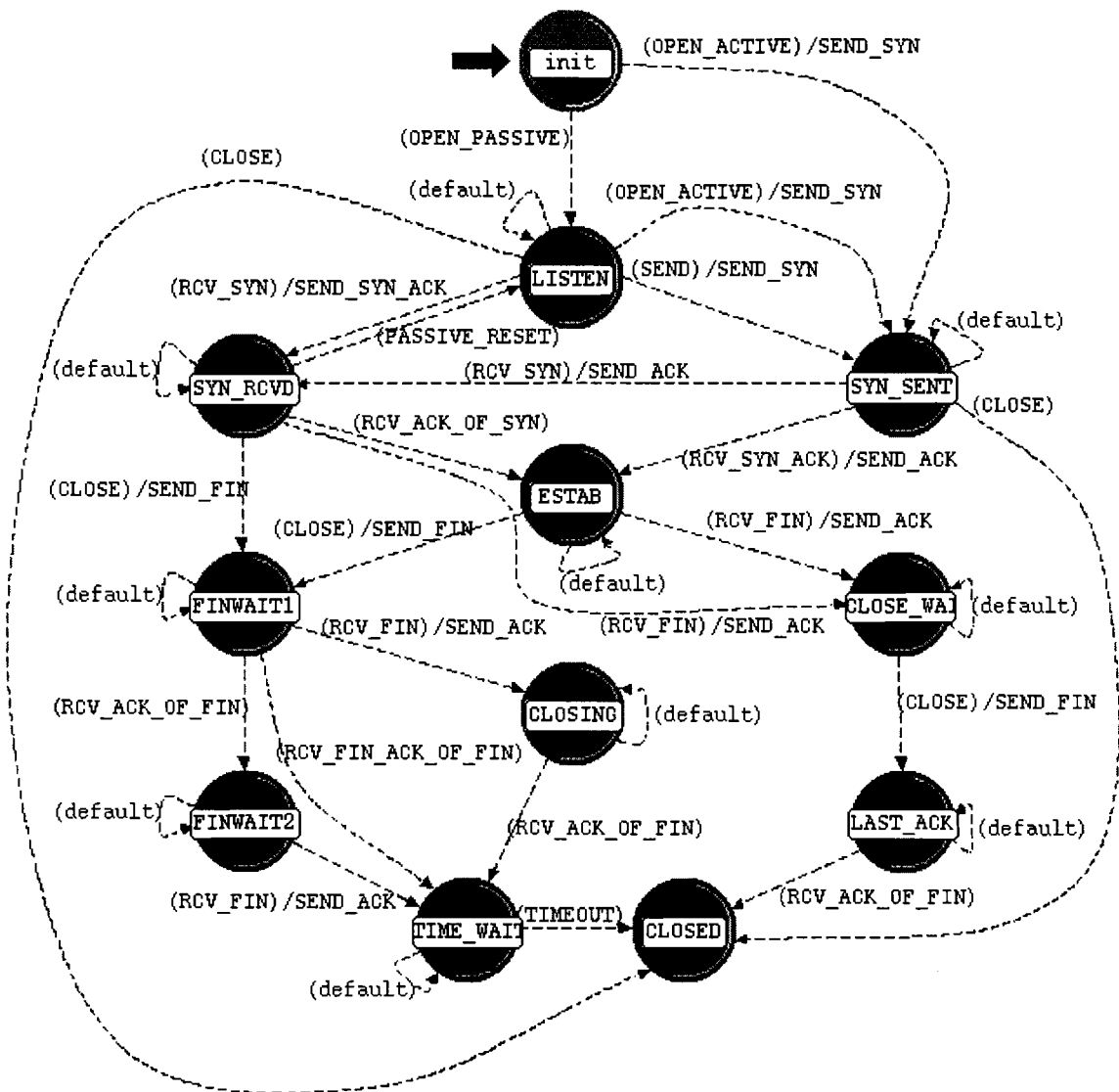
Figure 6.4. The *tcp_conn_v3* process model. The implementation of TCP-ADaLR requires modification to the function block of the process model.

49

CHAPTER 7: PERFORMANCE EVALUATION

We evaluate the performance of TCP-ADaLR using the OPNET network simulator in the absence and presence of errors and congestion. We discuss the error model used to simulate the GEO satellite link. We also discuss the simulated network topologies, simulation parameters, simulation scenarios, and performance metrics. We then discuss the simulation results.

7.1 Error model

The GEO satellite link is modelled as an additive white Gaussian noise (AWGN) memoryless (uncorrelated bit errors) channel. We select this model because the satellite client is a fixed user that has a line-of-sight (LoS) to the GEO satellite [65]. The satellite link exhibits random (uncorrelated) errors at various bit error rates (BERs) from 10^{-5} to 10^{-10} after forward error correction (FEC). Using the AWGN model, we calculate the packet error rate (PER) as

$$PER = 1 - (1-BER)^N, \qquad (7.1)$$

where N is the number of bits in the packet. We use 1,500 bytes (N = 12,000 bits) Ethernet packets. The error correction (ECC) threshold is the highest proportion of bit errors in a packet accepted by a receiver and forwarded to its output stream. The ECC threshold is equal to the PER when the BER is 10^{-10}. We use 10^{-10}, which is acceptable for wired Ethernet links. Packets with errors exceeding the ECC threshold are discarded by the receiver (lost). The ECC threshold is set to 1.2×10^{-6} as computed using (7.1).

Various BERs and their corresponding PERs for Ethernet packet size are shown in Table 7.1. For satellite links, typical average BER ranges from 10^{-5} to 10^{-8} [99]. An ideal satellite link is assumed to be error-free. The OPNET *PPP workstation advanced* model (TCP receiver) was modified to define the ECC threshold for accepted packets, as shown in Figure 7.1.

Table 7.1. Post-FEC BERs and corresponding PERs for the AWGN-modelled GEO satellite link calculated using (7.1).

BER	PER
10^{-9}	1.2×10^{-5}
10^{-8}	1.2×10^{-4}
10^{-7}	1.2×10^{-3}
10^{-6}	1.2×10^{-2}
10^{-5}	1.2×10^{-1}
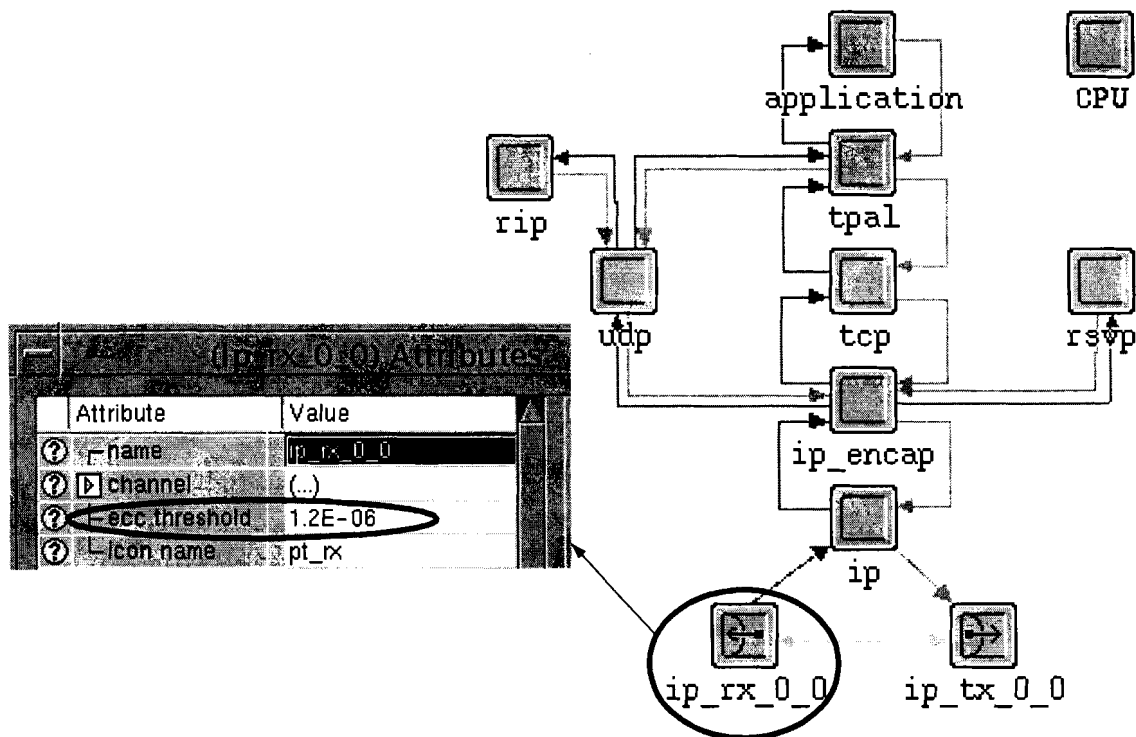
Figure 7.1. The OPNET *PPP workstation advanced* node model. We modified the transmission receiver (ip_rx_0_0) to set the error correction threshold for accepted packets.

7.2 Network topology

We consider two network topologies to evaluate various performance metrics for TCP-ADaLR. The network topology shown in Figure 7.2 is a typical hybrid terrestrial-satellite network connecting a fixed client (receiver) to a gateway through a GEO satellite in a bent-pipe configuration. The receiver has a line-of-sight (LoS) to the GEO satellite. The gateway is connected to the terrestrial wired network. This network topology is used to provide Internet service to home and corporate users. Demand for data and multimedia applications for home and corporate users increased by 6.6% and 9% (2001–2004), respectively, and is still increasing [8].
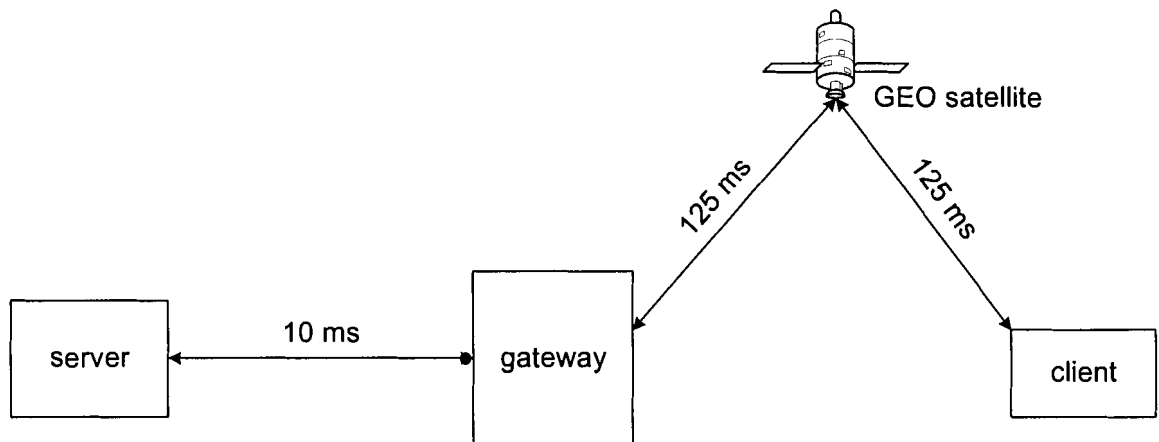
Figure 7.2. Network topology for direct to user hybrid terrestrial-satellite network. The shown link propagation delays are one-way.

The shown link propagation delays are one-way and remain constant during simulation, unless otherwise stated. The GEO satellite link between the client and the gateway is bi-directional with data rate of 2,048 kb/s in the downlink direction (satellite to client) and 256 kb/s in the uplink direction (client to satellite). This difference in the downlink and uplink capacities captures bandwidth asymmetry, a common characteristic of satellite link [20], [32]. Data transmission over satellite links employs downlink

bandwidth of the order 10 times or more in magnitude than the uplink [40]. The downlink and uplink paths have identical propagation delays. The terrestrial wired link between the gateway and the server is a full duplex 10 Mb/s Ethernet link. In all simulations scenarios, the Ethernet link is error-free.

7.3   Simulation scenarios and parameters

We employ the network topology shown in Figure 7.2 to evaluate performance of TCP-ADaLR in various scenarios. We consider an ideal case with no losses and cases with congestion losses only, with losses only due to satellite link errors, and, finally, with losses due to both congestion and satellite link errors. For the scenarios with congestion losses, we set finite buffer sizes of the gateway to 15 and 25 packets for FTP and HTTP applications, respectively. We modify the IP attributes of the OPNET *Ethernet4 slip8 gateway* node model in order to set the buffer size for the FTP application, as shown in Figure 7.3. In the scenarios with satellite link errors, we evaluate the performance of TCP variants for the PERs shown in Table 7.1. TCP-ADaLR considers cases when delayed ACK is enabled by the Internet hosts. We evaluate the performance of TCP-ADaLR without delayed ACK to investigate possible negative effects if the delayed ACK option is disabled.

We simulate both FTP and HTTP applications. The simulation parameters for the FTP and HTTP [100] applications are shown in Tables 7.2 and 7.3, respectively. All TCP variants use constant and identical parameters for the simulated Internet applications. TCP parameters used for simulation are shown in Table 7.4. We use identical set of parameters when the delayed ACK option is enabled, with the exception of the maximum

ACK delay and maximum ACK segment that are set to recommended values of 0.2 s and

2, respectively [14].
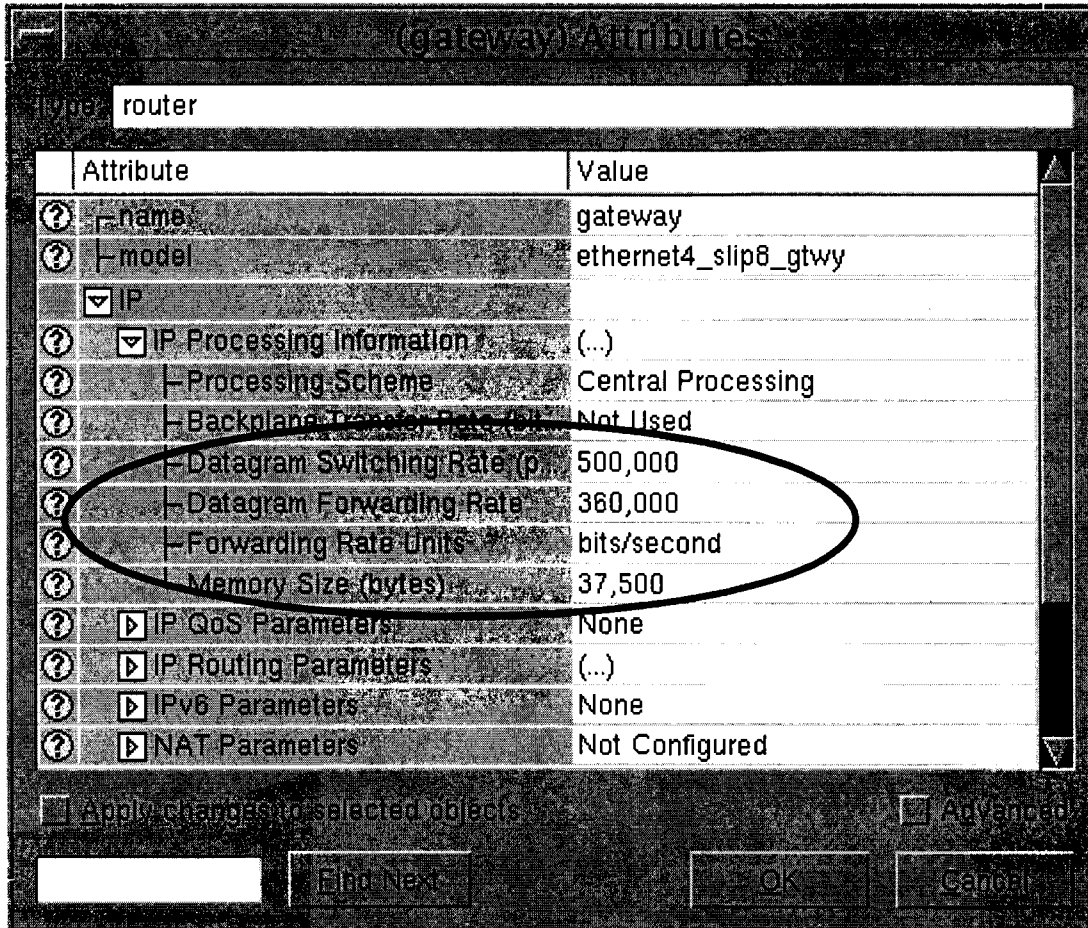
Figure 7.3. The OPNET *Ethernet4 slip8 gateway* model attributes. Shown are the modified IP module parameters employed to simulate congestion losses for the FTP application.

Table 7.2. FTP file download application parameters. The file inter-request time is large to ensure a single file download is completed during each simulation.

Attribute	Value
File inter-request time (s)	18,000
File inter-request time distribution	constant
File size (MB)	50
File size distribution	constant
Simulated time (hours)	5

Table 7.3. Simulated HTTP webpage download parameters.

Attribute	Value
HTTP specification	HTTP 1.1
Page inter-arrival time (s)	30
Page inter-arrival time distribution	constant
Main page object size (bytes)	10,710
Main page object size distribution	constant
Number of embedded page objects	15
Embedded object size (bytes)	7,758
Embedded object size distribution	constant
Simulated time (s)	1,000

Table 7.4. TCP parameters when the delayed ACK option is disabled (without delayed ACK). All parameters except two remain unchanged when the delayed ACK option is enabled (i.e., with delayed ACK).

TCP Parameters	Value
Sender maximum segment size (SMSS)	1,460 bytes
Slow start initial count	2 SMSS
Receiver's advertised window	65,535 bytes
Timer granularity	0.5 s
Persist time-out	1.0 s
Maximum ACK delay	0.0 s
Maximum ACK segment	1
Duplicate ACK threshold	3
Initial RTO	3.0 s
Minimum RTO	1.0 s
Maximum RTO	64.0 s
Retransmission threshold	6
RTT gain	0.125
RTT deviation coefficient	4
Deviation gain	0.25

7.4 Performance metrics

In various simulation scenarios, we evaluate and compare the performance of TCP-ADaLR, TCP SACK, and TCP NewReno, with and without delayed acknowledgements. TCP SACK and TCP NewReno are common Internet TCP implementations [20]. More than 50% of Internet servers employ TCP SACK [101], [102] while majority of others use TCP NewReno [101]. Hence, we consider two TCP-ADaLR variants: with SACK and with NewReno. (The TCP-ADaLR variants are named TCP-ADaLR SACK and TCP-ADaLR NewReno.) We consider performance metrics for the simulated Internet applications (FTP and HTTP), TCP, and the satellite link. The performance metrics include response times for FTP and HTTP applications, TCP goodput, TCP throughput, and satellite link throughput.

7.4.1 FTP download response time

The download response time is the time elapsed between sending an FTP request to an FTP server and receiving the complete response packet. It includes the signalling delay for the connection establishment and termination. The download response time is an indicator of the user-perceived latency of the FTP file download.

7.4.2 HTTP page response time

The page response time is the time elapsed between sending the HTTP request to the HTTP server and receiving the complete response of the entire web page with all contained embedded objects. The main page object and embedded objects are downloaded using a single TCP connection with HTTP 1.1. The HTTP paging response time is an indicator of the user-perceived latency of the web page retrieval.

7.4.3 TCP goodput

Goodput is the number of original bits (excluding retransmissions) correctly received by the TCP receiver per unit time during the duration of the connection. The received segment sequence number is also used as an indicator of goodput.

7.4.4 TCP throughput

TCP throughput is the traffic transmitted by the TCP sender to the TCP receiver. It is measured at the TCP receiver in terms of bytes as the average bytes per second forwarded by the TCP sender and received by the TCP receiver.

7.4.5 Satellite link throughput

Satellite link throughput, measured in b/s, is the average number of bits correctly received by the satellite link (satellite to client direction) for the duration of the file transfer.

7.4.6 Satellite link utilization

Satellite link utilization is the percentage fraction of the available satellite link capacity consumed by the data transmission. It is expressed as the ratio of the number of bits correctly transmitted over the satellite link per unit time and the satellite link data rate.

7.4.7 Calculation of percentage improvement

To compare the performance of TCP-ADaLR SACK and TCP SACK, we calculated the percentage improvement as

$$\frac{\text{TCP} - \text{ADaLR SACK } metric - \text{TCP SACK } metric}{\text{TCP SACK } metric} \times 100. \qquad (7.2)$$

Similarly, the percentage improvement for TCP-ADaLR NewReno was calculated as

$$\frac{\text{TCP} - \text{ADaLR NewReno } metric - \text{TCP NewReno } metric}{\text{TCP NewReno } metric} \times 100. \qquad (7.3)$$

Note that in simulation scenarios given in Sections 7.5.3 and 7.5.4, the largest percentages appear in the regions where the graphs overlap.

7.5 Simulation results

7.5.1 Ideal channel with no congestion or error losses

We first evaluate the performance of TCP-ADaLR in ideal channel conditions with no congestion or error losses. For the FTP application, we evaluate performance in terms of the download response time, TCP goodput, TCP throughput, satellite link throughput, and satellite link utilization. The performance of the HTTP application was evaluated in terms of the HTTP page response time.

7.5.1.1 Performance of FTP application

The download response time for the FTP application is shown in Table 7.5. TCP-ADaLR reduces the download response time of TCP SACK and TCP NewReno by ~23% and ~28% for cases with and without delayed ACK, respectively. The adaptive window (*cwnd* and *rwnd*) increase mechanisms enable TCP-ADaLR to transmit additional segments when there are no losses. TCP-ADaLR without delayed ACK outperforms TCP-ADaLR with delayed ACK by ~7%. The download response times for TCP SACK and TCP NewReno for cases with and without delayed ACK are comparable with a

difference of ~1%. Hence, TCP-ADaLR does not degrade performance of TCP connections without delayed ACK and yields better performance.

Table 7.5. FTP download response times for scenarios with ideal lossless satellite channel. For the case without delayed ACK, TCP-ADaLR shows ~28% shorter FTP download response times than TCP SACK and TCP NewReno.

	Download response time (s)	
Delayed ACK option	Enabled	Disabled
TCP-ADaLR SACK	360.6	333.4
TCP-ADaLR NewReno	360.6	333.4
TCP SACK	470.1	463.5
TCP NewReno	470.1	463.5

The received segment sequence number at the receiver is used as an indicator of the goodput of the ideal channel. The TCP goodput for the FTP application is shown in Figure 7.4. For cases with and without delayed ACK, TCP-ADaLR shows ~49% and ~50% higher goodput than TCP SACK and TCP NewReno, respectively. The TCP throughput is shown in Figure 7.5. For cases with and without delayed ACK, TCP-ADaLR exhibits ~53% and ~63% higher TCP throughput than TCP SACK and TCP NewReno, respectively. TCP-ADaLR is able to open its *cwnd* faster with the adaptive *cwnd* increase mechanism and, hence, transmit additional segments. When the *cwnd* exceeds the *rwnd* and if the modified *rwnd* value permits, the adaptive *rwnd* increase mechanism allows the TCP sender to transmit more segments than TCP SACK and TCP NewReno.
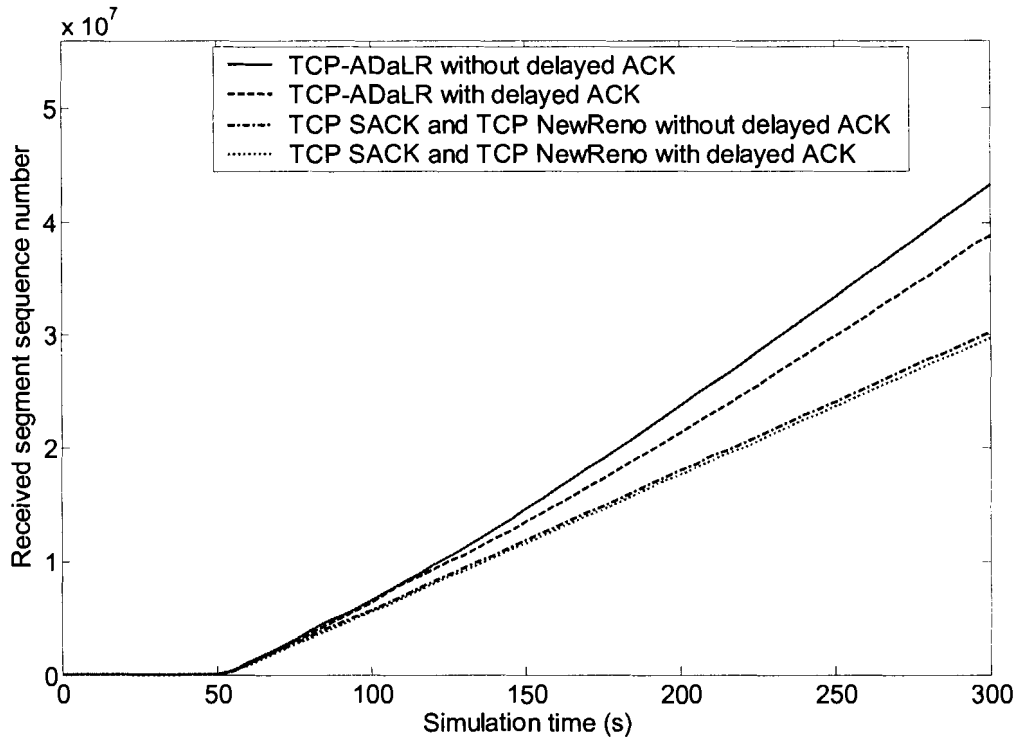
Figure 7.4. Goodput for scenarios with ideal lossless satellite channel. Received segment sequence number is used as an indicator of goodput. TCP-ADaLR exhibits the highest goodput when the delayed ACK option is disabled.
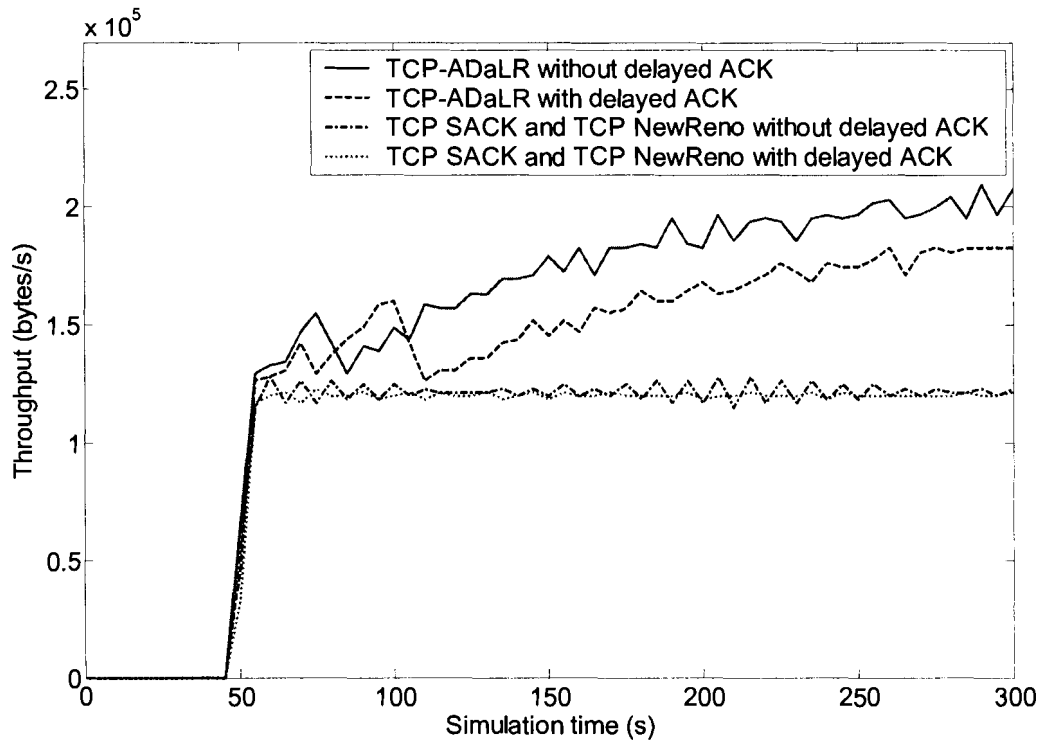
Figure 7.5. TCP throughput for scenarios with ideal lossless satellite channel. For cases without delayed ACK, TCP-ADaLR throughput is ~63% higher than TCP NewReno and TCP SACK.

