TCP with Adaptive Delay and Loss Response for Heterogeneous Networks

Modupe Omueti and Ljiljana Trajković
Simon Fraser University
Vancouver, British Columbia, Canada
{momueti, ljilja}@cs.sfu.ca

ABSTRACT

Long propagation delays and high bit error rates in heterogeneous networks with geostationary earth orbit (GEO) satellite links have negative impact on the performance of Transmission Control Protocol (TCP). In this paper, we propose modifications to TCP by introducing adaptive delay and loss response (TCP-ADaLR) to mitigate the adverse effects of satellite link characteristics. The proposed modifications incorporate delayed acknowledgment (ACK) recommended for Internet hosts. TCP-ADaLR introduces adaptive window increase and loss recovery mechanisms to address TCP performance degradation in satellite networks. We evaluate and compare the performance of TCP-ADaLR, TCP SACK, and TCP NewReno, with delayed ACK enabled and disabled. In the absence of losses, TCP-ADaLR exhibits the shortest user-perceived latency for HTTP and FTP applications. In the presence of only congestion losses, TCP-ADaLR shows comparable performance to TCP SACK and TCP NewReno. In the presence of only 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 the best performance in the absence of losses and in the presence of losses due to both congestion and errors. It also friendly to TCP NewReno, exhibits better fairness, and maintains TCP end-to-end semantics.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols – *Protocol Verification.*

General Terms

Algorithm, Performance

Keywords

Heterogeneous networks, GEO satellite networks, TCP, network simulation, performance evaluation.

1. INTRODUCTION

Transmission Control Protocol (TCP) provides connection-oriented byte-stream delivery services for Internet applications and carries up to 90% of Internet traffic [1]. The volume of Internet traffic has increased due to bandwidth-intensive multimedia and data applications that require high data rates [2]. These high data rates are offered through high-bandwidth geostationary earth orbit (GEO) satellite links. Through their easily scalable architecture and multicast capabilities [3], heterogeneous broadband GEO satellite networks provide global Internet access and coverage to areas with limited or no terrestrial cable infrastructure. Heterogeneous networks with GEO satellite links are more attractive for continuous coverage than networks employing non-GEO (NGEO) satellite links because of lower development risks compared to large number of satellite constellations required by NGEO satellites [4].

TCP was originally designed for wired networks with the underlying assumption that packet losses indicate congestion. Hence, TCP was enhanced by the congestion control algorithms [5], [6] to address congestion losses thus enabling TCP to perform well in wired networks. GEO satellite links are characterized by high bit error rate (BER), long propagation delay, and bandwidth asymmetry (different uplink and downlink bandwidth). TCP performs poorly in heterogeneous networks with GEO satellite links because of these 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, the long propagation delays of GEO satellite links result in large round trip times (RTTs), which negatively impact TCP performance. These large RTTs prevent a TCP sender from transmitting segments at the maximum rate causing reduced throughput and, hence, poor utilization of GEO satellite links.

In this paper, we propose TCP with adaptive delay and loss response (TCP-ADaLR) for improving TCP performance in heterogeneous networks with GEO satellite links. TCP-ADaLR is an end-to-end algorithm implemented as an extension to TCP SACK. It may also be applied to TCP NewReno. TCP SACK and TCP NewReno are prevalent TCP variants [7]. More than 50% of Internet servers are SACK-capable and majority of those that do not use SACK employ TCP NewReno for loss recovery [8]. The TCP-ADaLR algorithm is designed for the case when the TCP delayed acknowledgment (ACK) option is enabled. This TCP option [9] allows a TCP receiver to send an ACK for every second consecutive full-size packet received from the sender. (A full-size packet is equivalent to the sender maximum segment size (SMSS) packet.) TCP delayed ACK option is recommended for Internet hosts [9]. This option has been enabled by many current TCP implementations [10].

The paper is organized as follows: An overview of TCP and its congestion control algorithms is given in Section 2. In Section 3,

WICON 2007, October 22-24, 2007, Austin, Texas, USA.

we present an overview of satellite links, the impact of their characteristics on TCP, and a survey of related work. In Section 4, we describe the TCP-ADaLR algorithm and its mechanisms for improving the end-to-end performance of TCP in heterogeneous networks with GEO satellite links. Simulation scenarios, results of the performance evaluation, and comparison of TCP-ADaLR, TCP SACK, and TCP NewReno are presented in Section 5. We conclude with Section 6.

2. OVERVIEW OF THE TRANSMISSION CONTROL PROTOCOL

A TCP connection is established after the three-way handshake. To manage the volume and rate of data transmitted between a sender and a receiver, TCP employs four congestion control algorithms [5], [6]: slow start, congestion avoidance, fast retransmit, and fast recovery. A TCP sender transmits segments based on ACKs from a receiver. When a TCP sender receives an ACK, it increases the congestion window *cwnd* based on the number of acknowledged bytes and the receiver's advertised window *rwnd*. TCP increments the *cwnd* exponentially during the slow start phase and linearly during the congestion avoidance phase. The slow start threshold *ssthresh* determines the end of the slow start phase and the onset of the congestion avoidance phase. A TCP sender enters the fast retransmit phase when a segment loss is detected by three duplicate ACKs. After it receives an ACK of a new segment, a TCP sender then enters the fast recovery phase and resumes the congestion avoidance phase. When a segment loss is detected by retransmission timeout (RTO), a TCP sender retransmits the lost segment, resets the *ssthresh* value, and reverts to slow start.

