OPNET Model of TCP with Adaptive Delay and Loss Response for Broadband GEO Satellite Networks

Modupe Omueti and Ljiljana Trajković

Simon Fraser University
Vancouver, BC
Canada
E-mail: {momueti, ljilja}@cs.sfu.ca

Abstract

Transmission control protocol (TCP) provides reliable transport services for Internet applications. Broadband geostationary earth orbit (GEO) satellite networks play an important role in providing Internet access and network connectivity. They are characterized by long propagation delays and high bit error rates, which negatively affect TCP performance. We proposed a modification of TCP named TCP with adaptive delay and loss response algorithm (TCP-ADaLR) that improves TCP performance in broadband GEO satellite networks.

In this paper, we describe the TCP-ADaLR algorithm and its OPNET implementation. We evaluate and compare the performance of TCP-ADaLR, TCP SACK, and TCP NewReno, with and without delayed acknowledgment. To verify the algorithm implementation, we first consider simulation scenarios with an ideal lossless satellite link. We then consider simulation scenarios with congestion losses and show that TCP-ADaLR exhibits comparable performance to TCP SACK and TCP NewReno. In the presence of losses due to satellite link errors, TCP-ADaLR outperforms TCP SACK and TCP NewReno. In the presence of both congestion and error losses, TCP-ADaLR improves TCP goodput and throughput. In all simulation scenarios, TCP-ADaLR outperforms TCP SACK and TCP NewReno in terms of satellite link throughput and the user-perceived latency of HTTP web and FTP file download applications. TCP-ADaLR is fair to competing connections, friendly to TCP NewReno, and maintains the end-to-end semantics of TCP.

1. Introduction

Transmission control protocol (TCP) provides byte-stream transport services to Internet applications. It carries up to 90% of Internet traffic [1], [2]. The Internet is evolving into a seamless ubiquitous system with increasing demand for high-speed and bandwidth-intensive multimedia and data applications [3]. Broadband geostationary earth orbit (GEO) satellite networks are important to the Internet because they provide access and network connectivity through high-bandwidth GEO satellite links. These satellite networks have easily scalable architecture, multicast capabilities, and large coverage areas [4]. GEO satellite networks provide continuous coverage by using a single satellite. However, their non-GEO (NGEO) counterparts require a large number of satellite constellations that have higher development risks [5]. Hence, GEO satellite networks are more attractive than NGEO satellite networks.

TCP was originally designed for wired networks characterized by negligible random bit error rates (BERs). Hence, packet losses indicate congestion. TCP reacts to packet losses by reducing its sending rate with the congestion control algorithms

[6], [7]. If losses are not due to congestion, reducing the sending rate degrades TCP performance. Losses occur in GEO satellite networks because of high BERs [8] and, hence, degrade TCP performance. GEO satellite networks are also characterized by long propagation delays [9] that lead to large round trip times (RTTs) and poor satellite bandwidth utilization.

We have proposed TCP modifications collectively named TCP with adaptive delay and loss response algorithm (TCP-ADaLR) and have shown that TCP-ADaLR improves TCP performance in broadband GEO satellite networks [10]. We implemented TCP-ADaLR algorithm as an extension to TCP SACK [11]. These modifications are also applicable to TCP NewReno [12]. More than 50% of Internet servers employ TCP SACK while majority of others use TCP NewReno [13]. The TCP-ADaLR algorithm requires only sender-side modifications and, thus, maintains the end-to-end semantics of TCP. It also incorporates delayed acknowledgment (ACK) recommended for Internet hosts [14].

This paper is organized as follows. We present the motivation and description of the TCP-ADaLR algorithm in Section 2. In Section 3, we describe the OPNET implementation of TCP-ADaLR. Simulation results that verify the algorithm implementation and compare the performance of TCP-ADaLR, TCP SACK, and TCP NewReno are presented in Section 4. We conclude with Section 5.

2. TCP with Adaptive Delay and Loss Response Algorithm (TCP-ADaLR)

TCP is a connection-oriented transport protocol. Its transmission rate increases when the sender receives ACKs of transmitted segments. During the slow start phase, the long propagation delay of a GEO satellite link and the limited TCP window size prevent a TCP sender from achieving maximum throughput. For a receiver advertised window *rwnd* of 64KB, a GEO satellite link (RTT = 500 ms) with data rate of 2,048 kb/s may achieve only ~1,048 kb/s (~50%). Continuous decrease of the congestion window *cwnd*, caused by detection of random losses due to the high BERs of GEO satellite links, leads to TCP performance degradation. Performance of TCP in heterogeneous broadband networks employing GEO satellite links should be improved without violating the end-to-end semantics of TCP.

2.1 Overview of TCP-ADaLR

TCP-ADaLR is an end-to-end algorithm that introduces the following modifications at the sender: a scaling component ρ, adaptive window (*cwnd* and *rwnd*) increase, and loss recovery mechanisms [10]. The scaling component ρ is computed from the measured RTT taken from transmitted sample RTT segments. The adaptive window increase mechanisms enable a

TCP sender to better utilize the available GEO satellite link bandwidth in the absence of losses. The loss recovery mechanism compensates for delayed ACK and allows a TCP sender to recover faster from losses than TCP SACK.

2.2 Scaling Component

The smoothed RTT (*srtt*) and RTT variation *(rttvar)* are computed from measurements of transmitted sample RTT segments (*sampleRTT*). The retransmission timeout (RTO) is then computed from the *srtt* and *rttvar*. ρ is calculated as:

$$\rho = \left(\frac{sampleRTT \text{ s}}{1 \text{ s}}\right) \times 60, \qquad (1)$$

with selected minimum and maximum values set at 1 and 60 [10], respectively. The *cwnd* increments depend on the value of ρ.