The satellite link throughput is shown in Figure 7.6. TCP-ADaLR exhibits higher

satellite link throughput than TCP SACK or TCP NewReno. The higher TCP throughput

shown in Figure 7.5, leads to the higher satellite link throughput. The satellite link

utilization is shown in Figure 7.7. TCP-ADaLR exhibits better satellite link utilization

than TCP SACK and TCP NewReno. In the absence of losses, TCP-ADaLR transmits

additional segments more rapidly with the adaptive window (*cwnd* and *rwnd*) increase

mechanisms during the slow start and congestion avoidance phases. TCP-ADaLR

exhibits up to 56% higher satellite link utilization than TCP SACK and TCP NewReno

for the cases with delayed ACK. For the cases without delayed ACK, TCP-ADaLR

exhibits up to 68% higher satellite link utilization than TCP SACK and TCP NewReno.
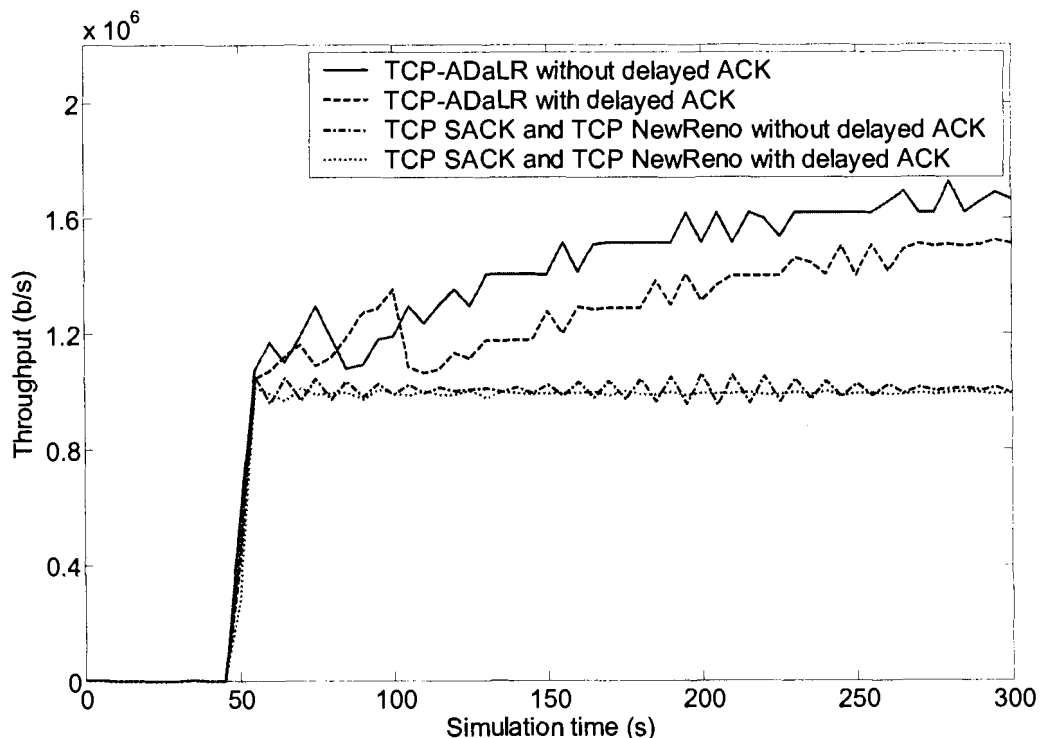
Figure 7.6. Satellite link throughput for scenarios with ideal lossless satellite link. For cases with and without delayed ACK, TCP-ADaLR exhibits ~53% and ~66% higher satellite link throughput than TCP SACK and TCP NewReno, respectively.

Figure 7.7. Satellite link utilization for scenarios with ideal lossless satellite channel. TCP-ADaLR achieves 80% of the link capacity while TCP SACK and TCP NewReno attain only 50% when the delayed ACK option is disabled.

7.5.1.1.1 Effect of increased propagation delay

We also evaluate the effect of increased propagation delay in the terrestrial segment of the network. The Ethernet one-way link propagation delay was increased to 50 ms. TCP-ADaLR outperforms TCP SACK and TCP NewReno even with the increased propagation delay because the *cwnd* increments in the slow start and congestion avoidance phases depend on the scaling component p computed from the TCP connection's RTT. TCP-ADaLR exhibits shorter download response time than TCP SACK and TCP NewReno, as shown in Table 7.6. Increasing propagation delay increases the download response time for all TCP variants. TCP-ADaLR exhibits the highest TCP goodput and TCP throughput shown in Figures 7.8 and 7.9, respectively. Similarly, the satellite link throughput for TCP-ADaLR is the highest, as shown in Figure 7.10. TCP-

ADaLR also exhibits the best link utilization, as shown in Figure 7.11. All TCP variants exhibit lower link utilization because of the increased propagation delay. However, TCP-ADaLR outperforms TCP SACK and TCP NewReno because the scaling component p has the maximum value of 60, equivalent to an RTT of 1 s.

Table 7.6. FTP download response time for scenarios with ideal lossless satellite channel and increased propagation delay. For cases with and without delayed ACK, TCP-ADaLR shows ~26% and ~32% shorter download response time than TCP SACK and TCP NewReno, respectively.

Delayed ACK option	Download response time (s)	
	Enabled	Disabled
TCP-ADaLR SACK	392.6	357.8
TCP-ADaLR NewReno	392.6	357.8
TCP SACK	533.5	526.3
TCP NewReno	533.5	526.3
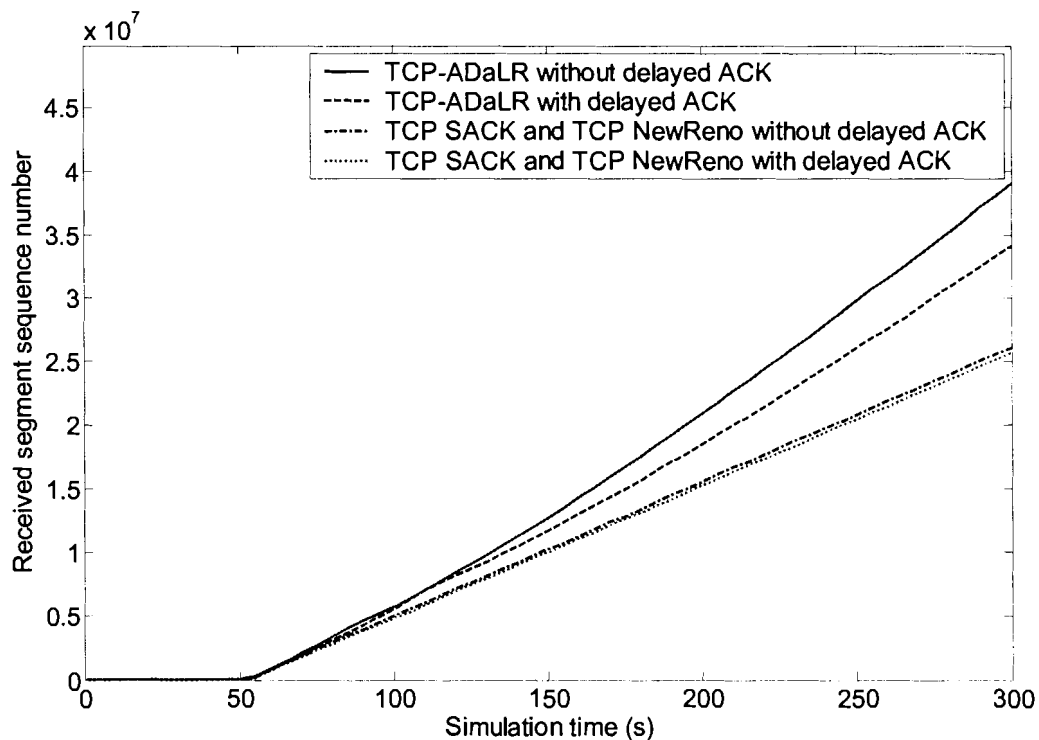
Figure 7.8. Goodput for scenarios with ideal lossless satellite channel and increased propagation delay. For cases with and without delayed ACK, TCP-ADaLR exhibits ~33% and ~50% higher goodput than TCP SACK and TCP NewReno, respectively.
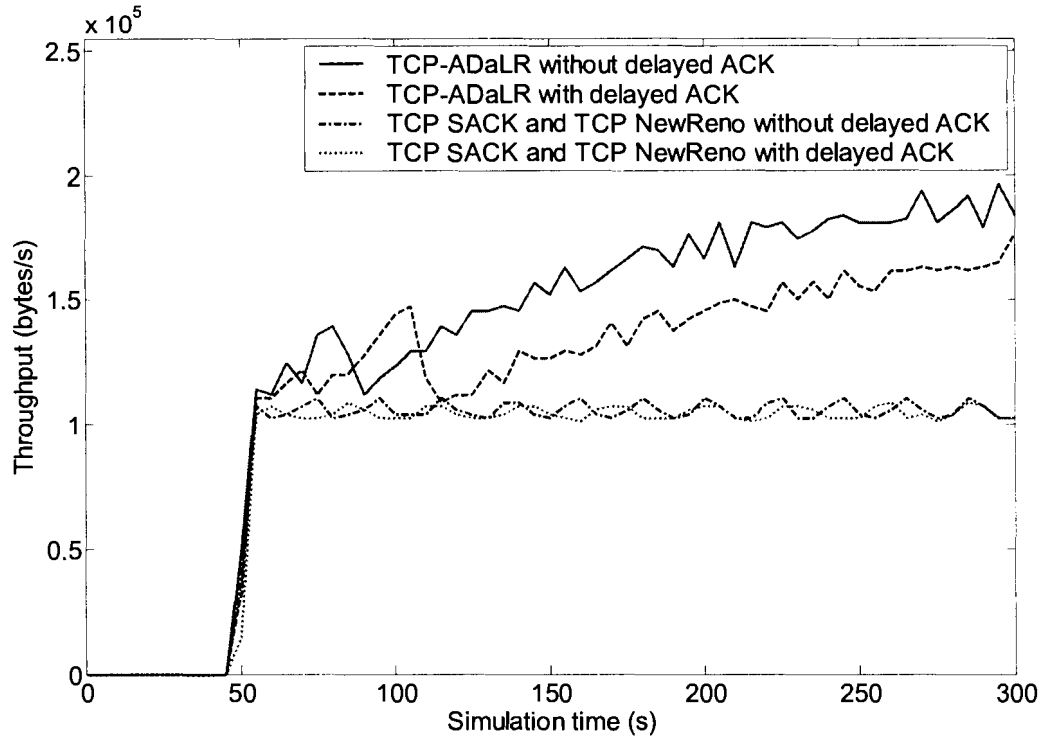
Figure 7.9. TCP throughput for scenarios with ideal lossless satellite channel and increased propagation delay. For cases with and without delayed ACK, TCP-ADaLR exhibits ~71% and ~79% higher TCP throughput than TCP SACK and TCP NewReno, respectively.
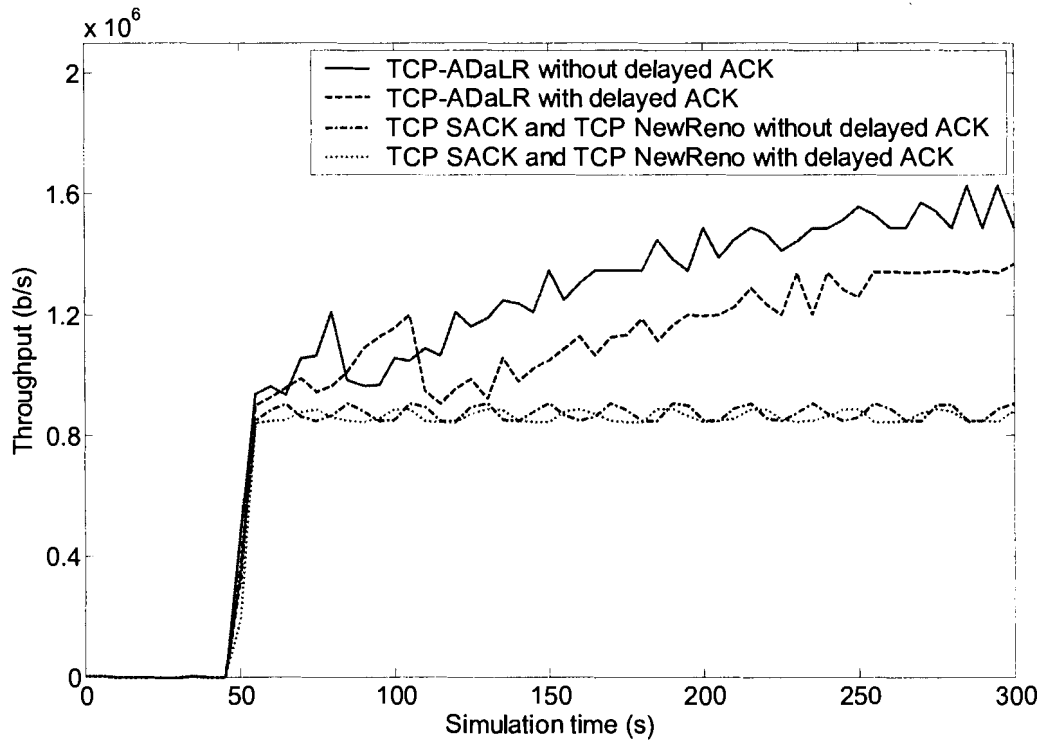
Figure 7.10. Satellite link throughput for scenarios with ideal lossless satellite channel and increased propagation delay. For cases with and without delayed ACK, TCP-ADaLR exhibits ~55% and ~63% higher throughput than TCP SACK and TCP NewReno, respectively.
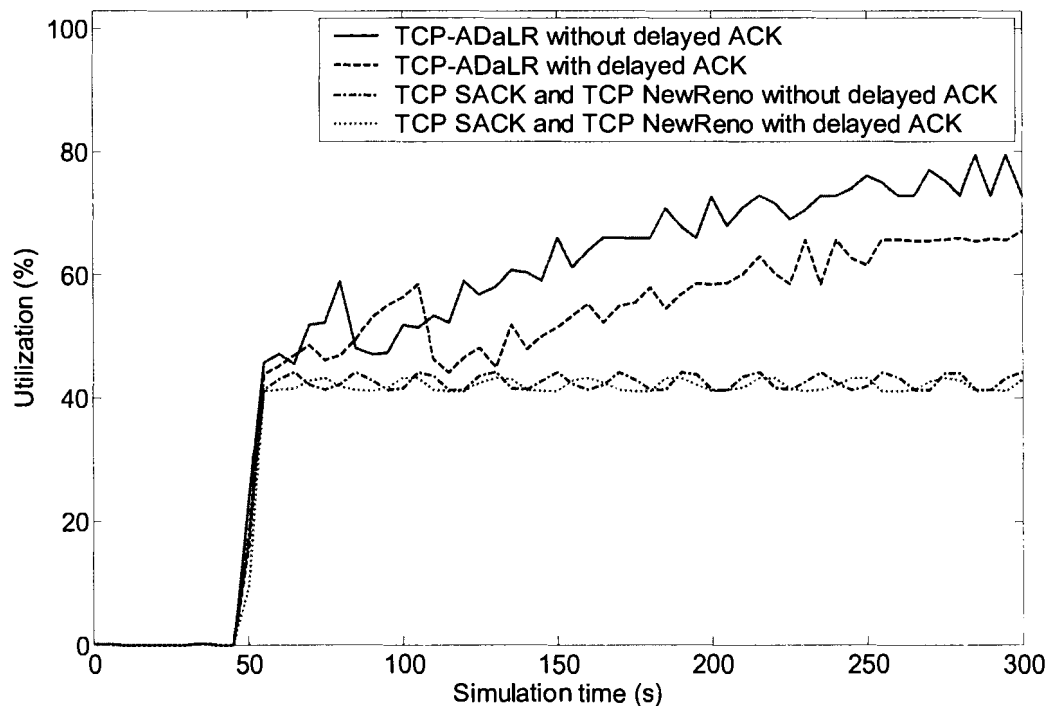
Figure 7.11. Satellite link utilization for scenarios with ideal lossless satellite channel and increased propagation delay. TCP-ADaLR exhibits ~80% peak percentage link utilization when the delayed ACK option is disabled.

7.5.1.1.2 Effect of increased satellite link data rate

We also evaluate the effect of increased satellite downlink and uplink data rates to 10 Mb/s and 1 Mb/s, respectively. TCP-ADaLR exhibits the best performance. TCP SACK and TCP NewReno exhibit comparable performance of 462.3 s and 459.7 s for both the 2 Mb/s link shown in Table 7.5 and 10Mb/s link shown in Table 7.7. TCP-ADaLR better utilizes the higher data rate than TCP SACK and TCP NewReno. High data rate is a characteristic of broadband GEO satellite networks. With increased satellite link data rates (10 Mb/s downlink and 1 Mb/s uplink), TCP-ADaLR shows ~21% and ~29% shorter download response times than TCP SACK and TCP NewReno for cases with and without delayed ACK, respectively. However, TCP SACK and TCP NewReno show only ~1% shorter download response times for cases with and without delayed,

respectively. Correspondingly, TCP-ADaLR shows higher TCP goodput, TCP throughput, satellite link throughput, and satellite link utilization, as shown in Figures 7.12–7.15. With the higher satellite link data rates, ACKs arrive faster at the TCP sender and, hence, TCP-ADaLR opens the *cwnd* and transmits additional segments faster.

Table 7.7. FTP download response time for scenarios with ideal lossless satellite channel and increased satellite link data rates. For cases with and without delayed ACK, TCP-ADaLR shows up to 38% and 48% shorter download response times than TCP SACK and TCP NewReno, respectively.

	Download response time (s)	
Delayed ACK option	Enabled	Disabled
TCP-ADaLR SACK	283.0	235.4
TCP-ADaLR NewReno	283.0	235.4
TCP SACK	462.3	459.7
TCP NewReno	462.3	459.7
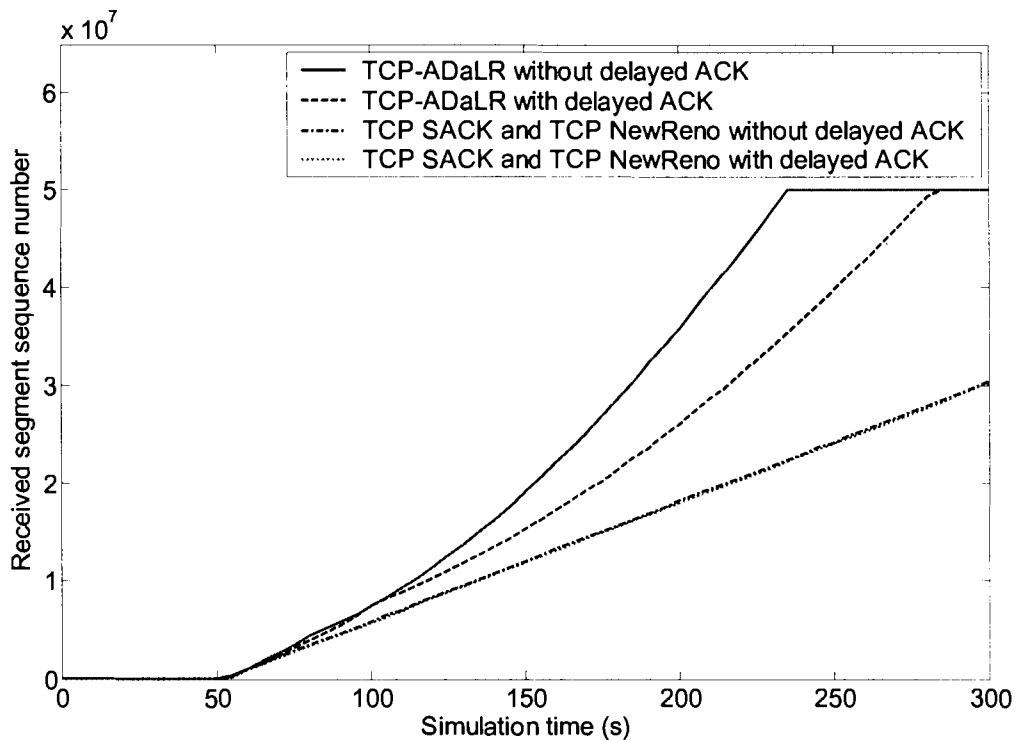
Figure 7.12. Goodput for scenarios with ideal lossless satellite channel and increased satellite link data rates. For cases with and without delayed ACK, TCP-ADaLR exhibits up to 66% and 138% higher goodput than TCP SACK and TCP NewReno, respectively.