A TCP receiver may increase efficiency by sending less than one ACK segment per data segment [9]. This TCP option is known as delayed ACK. If a TCP receiver does not enable the delayed ACK option, separate ACKs will be required for acknowledging data segments and sending window updates. If there is two-way data transfer, a delayed ACK may also be sent along with the data segment (ACK piggybacks with data). Hence, the delayed ACK option reduces protocol processing overhead [11]. The default interval before sending an ACK is 200 ms. However, a TCP receiver may not delay an ACK for more than 500 ms. Most TCP implementations use the default interval of 200 ms [9].

3. GEO SATELLITE LINKS

A satellite in a geostationary earth orbit (GEO) is circular in shape and lies in the plane of the equator. A GEO satellite orbits at an altitude of ~36,000 km above the earth surface with a period of 24 hours (earth rotation period). Hence, a GEO satellite appears to be stationary to observers from the earth. A single GEO satellite has a large footprint (satellite signal coverage area of the earth surface). Hence, receiving antennas positioned within this footprint of the satellite require no tracking capabilities.

3.1 Characteristics of GEO Satellite Links

GEO satellite links have characteristics that differ from terrestrial links [12]. These characteristics contribute to the degradation of TCP performance in satellite networks. GEO satellite links have long one-way propagation delays (~250 ms) due to high satellite altitudes. The RTT of a GEO satellite link is at least 500 ms and it depends on the satellite inclination. 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 and the RTT. GEO satellite links have a large BDP due to their long propagation delays and large bandwidth. Satellite links have different downlink and uplink channel capacities and, hence, exhibit bandwidth asymmetry [7]. Losses occur in GEO satellite networks due to high BERs ($\sim 10^{-6}$) [12] in satellite links.

3.2 Impact of GEO Satellite Link Characteristics on TCP Performance

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. Large BDP values require the presence of large amount of unacknowledged data in flight for TCP to maximally utilize the available network capacity. For a maximum *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. Bandwidth asymmetry in satellite networks results in traffic burstiness [13].

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. TCP throughput is also degraded when the delay or loss of an ACK on a downlink path having low bandwidth is misinterpreted as an indication of congestion. Hence, the data transmission rate is reduced accordingly.

3.3 Related Work

Various solutions that have been proposed for improving TCP performance in GEO satellite networks are classified as end-to-end, split-connection, and link-layer.

End-to-end solutions usually require modifications only at the TCP sender and/or receiver. They may also require that intermediate routers in the network support priority mechanisms. End-to-end solutions maintain the end-to-end semantics of TCP. The SACK [14] option allows a receiver to only indicate segments that were received. Hence, the sender may explicitly retransmit only the lost segments. TCP-Peach [15] requires priority mechanisms that will enable every intermediate router in the network path to discard low priority segments in the case of congestion. TCP Westwood [16] and TCP-Star [17] utilize bandwidth estimation mechanisms for adjusting the *cwnd* size. TCP Hybla [18] employs a time-scale modification algorithm to increment *cwnd* independent of RTTs during slow start and congestion avoidance. The algorithm assumes that transmission rate does not depend on the *rwnd*. TCP New Vegas [19] implements packet pairing to reduce the negative impact of delayed ACK on networks with large RTT such as GEO satellite networks.

In heterogeneous networks, TCP connections are split at intermediate nodes such as gateways. Split connections shield the satellite link characteristics from the terrestrial segment. In the satellite segment, satellite-optimized transport protocols are

utilized. Examples of intermediate nodes with split TCP connections are performance enhancing proxies (PEPs) [20] such as PEPsal [21]. However, split connections violate the end-to-end semantics of TCP.

Link layer solutions are classified as TCP-aware and TCP-unaware [22]. TCP-aware link layer solutions modify TCP header information and are incompatible with applications that require IP security. Snoop protocol [23] is an example of a TCP-aware link layer protocol applicable to GEO satellite networks. TCP-unaware solutions employ forward error correction (FEC) and automatic repeat request techniques.

4. TCP WITH ADAPTIVE DELAY AND LOSS RESPONSE ALGORITHM

We propose TCP with adaptive delay and loss response (TCP-ADaLR) algorithm for use in heterogeneous networks employing GEO satellite links. 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 used with TCP NewReno. The algorithm requires modifications only at the sender. The TCP-ADaLR algorithm modifications comprise of 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*). The TCP sender requires no a priori knowledge of satellite links present in the network. Hence, the proposed modifications may be applicable to any network with large RTT. We normalize the *sampleRTT* by 1 s (a common value of the minimum RTO in TCP implementations with a coarse grained timer [24]). The scaling component is computed as:

$$\rho = (sampleRTT \text{ s}/1 \text{ s}) \times 60. \quad (1)$$

The value of 60 is the minimum recommended value for the maximum RTO [9] normalized by 1 s. The lower bound of ρ is set to 1 to ensure that TCP employs the default algorithm for connections with extremely short RTTs. An upper bound of ρ is set to 60 to ensure that the value of ρ is not too large.

TCP-ADaLR improves TCP performance because it takes into account the effect of RTT on the *cwnd* by computing the scaling component and adaptively changing the *cwnd* based on measured RTTs. This allows faster transmission of additional segments.

4.1 Adaptive *CWND* Increase Mechanism

We divide the slow start phase into four sub-phases based on current *cwnd* size 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* (64 KB) and the largest value of the initial *cwnd* (16 KB) employed by TCP implementations. In each sub-phase, the increment in *cwnd* depends on the value of ρ and the presence or absence of losses during transmission. The breakpoint value (ρ = 15) corresponds to a *sampleRTT* of 250 ms. It was selected based on observing values of RTTs in simulations of 50 MB FTP file downloads in an ideal channel with TCP SACK, as shown in Table 1. If *sampleRTT* $\geq$ 250 ms, the simulated FTP download response time increases significantly. Hence, when $\rho \geq 15$ the *cwnd* is incremented using the adaptive increase mechanism. Its sub-phases are described in Algorithm 1.