2.3 Adaptive *cwnd* Increase Mechanism

At the TCP sender, the *cwnd* limits the amount of transmitted data. The *cwnd* increments are exponential during the slow start phase and linear during the congestion avoidance phase. The adaptive *cwnd* mechanism was proposed to divide the slow start phase into four sub-phases based on the current *cwnd*, slow start threshold *ssthresh*, and *flightsize* (total unacknowledged byte in the network). The *cwnd* increment factor depends on the sub-phases. A breakpoint $\rho = 15$ was selected to define the transition to the adaptive *cwnd* mechanism [10].

2.4 Adaptive *rwnd* Increase Mechanism

At the TCP receiver, the *rwnd* limits the amount of transmitted data. The number of transmitted bytes is the minimum of either the *cwnd* or *rwnd*. The adaptive *rwnd* increase mechanism is used to increment the *rwnd* when no losses have occurred [10]. It allows the transmission of additional segments thus improving bandwidth utilization of a GEO satellite link.

2.5 Loss Recovery Mechanism

The loss recovery mechanism modifies the fast recovery phase to enable the back-to-back transmission of two segments [10]. The default maximum ACK delay of 200 ms [14] is added to the current time used in the computation of the subsequent RTO value. Both modifications compensate for additional delays in sending an ACK if two back-to-back segments are not received by the TCP receiver when the delayed ACK option is enabled.

3. OPNET Implementation of TCP-ADaLR Algorithm

We implemented TCP-ADaLR in the OPNET (version 11.0.A) simulation tool [15]. Modifications were made to the OPNET node and process models of the TCP sender represented by the OPNET *Ethernet server advanced* model shown in Fig. 1. The OPNET TCP process model implements standard TCP features [7], [12], [16]. It also includes additional features such as SACK [11], delayed ACK [14], explicit congestion notification (ECN) [17], and timestamp options [18], [19]. These features are defined in the TCP attributes of the OPNET *Ethernet server advanced* model.

We modified the TCP process model associated with the server node model. The OPNET TCP implementation consists of a parent process *tcp_manager_v3* and a child process *tcp_conn_v3*. The *tcp_manager_v3* finite state machine (FSM), shown in Fig. 2, communicates with the session and network

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. We modified the OPEN state of the *tcp_manager_v3* FSM to invoke the modified *tcp_conn_v3* child process.
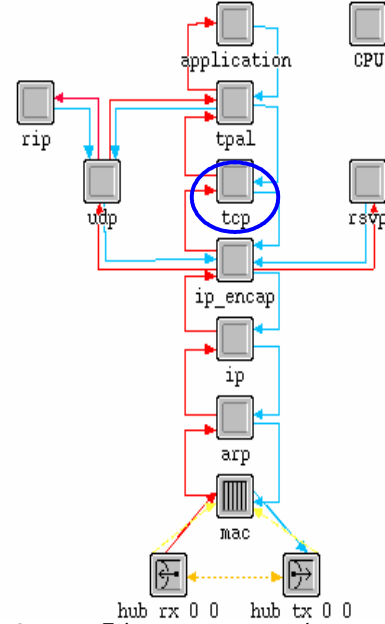

Fig. 1. The OPNET *Ethernet server advanced* node model. The implementation of TCP-ADaLR requires modifications to the two process models of the TCP module.
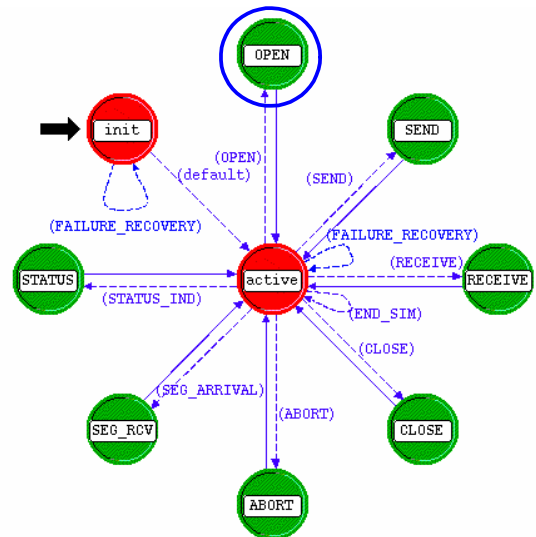

Fig. 2. The *tcp_manager_v3* process model. The OPEN state was modified to invoke the modified *tcp_conn_v3* child process when a new TCP connection is opened.

Additional modifications were made to functions defined in the *tcp_conn_v3* FSM shown in Fig. 3. We defined the scaling component ρ 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* using the *tcp_rtt_measurements_update* () function. This function was