66
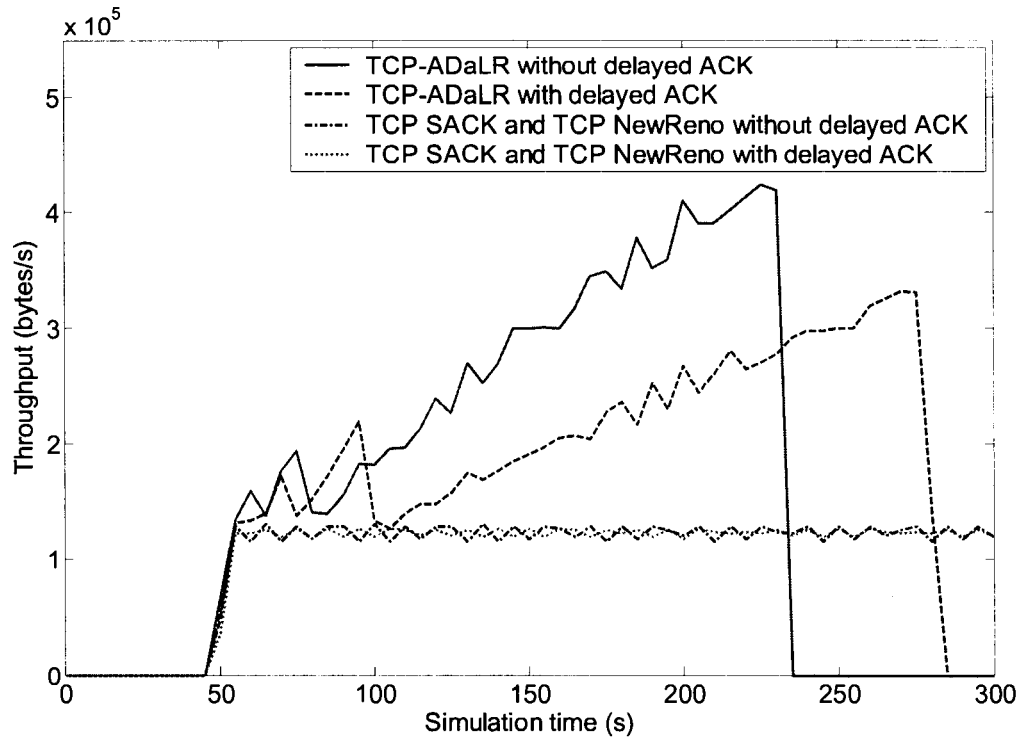
Figure 7.13. TCP throughput for scenarios with ideal lossless satellite channel and increased satellite link data rates. TCP-ADaLR exhibits increasing throughput until the file transfer is completed and the throughput reduces to zero.
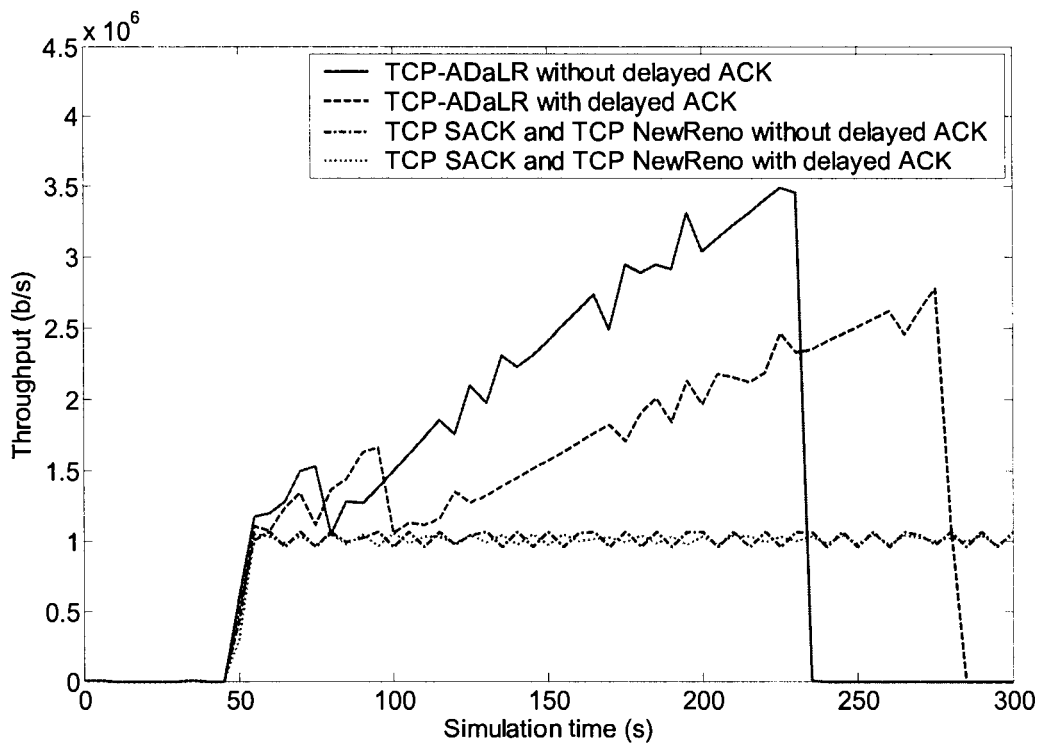
Figure 7.14. Satellite link throughput for scenarios with ideal lossless satellite channel and increased satellite link data rates. For cases with and without delayed ACK, TCP-ADaLR exhibits up to 150% and 250% higher satellite link throughput than TCP SACK and TCP NewReno, respectively.
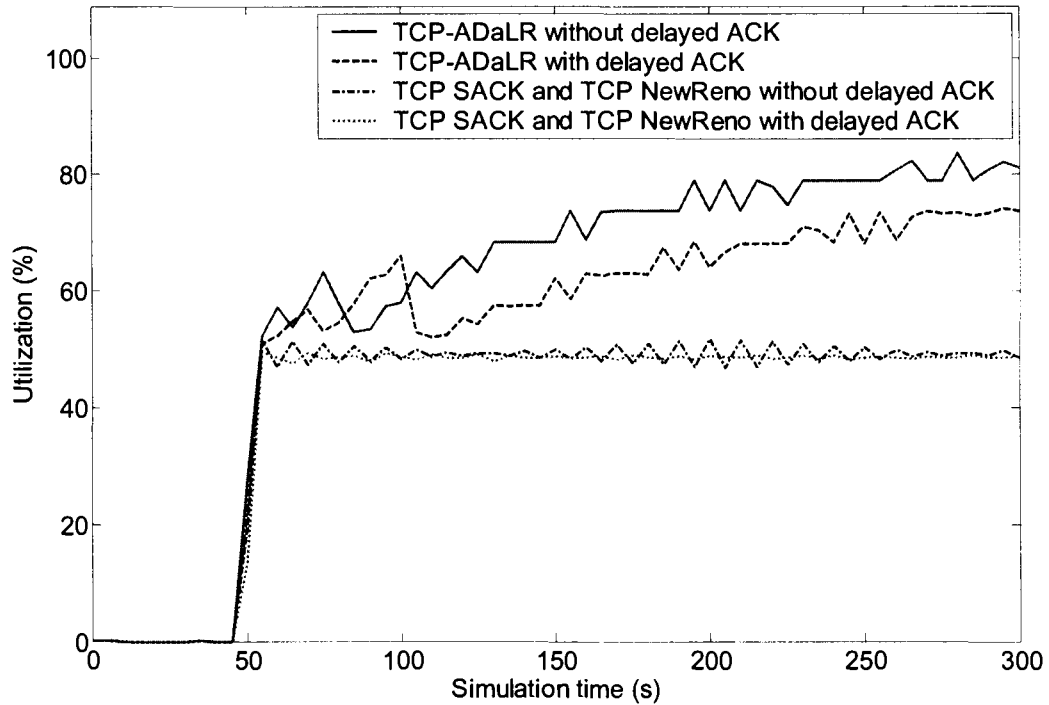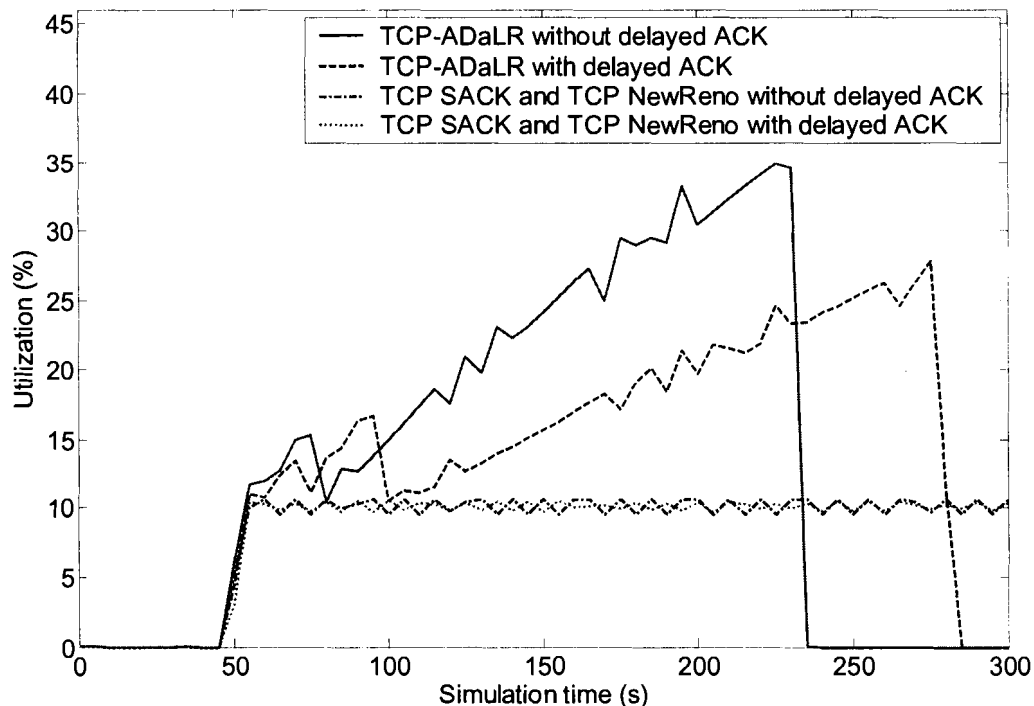
Figure 7.15. Satellite link utilization for scenarios with ideal lossless satellite channel and increased satellite link data rates. TCP-ADaLR exhibits higher utilization than TCP SACK and TCP NewReno.

7.5.1.1.3 Effect of varying FTP application file size

For the FTP application, we investigate the effect on TCP-ADaLR performance by varying file sizes as: 500 kB, 50 MB, 100 MB, 200 MB, 300 MB, 400 MB, and 500 MB. We evaluate the download response time, TCP throughput, satellite link throughput, and satellite link utilization for TCP-ADaLR and TCP SACK (In a scenario with ideal lossless satellite link, TCP SACK and TCP NewReno exhibit the identical performance. Similarly, TCP-ADaLR variants exhibit identical performance). We consider only the case when the delayed ACK option is enabled. The download response times for evaluated file sizes are shown in Figure. 7.16. TCP-ADaLR shows 9%–38% shorter download response times than TCP NewReno. As the file size increases, the download response time increases for both TCP-ADaLR and TCP NewReno.
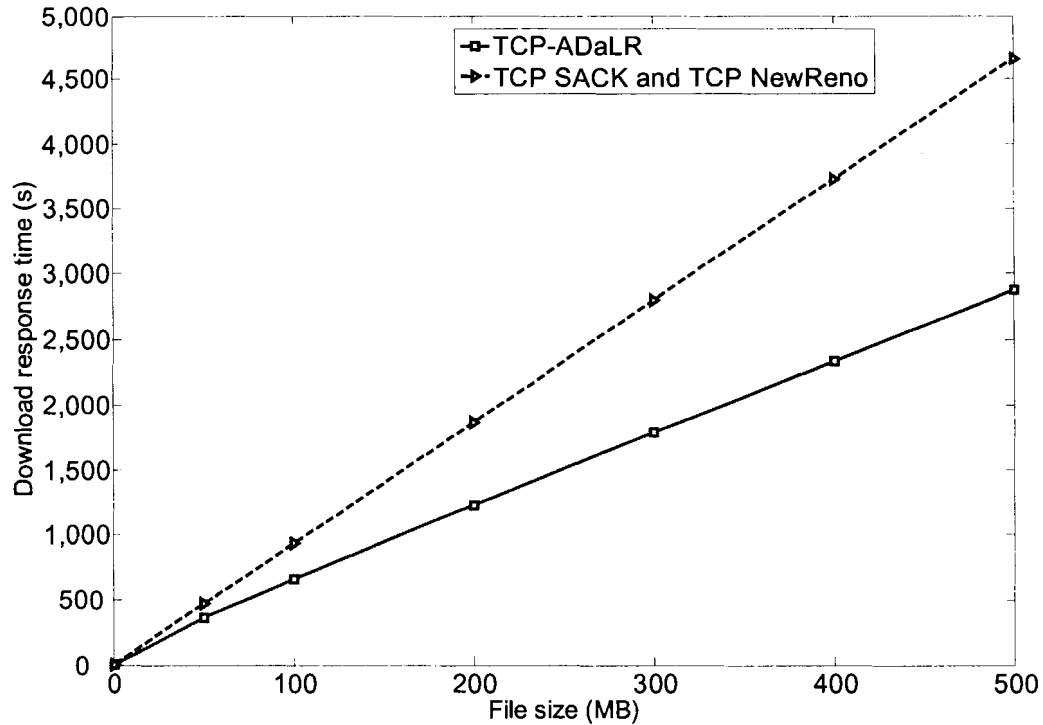
Figure 7.16. As the file size increases, TCP-ADaLR shows shorter download response time than TCP SACK and TCP NewReno.

TCP-ADaLR shows increasing TCP throughput, as shown in Figure 7.17. When the RTT is large, TCP-ADaLR adapts its transmission rate using the scaling component, which enables it to transmit more segments in each RTT with its adaptive *cwnd* increase mechanism. The adaptive *rwnd* increase mechanism also allows additional segments to be transmitted when the *cwnd* exceeds the *rwnd* and the *flightsize* does not exceed the *rwnd*. However, the throughput of the TCP NewReno connection remains unchanged when the file size is increased from 50 MB to 500 MB. When the file size is large, a larger percentage of the file is downloaded during the congestion avoidance phase. The congestion avoidance phase is more conservative with the linear *cwnd* increments. The *rwnd* limits maximum amount of data that can be transmitted when the *cwnd* has exceeded the *rwnd*. The long RTTs further prevent TCP throughput increase.
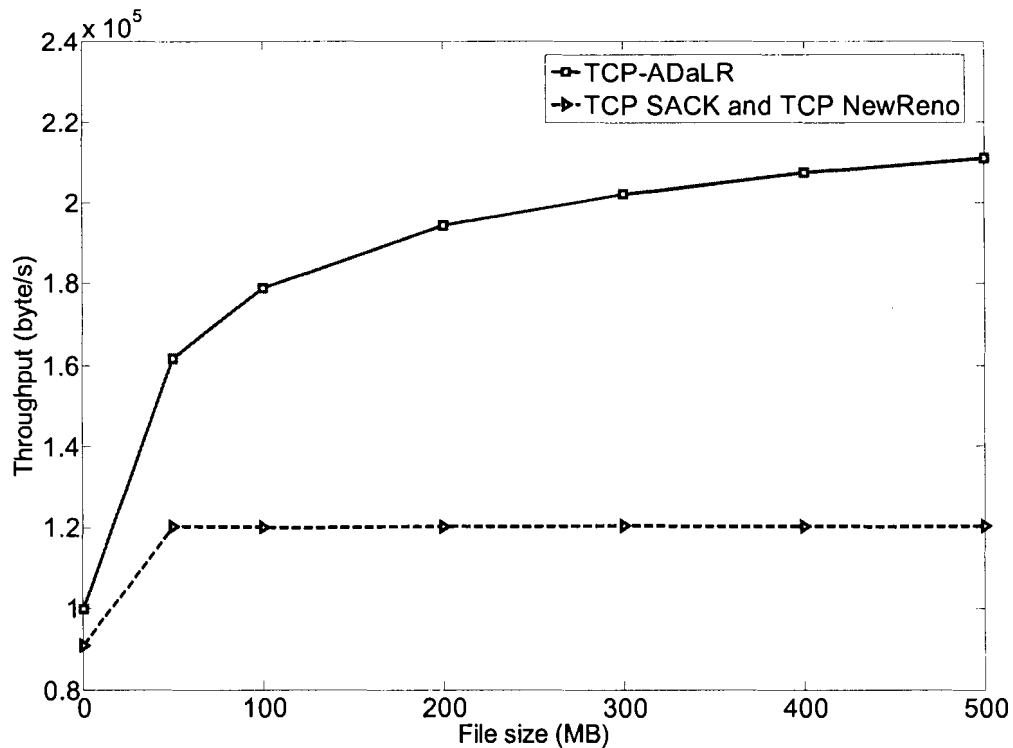
Figure 7.17. As the file size increases, TCP throughput of TCP-ADaLR increases 9%–75% compared to TCP SACK and TCP NewReno.

The satellite link throughput and satellite link utilization are shown in Figures 7.18 and 7.19, respectively. As expected, the satellite link utilization is lowest for TCP-ADaLR and TCP NewReno when the file size is 500 kB (the smallest file size). TCP-ADaLR exhibits increasing satellite link throughput and utilization for all file sizes up to 400 MB. The increasing TCP throughput enables transmission of additional segments, thus leading to the increased satellite link throughput and utilization, as shown in Figures 7.18 and 7.19. However, the satellite link throughput and utilization of the TCP NewReno connection decrease very slightly when the file size is larger than to 50 MB because of the slower transmission rate during congestion avoidance phase. TCP-ADaLR NewReno shows 57%–90% higher satellite link throughput than TCP NewReno. Hence, TCP-ADaLR exhibits performance scalability with increasing file sizes.
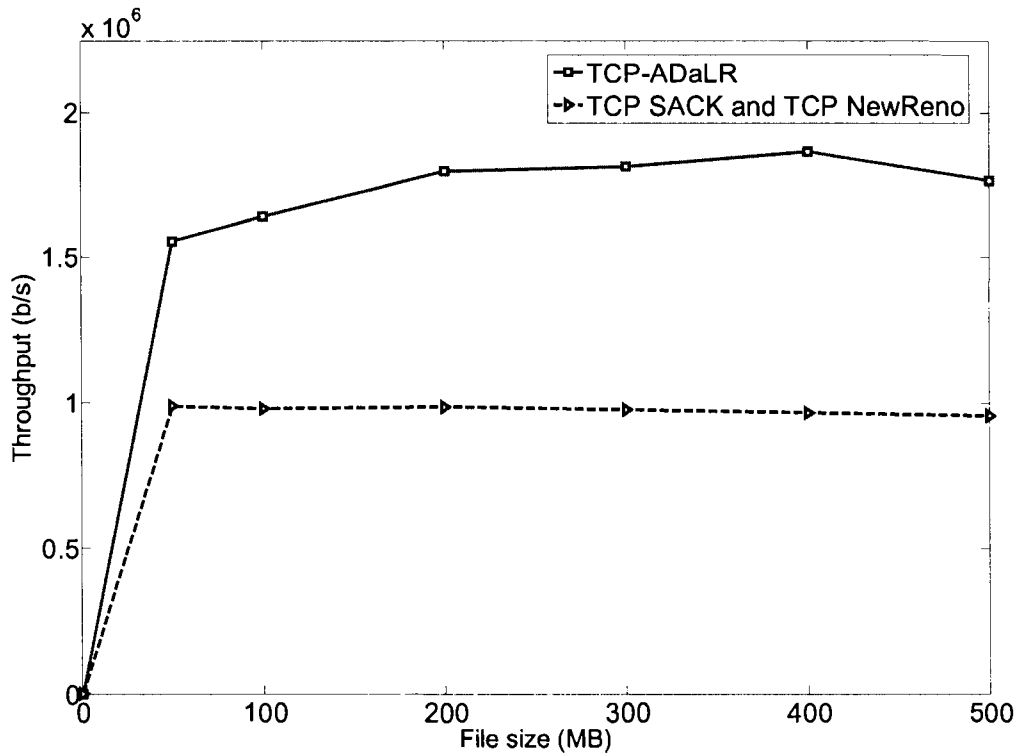
Figure 7.18. TCP-ADaLR exhibits up to 81% higher satellite link throughput than TCP SACK and TCP NewReno.
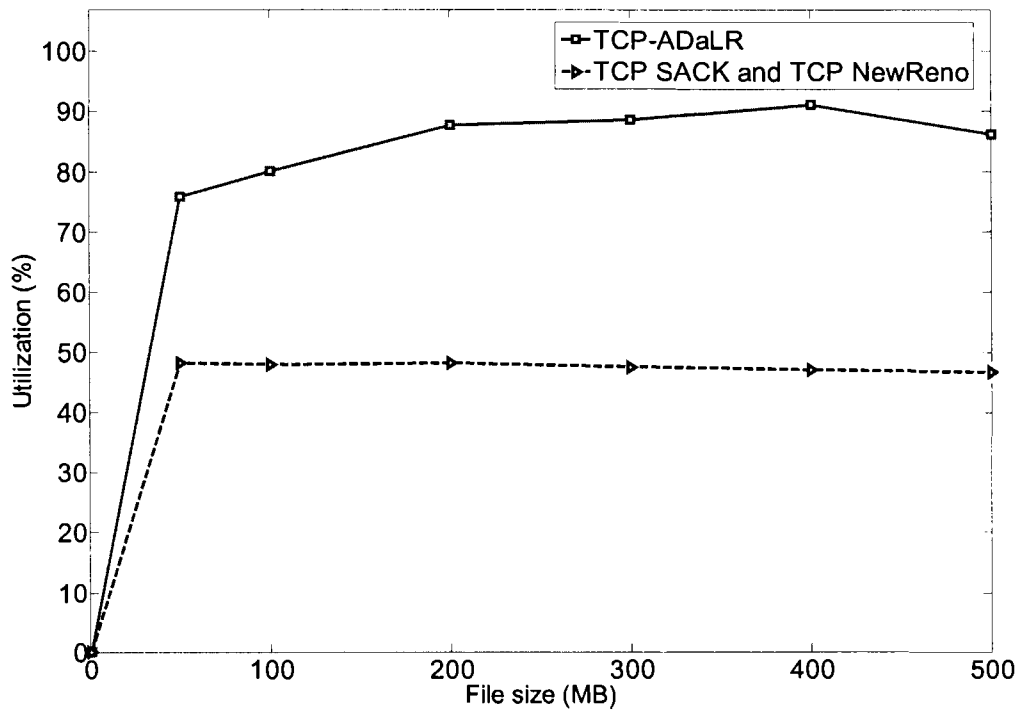
Figure 7.19. For all file sizes, TCP-ADaLR exhibits 57%–90% higher satellite link utilization than TCP SACK and TCP NewReno.

7.5.1.2 Performance of HTTP application

All TCP variants download identical number of web pages and embedded page objects. We evaluated the page response time of the HTTP application specified in Table 7.3. The main page object and the 15 embedded objects are completely downloaded and the complete web page is opened before the HTTP page response statistic is collected. The HTTP page response time for a single webpage is shown in Table 7.8. TCP-ADaLR shows ~10% and ~9% shorter page response times than TCP SACK and TCP NewReno with and without delayed ACK, respectively. Hence, TCP-ADaLR shows performance gains in the user-perceived latency of short-lived flows such as HTTP applications. The adaptive *cwnd* and *rwnd* increase mechanisms enable TCP-ADaLR to open the *cwnd* more rapidly and transmit additional segments after the IW of data segments (main page object). Hence, the HTTP webpage embedded objects are downloaded more quickly when TCP-ADaLR is employed. However, TCP SACK and TCP NewReno exhibit longer download response times for both cases with and without delayed ACK.

Table 7.8. HTTP page response time for scenarios with ideal lossless satellite channel. For cases with and without delayed ACK, TCP-ADaLR shows ~10% and ~9% shorter page response times than TCP SACK and TCP NewReno.

	Page response time (s)	
Delayed ACK option	Enabled	Disabled
TCP-ADaLR SACK	4.4	3.9
TCP-ADaLR NewReno	4.4	3.9
TCP SACK	4.9	4.3
TCP NewReno	4.9	4.3

7.5.2 Error-free satellite channel with only congestion losses

We evaluate the performance of TCP-ADaLR in the presence of congestion losses at a bottleneck gateway with a finite buffer size of 25 packets and 15 packets for the FTP and HTTP applications, respectively. We employ lower number of segments for the HTTP application because they are short-lived flows that occur in bursts.

7.5.2.1 Performance of FTP application

The download response time is comparable for all four TCP variants, as shown in Table 7.9. TCP-ADaLR SACK variant show slightly lower download response times than the other three TCP variants in the case with delayed ACK. For all TCP variants, the download response times with delayed ACK are lower than in the case without delayed ACK. The TCP goodput for cases with and without delayed ACK are shown in Figures 7.20 and 7.21, respectively. The TCP throughput, satellite link throughput, and satellite link utilization are shown in Figures 7.22–7.27. The TCP-ADaLR variants perform comparably to both TCP SACK and TCP NewReno because the adaptive *cwnd* and *rwnd* increase mechanisms lead to *cwnd* increments void of large bursts that may lead to performance degradation during congestion. Hence, TCP-ADaLR variants show no significant performance degradation in the presence of congestion. After the transmission rate is adjusted in response to congestion, the TCP ADaLR variants exhibit variations in TCP throughput attributed to the *cwnd* increments of the adaptive *cwnd* increase mechanism during the congestion avoidance phase, as shown in Figure 7. 22.

Table 7.9. FTP download response time for scenarios with only congestion losses. For both cases with and without delayed ACK, TCP-ADaLR variants exhibit download response times comparable to TCP SACK and TCP NewReno.