Table 1. FTP download response time for 50 MB file.

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

snd_max = maximum send sequence number (newest unacknowledged sequence number)
snd_una = sequence number of first unacknowledged segment (oldest unacknowledged sequence number)
snd_recover = sequence number denoting end of fast recovery (initialized to 0 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 ≤ ssthresh/4) && (flightsize < rwnd/4))
        set sub-phase = slow start sub-phase 1
    if ((cwnd > ssthresh/4) && (cwnd ≤ ssthresh/2)
    && (flightsize < rwnd/4))
        set sub-phase = slow start sub-phase 2
    if ((cwnd > ssthresh/4) && (flightsize ≥ rwnd/4)
    && (flightsize < rwnd/2))
        set sub-phase = slow start sub-phase 3
    if ((cwnd > ssthresh/2) && (flightsize ≥ rwnd/4)
    && (flightsize < rwnd/2))
        set sub-phase = slow start sub-phase 4
  }
```

Algorithm 1. Pseudo-code that describes the four sub-phases introduced by the adaptive *cwnd* increase mechanism.

The *cwnd* is incremented exponentially for $\rho < 15$, as in the default TCP slow start phase. For $\rho \geq 15$, we increase the *cwnd* by $(\sqrt{\rho}/4) \times$ SMSS when no losses have occurred. A delayed ACK of two outstanding segments will cause the transmission of four back-to-back segments that may result in micro-burstiness [25]. The value $(\sqrt{\rho}/4)$ is selected to accommodate the nonlinear increase of the FTP download response times (shown in Table 1) and micro-burstiness that may occur with delayed ACK enabled. Note that $(\sqrt{\rho}/4) \times$ SMSS lies in the range $(1 - 2) \times$ SMSS. A value of up to $2 \times$ SMSS is recommended to accommodate the delayed ACK option enabled [25]. If losses occur or if the conditions of the four sub-phases do not hold, the *cwnd* is increased as in the conventional TCP slow start phase. During the congestion avoidance phase, the *cwnd* is linearly incremented 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 $(\sqrt{\rho}/2) \times$ SMSS.

4.2 Adaptive *RWND* Increase Mechanism

The minimum of the *cwnd* or the *rwnd* determines the amount of data to be transmitted. The proposed algorithm adjusts the *rwnd* advertised by the TCP receiver to the sender using an adaptive *rwnd* increase mechanism that compensates for the 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. The modification to the *rwnd* is shown in Algorithm 2.