2

modified in order to compute the scaling component ρ, which depends on the value of *sampleRTT* (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 and the loss recovery mechanism used to set the *cwnd* during fast recovery. The modifications are shown in Algorithms 1 and 2.
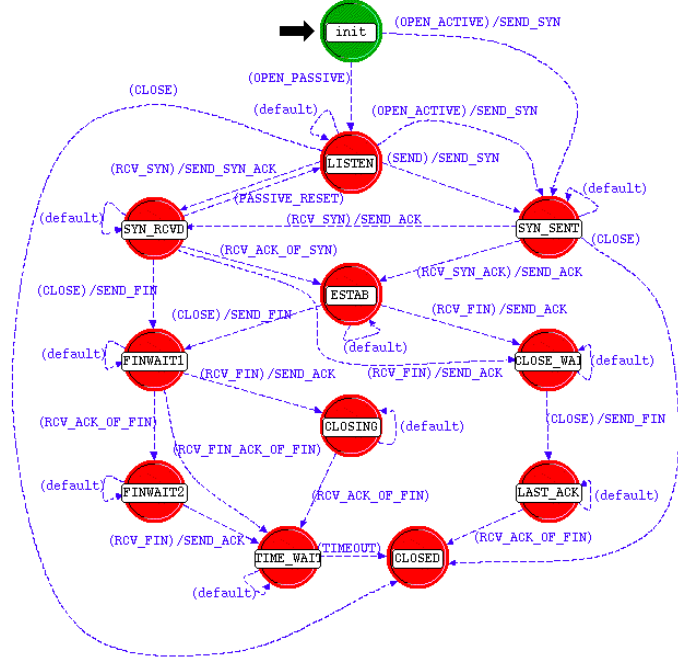

Fig. 3. The *tcp_conn_v3* **process model. TCP-ADaLR algorithm implementation of requires modifications of the function block of this process model.**

// *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 a 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)) // sub-phase 1
   {
    if ((*snd_recover* == 0) && ($\rho \geq 15$))
        increment *cwnd* by $\sqrt{\rho}/4 \times$ SMSS
    else
        increment *cwnd* exponentially as in TCP Reno
   }
 if ((*cwnd > ssthresh*/4) && (*cwnd ≤ ssthresh*/2) && (*flightsize < rwnd*/4)) // sub-phase 2
   {
    if ((*snd_recover* == 0) && ($\rho \geq 15$))
        increment *cwnd* by $\sqrt{\rho}/4 \times$ SMSS
    else
        increment *cwnd* exponentially as in TCP Reno
   }

if ((*cwnd > ssthresh*/4) && (*flightsize ≥ rwnd*/4) && (*flightsize < rwnd*/2)) // sub-phase 3
   {
    if ((*snd_recover* == 0) && ($\rho \geq 15$))
        increment cwnd by $\sqrt{\rho}/4 \times$ SMSS
    else
        increment *cwnd* exponentially as in TCP Reno
   }
 if ((*cwnd > ssthresh*/2) && (*flightsize ≥ rwnd*/4) && (*flightsize < rwnd*/2)) // sub-phase 4
   {
    if ((*snd_recover* == 0) && ($\rho \geq 15$))
        increment *cwnd* by $\sqrt{\rho}/4 \times$ SMSS
    else
        increment *cwnd* exponentially as in TCP Reno
   }
 }
// congestion avoidance phase
if (*cwnd > ssthresh*)
   {
   // entering fast recovery phase
   if (((*snd_una ≤ snd_recover*) && ($\rho \geq 15$) && (*snd_recover* != 0)) || (*flightsize ≥ rwnd*/2))
        increment *cwnd* by SMSS × SMSS / *cwnd*
   // exiting fast recovery phase
   else if (((*snd_una ≥ snd_recover*) && ($\rho \geq 15$) && (*snd_recover* != 0)) || (*flightsize < rwnd*/2))
        increment *cwnd* by $\sqrt{\rho}/2 \times$ SMSS × SMSS / *cwnd*
   else
        increment *cwnd* by SMSS × SMSS / *cwnd*
   }

Algorithm 1. Pseudo-code of the adaptive *cwnd* **increase mechanism.**

// 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 2. Pseudo-code of loss recovery mechanism for setting *cwnd* **during fast recovery.**

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. 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 [13] by limiting the number of retransmissions to three segments when the function is called.

3

4. Simulation Scenarios and Results

4.1 Network Model

The GEO satellite links are modeled as additive white Gaussian noise (AWGN) channels with the fixed receiver having a line-of-sight (LoS) to the GEO satellite. Hence, for the GEO satellite links we employ the OPNET unidirectional *point-to-point (PPP) link advanced* model. The downlink (satellite to receiver) and uplink (receiver to satellite) data rates are 2,048 kb/s and 256 kb/s, respectively. This difference in capacity captures bandwidth asymmetry, a characteristic of satellite links [20]. The GEO satellite links one-way propagation delays are set to 250 ms. For the link between the satellite gateway and the sender, we employ the OPNET full duplex 10 Mb/s *Ethernet link advanced* model. The one-way propagation delay of the Ethernet is set to 10 ms. We model the satellite gateway with the OPNET *Ethernet4 slip8 gateway* model. The fixed TCP receiver is modeled with the OPNET *PPP workstation advanced* model shown in Fig. 4. The OPNET network model is shown in Fig. 5.
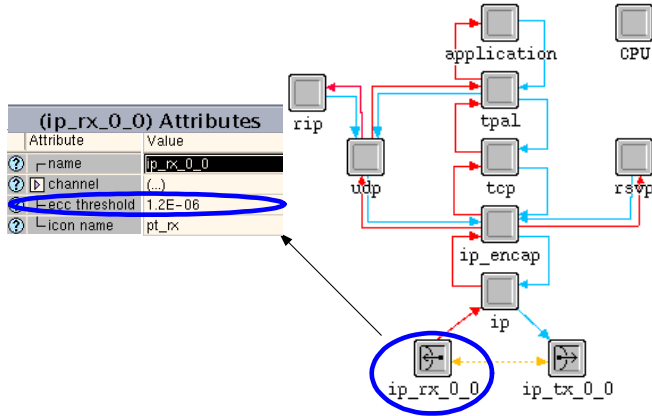