Delayed ACK option	Page response time (s)	
	Enabled	Disabled
TCP-ADaLR SACK	1212.7	1226.7
TCP-ADaLR NewReno	1228.0	1232.4
TCP SACK	1224.8	1226.7
TCP NewReno	1216.6	1226.7
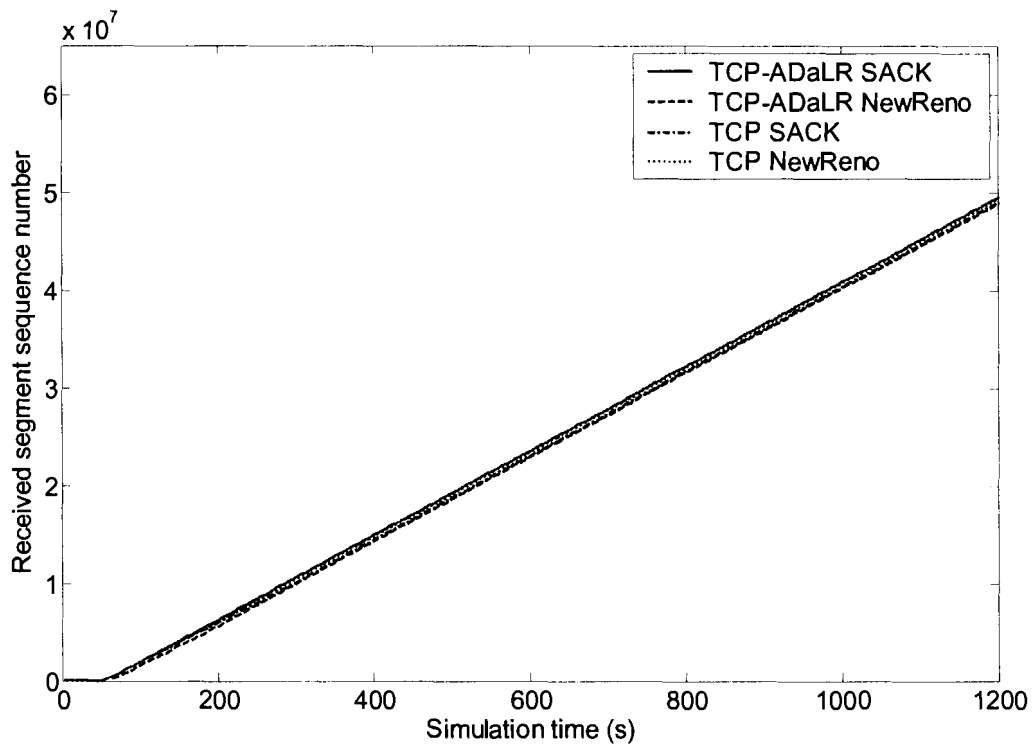
Figure 7.20. Goodput for scenarios with only congestion losses and delayed ACK enabled. Received segment sequence number is used as an indicator of goodput. The four TCP variants exhibit comparable goodput.
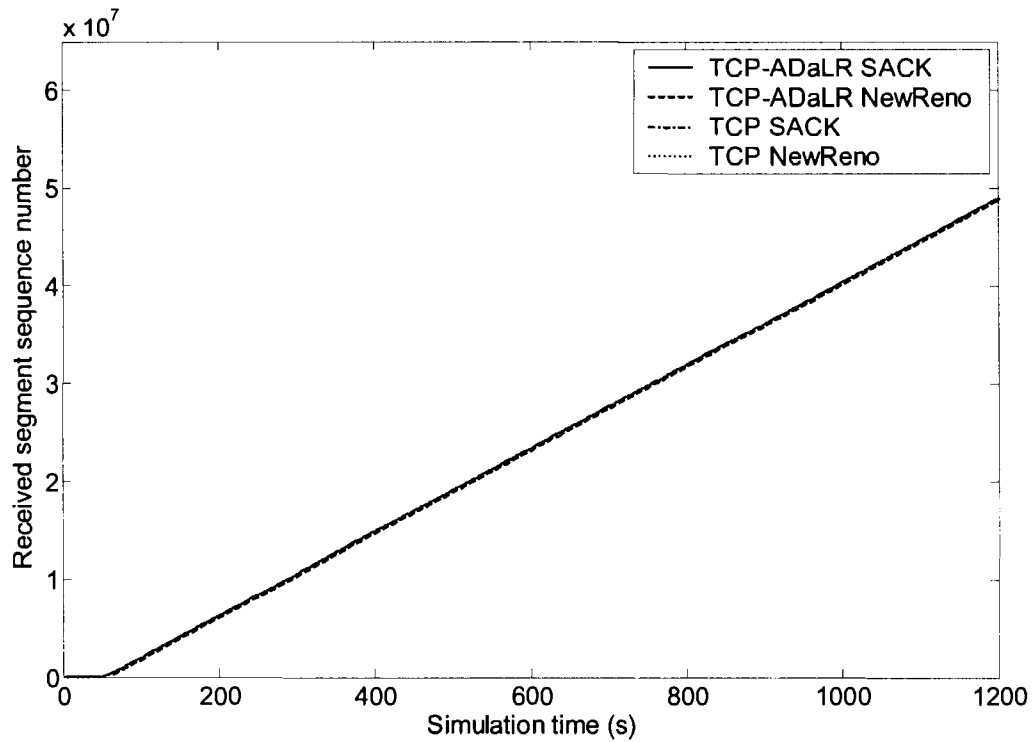
Figure 7.21. Goodput for scenarios with only congestion losses and delayed ACK disabled. The received segment sequence number is used as an indicator of goodput. The four TCP variants exhibit comparable goodput.
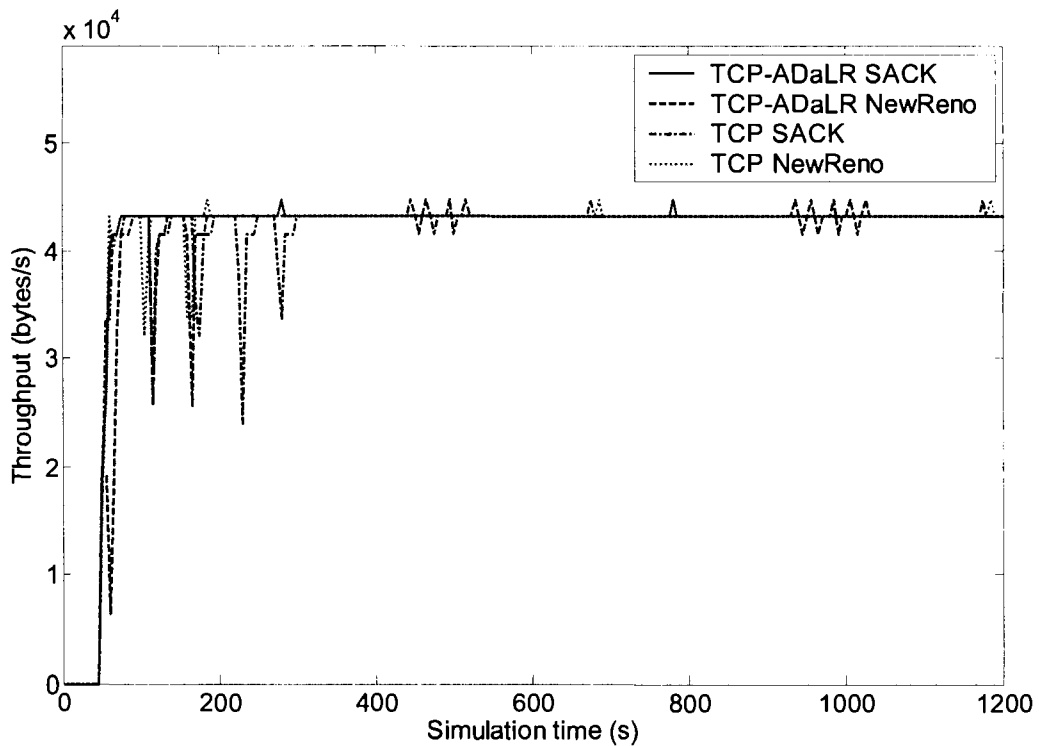
Figure 7.22. TCP throughput for scenarios with only congestion losses and delayed ACK enabled. The four TCP variants exhibit TCP throughput degradation when congestion losses are detected.
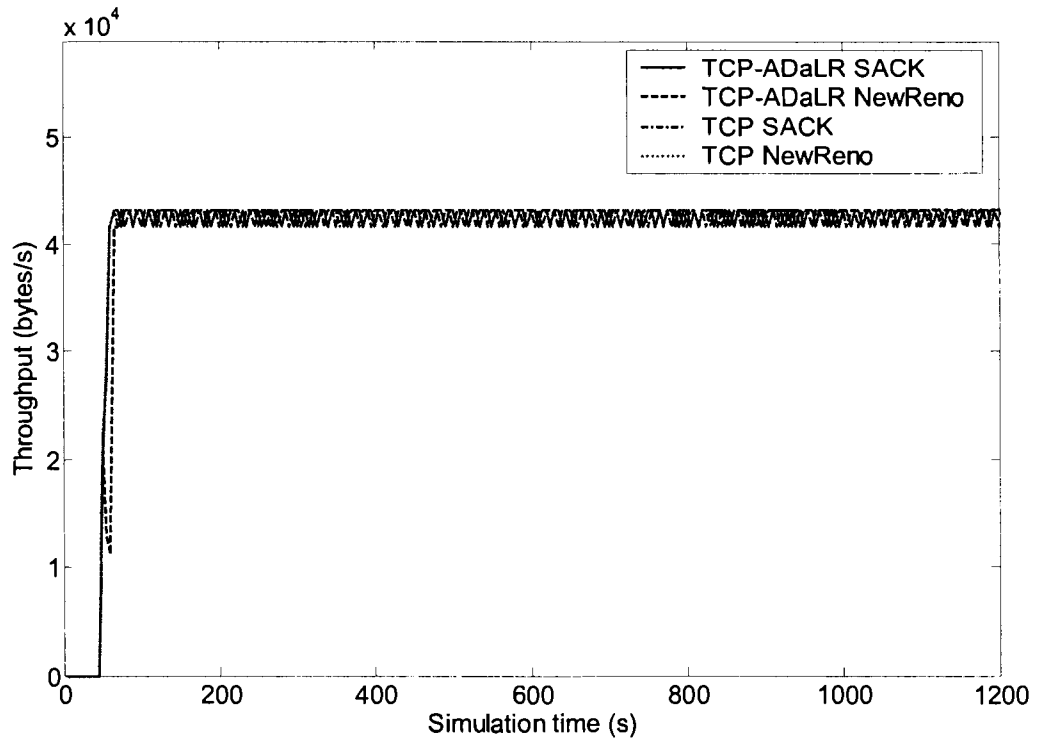
Figure 7.23. TCP throughput for scenarios with only congestion losses and delayed ACK disabled. The TCP throughput is comparable for the four TCP variants.
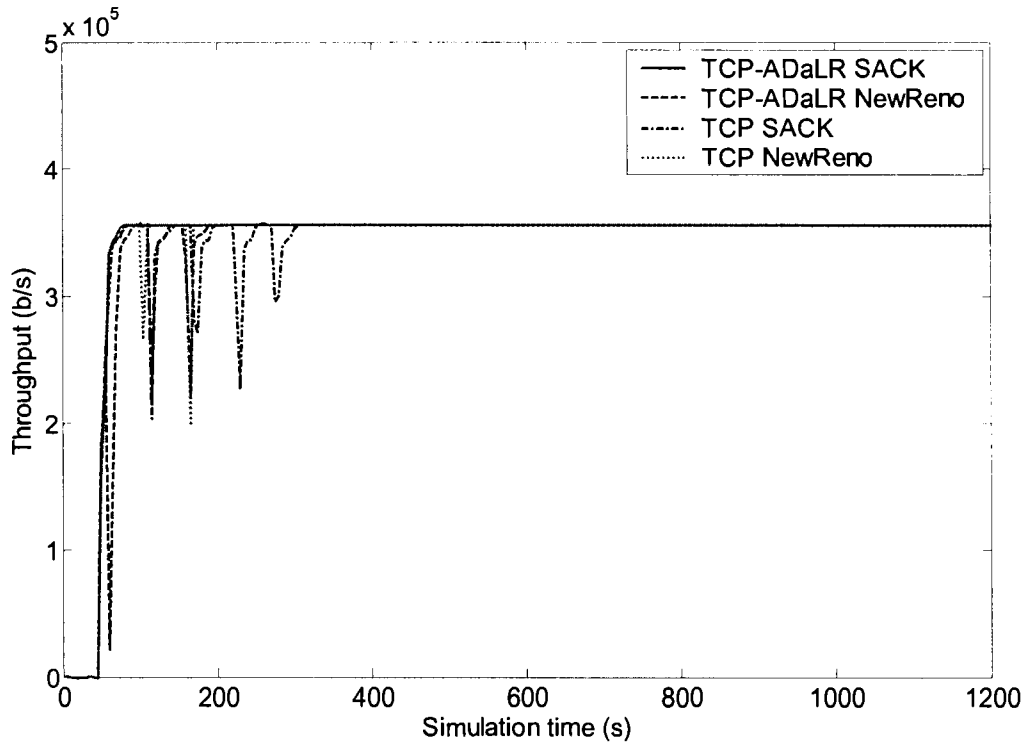
Figure 7.24. Satellite link throughput for scenarios with only congestion losses and delayed ACK enabled. The link throughput reduces when congestion losses are detected and attains steady state after transmission rate adjusted by the TCP congestion control algorithms in response to congestion.
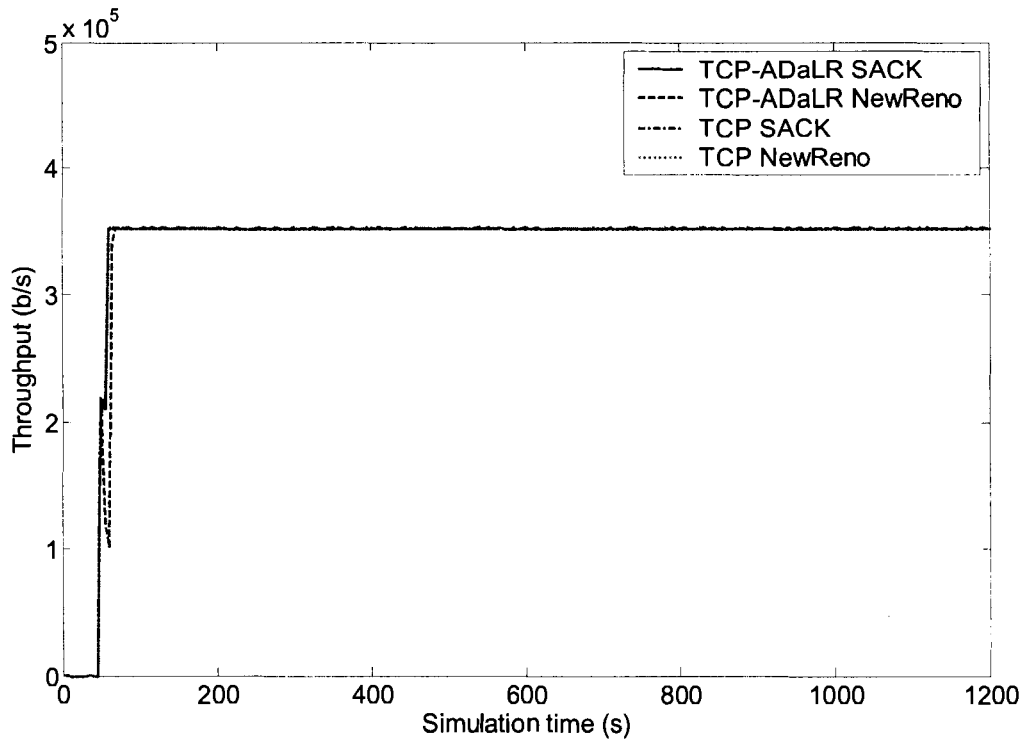
Figure 7.25. Satellite link throughput for scenarios with only congestion losses and delayed ACK disabled. Satellite link throughput is comparable for the four TCP variants.
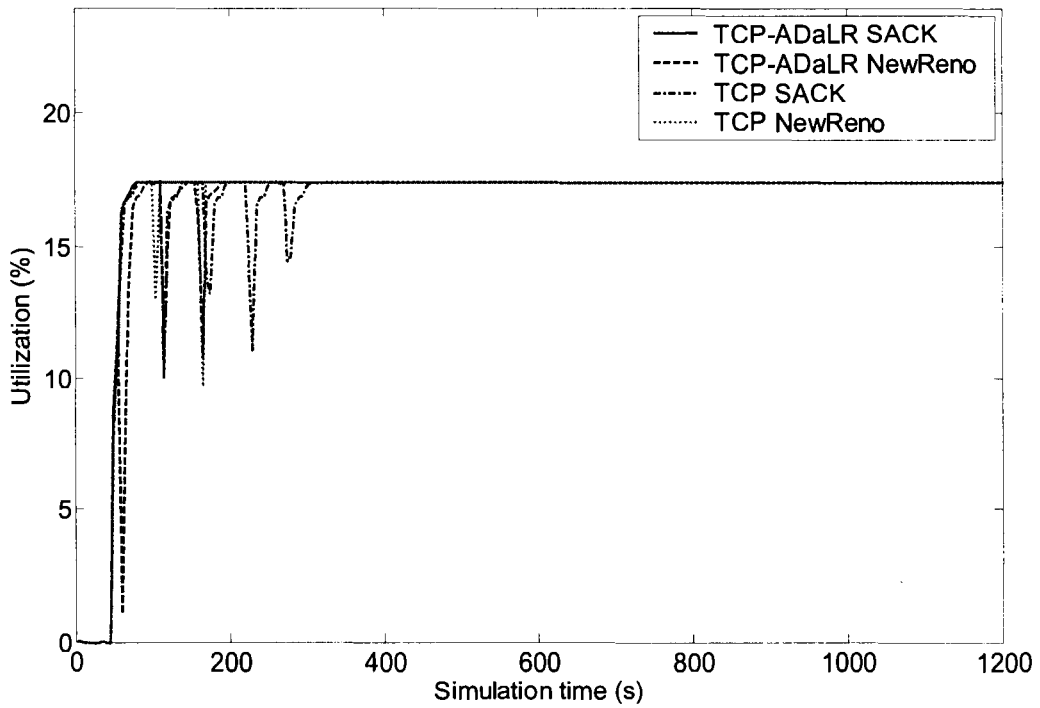
Figure 7.26. Satellite link utilization for scenarios with only congestion losses and delayed ACK enabled Satellite link utilization decreases when congestion losses are detected.
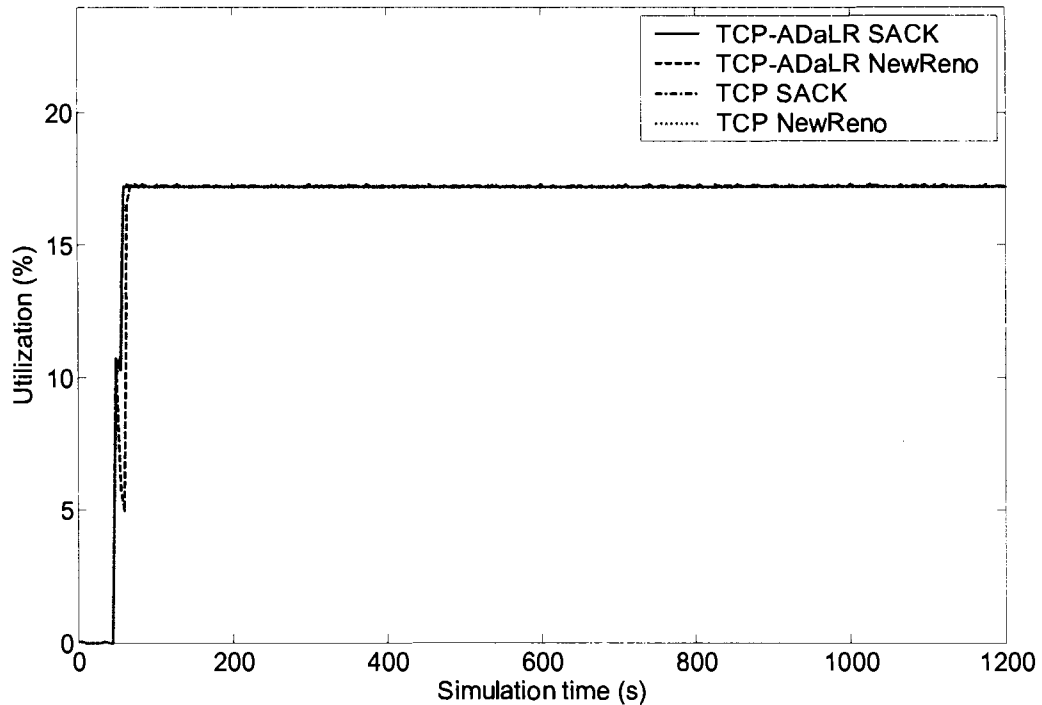
Figure 7.27. Satellite link utilization for scenarios with only congestion losses and delayed ACK disabled. Satellite link utilization is comparable for the four TCP variants.

7.5.2.2 Performance of HTTP application

We evaluate performance of the HTTP application in terms of the page response time. The increased page response time indicates the impact of congestion losses (compared to the ideal case) as shown in Table 7.10. TCP-ADaLR SACK shows the best performance while TCP-ADaLR NewReno exhibits the second best performance in both cases with and without delayed ACK. TCP-ADaLR variants exhibit shorter HTTP page responses times than TCP SACK and TCP NewReno and, hence, show performance gains. HTTP transfers occur in short bursts. When losses occur, the adaptive *cwnd* increase mechanism enables the transmission of additional segments after fast recovery. Hence, the HTTP transfers are completed faster than TCP SACK and TCP NewReno. The adaptive *rwnd* mechanism allows at least two back-to-back segments to be

transferred during the recovery process to compensate for delayed ACK. Hence, the main page and embedded objects are downloaded in quick successions with the TCP-ADaLR algorithm.

Table 7.10. HTTP page response time for scenarios with only congestion losses. For cases with and without delayed ACK, TCP-ADaLR SACK exhibits 33% and 12% shorter page response times than TCP SACK, respectively.

	Page response time (s)	
Delayed ACK option	Enabled	Disabled
TCP-ADaLR SACK	11.0	10.3
TCP-ADaLR NewReno	11.0	11.1
TCP SACK	13.8	11.7
TCP NewReno	16.6	11.7

7.5.3 Satellite channel with only error losses

We evaluate the performance of TCP-ADaLR in the presence of losses due to only satellite link errors for the various BER values shown in Table 7.1. (Typical average post-FEC BER of satellite links is 10^{-5} to 10^{-8} [99].) For each BER value, we use different random seed numbers and compute the average values using 95% confidence intervals.

7.5.3.1 Performance of FTP application

For the FTP application, we evaluate average values of the download response time, TCP goodput, TCP throughput, satellite link throughput, and satellite link utilization as a function of the BER values. The download response time for cases with and without delayed ACK are shown in Figures 7.28 and 7.29, respectively. For all TCP variants, the negative effect of losses due to satellite link errors is evident in the increasing response times as the BER increases. For the simulated BER values, TCP-ADaLR SACK shows 13%–37% and 6%–31% shorter download response time than TCP

79

SACK with and without delayed ACK, respectively. TCP-ADaLR NewReno exhibits 5%–26% and 2%–26% shorter download response time than TCP NewReno with and without delayed ACK, respectively. Performance gains increase at high BER values. The TCP-ADaLR variants are robust against heavy losses resulting from high BER values. The loss recovery mechanism enables quicker recovery from the losses due to satellite link errors. The adaptive window increase mechanisms also enable faster transmission of additional segments. TCP-ADaLR SACK and TCP-ADaLR NewReno show similar performance at low BER values. However, at high BER values, TCP-ADaLR SACK exhibits better performance than TCP-ADaLR NewReno. Hence, TCP-ADaLR offers better performance improvement with the SACK option.
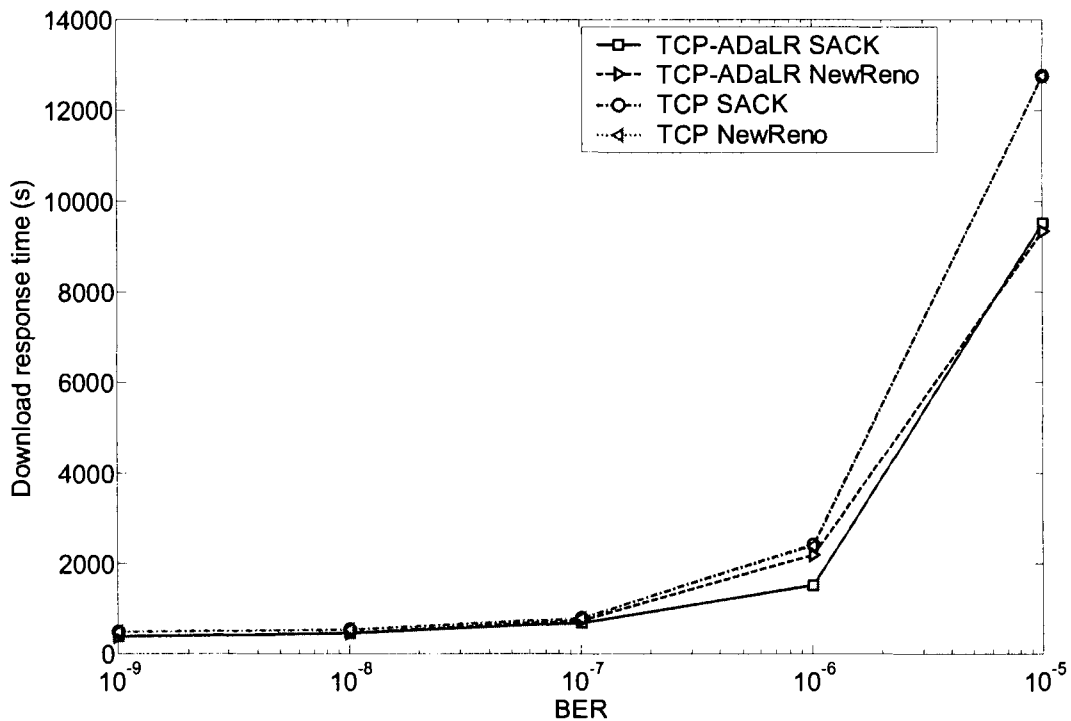
Figure 7.28. FTP download response time for scenarios with only error losses and delayed ACK enabled. TCP-ADaLR SACK exhibits up to 37% shorter download response time than TCP SACK.
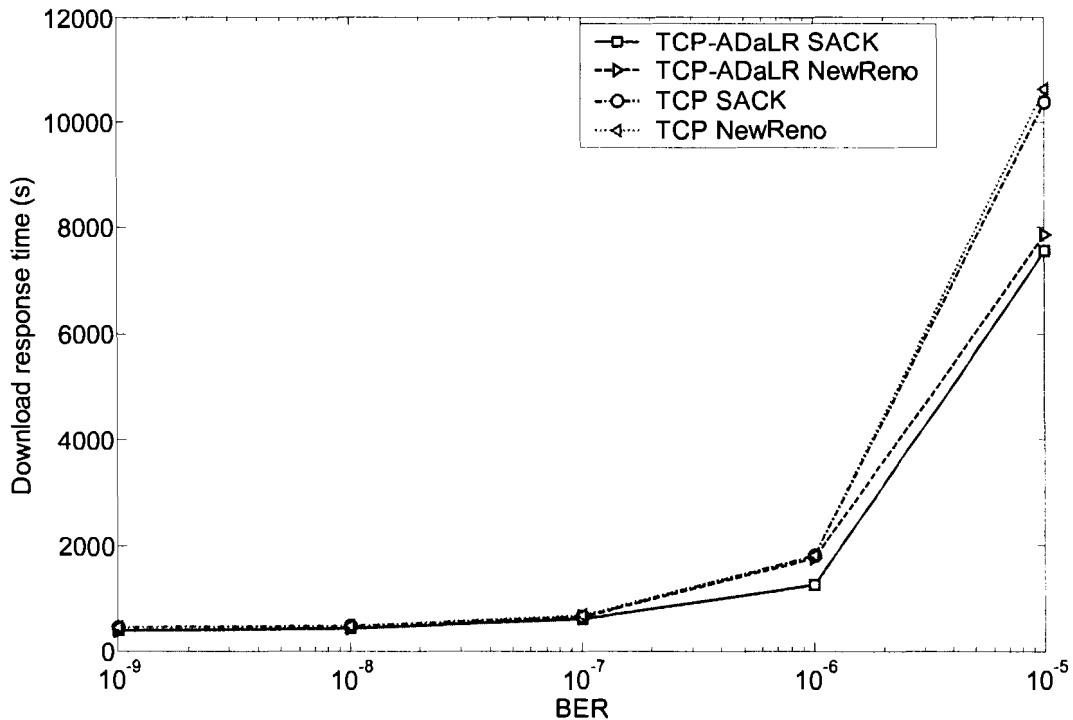
Figure 7.29. FTP download response time for scenarios with only error losses and delayed ACK disabled. At BER value of 10^{-6}, TCP-ADaLR SACK shows ~31% shorter download response time than TCP SACK.