```
rtt_dev_gain = RTT deviation gain
if (ρ ≥ 15)
{
    if (flightsize > rwnd)
        do nothing
    // slow start phase or congestion avoidance phase
    if ((cwnd < ssthresh) || (cwnd > ssthresh))
        {
        // no losses have occurred
        if (snd_recover = = 0)
            set rwnd to rwnd + rtt_dev_gain × ρ × SMSS
        //losses have occurred and in fast recovery phase
        else if ((snd_una ≤ snd_recover) && (snd_recover
        != 0))
            set rwnd to rwnd + SMSS
        else
            do nothing
            }
}
```

Algorithm 2. Pseudo-code that describes the *rwnd* modification introduced by the adaptive *rwnd* increase mechanism.

4.3 Loss Recovery Mechanism

During the fast recovery phase, the minimum value of *cwnd* is set to 2 × SMSS rather than 1 × SMSS when the number of acknowledged bytes is greater than the current value of *cwnd*. This prevents the *cwnd* from being reduced to zero. By adjusting the value of *cwnd*, at least two segments may be transmitted back-to-back during the fast recovery phase to ensure that the TCP sender is able to receive ACKs in time when the delayed ACK option is enabled. If an RTO occurs, 200 ms is added to the current time to prevent premature expiration of the RTO timer, which may lead to false retransmissions when the delayed ACK option is enabled. The setting of the *cwnd* during the fast recovery phase is shown in Algorithm 3.

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

Algorithm 3. Pseudo-code that describes the loss recovery mechanism for setting *cwnd* during the fast recovery phase.

5. PERFORMANCE EVALUATION

We model the GEO satellite link as an additive white Gaussian noise (AWGN) channel with the satellite client being a fixed user that has a line of sight to the GEO satellite [18]. The packet error rate (PER) was calculated as:

$$PER = 1 - (1 - BER)^n, \quad (2)$$

where n is the number of bits in each packet. We use Ethernet packets of 1,500 bytes (n = 12,000 bits). For satellite links, typical average BER ranges from 10^{-5} to 10^{-8} [26]. The network topology used for performance evaluation of TCP-ADaLR is shown in Figure 1. The shown link propagation delays are one-way and remain unchanged during simulation, unless otherwise stated. The GEO satellite link between the client and the gateway is bi-directional with data rates 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 capacity models bandwidth asymmetry.

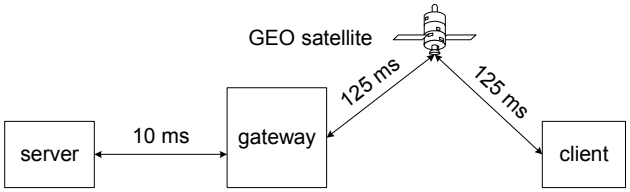

Figure 1. Heterogeneous network topology.

We evaluate the performance of TCP-ADaLR in various scenarios using the OPNET network simulator [27]. We consider an ideal case with no losses and cases with only congestion losses, with losses only due to satellite link errors, and, finally, with losses due to both congestion and satellite link errors. For the congestion scenario, we set gateway buffer size to 15 and 25 packets for HTTP and FTP applications, respectively. The simulation parameters for the HTTP [28] and FTP applications are shown in Tables 2 and 3. All TCP variants have constant parameters. We evaluate the performance of TCP-ADaLR without delayed ACK to investigate possible negative effects if the delayed ACK option is disabled. TCP parameters used for simulation are shown in Table 4. Identical set of parameters are used when the delayed ACK option is enabled, with the exception of the maximum ACK delay and the maximum number of ACK segments that are set to recommended values of 0.2 s and 2 [9], respectively. In this study, we have not compared performance of TCP-ADaLR with other TCP variants (TCP Westwood and TCP Jersey) and those specifically designed for satellite networks (TCP-Peach, TCP Hybla, TCP NewVegas, and TCP-Star). These protocols, unlike TPC NewReno, TCP Reno, and TCP SACK, are not currently available in the OPNET network simulator.