Fig. 4. The OPNET *PPP workstation advanced* **node model. We modified the transmission receiver (ip_rx_0_0) to set the error correction threshold for accepted packets.**
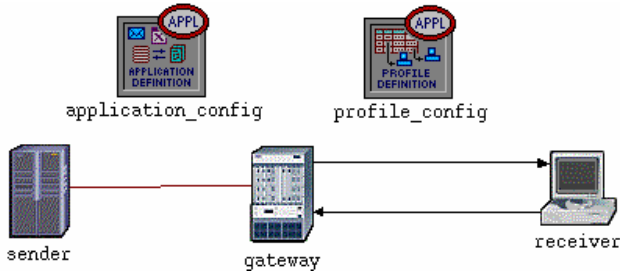

Fig. 5. OPNET network model of a heterogeneous network employing asymmetric GEO satellite links.

4.2 Simulation Scenarios

We evaluate and compare the performance of TCP-ADaLR, TCP SACK, and TCP NewReno by employing four simulation scenarios. We first verify the TCP-ADaLR algorithm implementation by simulating ideal lossless satellite links. After the verification, we then consider two TCP-ADaLR variants: with SACK and with NewReno. We introduce congestion losses at the satellite gateway by modifying the attributes of its Internet protocol (IP) module to set limited buffer size for the FTP application, as shown in Fig. 6. In the next scenario, we introduce random losses due to satellite link errors for various BERs. We modify the OPNET *PPP workstation advanced model* (TCP receiver) to define the error correction (ECC) threshold for accepted packets. Packets with errors exceeding the

ECC threshold are discarded (lost). We then consider losses due to both congestion and satellite link errors. For all scenarios, we simulate cases with and without delayed ACK. Selected TCP simulation parameters are shown in Table 1. (Values in parentheses indicate the case with delayed ACK.) We evaluate response times for HTTP and FTP applications with parameters shown in Tables 2 and 3, respectively. We also evaluate TCP goodput, TCP throughput, satellite link throughput, and TCP-ADaLR fairness and friendliness.
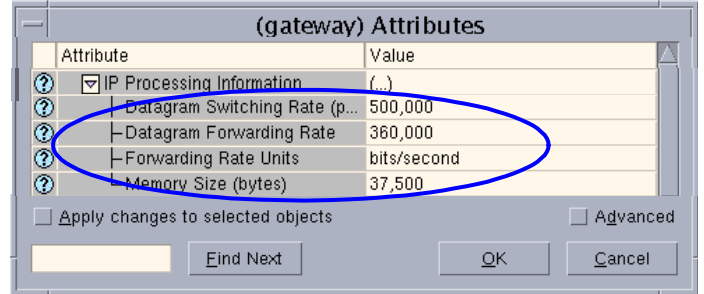

Fig. 6. The OPNET *Ethernet4 slip8 gateway* **model attributes.**

Table 1. TCP simulation parameters.

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 ACK segment	1 (2)
Duplicate ACK threshold	3

Table 2. Simulated HTTP webpage download parameters.

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

Table 3. Simulated FTP file download parameters.

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

4.2.1 Scenario with Ideal Lossless Satellite Link

We verify the OPNET implementation of TCP-ADaLR in an ideal lossless satellite link. The response times of the HTTP and FTP applications are shown in Tables 4 and 5, respectively. The TCP goodput and satellite link throughput are shown in Figs. 7 and 8, respectively. Received segment sequence number is used as an indicator of goodput. The adaptive window increase mechanisms enable TCP-ADaLR to transmit additional segments faster than TCP SACK and TCP NewReno. For cases with and without delayed ACK, TCP-ADaLR achieves ~75% and ~80% of the satellite link capacity, respectively. Hence, TCP-ADaLR does not have negative impact on connections without delayed ACK. However, for cases with and without delayed ACK, TCP SACK and TCP NewReno attain only ~50% of the satellite link capacity.

Table 4. HTTP page response time for the scenario with ideal lossless satellite link. For the case without delayed ACK, TCP-ADaLR shows ~9% shorter page response times than TCP SACK and TCP NewReno.

TCP variant	Page response time (s)	
	With delayed ACK	Without delayed ACK
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 5. FTP download response time for the scenario with ideal lossless satellite link. For the case without delayed ACK, TCP-ADaLR shows ~28% shorter FTP download response times than TCP SACK and TCP NewReno.

