Performance Evaluation of TCP Tahoe, Reno, Reno with SACK, and NewReno Using OPNET Modeler

Laxmi Subedi, Mohamadreza Najiminaini, and Ljiljana Trajković
Simon Fraser University
Vancouver, British Columbia
Canada
E-mail: {lsa38, mna28, and ljilja}@cs.sfu.ca

Abstract

Over the past two decades, numerous Transmission Control Protocol (TCP) congestion control algorithms have been proposed for deployed wireless and wired networks. The main goal of these algorithms was to successfully manage congestion, reliably handle loss, and minimize transmission errors. In this paper, we use OPNET Modeler to simulate four TCP versions (Tahoe, Reno, Reno with Selective Acknowledgment (SACK), and NewReno) in several wireless and wired networks. We analyze congestion window maintenance and recovery process for each TCP algorithm by observing congestion window size, file download response time, throughput, and goodput. We observe that in wireless networks with signal attenuation, fading, and multipath, TCP Reno has larger congestion window size, shorter file download response time, and higher throughput and goodput than the remaining three TCP algorithms. In case of wired networks, TCP Reno with SACK has the best overall performance.

1. Introduction

Transmission control protocol (TCP) [1], [2] is the predominant Internet protocol and carries approximately 90% of Internet traffic in today's heterogeneous wireless and wired networks. TCP is widely used as a connection oriented transport layer protocol that provides reliable packet delivery over unreliable links. TCP does not depend on the underlying network layers and, hence, design of various TCP flavors is based on the properties of wired networks. However, TCP congestion control algorithms may not perform well in heterogeneous networks.

TCP congestion control algorithms have been originally proposed based on the assumption that congestion is the only cause for packet loss. However, wireless networks have higher bit error rates due to weather conditions, obstacles, multipath interferences, mobility of wireless end-devices, and signal attenuation and fading, which may lead to packet loss. Various TCP algorithms and techniques have been proposed to improve congestion and reduce the non-congestion related packet loss. TCP Tahoe, TCP Reno, TCP Reno with Selective Acknowledgement (SACK), TCP NewReno, TCP Vegas, and TCP Binary Increase Congestion (BIC) are examples of proposed end-to-end solutions. Snoop-TCP is a link layer control approach while M-TCP and I-TCP are split connection approaches. They are all proposed to improve network performance [3]–[9]. The end-to-end techniques are the most promising because they require changes only to the end systems rather than to the intermediate nodes. These end-to-end control approaches are used in today's deployed networks.

In this paper, we perform a comparative performance analysis of four TCP congestion control algorithms (Tahoe, Reno, Reno with SACK, and NewReno) in wireless and wired networks. Analysis of these algorithms is performed using the OPNET Modeler [10]. In Section 2, we provide literature survey. Description of simulated network topologies is given in Section 3 while simulation scenarios and results are described in Section 4. We conclude with Section 5.

2. Transmission Control Protocol (TCP) Algorithms

In case of end-to-end control approaches, network layer does not provide information regarding congestion to the transport layer. Network congestion is inferred only by packet loss or delay (triple duplicate ACKs or timeout) and TCP decreases its congestion window size accordingly. Mechanisms devised to control congestion window are slow start, additive increase and multiplicative decrease (AIMD), congestion avoidance, fast retransmit, and fast recovery [11]–[14]. Various TCP flavors employ these congestion control algorithms to regulate the sending rate as a function of perceived congestion.

2.1 TCP Tahoe

TCP Tahoe was the first algorithm to employ three transmission phases: slow start, congestion avoidance, and fast retransmit [15], [16].

2.2 TCP Reno

TCP Reno is the most widely adopted Internet TCP protocol. It employs four transmission phases: slow start, congestion avoidance, fast retransmit, and fast recovery. When packet loss occurs in a congested link due to buffer overflow in the intermediate routers, either the sender receives three duplicate acknowledgments or the sender's retransmission timeout (RTO) timer expires. In case of three duplicate ACKs, the sender activates TCP fast retransmit and recovery algorithms and reduces its congestion window size to half. It then linearly increases congestion window, similar to the case of congestion avoidance. This increase in transmission rate is slower than in the case of slow start and helps relieve congestion. TCP Reno fast recovery algorithm improves TCP performance in case of a single packet loss within a window of data. However, performance of TCP Reno suffers in case of multiple packet losses within a window of data [17].