Table 2. HTTP application parameters.

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

Table 3. FTP application parameters.

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

Table 4. TCP simulation parameters when the delayed ACK option is disabled. Values in parentheses indicate when the delayed ACK option is enabled.

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
Maximum ACK delay	0.0 s (0.2 s)
Maximum number of ACK segments	1 (2)
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

5.1 Ideal Lossless Satellite Channel

The HTTP page response time for a single webpage is shown in Table 5. The main page object and the 15 embedded objects are completely downloaded and the entire web page is open before the HTTP page response statistic is collected. TCP-ADaLR exhibits better performance compared to TCP SACK and TCP NewReno. TCP-ADaLR exhibits the shortest HTTP page response time that is 10% and 9% smaller with delayed ACK enabled and disabled, respectively. Hence, TCP-ADaLR exhibits reduced user-perceived latency of short-lived flows such as in the case of HTTP applications. The download response time for the FTP application is shown in Table 6. TCP-ADaLR exhibits 23% and 28% shorter FTP download response time with delayed ACK enabled and disabled, respectively. The TCP throughput of TCP-ADaLR for the ideal channel is shown in Figure 2. TCP-ADaLR shows up to 53% and 63% higher TCP throughput with delayed ACK enabled and disabled, respectively. The higher TCP-ADaLR throughput and goodput imply the higher satellite downlink utilization, as shown in Figure 3. TCP-ADaLR exhibits up to 28% and 34% higher satellite link utilization with delayed ACK enabled and disabled, respectively. TCP-ADaLR does not have negative impact on connections with delayed ACK disabled. TCP-ADaLR outperforms TCP SACK and TCP NewReno because the adaptive *cwnd* and *rwnd* window increase mechanisms enable faster transmission of additional segments when there are no losses.

Table 5. HTTP page response time for the four TCP variants in the scenario with an ideal lossless satellite channel.

	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

Table 6. FTP download response time the four TCP variants in the scenario with an ideal lossless satellite channel.

	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

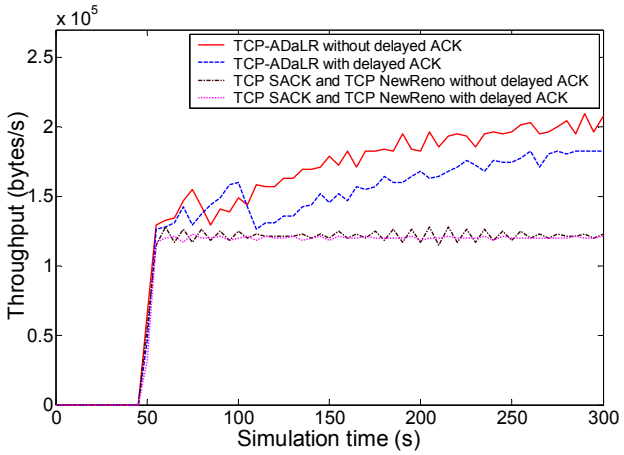

Figure 2. Scenario with an ideal lossless satellite channel. TCP-ADaLR shows up to 63% higher throughput than TCP SACK and TCP NewReno with delayed ACK disabled.

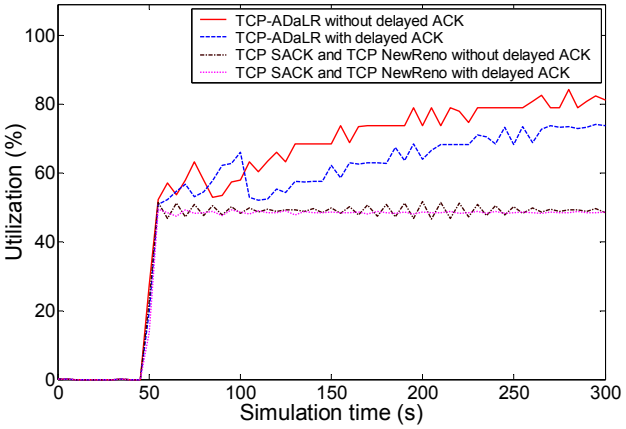

Figure 3. Scenario with an ideal lossless satellite channel. With delayed ACK disabled, TCP-ADaLR reaches 80% utilization of the satellite link capacity while TCP SACK and TCP NewReno attain only 50% satellite link utilization.

5.2 Ideal Satellite Channel with only Congestion Losses

We consider two variants of the proposed TCP-ADaLR algorithm: with SACK and with NewReno. The increased HTTP page response time indicates the impact of congestion losses (compared to the ideal case), as shown in Table 7. TCP-ADaLR SACK shows the best performance in both cases with delayed ACK enabled and disabled. The FTP download response time is

comparable for all four TCP variants, as shown in Table 8. The TCP throughput, TCP goodput, and satellite link utilization for the case with delayed ACK enabled are shown in Figures 4–6. The received segment sequence number at the receiver indicates the goodput. TCP-ADaLR performs 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 degrade performance during congestion. In the case with delayed ACK disabled, the four TCP variants exhibit similar patterns as those with delayed ACK enabled [29]. Hence, TCP-ADaLR variants show no significant performance degradation in the presence of congestion.