The TCP goodput and TCP throughput are shown in Figures 7.30–7.33. The TCP-ADaLR SACK shows 16%–61% and 7%–46% higher throughput than TCP SACK with and without delayed ACK, respectively. TCP-ADaLR NewReno exhibits 6%–36% and 2%–35% higher throughput than TCP NewReno with and without delayed ACK, respectively. The adaptive *cwnd* increase mechanism causes additional segments to be rapidly sent after losses have occurred. The adaptive *rwnd* increase and loss recovery mechanisms allow at least two segments to be transmitted back-to-back when losses are detected in order to compensate for delayed ACK. These mechanisms enable TCP-ADaLR to recover more quickly than TCP SACK and TCP NewReno. For corresponding BERs, TCP variants in scenarios with delayed ACK exhibit better performance than TCP

variants in scenarios without delayed ACK. Correspondingly, the satellite link throughput and utilization decrease with increasing BER, as shown in Figures 7.34–7.37.

For all TCP variants, the TCP goodput and TCP throughput degrade considerably at BERs higher than 10^{-8} [103]. The long propagation delays prevent quick recovery from losses and exacerbate the performance degradation. Hence, it is difficult to sustain the higher throughputs at BER values lower than 10^{-8}. When heavy losses occur and TCP throughput degrades, fewer segments are transmitted due to several RTO timer expirations in the absence of duplicate ACKs. When the delayed ACK option is enabled, it exacerbates the inability of TCP to recover quickly from the losses. Hence, for the four TCP variants, the satellite link utilization decreases. TCP-ADaLR SACK is the most robust variant in the presence of heavy losses due to satellite link errors.
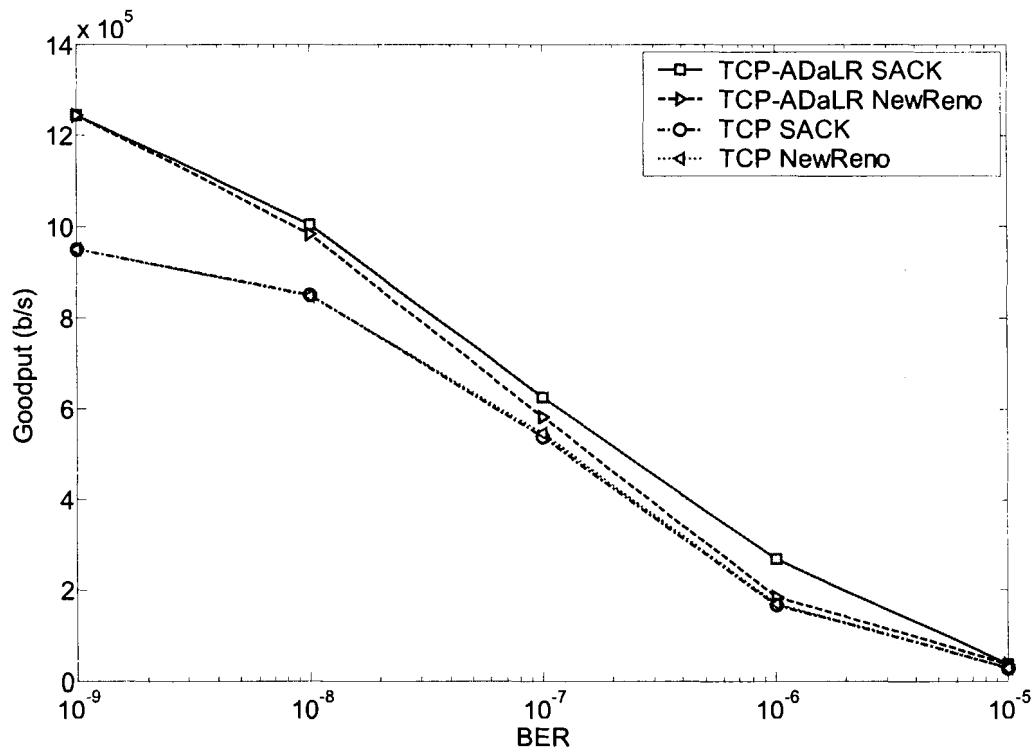
Figure 7.30. Goodput for scenarios with only error losses and delayed ACK enabled. TCP-ADaLR SACK exhibits up to 27% higher goodput than TCP SACK.
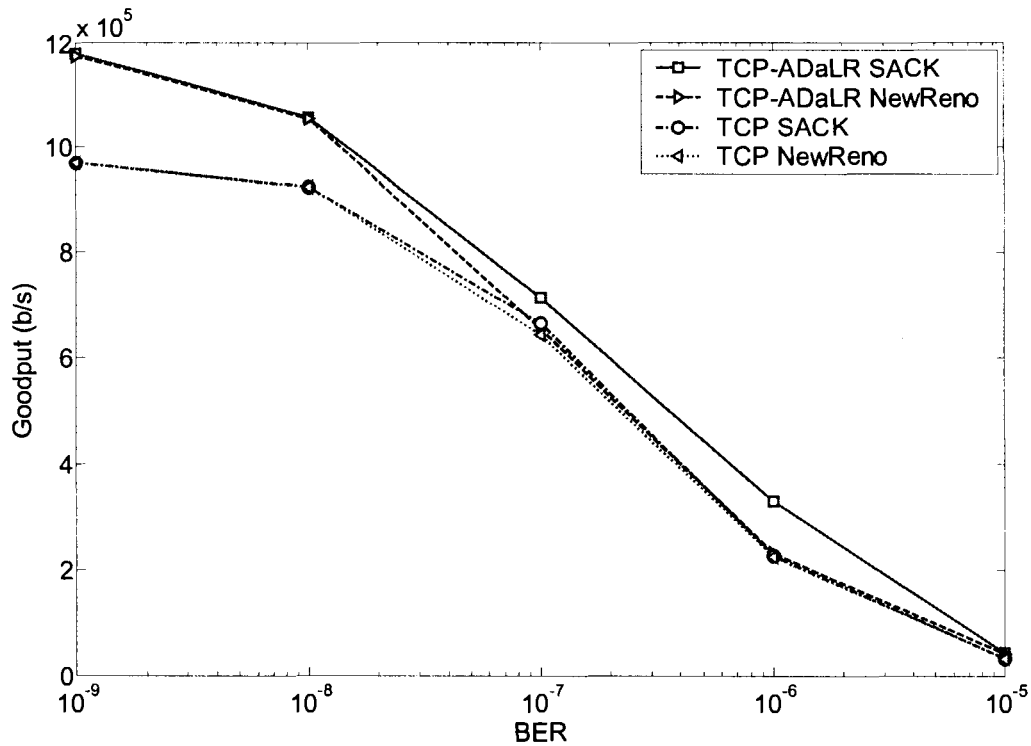
Figure 7.31. Goodput for scenarios with only error losses and delayed ACK disabled. TCP-ADaLR SACK shows 7%–46% higher goodput than TCP SACK.
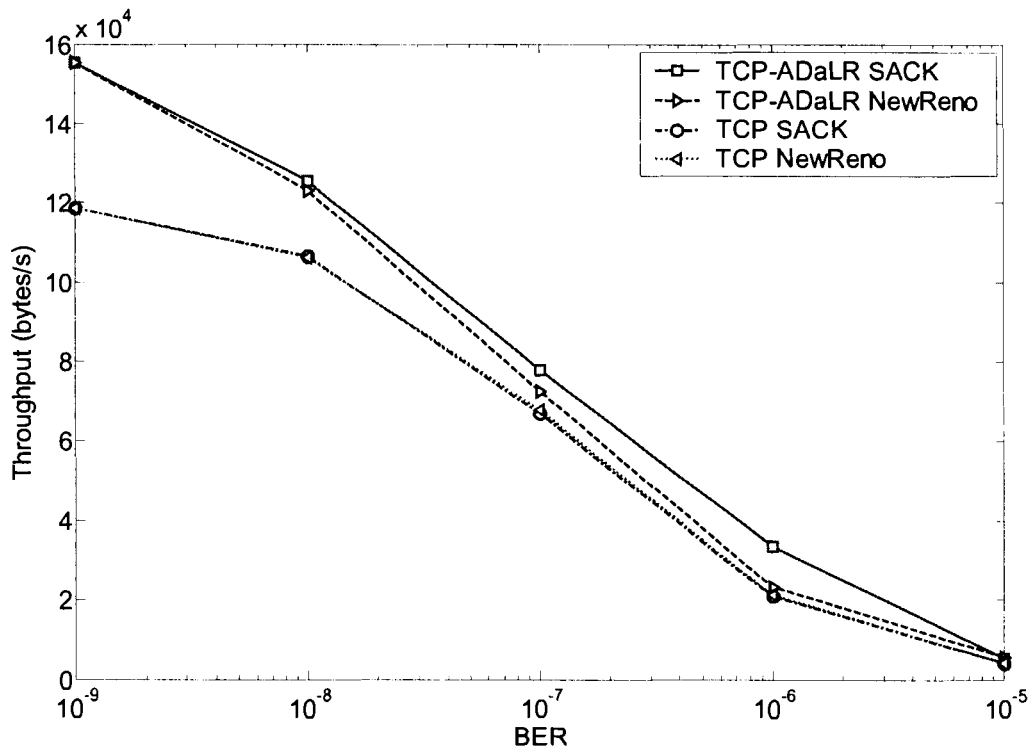
Figure 7.32. TCP throughput for scenarios with only error losses and delayed ACK enabled. TCP-ADaLR SACK and TCP-ADaLR NewReno exhibit higher TCP throughputs than TCP SACK and TCP NewReno, respectively.
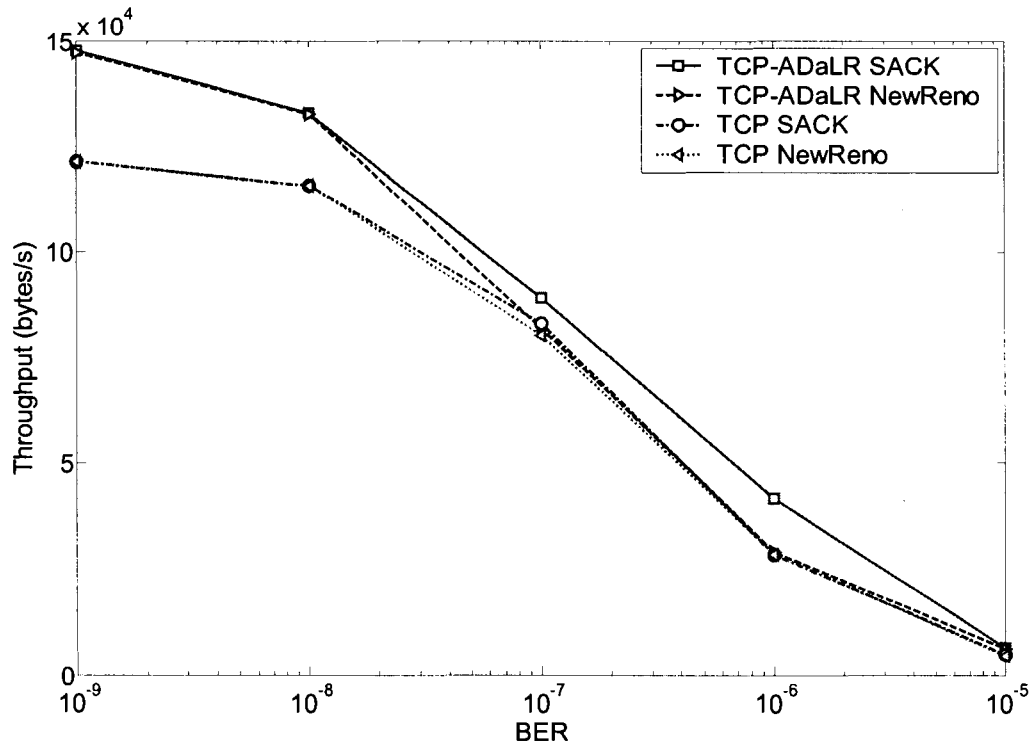
Figure 7.33. TCP throughput for scenarios with only error losses and delayed ACK disabled. For all BER values, TCP-ADaLR SACK exhibits the highest throughput.
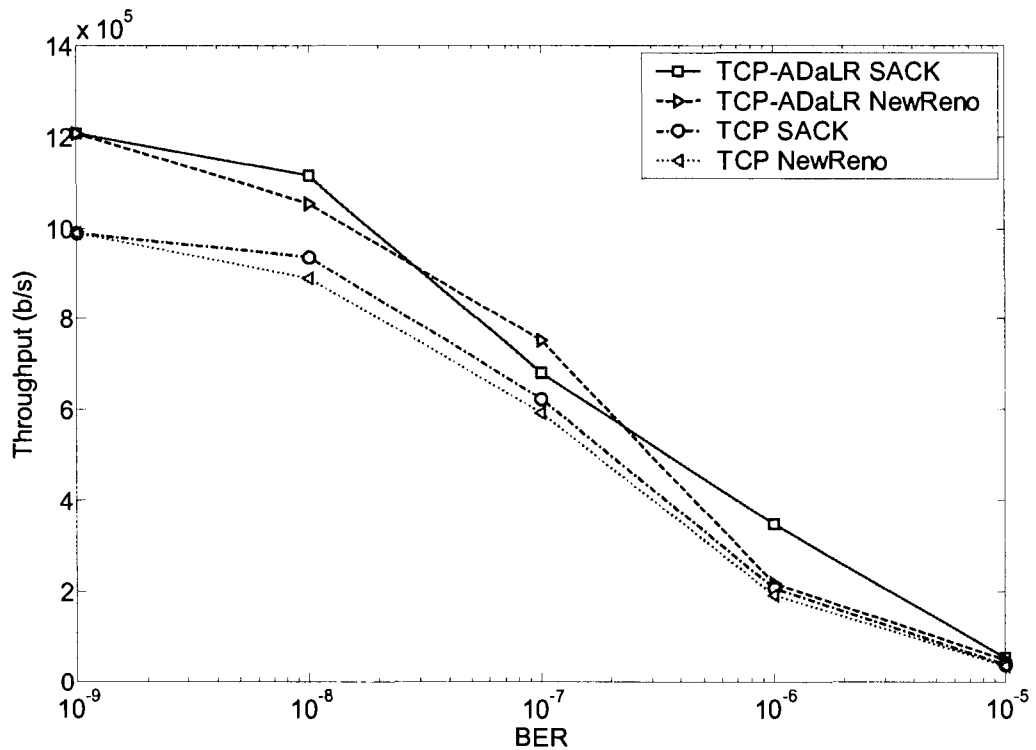
Figure 7.34. Satellite link throughput for scenarios with only error losses and delayed ACK enabled. TCP-ADaLR SACK exhibits up to 73% higher satellite link throughput than TCP SACK.
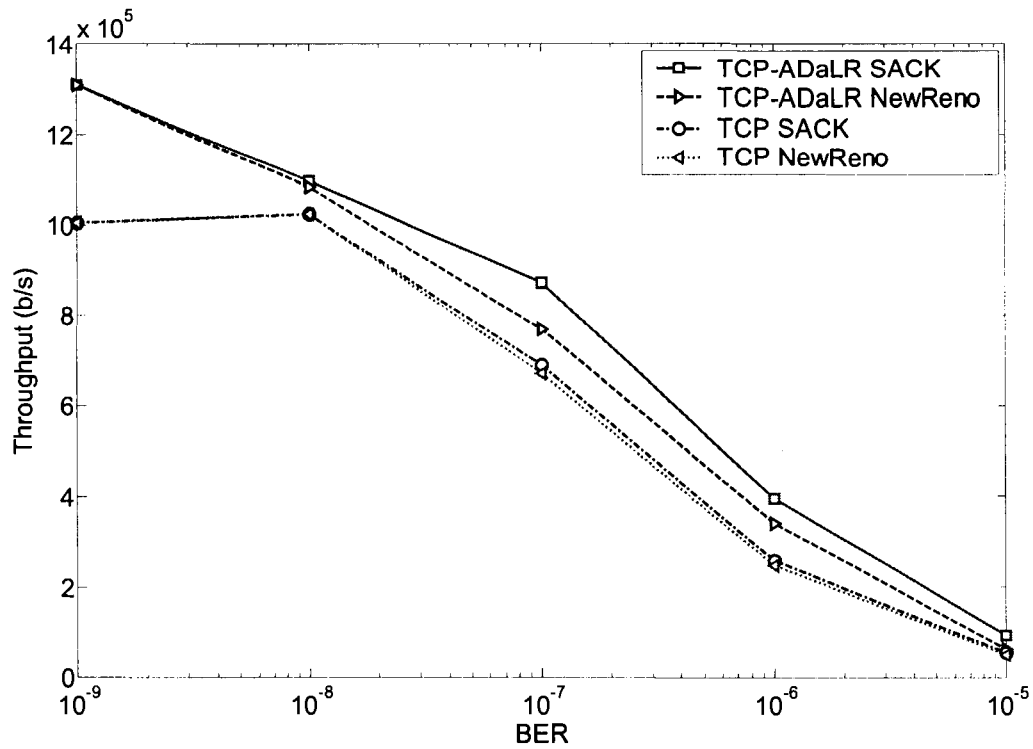
Figure 7.35. Satellite link throughput for scenarios with only error losses and delayed ACK disabled. For all BER values, TCP-ADaLR SACK exhibits the highest satellite link throughput.
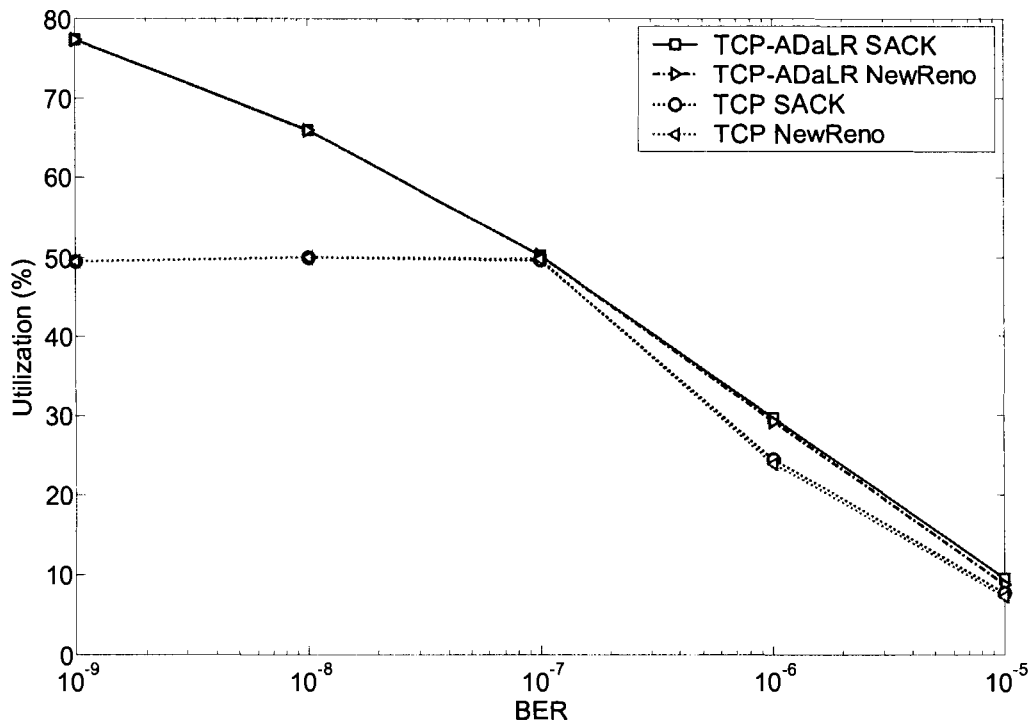
Figure 7.36. Satellite link utilization for scenarios with only error losses and delayed ACK enabled. TCP-ADaLR SACK and TCP-ADaLR NewReno exhibit comparable link utilization higher than TCP SACK and TCP NewReno.
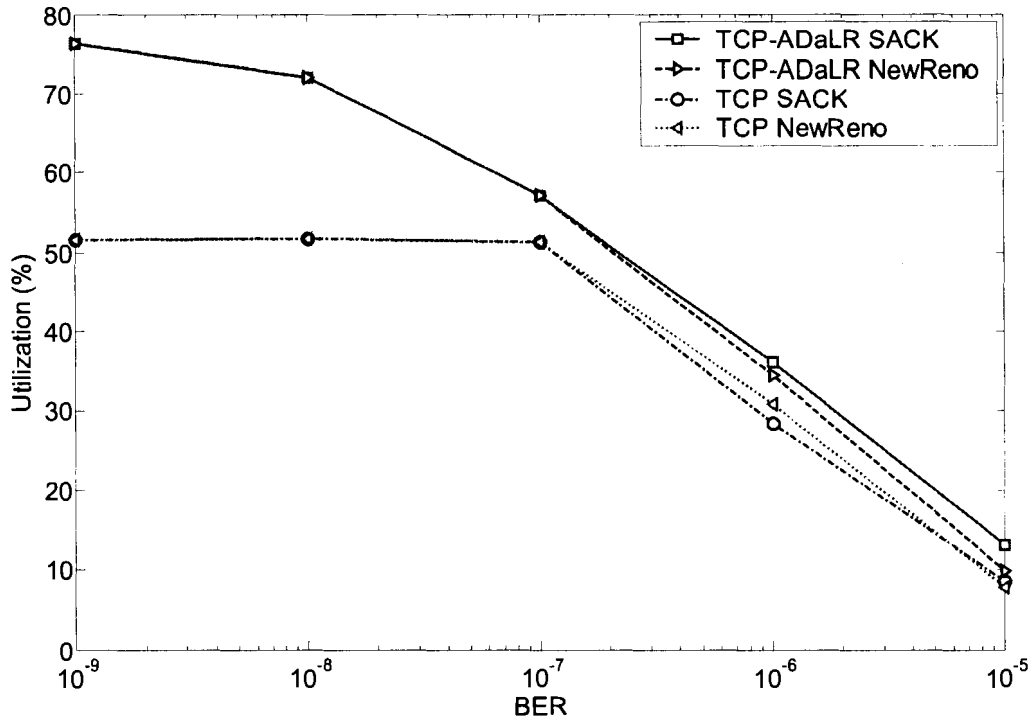
Figure 7.37. Satellite link utilization for scenarios with only error losses and delayed ACK disabled. TCP-ADaLR variants exhibit up to 46% higher satellite link utilization than TCP SACK and TCP NewReno.

7.5.3.2 Performance of HTTP application

The page response times for cases with and without delayed ACK are shown in Figures 7.38 and 7.39, respectively. Both TCP-ADaLR variants exhibit similar page response times and outperform both TCP SACK and TCP NewReno. When all outstanding segments have been acknowledged, the adaptive *cwnd* increase mechanism enables rapid transmission of segments after loss recovery. The adaptive *rwnd* increase and loss recovery mechanisms enable TCP-ADaLR to recover more quickly from losses than TCP SACK or TCP NewReno. For cases with and without delayed ACK, the performance gains by TCP-ADaLR SACK and TCP-ADaLR NewReno are highest when the BER is lower than 10^{-6}. For cases with and without delayed ACK, TCP-ADaLR variants exhibit 2%–12% and 7%–23% shorter page response time than TCP SACK and

TCP NewReno, respectively. When the BER is lower than 10^{-6}, the TCP variants without delayed ACK exhibit better performance than with delayed ACK. At BER values higher than 10^{-7}, the four TCP variants show similar page response times for cases with and without delayed ACK.

The four TCP variants suffer performance degradation at BERs higher than 10^{-7} because the increasing BER causes additional segment losses. Several lost TCP segments may compose a complete web page and embedded objects. Hence, a webpage download request may be completed in multiples of the usual response time, as indicated by the higher values of page response times.

Figure 7.38. HTTP page response time for scenarios with only error losses and delayed ACK enabled. TCP-ADaLR SACK and TCP-ADaLR NewReno show 2%–12% shorter page response times than TCP SACK and TCP NewReno.

Figure 7.39. HTTP page response time for scenarios with only error losses and delayed ACK disabled. TCP-ADaLR SACK and TCP-ADaLR NewReno exhibit 7%–23% shorter page response times than TCP SACK and TCP NewReno.

7.5.4 Satellite channel with both congestion and error losses

We evaluate the performance of TCP-ADaLR in the presence of both congestion losses at the bottleneck gateway and losses due to satellite link errors. The bottleneck gateway had finite buffer sizes for FTP and HTTP applications. We test various BER values shown in Table 7.1. For each BER value, we use different random seed numbers and compute the average values using 95% confidence intervals.