2.3 SACK

SACK algorithm allows a TCP receiver to acknowledge out-of-order segments selectively rather than cumulatively by acknowledging the last correctly in order received segment. The receiver acknowledges packets received out of order and the sender then retransmits only the missing data segments instead of sending all unacknowledged segments. TCP Reno with SACK

behaves similarly to TCP Tahoe and TCP Reno, which are robust in case of out of order packet arrivals. However, TCP with SACK helps improve performance in case of multiple packet losses. During the fast recovery phase, SACK maintains a variable called *pipe* that represents the estimated number of outstanding packets [17]. The sender only sends new or retransmitted data when the estimated number of packet in a router is smaller than the congestion window. The *pipe* variable is incremented by one when the sender either sends a new segment or retransmits an old one. It is decremented by one when the sender receives the duplicate ACK with a SACK option [18].

2.4 TCP NewReno

TCP NewReno is a modification of TCP Reno. It improves retransmission process during the fast recovery phase of TCP Reno. TCP NewReno can detect multiple packet losses. It does not exit the fast recovery phase until all unacknowledged segments at the time of fast recovery are acknowledged. Thus, as in TCP Reno, it overcomes reducing the congestion window size multiple times in case of multiple packet losses. The remaining three phases (slow start, congestion avoidance, and fast retransmit) are similar to TCP Reno. TCP NewReno exits fast recovery after receiving acknowledgement of all unacknowledged segments. It then sets congestion window size to slow start threshold and continues the congestion avoidance phase [19]. It retransmits the next segment when it receives a partial acknowledgment [20]. (Partial acknowledgments are the acknowledgments that do not acknowledge all outstanding packets at the onset of the fast recovery.)

2.5 Comparison of TCP Algorithms

Considerable research has been conducted to analyze the effect of various algorithms employed to recover and maintain congestion window in various network scenarios. The behavior of heterogeneous networks is unpredictable and, hence, no algorithm performs well in all cases. Many algorithms do not perform well in deployed networks because the effect of various network parameters and network variables cannot be accurately predicted. These algorithms are often only evaluated in simulated environments. However, many developed algorithms improve network performance and aim to achieve maximum utilization of end-to-end resources.

TCP NewReno, a modified version of TCP Reno, avoids several TCP Reno performance issues when multiple packet losses occur within a window of data [17]. However, it still limits the performance of TCP in the absence of SACK. Without selective acknowledgments, TCP implementations are constrained to either retransmit at most one dropped packet per round-trip time or to retransmit packets that might have already been successfully delivered. Hence, SACK performs better in case of multiple packet losses. The case of wireless networks, where frequent disconnections may occur, has not been considered [17].

TCP Reno with SACK shows lower bandwidth utilization in case of congested links and has lower goodput than TCP Reno and TCP NewReno [21]. Hence, performance of TCP Reno with SACK deteriorates in case of congested network. In case of non-congested network all three considered algorithms have comparable performance.

Performance of TCP Reno, Reno with SACK, and NewReno was compared using the ns-2 simulator [19]. It was found that increasing the bandwidth-delay product led to performance degradation regardless of TCP versions and the bottleneck buffer size. TCP NewReno outperforms TCP Reno and TCP Reno with SACK when no packet losses occur during the slow-start phase.

Analysis of various TCP algorithms over wireless links with correlated packet losses indicated that TCP NewReno often performs worse than TCP Tahoe because of the inefficient fast recovery algorithm [22]. Under certain conditions, the performance depends not only on the bandwidth-delay product but also on the nature of timeout. The ns–2 simulations indicated that SACK has the best and the most robust performance over wireless channels.