Table 6. HTTP page response time for the four TCP variants in the scenario with only congestion losses.

	Download 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

Table 7. FTP download response time for the four TCP variants in the scenario with only congestion losses.

	Download response time (s)	
Delayed ACK option	Enabled	Disabled
TCP-ADaLR SACK	1,212.7	1,226.7
TCP-ADaLR NewReno	1,228.0	1,232.4
TCP SACK	1,224.8	1,226.7
TCP NewReno	1,216.6	1,226.7

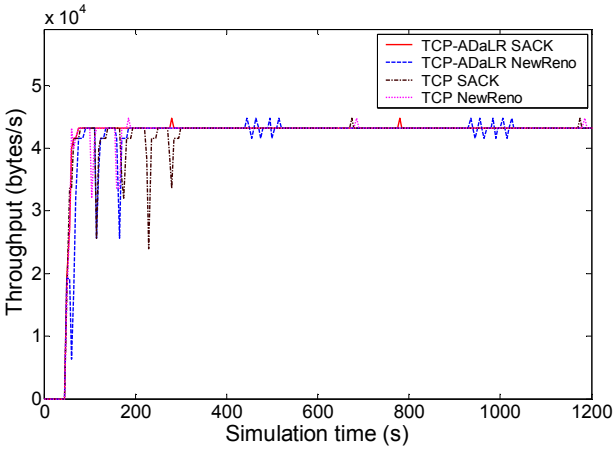

Figure 4. Scenario with only congestion losses and delayed ACK enabled. TCP throughput degrades for the four TCP variants when congestion losses are detected.

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 each BER value, we use different random seed numbers and compute the average values using 95% confidence intervals. The average HTTP page response times for the case with delayed ACK enabled are shown in Figure 7. Both TCP-ADaLR variants exhibit similar HTTP page response times and outperform both TCP SACK and TCP NewReno. The average FTP download response time is shown in Figure 8. For the four TCP variants, the FTP download response time increases with higher BER values. TCP-ADaLR SACK is the most robust variant in the presence of losses and shows 13%–37% shorter FTP download response time than TCP SACK. The TCP goodput, TCP throughput, and satellite link utilization for the case with delayed ACK enabled are shown in Figures 9–11. TCP-ADaLR SACK shows 16%–61% higher TCP throughput than TCP SACK. The case with delayed ACK disabled exhibits the same pattern as with delayed ACK enabled [29]. TCP-ADaLR variants outperform TCP SACK and TCP NewReno because the adaptive *cwnd* increase mechanism causes additional segments to be rapidly sent after losses have occurred. Furthermore, 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 ACKs. Hence, these mechanisms enable TCP-ADaLR to recover more quickly than TCP SACK and TCP NewReno.

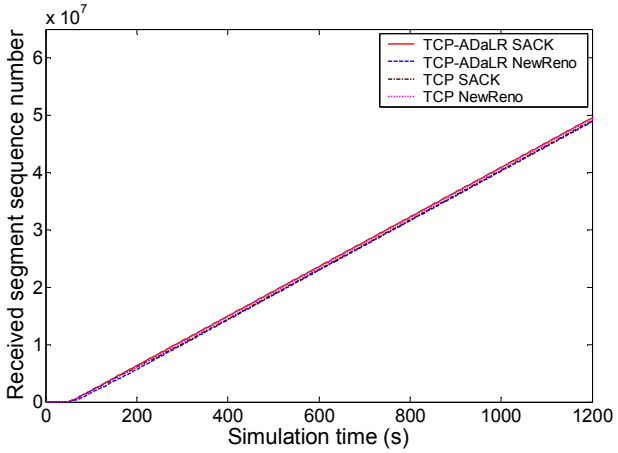

Figure 5. Scenario with only congestion losses and delayed ACK enabled. TCP-ADaLR variants exhibit TCP goodput comparable to TCP SACK and TCP NewReno.

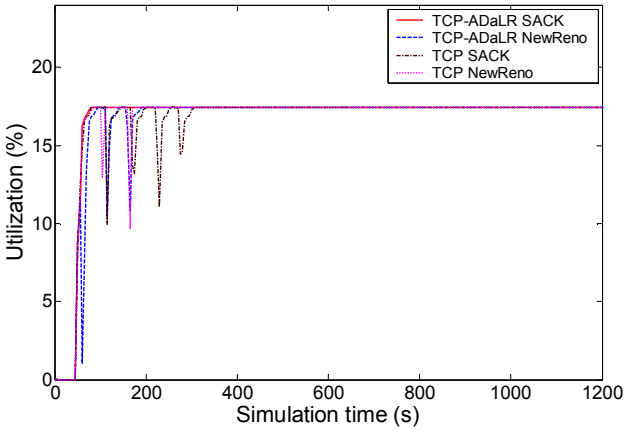

Figure 6. Scenario with only congestion losses and delayed ACK enabled. Satellite link utilization decreases when congestion losses are detected.

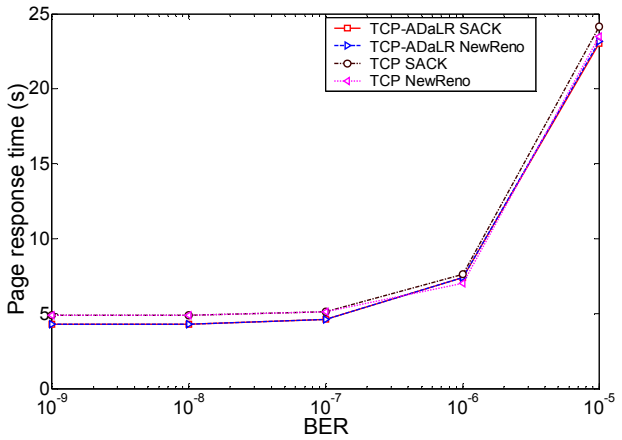

Figure 7. Scenario with only satellite link error losses and delayed ACK enabled. TCP-ADaLR SACK and TCP-ADaLR NewReno exhibit 2%–12% shorter HTTP page response time than TCP SACK and TCP NewReno.

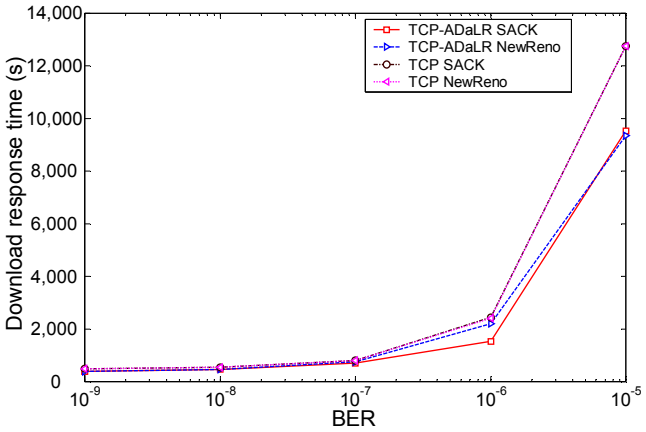

Figure 8. Scenario with only satellite link error losses and delayed ACK enabled. TCP-ADaLR SACK exhibits up to 37% shorter FTP download response time than TCP SACK.

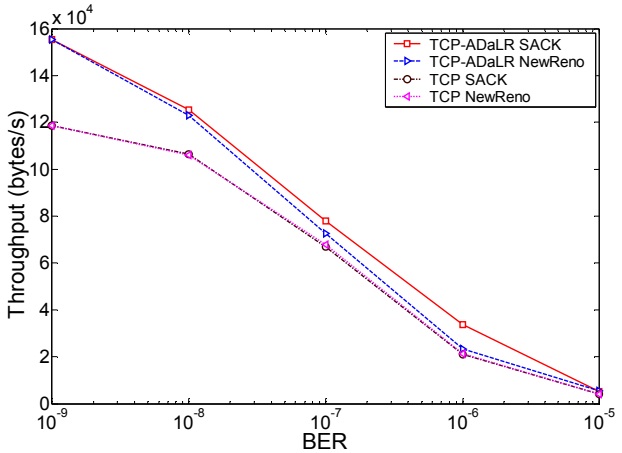

Figure 9. Scenario with only satellite link error losses and delayed ACK enabled. TCP-ADaLR variants show higher TCP throughput than TCP SACK and TCP NewReno.

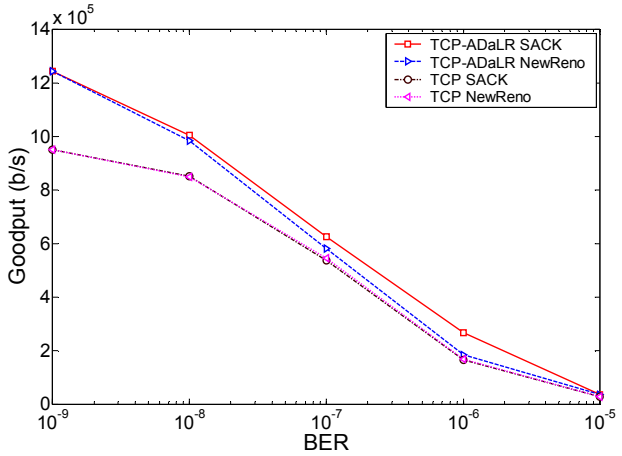

Figure 10. Scenario with only satellite link error losses and delayed ACK enabled. TCP-ADaLR SACK exhibits up to 27% higher TCP goodput than TCP SACK.

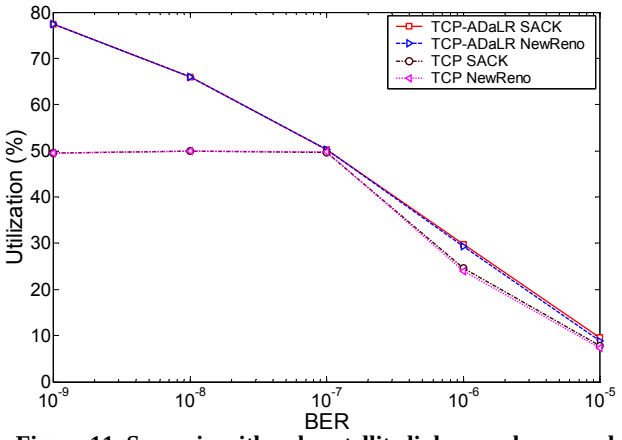

Figure 11. Scenario with only satellite link error losses and delayed ACK enabled. TCP-ADaLR SACK and TCP-ADaLR NewReno exhibit up to 76% higher satellite link utilization than TCP SACK and TCP NewReno.

5.4 Satellite Channel with both Congestion and Error Losses