7.5.4.1 Performance of FTP application

In the presence of error and congestion losses, the two TCP-ADaLR variants show comparable FTP download response time with TCP SACK and TCP NewReno when the BER value is 10^{-7} and lower, as shown in Figures 7.40 and 7.41. The TCP goodput, TCP throughput, satellite link throughput, and utilization are shown in Figures

7.42–7.49. At lower BERs, TCP-ADaLR variants exhibit comparable performance to TCP SACK and TCP NewReno because losses are mainly due to congestion. At higher BERs, TCP-ADaLR variants exhibit better performance than TCP SACK and TCP NewReno because satellite link errors are the more prevalent cause of losses. For the case with delayed ACK at BER value of 10^{-6} and higher, TCP-ADaLR SACK exhibits up to 29% shorter download response time than TCP SACK. For the case without delayed ACK, TCP-ADaLR NewReno exhibits shorter download response time than TCP NewReno. At higher BERs, the adaptive *cwnd* increase mechanism enables quick recovery from segment losses when all outstanding segments have been acknowledged. The adaptive *rwnd* increase and loss recovery mechanisms enable TCP-ADaLR SACK to recover more quickly from losses than either TCP SACK and TCP NewReno.
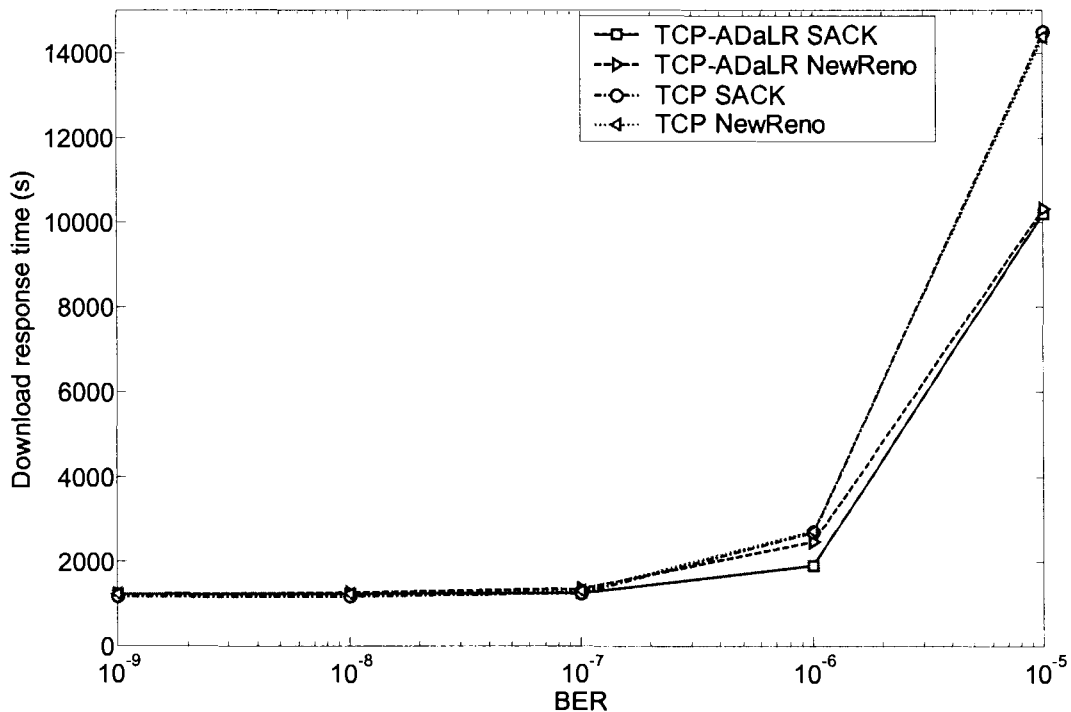
Figure 7.40. FTP download response time for scenarios with both congestion and error losses and delayed ACK enabled. When BER is higher than 10^{-7}, TCP-ADaLR variants exhibit shorter download response times than TCP SACK and TCP NewReno.

89
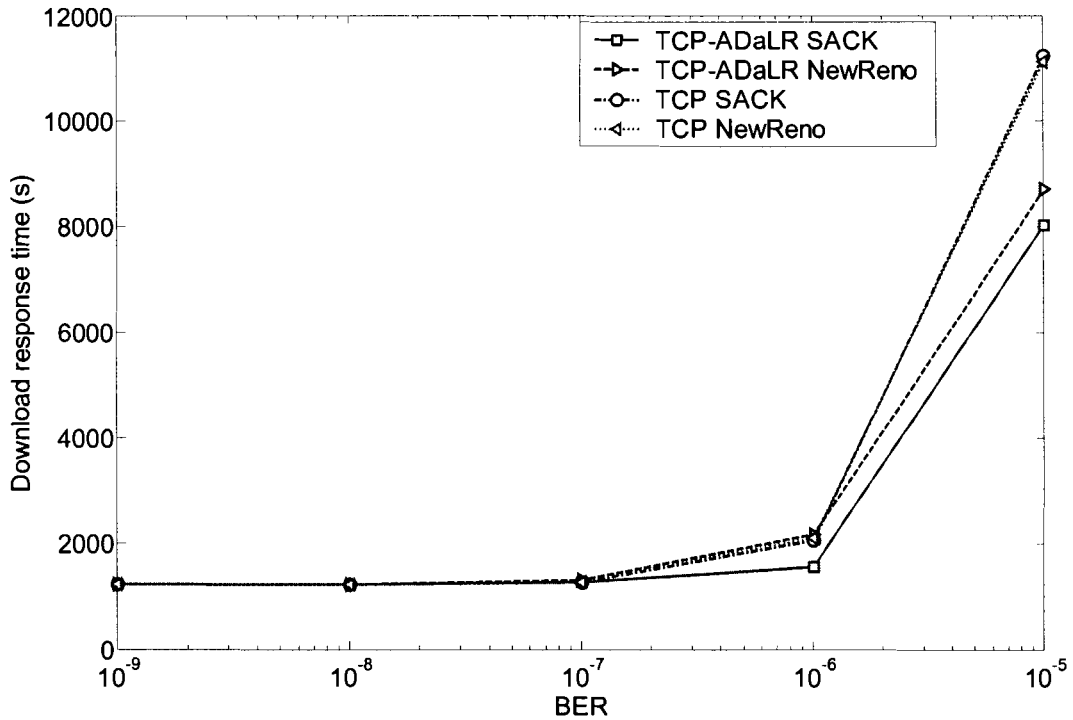
Figure 7.41. FTP download response time for scenarios with both congestion and error losses and delayed ACK disabled. TCP-ADaLR SACK exhibits 23%–28% shorter download response times than TCP SACK.
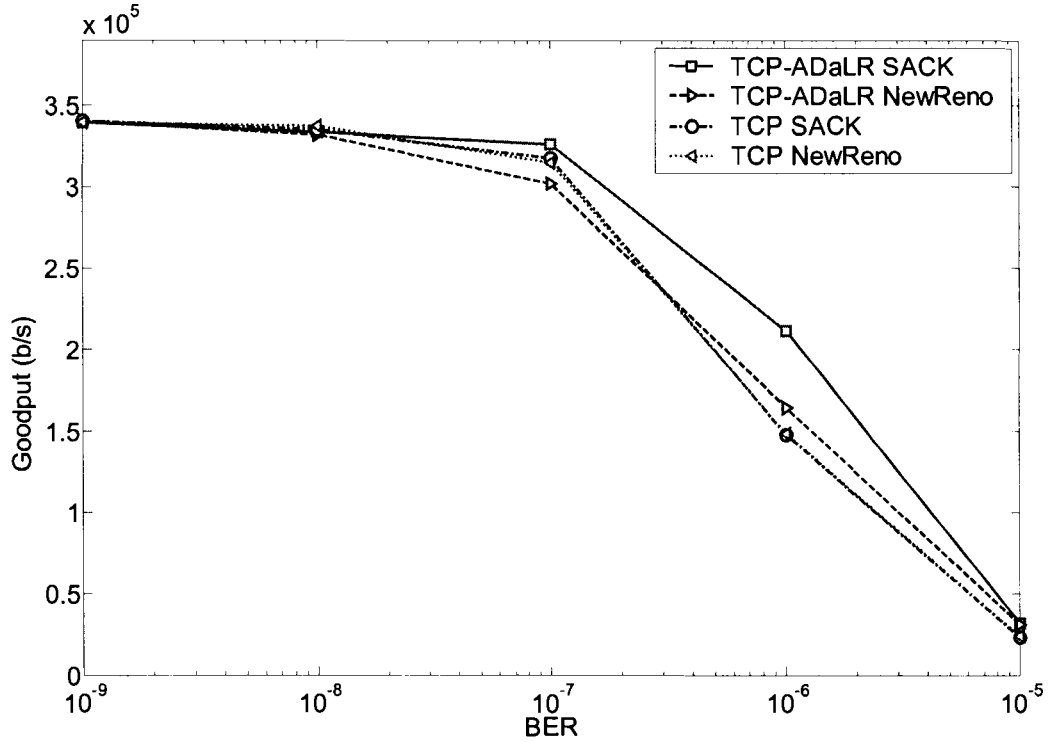
Figure 7.42. Goodput for scenarios with both congestion and error losses and delayed ACK enabled. TCP-ADaLR SACK exhibits 36%–43% higher goodput than TCP SACK and TCP NewReno at BER higher than 10^{-7}.
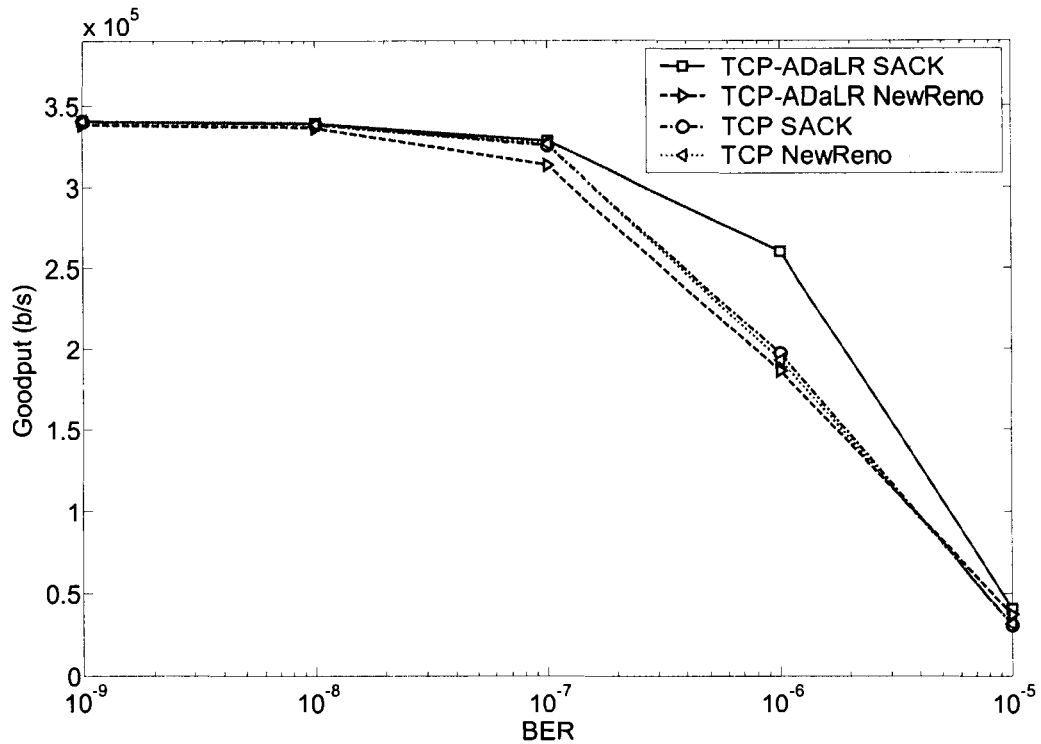
Figure 7.43. Goodput for scenarios with both congestion and error losses and delayed ACK disabled. TCP-ADaLR SACK shows 32% higher goodput than TCP SACK for BER values of 10^{-6} and 10^{-5}, respectively.
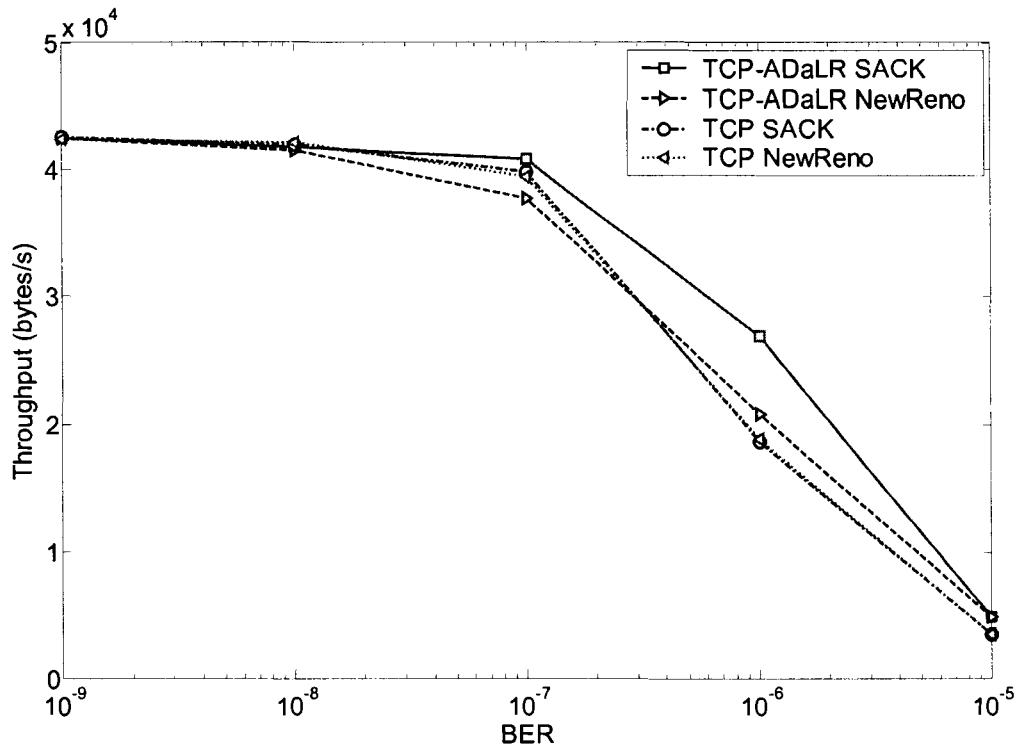
Figure 7.44. TCP throughput for scenarios with both congestion and error losses and delayed ACK enabled. For BER values higher than 10^{-7}, TCP-ADaLR SACK and TCP-ADaLR NewReno exhibit 42%–43% and 10%–39% higher throughput than TCP SACK and TCP NewReno, respectively.
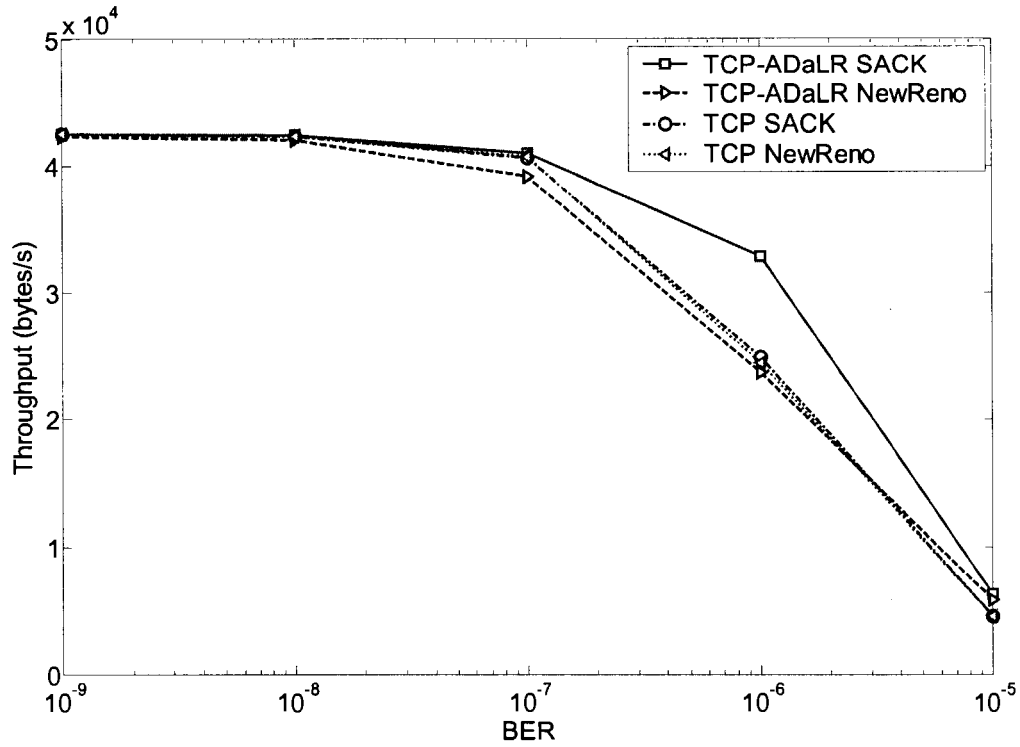
Figure 7.45. TCP throughput for scenarios with both congestion and error losses and delayed ACK disabled. At BER values higher than 10^{-7}, TCP-ADaLR SACK exhibits 32%–39% higher throughput than TCP SACK.
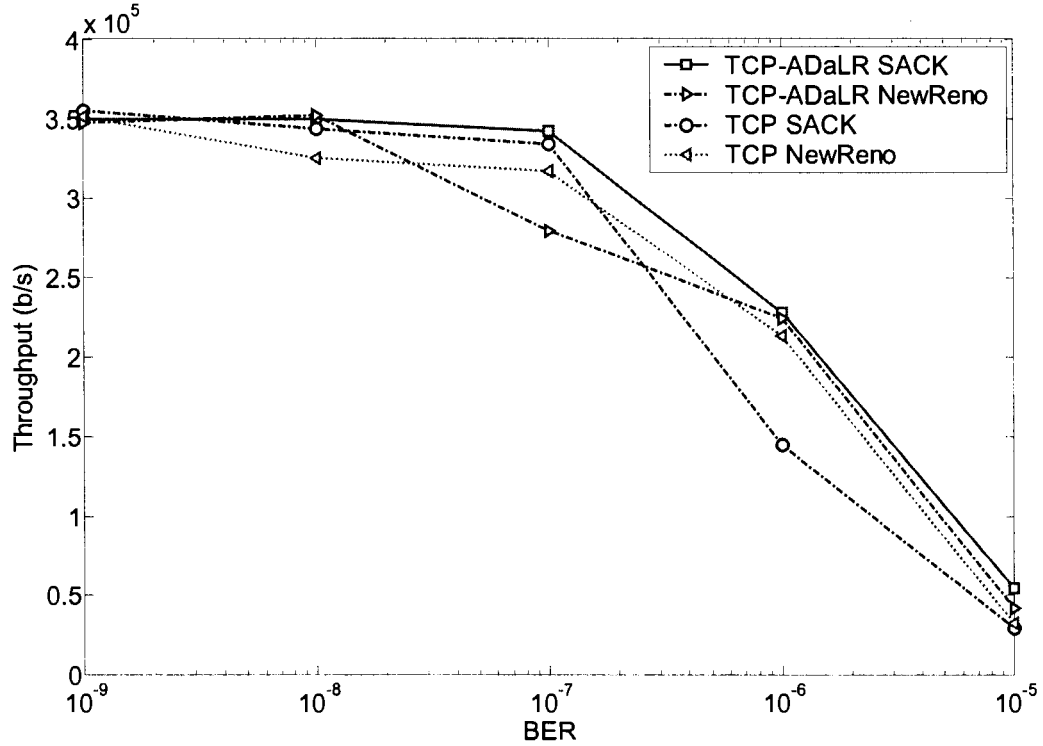
Figure 7.46. Satellite link throughput for scenarios with both congestion and error losses and delayed ACK disabled is comparable for all TCP variants at low BER values. TCP-ADaLR SACK shows 57%–86% higher satellite link throughput than TCP SACK with delayed ACK.
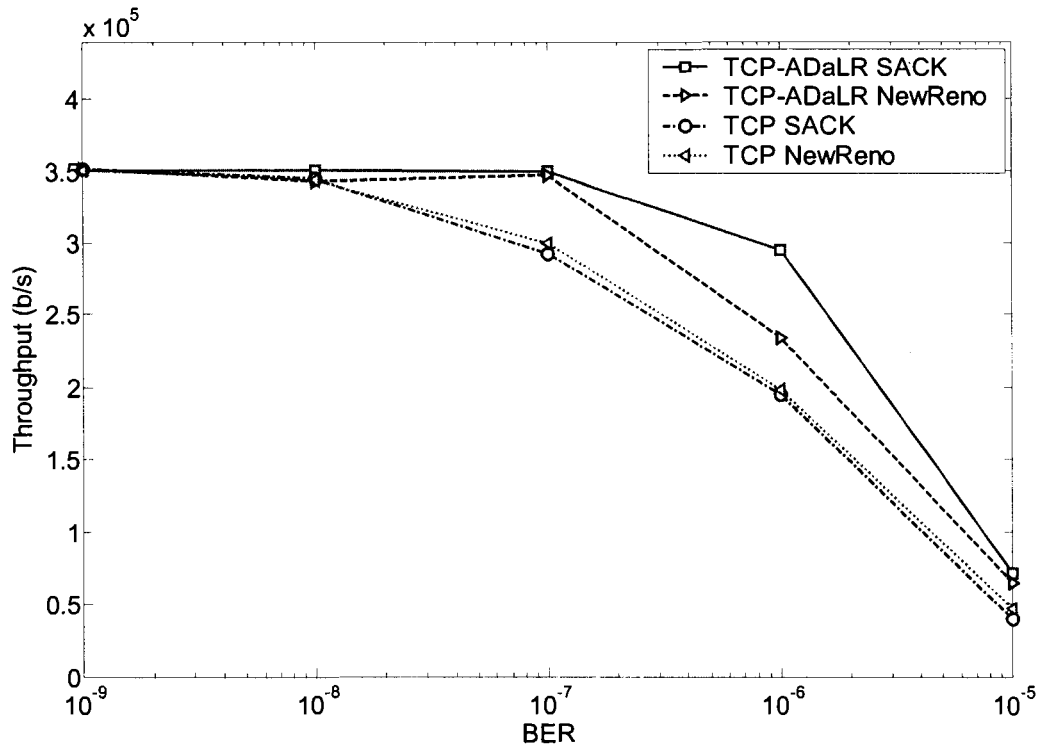
Figure 7.47. Satellite link throughput for scenarios with both congestion and error losses and delayed ACK disabled. TCP-ADaLR SACK shows 51%–79% higher satellite link throughput than TCP SACK.
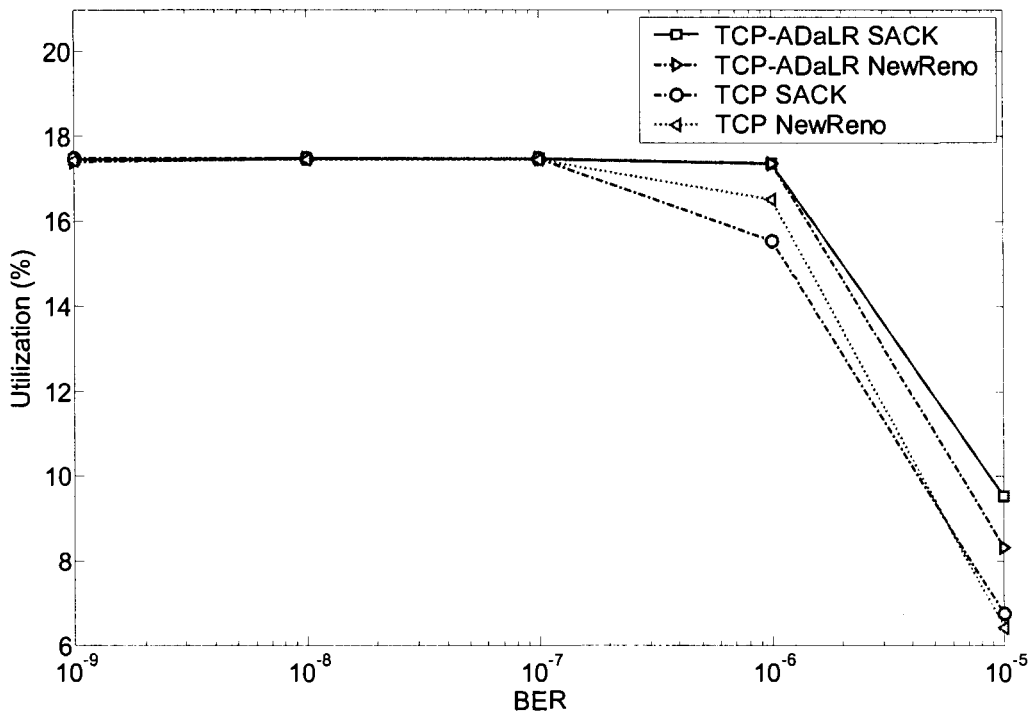
Figure 7.48. Satellite link utilization for scenarios with both congestion and error losses and delayed ACK enabled. The satellite link utilization for the four TCP variants is severely reduced because of the heavy losses caused by congestion and satellite link errors.
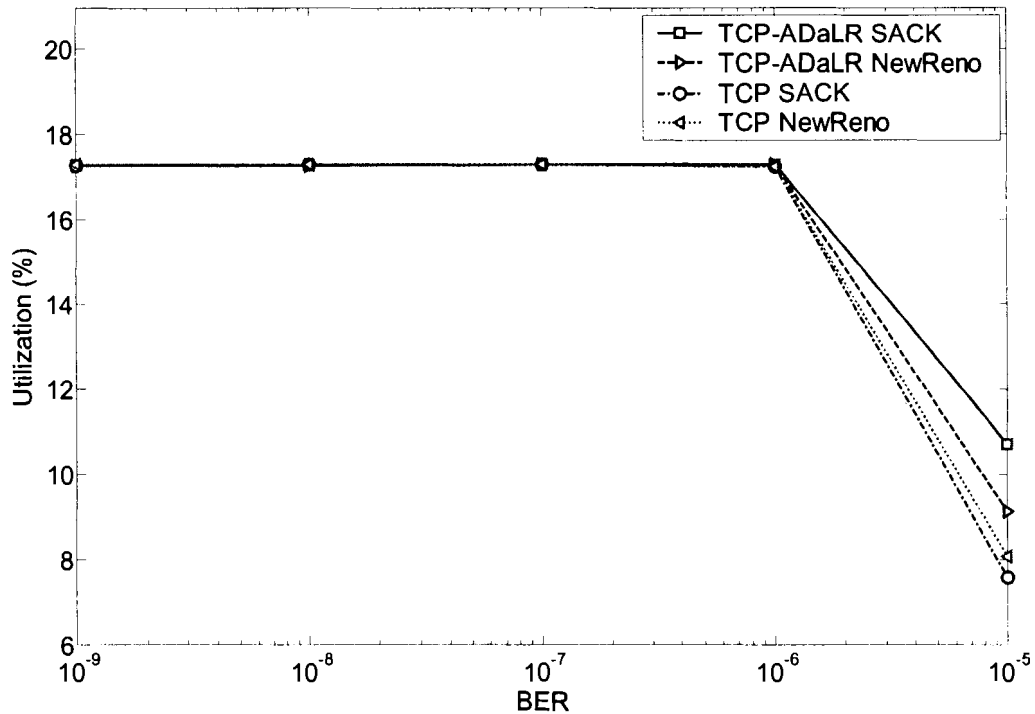
Figure 7.49. Satellite link utilization for scenarios with both congestion and error losses and delayed ACK disabled. At BER greater than 10^{-7}, TCP-ADaLR SACK exhibits the best performance.