TCP variant	Download response time (s)	
	With delayed ACK	Without delayed ACK
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
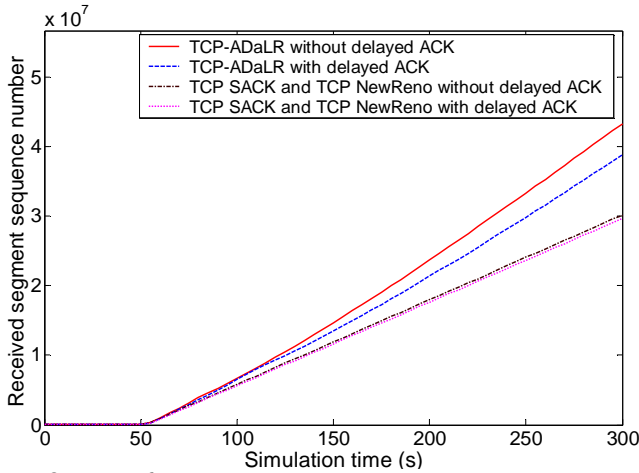
Fig. 7. Goodput for the scenario with ideal lossless satellite link. For the case without delayed ACK, TCP-ADaLR shows ~50% higher goodput than TCP SACK and TCP NewReno.
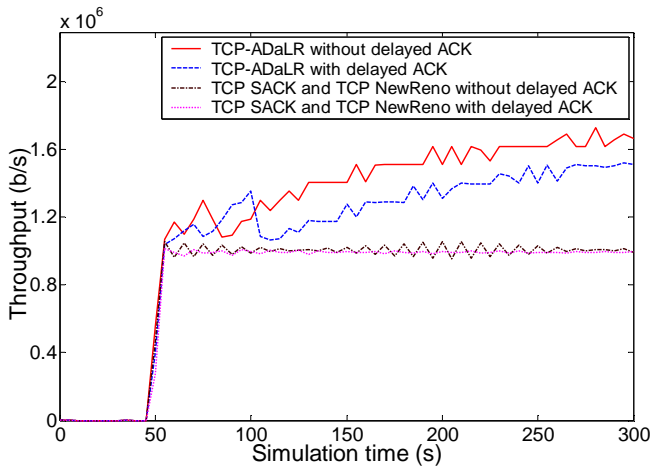

Fig. 8. Satellite link throughput for the scenario with ideal lossless satellite link. For the case without delayed ACK, TCP-ADaLR shows ~66% higher satellite link throughput than TCP SACK and TCP NewReno.

4.2.2 Scenario with only Congestion Losses

We introduce congestion losses by limiting the buffer size of the satellite gateway, as shown in Fig. 6. The HTTP and FTP response times are shown in Tables 6 and 7, respectively. For the simulated HTTP application, the TCP-ADaLR variants exhibit shorter page response times than TCP SACK and TCP NewReno and, hence, show performance gains. For the FTP file download application, TCP-ADaLR exhibits comparable download response times to TCP SACK and TCP NewReno. The four TCP variants in the case without delayed ACK exhibit ~1% shorter download response times than the four TCP variants in the case with delayed ACK. Hence, for both cases with and without delayed ACK the four TCP variants exhibit similar performance. The TCP and satellite link throughputs for the case without delayed ACK are shown in Figs. 9 and 10, respectively. The adaptive window increase mechanisms prevent large bursts that may cause degraded TCP performance during congestion. The performance degradation of the four TCP variants reflects the impact of congestion losses.

Table 6. HTTP page response time for the scenario with only congestion losses. For cases with and without delayed ACK, TCP-ADaLR SACK shows ~33% and ~12% shorter page response times than TCP SACK and TCP NewReno.

TCP variant	Page response time (s)	
	With delayed ACK	Without delayed ACK
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 scenario with only congestion losses. For cases with and without delayed ACK, TCP-ADaLR variants exhibit download response times comparable to TCP SACK and TCP NewReno.

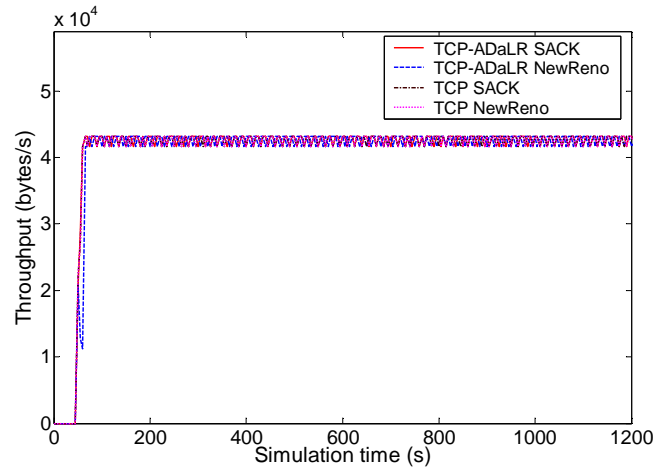| TCP variant	Page response time (s)	
	With delayed ACK	Without delayed ACK
TCP-ADaLR SACK	1,212.7	1,226.7
TCP-ADaLR NewReno	1,228.0	1,232.4
TCP SACK	1,224.8	1,226.7
TCP NewReno	1,216.6	1,226.7

Fig. 9. TCP throughput for the scenario with only congestion losses and delayed ACK disabled. The TCP throughput is comparable for the four TCP variants.
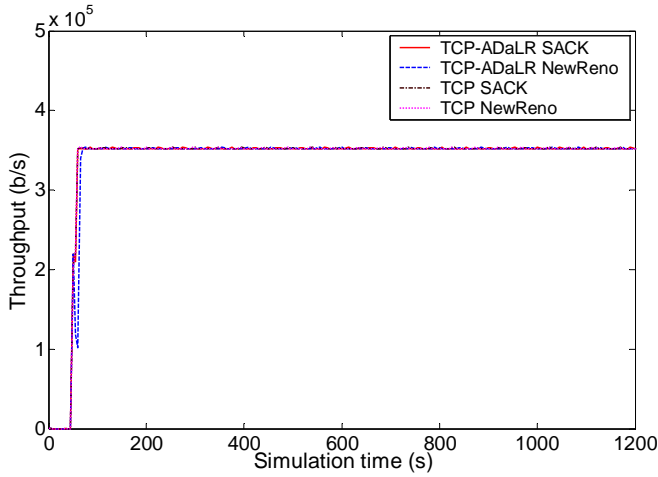
Fig. 10. Satellite link throughput for the scenario with only congestion losses and delayed ACK disabled. Satellite link throughput is comparable for the four TCP variants.