The increased HTTP page response time reflects the effect of both congestion and error losses, as shown in Figure 12. TCP-ADaLR SACK shows the best performance in both cases with delayed ACK enabled and disabled. TCP-ADaLR SACK exhibits up to 32% shorter HTTP page response time than TCP SACK with delayed ACK enabled. The short and bursty nature of HTTP transfers ensures small number of outstanding unacknowledged bytes. Hence, when losses occur, the adaptive *cwnd* increase and loss recovery mechanisms enable faster completion of the HTTP transfers than with conventional TCP SACK and TCP NewReno. However, TCP-ADaLR NewReno performs worse than TCP NewReno with delayed ACK enabled. The higher HTTP page response time exhibited by TCP-ADaLR NewReno may be caused by its initial high transmission rate and, hence, the loss of several original and retransmitted segments. This is the only simulation scenario where TCP-ADaLR NewReno performs worse than TCP NewReno.

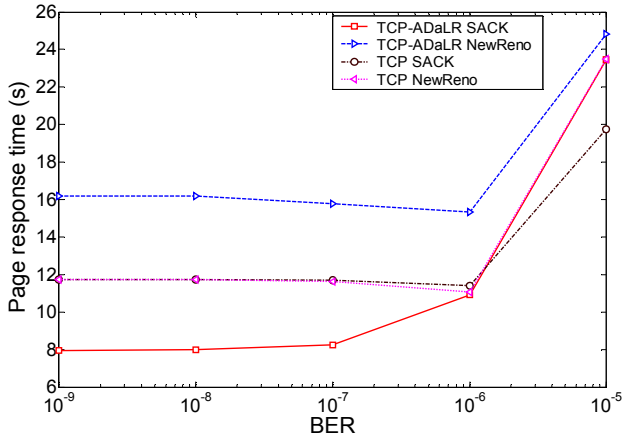

Figure 12. HTTP page response time for the four TCP variants in the scenario with congestion losses, satellite link error losses, and delayed ACK enabled.

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 BER values are 10^{-7} and lower, as shown in Figure 13. At these lower BER values, the more prevalent cause of losses is congestion. Hence, TCP SACK and TCP NewReno exhibit 1%–4% shorter FTP download response times than the TCP-ADaLR variants. For BER $\geq 10^{-6}$ in the case with delayed ACK enabled, TCP-ADaLR SACK exhibits 28%–29% shorter FTP download response time than TCP SACK. The case with delayed ACK enabled exhibits similar performance [29]. At higher BER values, the link error is the more prevalent cause of losses. Hence, 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 TCP SACK and TCP NewReno. The TCP goodput, TCP throughput, and satellite link utilization exhibit similar performance to the FTP download response time, as shown in Figures 14–16. TCP-ADaLR variants outperform TCP SACK and TCP NewReno when the BER value of the GEO satellite link exceeds 10^{-7}.

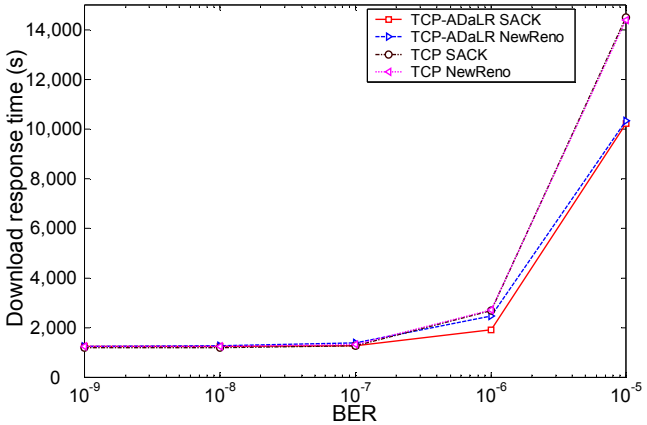

Figure 13. FTP download response time for the four TCP variants in the scenario with congestion losses, satellite link error losses, and delayed ACK enabled.

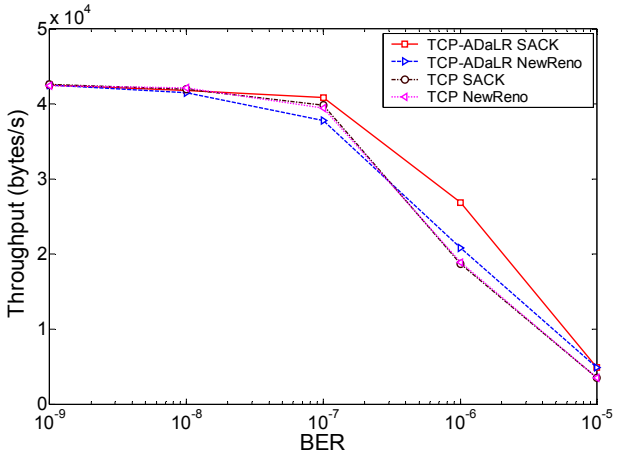

Figure 14. Scenario congestion losses, satellite link error losses, and delayed ACK enabled. For BER values higher than 10^{-7}, TCP-ADaLR SACK exhibits 42%–43% higher TCP throughput than TCP SACK with delayed ACK enabled.

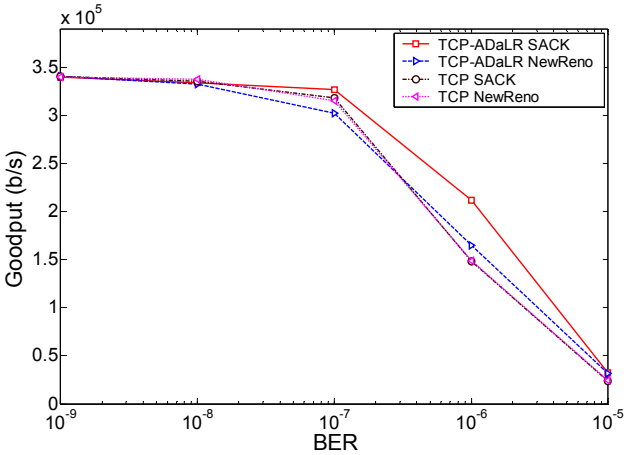

Figure 15. Scenario congestion losses, satellite link error losses, and delayed ACK enabled. For BER values higher than 10^{-7}, TCP-ADaLR SACK exhibits 36%–43% higher TCP goodput than TCP SACK with delayed ACK enabled.

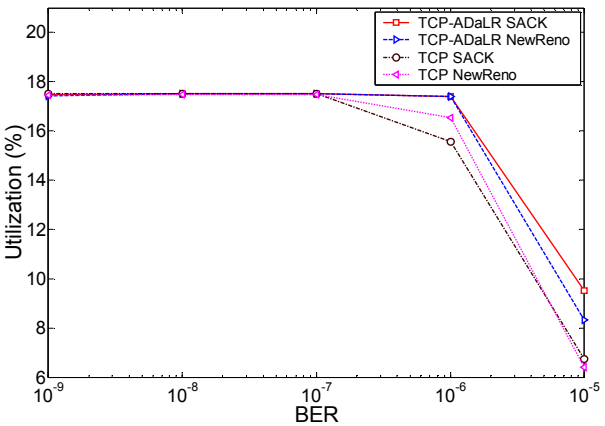

Figure 16. Satellite link utilization for the four TCP variants in the scenario with congestion losses, satellite link error losses, and delayed ACK enabled.

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 [30] defined as:

$$Fairness = \frac{(\sum_{j=1}^{n} t_j)^2}{n \times (\sum_{j=1}^{n} t_j^2)}, \quad (3)$$

where n is the number of competing connections and t_j is the average throughput of the jth connection. The fairness 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. Common TCP variants such as TCP SACK and TCP NewReno are known to be fair when the competing connections have similar RTTs [18]. However, if the competing connections have different RTTs, the connections with shorter RTTs consume a larger fraction of the available bottleneck link 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 in the absence of losses for an FTP application. The network topology is shown in Figure 17. All links are bi-directional with 10 Mb/s data rates. Shown are one-way link propagation delays. We test six TCP connections with different RTTs using two fairness scenarios where TCP connections employ TCP-ADaLR or TCP NewReno. In the friendliness scenario, we test six connections with identical RTTs (25 ms). Three connections employ TCP-ADaLR and the remaining three connections employ TCP NewReno.