7.5.4.2 Performance of HTTP application

We evaluate performance of the HTTP application in terms of the page response time. The increased HTTP page response time indicates the effect of both congestion and error losses, as shown in Figure 7.50 and 7.51.

TCP-ADaLR SACK shows the best performance in both cases with and without delayed ACK. Hence, the SACK option improves the TCP-ADaLR algorithm performance for the HTTP application. TCP-ADaLR SACK shows up to 32% and 26% shorter page response times than TCP SACK for cases with and without delayed ACK, respectively. The short and bursty nature of HTTP transfers ensures small number of outstanding unacknowledged bytes. Hence, when losses occur, the adaptive *cwnd* increase mechanism enables the HTTP web pages to open more quickly with TCP-

ADaLR than with TCP SACK and TCP NewReno. The adaptive *rwnd* and loss recovery mechanisms compensate for delayed ACK by allowing at least two segments to be transferred back-to-back during the fast recovery phase. Hence, the HTTP transfers are completed faster with the TCP-ADaLR. TCP-ADaLR NewReno outperforms TCP NewReno when the delayed ACK option is disabled. However, TCP-ADaLR NewReno performs worse than TCP NewReno when the delayed ACK option is enabled. This performance reduction may be due to loss of several original and retransmitted segments due to the initial high transmission rate of the TCP-ADaLR algorithm. In the absence of losses at the start of transmission, TCP-ADaLR opens the *cwnd* faster than TCP SACK and TCP NewReno, thus resulting in an initial high transmission rate. This is the only scenario we observed where TCP-ADaLR NewReno performs worse than TCP NewReno.
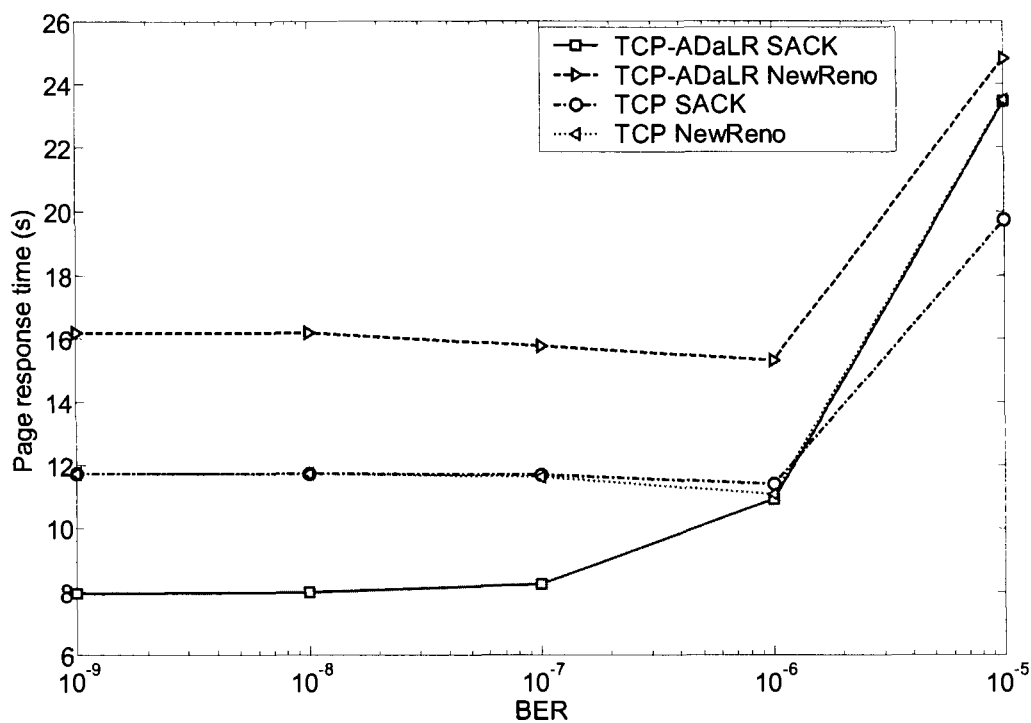
Figure 7.50. HTTP page response time for scenarios with both congestion and error losses and delayed ACK enabled. For most BER values, TCP-ADaLR SACK exhibits the best performance.
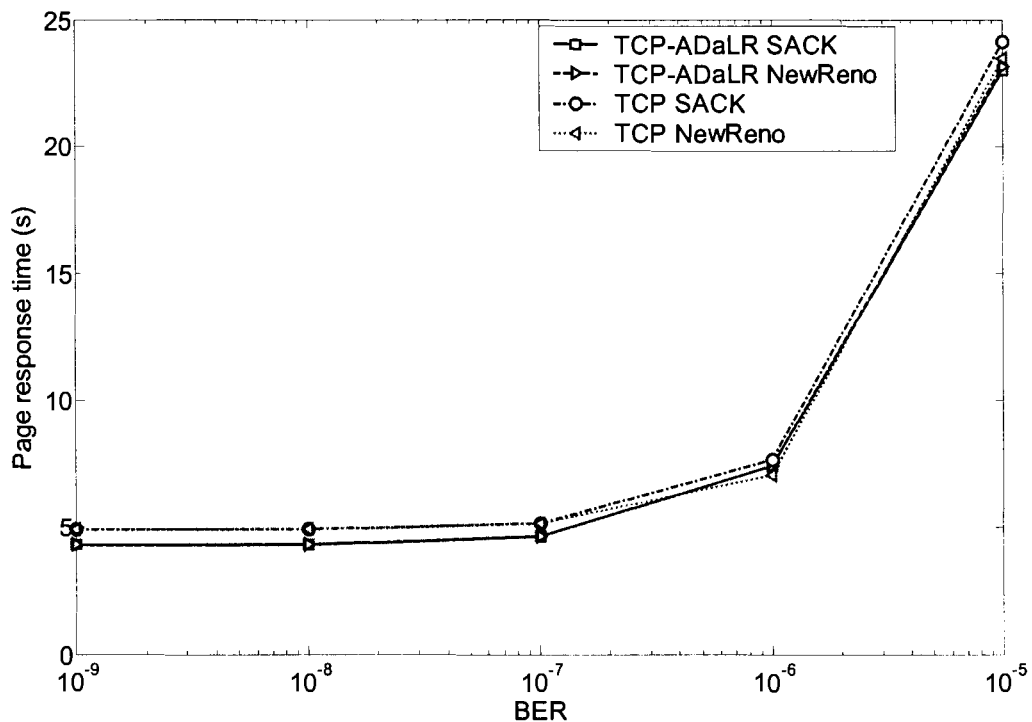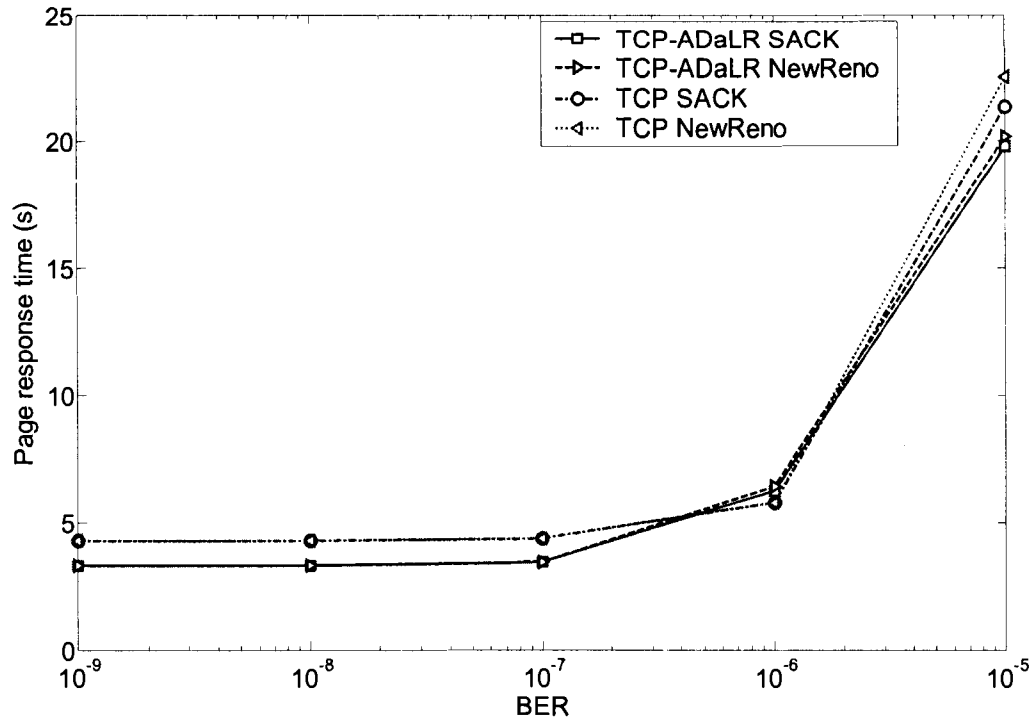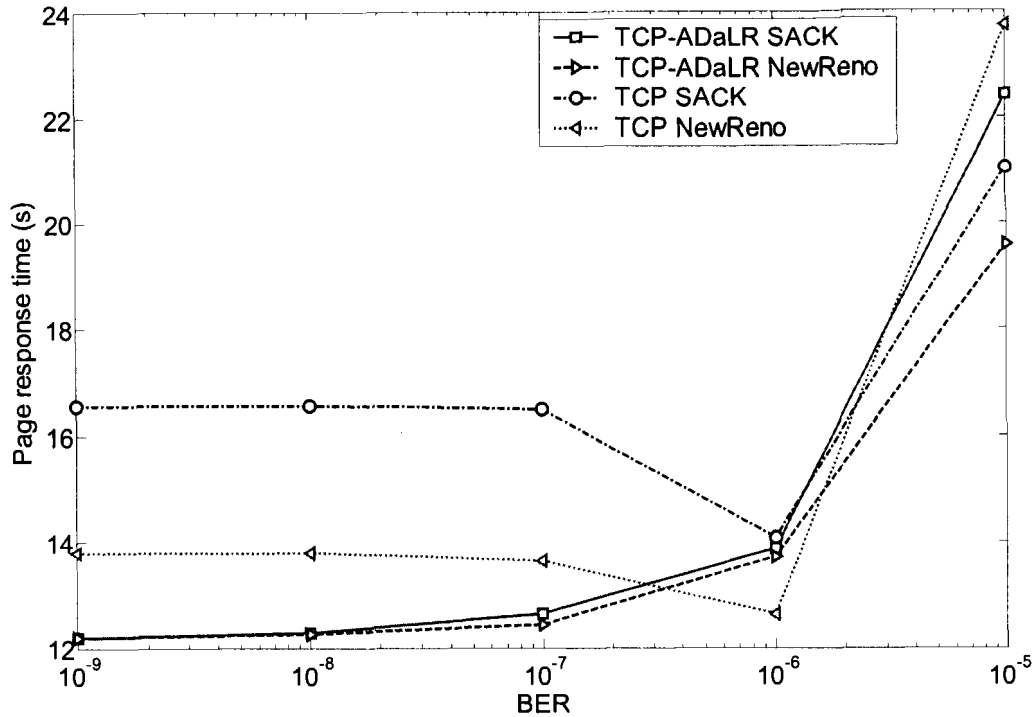
Figure 7.51. HTTP page response time for scenarios with both congestion and error losses and delayed ACK disabled. TCP-ADaLR SACK and TCP-ADaLR NewReno exhibit comparable performance and outperform both TCP SACK and TCP NewReno.

7.5.5 Fairness and friendliness

An important feature of TCP is its ability to ensure a fair division among multiple competing connections. A TCP variant is fair if coexisting connections achieve equal bandwidth allocation. Friendliness refers to coexisting TCP connections with distinct TCP variants having a fair share of the available bandwidth. We employ the Jain's metric of fairness [23] defined as

$$Fairness = \frac{(\sum_{j=1}^{n} t_j)^2}{n \times (\sum_{j=1}^{n} t_j^2)},$$ (7.4)

where n is the number of competing connections and t_j is the average throughput of the jth connection. The Jain's metric of fairness is used to evaluate both fairness and

friendliness. The fairness/friendliness metric has a value between $1/n$ and 1, where $1/n$ corresponds to unfair and 1 to fair (equal) bandwidth allocation for all n connections.

Max-min fairness [104] for multiple competing connections using the identical protocols is defined as

$$Fairness \ = \ \frac{t_{j_{min}}}{t_{j_{max}}},\qquad (7.5)$$

where $t_{j_{min}}$ and $t_{j_{max}}$ are the minimum and maximum throughputs, respectively. The max-min fairness metric is also used to evaluate fairness.

Common TCP variants such as TCP SACK and TCP NewReno are known to be fair when the competing connections have similar RTTs [65]. However, if the competing connections have different RTTs, the connections with shorter RTTs consume a larger fraction of the available bottleneck bandwidth and starve connections with longer RTTs.

TCP variants in deployed networks are expected to coexist and share bottleneck links among connections of distinct RTTs. We evaluate the fairness and friendliness of TCP-ADaLR NewReno in the absence of losses for an FTP application. (TCP-ADaLR NewReno and TCP-ADaLR SACK exhibit identical performance in the absence of losses. Similarly TCP NewReno and TCP SACK exhibit identical performance in the absence of losses.) All TCP connections are configured with the delayed ACK option enabled. We evaluate fairness and friendliness using the network model shown in Figure 7.52. All links, including the GEO satellite link, are bi-directional with 10 Mb/s data rate. Shown are one-way propagation delays. A single client connects to a single server and corresponding server-client pairs have the same subscript notation. We test six TCP

connections with various RTTs using two fairness scenarios where TCP connections employ TCP-ADaLR NewReno and TCP NewReno, respectively. We evaluate fairness using (7.4) and (7.5). In the friendliness scenario, we test three TCP-ADaLR NewReno and three TCP NewReno coexisting connections. We evaluate the friendliness of TCP-ADaLR NewReno using (7.4).
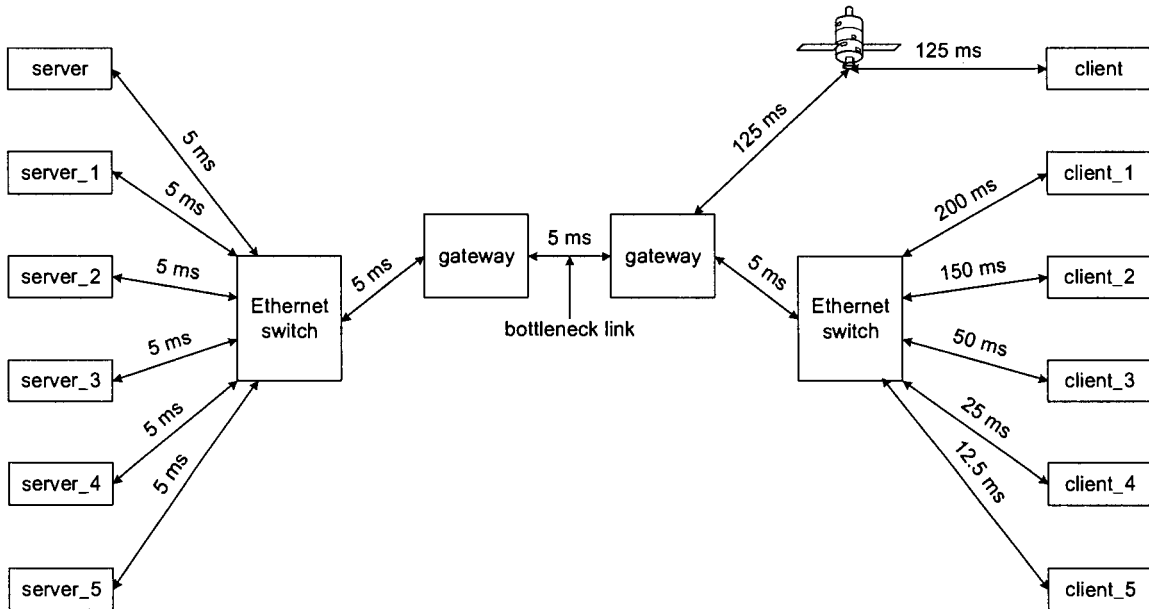
Figure 7.52. Network configuration for evaluating TCP fairness and TCP friendliness of TCP-ADaLR NewReno.

7.5.5.1 TCP-ADaLR fairness

The average throughput of the six TCP-ADaLR NewReno and six TCP NewReno connections are shown in Tables 7.11 and 7.12, respectively. The longest (500 ms) RTT connection using TCP-ADaLR NewReno has average throughput 47% higher than the corresponding TCP NewReno 500 ms RTT connection. Conversely, the average throughput of the shortest RTT connection using TCP-ADaLR NewReno drops by 12%. The fairness values are shown in Table 7.13. With both metrics of fairness, TCP-ADaLR

NewReno reduces the penalty caused by long RTT connections and exhibits better fairness than TCP NewReno.

Table 7.11. Average throughput achieved by six competing TCP-ADaLR NewReno connections, each with distinct RTTs.

RTT (ms)	Average throughput (bytes/s)
25	283,404.6
50	281,750.6
100	268,984.6
300	195,099.8
400	175,343.8
500	160,897.4

Table 7.12. Average throughput achieved by six competing TCP NewReno connections, each with distinct RTTs.

RTT (ms)	Average throughput (bytes/s)
25	322,418.0
50	300,629.1
100	263,129.2
300	158,601.5
400	129,560.8
500	109,239.5

Table 7.13. TCP fairness values of TCP-ADaLR NewReno and TCP NewReno using the Jain's metric of fairness and max-min fairness metric.

TCP Variant	$Fairness = \dfrac{(\sum_{j=1}^{n} t_j)^2}{n \times (\sum_{j=1}^{n} t_j^2)}$	$Fairness = \dfrac{t_{j_{min}}}{t_{j_{max}}}$
TCP-ADaLR NewReno	0.9510	0.5677
TCP NewReno	0.8650	0.3388

99

In the friendliness scenario, the average throughput of each competing connection is shown in Table 7.14. The friendliness value (close to 1), shown in Table 7.15, confirms that TCP-ADaLR NewReno is TCP-friendly. Hence, the three long RTT connections have a fair share of the bottleneck link's available bandwidth.

Table 7.14. Average throughput achieved by six competing TCP connections using distinct TCP variants.

RTT (ms)	TCP variant	Average throughput (bytes/s)
25	TCP NewReno	253,202.9
50	TCP NewReno	231,009.8
100	TCP NewReno	206,831.7
300	TCP-ADaLR NewReno	211,369.5
400	TCP-ADaLR NewReno	188,513.4
500	TCP-ADaLR NewReno	177,309.9

Table 7.15. TCP friendliness of TCP-ADaLR NewReno and TCP NewReno competing connections.

TCP variant mix	$Fairness = \dfrac{(\sum_{j=1}^{n} t_j)^2}{n \times (\sum_{j=1}^{n} t_j^2)}$	$Fairness = \dfrac{t_{j_{min}}}{t_{j_{max}}}$
TCP-ADaLR NewReno and TCP NewReno	0.9859	0.7000

CHAPTER 8: CONCLUSIONS AND FUTURE WORK

The importance of broadband GEO satellite networks in providing core high-speed backbone and broadband access for the next generation Internet cannot be over emphasized. TCP as the dominant transport protocol will continue to provide byte-stream transport layer services for evolving Internet applications.

In this thesis, we proposed the TCP-ADaLR algorithm (TCP with adaptive delay and loss response) to reduce the adverse impact of the long propagation delays and high BERs on TCP performance in heterogeneous networks with GEO satellite links. In each simulation scenario, we considered the case with the TCP delayed ACK option enabled and the case with the TCP delayed ACK option disabled. The TCP-ADaLR algorithm was implemented as an extension to TCP SACK. We also evaluated the algorithm performance when implemented as an extension to TCP NewReno. We named the two TCP-ADaLR variants TCP-ADaLR SACK and TCP-ADaLR NewReno. Simulation results indicated that TCP-ADaLR improves the end-to-end performance of TCP for HTTP and FTP applications in the absence of losses with delayed ACK enabled and with delayed ACK disabled. The TCP-ADaLR algorithm reduced the response times for downloading HTTP WebPages and FTP files. In the presence of only congestion losses, both TCP-ADaLR SACK and TCP-ADaLR NewReno show comparable performance to TCP SACK and TCP NewReno. In the presence of only error losses, TCP-ADaLR SACK outperforms TCP SACK and TCP NewReno and improves the average TCP throughput, TCP goodput, and satellite link utilization. TCP-ADaLR SACK also shows

better performance than TCP SACK and TCP NewReno in the presence of both congestion and error losses. Hence, TCP-ADaLR SACK exhibits the best performance in all cases followed by TCP-ADaLR NewReno. In each simulation scenario, TCP-ADaLR with delayed ACK disabled outperforms TCP-ADaLR with delayed ACK enabled. Hence, TCP-ADaLR does not degrade performance of TCP connections with delayed ACK disabled and yields better performance.

Increasing the propagation delay of the terrestrial segment reduces the performance of the four TCP variants. However, TCP-ADaLR exhibits better throughput and goodput performance than TCP SACK and TCP NewReno. High data rate is an important feature of broadband networks. With increased link data rate, TCP-ADaLR shows higher satellite link throughput and better link utilization than TCP SACK and TCP NewReno.

The deployment of TCP-ADaLR in heterogeneous networks requires modifications only at the TCP sender. These modifications place additional albeit minimal processing and memory overheads at the TCP sender. The TCP-ADaLR algorithm does not require modifications or introduction of packet prioritization mechanisms at intermediate network nodes. No modifications are required at the TCP receiver. TCP-ADaLR is fair to competing connections with different RTTs. It is also friendly to TCP NewReno connections. Hence, it may be deployed in networks with other TCP variants. Finally, TCP-ADaLR maintains the end-to-end semantics of TCP.

An important area of future research is performance comparison of TCP-ADaLR with TCP variants designed for satellite networks such as TCP Westwood, TCP Hybla, TCP-Peach, TCP NewVegas, and TCP-Star, which are currently not available in OPNET.

Furthermore, a mechanism to distinguish between congestion losses and error losses could be introduced to the TCP-ADaLR algorithm to further improve its performance in satellite networks. The performance of TCP-ADaLR algorithm may also be evaluated for NGEO networks, which exhibit characteristics such as frequent handovers and varying propagation delays.

APPENDIX A: FEATURES OF OPNET SIMULATIONS

In this Appendix, we present details of the simulation and simulated times for the four simulation scenarios.

In the OPNET discrete event simulator, simulation variables are defined in the *simulation set info* menu, as shown in Figure A.1. The *duration* is the specified real-time simulated (simulated time). The *seed* is the random seed number employed for the simulation run. Multiple seed numbers may be defined with the *multiple seed values* attribute. Four seed values were employed in the scenarios with various BER values. The *values per statistic* is the collection interval between collecting consecutive result data. Hence, for a *duration* = 18,000 s and *values per statistic* = 3,600, the interval between collecting consecutive result data is 5 s. The *update interval* is the number of events that are required before a simulation update is generated by the OPNET discrete event simulator. A higher value of *update interval* implies a longer interval update period. The *update interval* attribute can be set to a high value to reduce the number of updates received during simulation and, and, thus reduce the simulation time. The OPNET default is 500,000 events. However, we set this variable to 1, 000, 000 events for all simulations. The simulation time is the actual real-time that elapses for the simulation to be completed. A simulation may completed before the *duration* elapses as in the case when simulated application (FTP or HTTP) download is completed.
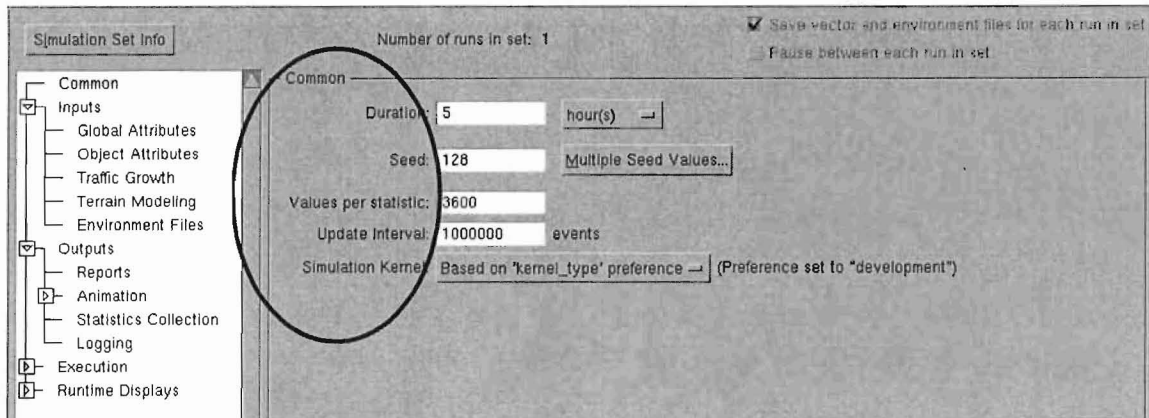
Figure A.1. The OPNET *simulation set info* menu for specifying simulator parameters. The common attributes include *duration*, *seed*, *values per statistic*, *update interval*, and *simulation kernel*.