4.2.3 Scenario with only Satellite Link Error Losses

In the simulation scenario with only satellite link error losses, we evaluate the performance of the four TCP variants for various BERs (10^{-5} to 10^{-9}). We employ different random seed numbers for each BER value and compute average values using 95% confidence interval. The HTTP page and FTP download response times for the case without delayed ACK are shown in Figs. 11 and 12, respectively. TCP-ADaLR SACK exhibits the shortest response times for both HTTP and FTP applications. 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. The TCP goodput and satellite link throughput for the case without delayed ACK are shown in Figs. 13 and 14, respectively. TCP-ADaLR SACK exhibits the best performance. Similarly, for the case with delayed ACK, TCP-ADaLR SACK exhibits the best performance [10]. TCP-ADaLR variants outperform TCP SACK and TCP NewReno because the loss recovery mechanism enables faster recovery than TCP SACK and TCP NewReno. Furthermore, during the congestion avoidance phase, the adaptive window increase mechanism enables transmission of additional segments after loss recovery.
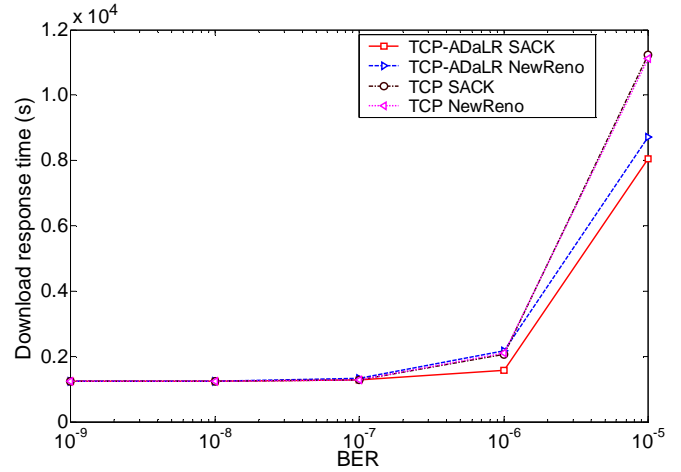

Fig. 12. FTP download response time for the scenario with only satellite link error losses and delayed ACK disabled. TCP-ADaLR SACK show up to 31% shorter download response time than TCP SACK.
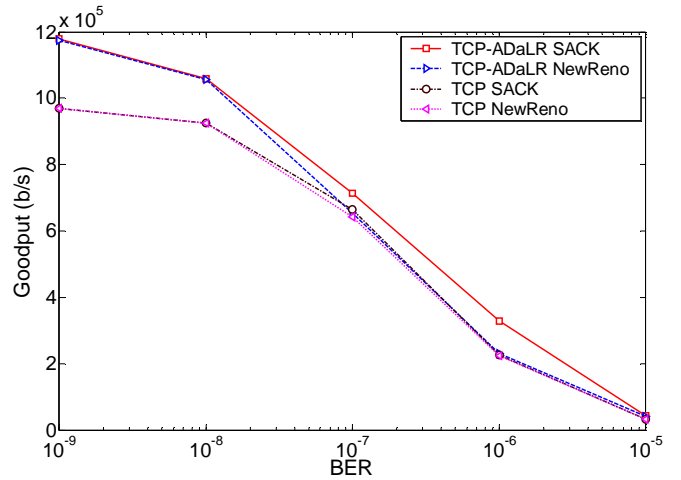

Fig. 13. Goodput for the scenario with only satellite link error losses and delayed ACK disabled. TCP-ADaLR SACK NewReno show up to 46% higher goodput than TCP SACK and TCP NewReno.
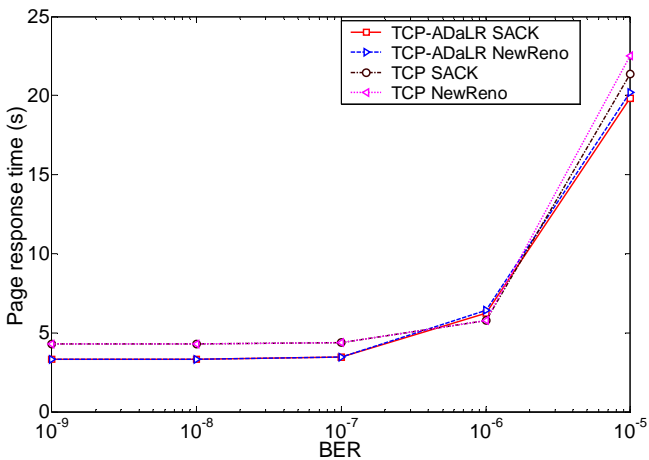

Fig. 11. HTTP page response time for the scenario with only satellite link error losses and delayed ACK disabled. TCP-ADaLR SACK and TCP-ADaLR NewReno exhibit up to 23% shorter page response times than TCP SACK and TCP NewReno.
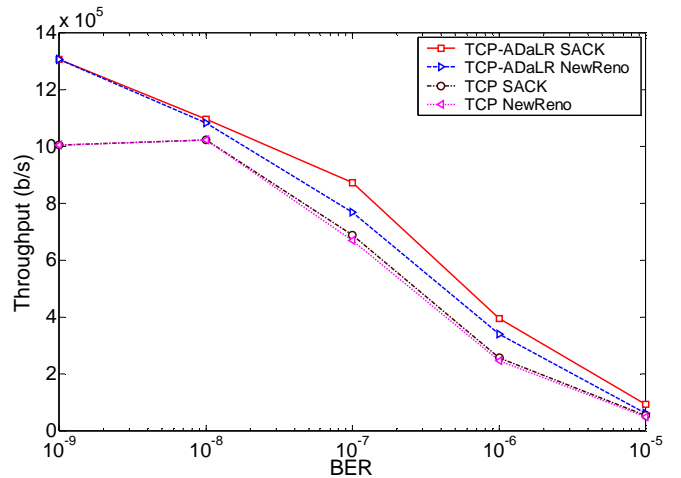

Fig. 14. Satellite link throughput for the scenario with only satellite link error losses and delayed ACK disabled. TCP-ADaLR SACK exhibits up to 73% higher satellite link throughput than TCP SACK.