In summary, reported studies indicate variable performance of TCP algorithms. These variations may be due to a variety of assumptions, network parameters, and network topologies. The purpose of developing these algorithms is to maintain and recover the congestion window in order to increase the utilization of network resources in case of congestion without overwhelming intermediate routers and, thus, provide fastest response to the client.

3. OPNET Simulated Network Topologies

We develop a simple client server OPNET model to evaluate performance of TCP algorithms in wireless networks when packet losses occur due to signal attenuation. The simulated network shown in Fig. 1consists of a mobile client, a file transfer protocol (FTP) server connected with wired router by a 10 Mbps link, and a wireless router (base station for a mobile client) connected to a wired router by a 1.5 Mbps link. A 50 MB file is transferred using the FTP application from the server to the mobile client via wired and wireless routers.

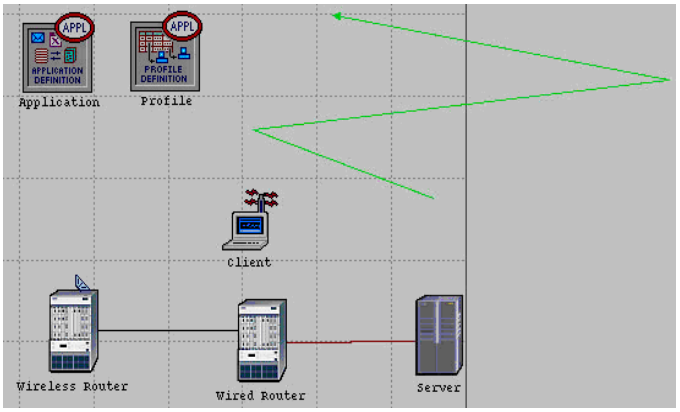
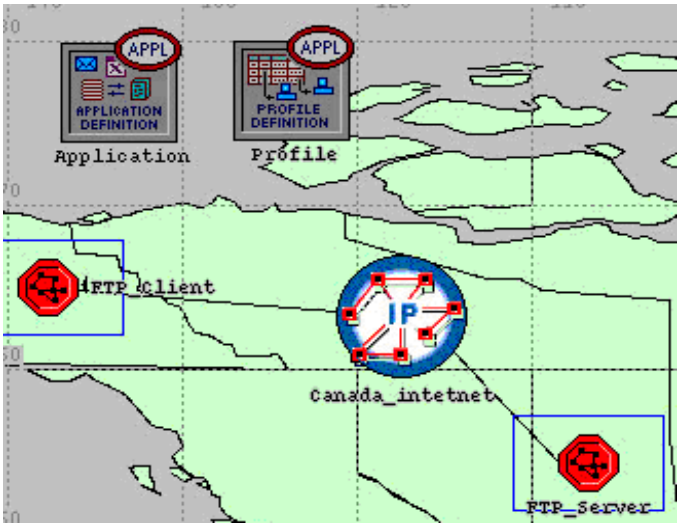
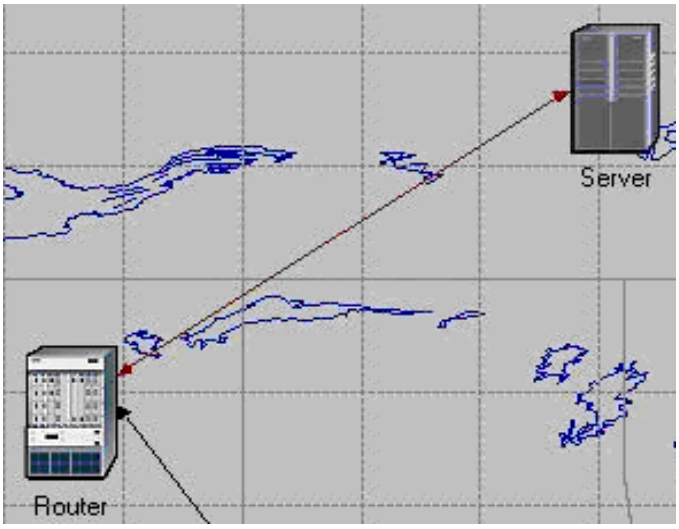


Fig. 1. OPNET model of the wireless client-server network.