The average throughput values of the six TCP NewReno and six TCP-ADaLR connections are shown in Table 9. The 500 ms RTT connection using TCP-ADaLR 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 reduces by ~12%. The fairness values are shown in Table 10. TCP-ADaLR reduces the penalty caused by long RTT connections and exhibits better fairness than TCP NewReno. In the friendliness scenario, the average throughput of each competing connection is shown in Table 11. The friendliness value, shown in Table 12, confirms that TCP-ADaLR is TCP-friendly. Hence, the six coexisting TCP-ADaLR and TCP NewReno connections have a fair share of the bottleneck link's available bandwidth.

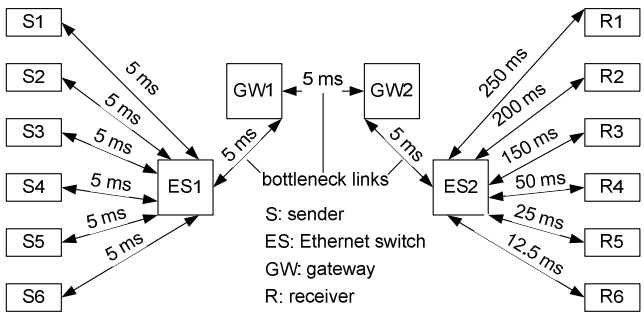

Figure 17. Network topology used to evaluate fairness and friendliness.

Table 9. Average throughput achieved by six competing TCP-ADaLR and TCP NewReno connections.

RTT (ms)	Average throughput (bytes/s)	
	TCP ADaLR	TCP NewReno
25	283,404.6	322,418.0
50	281,750.6	300,629.1
100	268,984.6	263,129.2
300	195,099.8	158,601.5
400	175,343.8	129,560.8
500	160,897.4	109,239.5

Table 10. TCP fairness values of TCP-ADaLR and TCP NewReno using the Jain's fairness index.

TCP variant	Fairness
TCP-ADaLR	0.9510
TCP NewReno	0.8650

Table 11. Average throughput achieved by six competing connections using distinct TCP variants. RTT is set to 25 ms.

TCP variant	Average throughput (bytes/s)
TCP-ADaLR	354,451.6
TCP-ADaLR	356,565.9
TCP-ADaLR	356,906.8
TCP NewReno	352,012.7
TCP NewReno	351,913.3
TCP NewReno	351,748.1

Table 12. TCP friendliness of TCP-ADaLR and TCP NewReno competing connections.

TCP variant mix	Friendliness
TCP-ADaLR and TCP NewReno	0.99996

6. CONCLUSIONS

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. We considered cases with both delayed ACK enabled and 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. 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 both delayed ACK enabled and disabled. The TCP-ADaLR algorithm reduced the response times for downloading HTTP web pages and FTP files. In the presence of only congestion losses, TCP-ADaLR variants 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. In each 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.

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 routers. 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 is deployable in networks with other TCP variants. Finally, TCP-ADaLR maintains the end-to-end semantics of TCP.

7. ACKNOWLEDGMENTS

The authors would like to thank S. Lau, R. Narayanan, B. Vujičić, S. Vujičić, and W. Zeng from the Communication Networks Laboratory at SFU for constructive suggestions and comments.

8. REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

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

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

[7] 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.

[8] 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.

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

[10] 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.

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

[12] 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.

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

[23] 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.

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

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

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

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

[28] 3GPP/TSG-C.R1002, "1xEV-DV evaluation methodology (v14)," June 2003.

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

[30] 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.