4.2.4 Scenario with both Congestion and Error Losses

The HTTP and FTP response times for the case without delayed ACK are shown in Figs. 15 and 16, respectively. The TCP and satellite link throughputs for the case without delayed ACK are shown in Figs. 17 and 18, respectively. 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 losses are mainly due to link errors. For the case with delayed ACK, TCP-ADaLR variants exhibit similar performance [10].
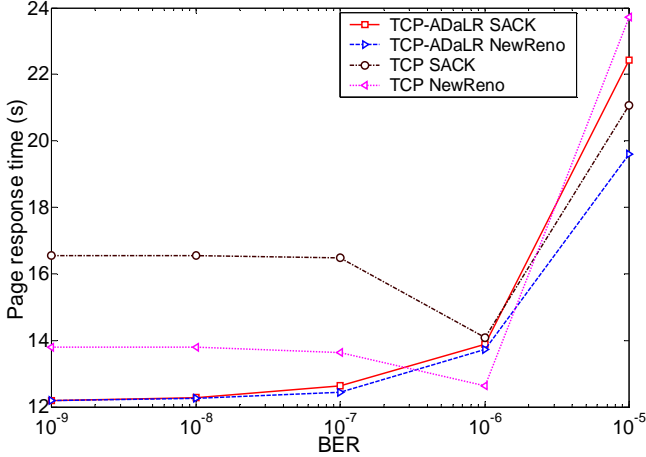

Fig. 15. HTTP page response time for scenarios with both congestion and error losses and delayed ACK disabled. TCP-ADaLR SACK exhibits up to 26% shorter page response times than TCP SACK.
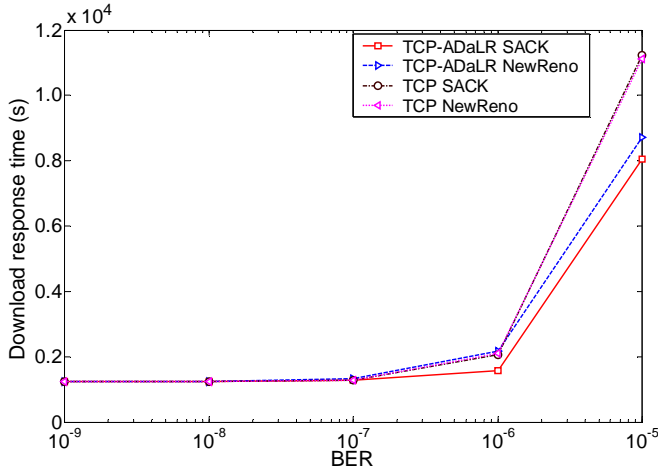

Fig. 16. FTP download response time for scenarios with both congestion and error losses and delayed ACK disabled. TCP-ADaLR SACK exhibits up to 28% shorter download response times than TCP SACK.

4.2.4 Fairness and Friendliness

The Jain's metric of fairness [21] is defined as:

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

where FI is the fairness (friendliness) index, n is the number of competing connections, and t_j is the average throughput of the jth connection. FI has values in the range $1/n \le FI \le 1$, where $1/n$ and 1 correspond to unfair and fair FI values, respectively.
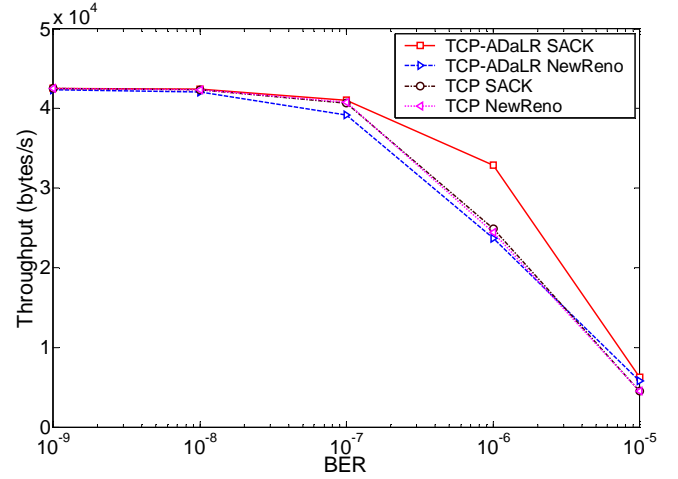