A.1. FTP file download application

For the FTP file download application, the simulated time for all simulation scenarios was 5 hours (18,000 s). A single 50 MB file was downloaded in each scenario unless otherwise stated. However, each simulation scenario exhibited different simulation times for the four TCP variants, as shown in Tables A.1–A.5. The simulation time is shorter than the simulated time (18,000 s) in all scenarios because the simulated application (FTP file download) was completed before the simulated time (*duration*) elapsed. The simulation time increased with file size, as shown in Table A.2, because a larger file requires a higher number of events to take place before the file download is completed.

Table A.1. Simulation times for the four TCP variants with delayed ACK enabled in the ideal lossless satellite channel scenario for the simulated 50 MB FTP file download application.

TCP variant	Simulated time (s)	Simulation time (s)
TCP-ADaLR SACK	18,000	179
TCP-ADaLR NewReno	18,000	179
TCP SACK	18,000	184
TCP NewReno	18,000	184

Table A.2. Simulation times (s) for TCP-ADaLR NewReno and TCP NewReno with delayed ACK enabled for various file sizes in the ideal lossless satellite channel scenarios for evaluating the effect of increasing file size.

TCP variant	Simulated time (s)	File size (MB)				
		100	200	300	400	500
TCP-ADaLR NewReno	18,000	306	741	1,124	1,447	
TCP NewReno	18,000	309	732	1,123	1,455	

Table A.3. Simulation times for the four TCP variants with delayed ACK enabled in the scenarios with only congestion losses for the simulated 50 MB FTP file download application.

TCP variant	Simulated time (s)	Simulation time (s)
TCP-ADaLR SACK	18,000	192
TCP-ADaLR NewReno	18,000	257
TCP SACK	18,000	195
TCP NewReno	18,000	187

Table A.4. Simulation times (s) for the four TCP variants with delayed ACK enabled in the scenarios with only error losses for the simulated 50 MB FTP file download application.

TCP variant	Simulated time (s)	Bite error rate				
		10^{-9}	10^{-8}	10^{-7}	10^{-6}	10^{-5}
TCP-ADaLR SACK	18,000	189	190	190	197	243
TCP-ADaLR NewReno	18,000	183	183	190	201	243
TCP SACK	18,000	185	184	186	201	221
TCP NewReno	18,000	185	188	194	195	219

Table A.5. Simulation times for the four TCP variants with delayed ACK enabled in the scenarios with both congestion and error losses for the simulated 50 MB FTP file download application.

TCP variant	Simulated time (s)	Bite error rate				
		10^{-9}	10^{-8}	10^{-7}	10^{-6}	10^{-5}
TCP-ADaLR SACK	18,000	189	183	189	203	244
TCP-ADaLR NewReno	18,000	189	183	187	201	244
TCP SACK	18,000	198	183	187	195	214
TCP NewReno	18,000	179	183	186	194	271

A.2. HTTP web page download application

For the HTTP web page download application (Table 7.3), the simulated time for all simulation scenarios was 600 s. Each simulation scenario exhibited comparable simulation times for the four TCP variants, as shown in Tables A.6–A.9.

Table A.6. Simulation times for the four TCP variants with delayed ACK enabled in the ideal lossless satellite channel scenarios for the simulated HTTP web page download application.

TCP variant	Simulated time (s)	Simulation time (s)
TCP-ADaLR SACK	600	13
TCP-ADaLR NewReno	600	13
TCP SACK	600	13
TCP NewReno	600	13

Table A.7. Simulation times for the four TCP variants with delayed ACK enabled in the scenarios with only congestion losses for the simulated HTTP web page download application.

TCP variant	Simulated time (s)	Simulation time (s)
TCP-ADaLR SACK	600	14
TCP-ADaLR NewReno	600	15
TCP SACK	600	13
TCP NewReno	600	13

Table A.8. Simulation times (s) for the four TCP variants with delayed ACK enabled in the scenarios with only error losses for the simulated HTTP web page download application.

TCP variant	Simulated time (s)	Bite error rate				
		10^{-9}	10^{-8}	10^{-7}	10^{-6}	10^{-5}
TCP- ADaLR SACK	600	14	14	13	14	13
TCP- ADaLR NewReno	600	12	14	13	13	13
TCP SACK	600	12	13	13	13	13
TCP NewReno	600	12	14	13	13	13

Table A.9. Simulation times (s) for the four TCP variants with delayed ACK enabled in the scenarios with both congestion and error losses for the simulated HTTP web page download application.

TCP variant	Simulated time (s)	Bite error rate				
		10^{-9}	10^{-8}	10^{-7}	10^{-6}	10^{-5}
TCP-ADaLR SACK	600	12	14	14	14	14
TCP-ADaLR NewReno	600	11	14	14	14	15
TCP SACK	600	12	14	13	13	14
TCP NewReno	600	12	14	13	14	13

REFERENCE LIST

[1] V. Cerf and R. Kahn, "A protocol for packet network intercommunication," *IEEE Trans. Commun.*, vol. 22, no. 5, pp. 637–648, May 1974.

[2] J. Postel, Ed., "Transmission Control Protocol," *RFC 793*, Sept. 1981.

[3] W. Stevens, *TCP Illustrated Volume 1: The Protocols*. Reading, MA: Addison-Wesley, 1994.

[4] J. Postel, Ed., "Internet Protocol," *RFC 791*, Sept. 1981.

[5] M. Fomenkov, K. Keys, D. Moore, and K. Claffy, "Longitudinal study of Internet traffic in 1998-2003," in *Proc. ACM Winter Int. Symp. Inf. and Commun. Technol.*, Cancun, Mexico, Jan. 2004, pp. 1–6.

[6] C. Fraleigh et al., "Packet-level traffic measurements from the Sprint IP backbone," *IEEE Netw.*, vol. 17, no. 6, pp. 6–16, Nov./Dec. 2003.

[7] C. Williamson, "Internet traffic measurement," *IEEE Internet Comput.*, vol. 5, no. 6, pp. 70–74, Nov./Dec. 2001.

[8] A. Jamalipour, M. Marchese, H. Cruickshank, J. Neal, and S. Verma, "Broadband IP Networks via satellites-part II," *IEEE J. Select. Areas Commun.*, vol. 22, no. 3, pp. 433–437, Apr. 2004.

[9] M. Allman, V. Paxson, and W. Stevens, "TCP congestion control," *RFC 2581*, Apr. 1999.

[10] V. Jacobson, "Congestion avoidance and control," in *Proc. ACM SIGCOMM Symp. on Commun. Archit. and Protocols*, Stanford, CA, Aug. 1988, pp. 314–329.

[11] T-L. Pham, G. Schneider, S. Goose, and A. Pizano, "Composite device computing environment: a framework for situated interaction using small screen devices," *Pers. and Ubiquitous Comput.*, vol. 5, no. 1, pp. 25–28, Feb. 2001.

[12] A. Jamalipour, M. Marchese, H. Cruickshank, J. Neal, and S. Verma, "Broadband IP Networks via satellites-part I," *IEEE J. Select. Areas Commun.*, vol. 22, no. 2, pp. 213–217, Feb. 2004.

[13] R. A. Peters and M. Farrell, "Comparison of LEO and GEO satellite systems to provide broadband services," in *Proc. 21st AIAA Int. Commun. Satellite Syst. Conf. and Exhibit*, Yokohama, Japan, Apr. 2003, AIAA–2003–2246.

[14] R. Braden, "Requirements for Internet hosts–communication layers," *RFC 1122*, Oct. 1989.

[15] V. Paxson, "Automated packet trace analysis of TCP implementations," in *Proc. ACM SIGCOMM Conf. on Appl., Technol., Archit., and Protocols for Comput. Commun.*, Cannes, France, Sept. 1997, pp. 167–179.

[16] T. Lang and D. Floreani, "The impact of delayed acknowledgements on TCP performance over satellite links," in *Proc. ACM First Workshop on Wireless Mobile Internet*, Rome, Italy, July 2001, pp. 56–61.

[17] S. Floyd, T. Henderson, and A. Gurtov, "The NewReno modification to TCP's fast recovery algorithm," *RFC 3782*, Apr. 2004.

[19] OPNET Modeler software [Online]. Available: http://www.opnet.com/products/modeler/home.html.

[18] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanov, "TCP selective acknowledgement options," *RFC 2018*, Oct. 1996.

[20] A. Gurtov and S. Floyd, "Modeling wireless links for transport protocols," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 2, pp. 85–96, Apr. 2004.

[21] D. E. Comer, *Internetworking with TCP/IP: Principles, Protocols, and Architecture*. vol. 1. Upper Saddle River, NJ: Prentice Hall, 2000, pp. 209–218.

[22] A. Jamalipour, *The Wireless Mobile Internet: Architectures, Protocols and Services*. West Sussex, England: John Wiley & Sons, 2003, pp. 326–329.

[23] D. Chiu and R. Jain, "Analysis of the increase/decrease algorithms for congestion avoidance in computer networks," *J. of Comput. Netw. ISDN Syst.*, vol. 17, no. 1, pp. 1–14, June 1989.

[24] M. Allman, S. Floyd, and C. Patridge, "Increasing TCP's initial window," *RFC 2414*, Sept. 1998.

[25] P. Karn and C. Partridge, "Improving round-trip time estimates in reliable protocols," in ACM *Trans. Comput. Syst.*, vol. 9, no. 4, pp.364–373, Nov. 1991.

[26] V. Paxson and M. Allman, "Computing TCP's retransmission timer," *RFC 2988*, Nov. 2000.

[27] M. Emmelmann, "Effects of advertised receive buffer size and timer granularity on TCP performance over erroneous links in a LEO satellite network," in *Proc. IEEE GLOBECOM*, Taipei, Taiwan, Nov. 2002, pp. 2955–2958.

[28] S. Floyd, "TCP and explicit congestion notification," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 24, no. 5, pp. 10–23, Oct. 1994.

[29] M. Allman and V. Paxson, "On estimating end-to-end network path properties," in *Proc. ACM SIGCOMM Conf. on Appl., Technol., Archit., and Protocols for Comput. Commun.*, Cambridge, MA, Sept. 1999, pp. 263–274.

[30] D. D. Clark, "Window and acknowledgement strategy in TCP," *RFC 813*, July 1982.

[31] C. Joo and S. Bahk, "Increasing TCP capacity in wireless multihop networks," in *Web and Commun. Technol. and Internet-Related Social Issues–HSI, Lecture Notes in Comput. Science*. Springer, Berlin: vol. 3597, pp. 37–44, 2005.

[32] H. Balakrishnan, V. Padmanabhan, and Randy Katz, "The effects of asymmetry on TCP performance," in *Proc. ACM Int. Conf. on Mobile Comput. and Netw. (MobiCom)* 2005, Budapest, Hungary, Sept. 1997, pp. 77–89.

[33] M. Allman, Ed., "Ongoing TCP Research Related to Satellites," *RFC 2760*, Feb. 2000.

[34] J. Chen, Y. Z. Lee, M. Gerla, and M. Y. Sanadidi, "TCP with delayed ack for wireless networks," in *Proc. IEEE/CreateNet BROADNETS 2006*, San Jose, CA, USA, Oct. 2006.

[35] B. R. Elbert, *Introduction to Satellite Communication*, 2nd ed. Norwood, MA: Artech House, 1999.

[36] B. R. Elbert, *The Satellite Communications Applications Handbook*, 2nd ed. Norwood, MA: Artech House, 2004.

[37] L. Wood, "Satellite constellation networks," chapter 2 in *Internetworking and Computing over Satellite Networks*. Y. Zhang, Ed. Norwell, MA: Kluwer Academic Publishers, 2003, pp. 13–34.

[38] M. Allman, D. Glover, and L. Sanchez, "Enhancing TCP over satellite links using standard mechanisms," *RFC 2488*, Jan. 1999.

[39] Y. Shang and M. Hadjitheodosiou, "TCP splitting protocol for broadband and aeronautical satellite network," in *Proc. 23rd IEEE Digital Avionics Syst. Conf.*, Salt Lake City, UT, Oct.2004, vol. 2, pp. 11.C.3-1–11.C.3-9.

[40] T. R. Henderson and R. H. Katz, "Transport protocol for Internet-compatible satellite networks," *IEEE J. Select. Areas Commun.*, vol. 17, no. 2, pp. 326–344, Feb. 1999.

[41] A. Jamalipour, "Broad-band satellite networks – the global IT bridge," *in Proc. IEEE*, vol. 89, no. 1, pp. 88–104, Jan. 2001.

[42] C. Patridge and T. J. Shepard, "TCP/IP performance over satellite links," *IEEE Netw.*, vol. 11, no. 5, pp. 44–49, Sept./Oct. 1997.

[43] S. Subramanian, S. Sivakumar, W. J. Phillips, and W. Robertson, "Investigating TCP performance issues in satellite networks," in *Proc. Third IEEE Commun. Netw. and Services Research Conf.*, Halifax, NS, Canada, May 2005, pp. 327–332.

[44] J. Sing and B. Soh, "TCP performance over geostationary satellite links: problems and solutions," in *Proc. 12th IEEE Int. Conf. on Netw.*, Guadeloupe, French Caribbean, Nov. 2004, vol. 1, pp. 14–18.

[45] I. F. Akyildiz, G. Morabito, and S. Palazzo, "Research issues for transport protocols in satellite IP networks," IEEE *Pers. Commun. Mag.*, vol. 8, no. 3, pp. 44–48, June 2001.

[46] V. Jacobson, R. Braden, and D. Borman, "TCP extensions for high performance," *RFC 1323*, May 1992.

[47] N. Ghani and S. Dixit, "TCP/IP enhancements for satellite networks," *IEEE Commun. Mag.*, vol. 37, no. 7, pp. 64–72, July 1999.

[48] J. Mogul and S. Deering, "Path MTU discovery," *RFC 1191*, Nov. 1990.

[49] C. Barakat, N. Chaher, W. Dabbous, and E. Altman, "Improving TCP/IP over geostationary satellite links," in *Proc. IEEE GLOBECOM*, Rio de Janeiro, Brazil, Dec. 1999, vol. 1b, pp. 781–785.

[50] V. Padmanabhan and R. Katz, "TCP fast start: a technique for speeding up web transfers," in *Proc. IEEE GLOBECOM Internet Mini-Conf.*, Sydney, Australia, Nov. 1998.

[51] M. Gerla, W. Weng, and R. L. Cigno, "BA-TCP: a bandwidth aware TCP for satellite networks," in *Proc. Eighth IEEE Int. Conf. on Comput. Commun. and Netw.*, Las Vegas, NV, Oct. 1999, pp. 204–207.

[52] J. Peng, P. Andreadis, C. Belisle, and M. Barbeau, "Improving TCP performance over long delay satellite links," *OPNETWORK 2001*, Washington, DC, Aug. 2001.

[53] I. F. Akyildiz, G. Morabito, and S. Palazzo, "TCP-Peach: a new congestion control scheme for satellite IP networks," *IEEE/ACM Trans. Netw.*, vol. 9, no. 3, pp. 307–321, June 2001.

[54] I. F. Akyildiz, X. Zhang, and J. Fang, "TCP-Peach+: enhancement of TCP-Peach for satellite IP networks," *IEEE Commun. Lett.*, vol. 6, no. 7, pp. 303–305, July 2001.

[55] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang, "TCP Westwood: end-to-end congestion control for wired/wireless networks," *Wireless Netw.*, vol. 8, no. 5, pp. 467–479, Sept. 2002.

[56] R. Wang, G. Pau, K. Yamada, M. Y. Sanadidi, and M. Gerla, "TCP startup performance in large bandwidth delay networks," in *Proc. IEEE INFOCOM 2004*, Hong Kong, China, Mar. 2004, vol. 2, pp. 796–805.

[57] G. Yang, R. Wang, M. Gerla, and M. Y. Sanadidi, "TCP bulk repeat," *Comput. Commun.*, vol. 28, no. 5, pp. 507–518, Mar. 2005.

[58] S. Floyd, "HighSpeed TCP for large congestion windows," *RFC 3649*, Dec. 2003.

[59] T. Kelly, "Scalable TCP: improving performance in high speed wide area networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 2, pp. 83–91, Apr. 2003.

[60] G. Giambene and D. Miorandi, "A simulation study of scalable TCP and highSpeed TCP in geostationary satellite networks," *Telecommun. Syst.*, vol. 30, no. 4, pp. 297–320, Dec. 2005.

[61] K-F. Leung and K. L. Yeung, "TCP-Swift: an end-to-end host enhancement scheme for TCP over satellite IP networks," in *Proc. Ninth IEEE Int. Symp. on Comput. and Commun.*, Alexandria, Egypt, July 2004, vol. 1, pp. 551–555.

[62] H. Xu and S. Wu, "A priority-based TCP congestion control strategy in satellite IP networks," in *Proc. Int. Conf. on Commun., Circuits, and Syst.*, Hong Kong, China, May 2005, pp. 402–406.

[63] H. Obata, K. Ishida, S. Takeuchi, and S. Hanasaki, "TCP-STAR: TCP congestion control method for satellite Internet" *IEICE Trans. Commun.*, vol. E89-B, no. 6, pp. 1766–1773, June 2006.

[64] K. Xu, Y. Tian, and N. Ansari, "TCP-Jersey for wireless IP communications," *IEEE J. Select. Areas Commun.*, vol. 22, no. 4, pp. 747–756, May 2004.

[65] C. Caini and R. Firrincieli, "TCP Hybla: a TCP enhancement for heterogeneous networks," *Int. J. Satellite Commun. Netw.*, vol. 22, no. 5, pp. 547–566, Sept. 2004.

[66] J. Sing and B. Soh, "TCP New Vegas: improving the performance of TCP Vegas over high latency links," in *Proc. Fourth IEEE Int. Symp. on Netw. Comput. and Appl.*, Cambridge, MA, July 2005, pp. 73–82.

[67] L. S. Brakmo and L. L. Peterson, "TCP Vegas: end to end congestion avoidance on a global Internet," *IEEE J. Select. Areas Commun.*, vol. 13, no. 8, pp. 1465–1480, Oct. 1995.

[68] T. R. Henderson and R. H. Katz, "TCP performance over satellite channels," University of California, Berkeley, CA, Tech. Rep. CSD-99-1083, Dec. 1999.

[69] Y. Zhang, D. Delucia, B. Ryu, and S. Dao, "Satellite communications in the global Internet: issues, pitfalls, and potential," in *Proc. Seventh ISOC Int. Netw. Conf.*, Kuala Lumpur, Malaysia, June 1997.

[70] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby, "Performance enhancing proxies intended to mitigate link-related degradations," *RFC 3135*, June 2001.

[71] D. Velenis, D. Kalogeras, and B. Maglaris, "SaTPEP: a TCP performance enhancing proxy for satellite link," in *Networking: Second Int. IFIP-TC6 Netw. Conf., Lecture Notes in Comput. Science*. Springer, Berlin, vol. 2345, pp. 1233–1238, 2002.

[72] C. Caini, R. Firrincieli, and D. Lacamera, "PEPsal: a performance enhancing proxy designed for TCP satellite connections," in *Proc. 63rd IEEE Veh. Technol. Conf.*, Melbourne, Australia, Feb. 2006, vol. 6, pp. 2607–2611.

[73] G. Ciccarese, M. De Blas, L. Patrono, P. Marra, and G. Tomasicchio, "An IPSec-aware TCP PEP for integrated mobile satellite networks," in *Proc. 15th IEEE Int. Symp. on Pers., Indoor, and Mobile Radio Commun. (PIMRC) 2004*, Barcelona, Spain, Sept. 2004, vol. 4, pp. 2362–2366.

[74] U. Lee and S. F. Midkiff, "Quality of service for TCP over satellite links in congested networks," in *Proc. IEEE Wireless Commun. and Netw. Conf. (WCNC) 2005*, New Orleans, LA, USA, Mar. 2005, vol. 3, pp. 1515–1520.

[75] J. Ishac and M. Allman, "On the performance of TCP spoofing in satellite networks," in *Proc. IEEE MILCOM*, Monterey, CA, Oct. 2001, vol. 1, pp. 700–704.

[76] E. A. Faulkner, A. P. Worthen, J. B. Schodorf, and J. D. Choi, "Interactions between TCP and link layer protocols on mobile satellite links," in *Proc. IEEE MILCOM*, Monterey, CA, Nov. 2004, vol. 1, pp. 535–541.

[77] E. Amir, H. Balakrishnan, S. Seshan, and R. H. Katz, "Efficient TCP over networks with wireless links," in *Proc. IEEE Fifth Workshop on Hot Topics in Operating Syst. (HotOS-V) 1995*, Orcas Island, WA, May 1995, pp. 35–40.

[78] H. Balakrishnan, S. Seshan, and R. H. Katz, "Improving reliable transport and handoff performance in cellular wireless networks," *Wireless Netw.*, vol. 1, no. 4, pp. 469–481, Dec. 1995.

[79] J. Sing and B. Soh, "On the use of snoop with geostationary satellite links," in *Proc. Third IEEE Int. Conf. on Inf. Technol. and Appl. (ICITA) 2005*, Sydney, Australia, July 2005, vol. 2, pp. 689–694.

[80] J. S. Stadler, "A link layer protocol for efficient transmission of TCP/IP via satellite," in *Proc. IEEE MILCOM*, Monterey, CA, Nov. 1997, vol. 2, pp. 723–727.

[81] W. G. Zeng and Lj. Trajković, "TCP packet control for wireless networks," in *Proc. IEEE Int. Conf. on Wireless and Mobile Comput., Netw., and Commun. (WiMob) 2005*, Montreal, QC, Canada, Aug. 2005, vol. 2, pp. 196–203.

[82] R. C. Durst, G. J. Miller, and E. J. Travis, "TCP extensions for space communications," *Wireless Netw.*, vol. 3, no. 5, pp. 389–403, Oct. 1997.

[83] R. Wang, V Bandekodige, and M. Banerjee, "An experimental evaluation of link delay impact on throughput performance of TCP and SCPS-TP in space communications," in *Proc. 60th IEEE Veh. Technol. Conf.*, Los Angeles, CA, Sept. 2004, vol. 6, pp. 4061–4065.

[84] R. H. Wang and S. Horan, "Performance evaluation of TCP and its extensions over lossy links in a small satellite environment," in *Proc. IEEE Int. Conf. on Commun.*, Seoul, Korea, May 2005, vol. 3, pp. 1478–1482.

[85] T. R. Henderson and R. H. Katz, "Transport protocols for Internet-compatible satellite networks," *IEEE J. Select. Areas Commun.*, vol. 17, no. 2, pp. 326–344, Feb. 1999.

[86] *B-ISDN Signalling ATM Adaptation Layer–Service Specific Connection Oriented Protocol*, ITU-T Recommendation Q.2110, 1994.

[87] M. E. Elaasar, M Barbeau, E. Kranakis, and Z. Li, "Satellite transport protocol handling bit corruption, handoff and limited connectivity," *IEEE Trans. Aerosp. and Electron. Syst.*, vol. 41, no. 2, pp. 489–502, Apr. 2005.

[88] I. F. Akyildiz, O. B. Akan, and G. Morabito, "A rate control scheme for adaptive real-time applications in IP networks with lossy links and long RTTs," *IEEE/ACM Trans. Netw.*, vol. 13, no. 3, pp. 554–567, June. 2005.

[89] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 4, pp. 89–102, Oct. 2002.

[90] K. Zhou, K. L. Yeung, and V. O. K. Li, "P-XCP: a transport layer protocol for satellite IP networks," in *Proc. IEEE GLOBECOM*, Dallas, TX, Dec. 2004, vol. 5, pp. 2707–2711.

[91] A. Kapoor, A. Falk, T. Faber, and Y. Pryadkin, "Achieving faster access to satellite link bandwidth," in *Proc. IEEE INFOCOM 2005*, Miami, FL, USA, Mar. 2005, vol. 4, pp. 2870–2875.

[92] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson, "Stream Control Transmission Protocol," *RFC 2960*, Oct. 2000.

[93] S. Fu and M. Atiquzzan, "SCTP: state of the art in research, products, and technical challenges," *IEEE Commun. Mag.*, vol. 42, no. 4, pp. 64–76, Apr. 2004.

[94] M. Omueti and Lj. Trajković, "TCP with adaptive delay and loss response for heterogeneous networks," to be presented at *Third Int. Wireless Internet Conf. (WICON) 2007*, Austin, TX, Oct. 2007.

[95] M. Allman, "TCP congestion control with appropriate byte counting (ABC)," *RFC 3465*, Feb. 2003.

[96] E. Blanton and M. Allman, "On the impact of bursting on TCP performance," in *Passive and Active Measurement (PAM) 2005, Lecture Notes in Comput. Science.* Springer, Berlin: vol. 3431, pp. 1–12, Mar. 2005.

[97] M. Omueti and Lj. Trajković, "OPNET model of TCP with adaptive delay and loss response for broadband GEO satellite networks," to be presented at *OPNETWORK 2007*, Washington, DC, Aug. 2007.

[98] OPNET product documentation, V.11.0.A., OPNET Technologies Inc., Bethesda, MD, 2004.

[99] J. Zhu, S. Roy, and J. H. Kim, "Performance modelling of TCP enhancements in terrestrial-satellite hybrid networks," *IEEE/ACM Trans. Netw.*, vol. 14, no. 4, pp. 753–766, Aug. 2006.

[100] 3GPP/TSG-C.R1002, "1xEV-DV evaluation methodology (v14)," June 2003.

[101] A. Medina, M. Allman, and S. Floyd, "Measuring the evolution of transport protocols in the Internet," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 2, pp. 37–52, Apr. 2005.

[102] F. Vacirca, F. Ricciato, and R. Pilz, "Large-scale RTT measurements from an operational UMTS/GPRS network," in *Proc. First Int. Wireless Internet Conf. (WICON) 2005*, Budapest, Hungary, July 2005, pp. 190–197.

[103] Y. Chotikapong, H. Cruickshank, and Z. Sun, "Evaluation of TCP and Internet traffic via low earth orbit satellites," *IEEE Pers. Commun. Mag.*, vol. 8, no. 3, pp. 28–34, June 2001.

[104] D. P. Bertsekas and R. G. Gallager, *Data Networks*, 2nd ed., Englewood Cliffs, NJ: Prentice Hall, 1992.