Fig. 17. TCP throughput for scenarios with both congestion and error losses and delayed ACK disabled. At BER values higher than 10-7, TCPADaLR SACK exhibits up to 39% higher TCP throughput than TCP SACK.
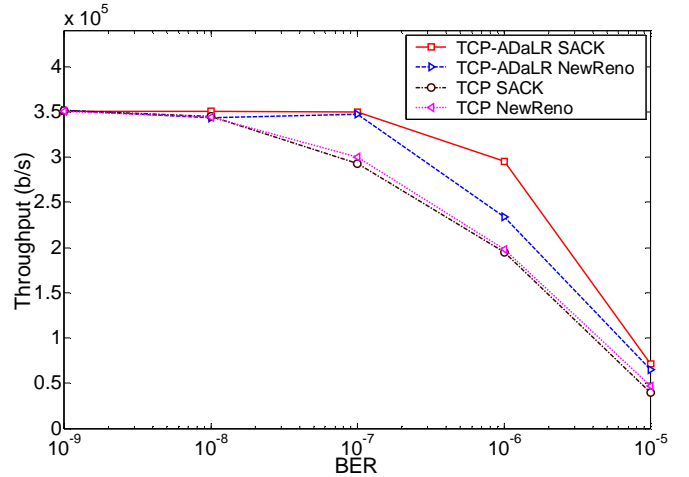

Fig. 18. Satellite link throughput for scenarios with both congestion and error losses and delayed ACK disabled. TCP-ADaLR SACK shows up to 79% higher satellite link throughput than TCP SACK.

We evaluate TCP fairness and friendliness using an FTP file download application (Table 3) in the network shown in Fig. 19. Links are 10 Mb/s full duplex Ethernet with 5 ms one-way propagation delays, unless otherwise stated. The one-way link propagation delays between receivers (0 to 5) and Ethernet switch_1 are 250 ms, 200 ms, 150 ms, 50 ms, 25 ms, and 12.5 ms, respectively. The bottleneck link is located between the two gateways. In two fairness scenarios with TCP-ADaLR and with TCP New Reno, we consider six TCP connections with identical TCP variants. In the friendliness scenario, we consider six co-existing connections. Three connections with shorter RTTs use TCP NewReno while the remaining three with longer RTTs use TCP-ADaLR. The fairness indices are shown in Table 8. TCP-ADaLR exhibits 9.9% higher FI than TCP NewReno. Hence, TCP-ADaLR is fairer to longer RTT connections without starving the shorter RTT connections at the bottleneck link. The FI (friendliness) for the six co-existing TCP ADaLR and NewReno connections is shown in Table 9. The FI (friendliness) indicates that TCP-ADaLR is friendly to co-existing connections employing distinct TCP variants.
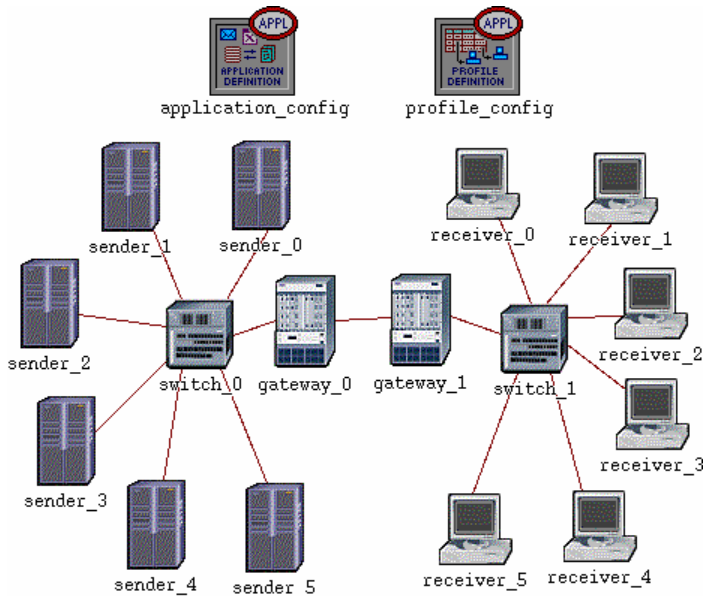
Fig. 19. Network used to evaluate fairness and friendliness of TCP-ADaLR and TCP NewReno.

Table 8. Fairness indices of TCP-ADaLR and TCP NewReno.

TCP variant	Fairness
TCP-ADaLR	0.9510
TCP NewReno	0.8650

Table 9. Friendliness index of six co-existing TCP-ADaLR and TCP NewReno connections.

TCP variant mix	Friendliness
TCP-ADaLR and TCP NewReno	0.9859

5. Conclusion

In this paper, we presented the OPNET implementation of the TCP-ADaLR algorithm. Two variants of TCP-ADaLR were considered: with SACK and with NewReno. We simulated scenarios with an ideal lossless satellite link to verify the algorithm implementation. We then considered scenarios with only congestion losses, only satellite link error losses, and both congestion and satellite link error losses. In each scenario, we considered cases with and without delayed ACK. We evaluated the page response times for an HTTP web application. We also evaluated the download response time, TCP goodput, TCP throughput, and satellite link throughput for an FTP file download application. The performance results show that TCP-ADaLR SACK always exhibits the best overall performance, followed by TCP-ADaLR NewReno.

The deployment of TCP-ADaLR in existing networks requires modifications only at the TCP sender thus introducing additional albeit minimal processing and memory overheads at the TCP sender. No modifications are required at the intermediate nodes and the TCP receiver. Implementation of the TCP-ADaLR algorithm does not violate the end-to-end semantics of TCP. Hence, TCP-ADaLR is compatible with IP security used for IP payload encryption and authentication. TCP-ADaLR may be deployed in networks with competing connections and distinct TCP variants because it is fair to competing connections and friendly to TCP NewReno.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

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

[8] 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.

[9] 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.

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

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

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

[13] 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.

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

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

[16] J. Postel, Ed., "Transmission control protocol," *RFC 793*, Sept. 1981.

[17] K. Ramakrishnan, S. Floyd, and D. Black, "The addition of explicit congestion notification to IP," *RFC 3168*, Sept. 2001.

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

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

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