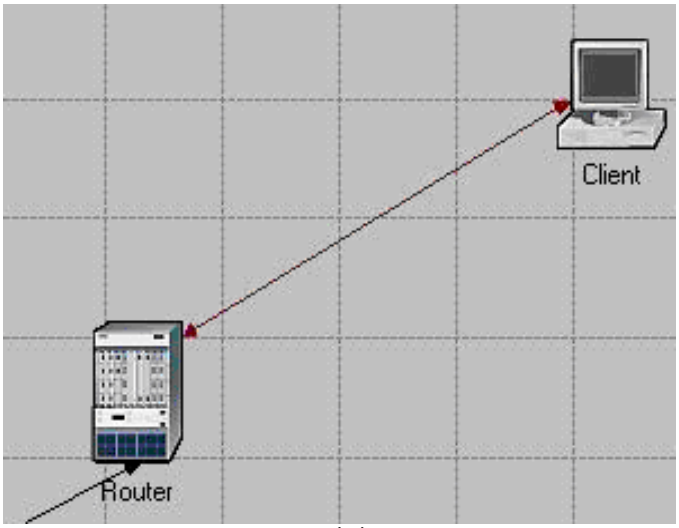
The OPNET model of the wired client server in a wide area network is shown in Fig. 2. The network consists of an Ethernet server and a client connected to the corresponding wired routers via 10 Mbps wired links. Two routers are connected to the Internet cloud via a 1.5 Mbps wired link. FTP application is used to transfer 10 MB file from the server to the client.



(a)



(b)



(c)

Fig. 2. OPNET model of the wired network: (a) Topology of the wide area network. (b) Server and router located within the FTP server subnet. (c) Client and router located within the FTP client subnet.

4. Simulation Scenarios and Results

We use the OPNET Modeler to simulate wireless and wired networks with four TCP versions: Tahoe, Reno, Reno with SACK, and NewReno. We analyze the effect of four performance parameters: congestion window size, file download response time, throughput, and goodput for the four TCP versions.

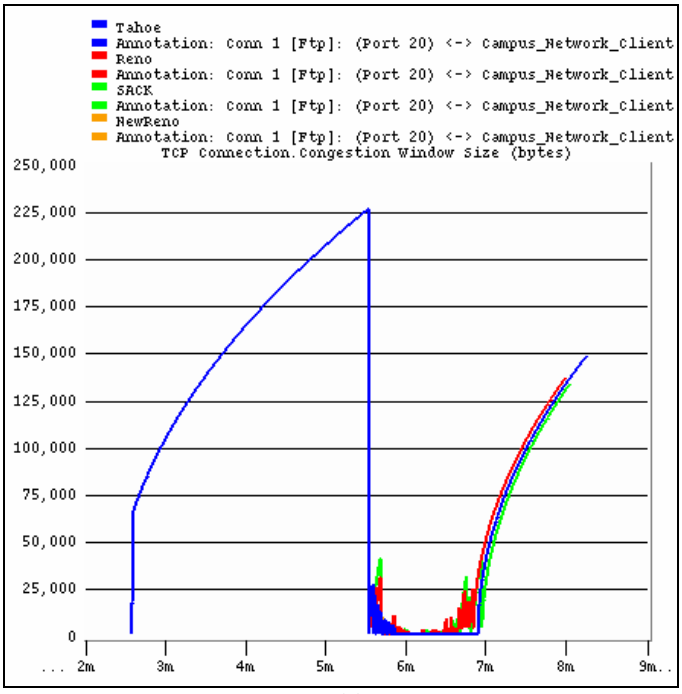
Selected TCP simulation parameters are shown in Table 1. For all TCP versions, TCP parameters such as slow start initial count and initial, minimum, and maximum retransmission time out (RTO) are set to the default values.

Table 1. TCP OPNET simulation parameters.

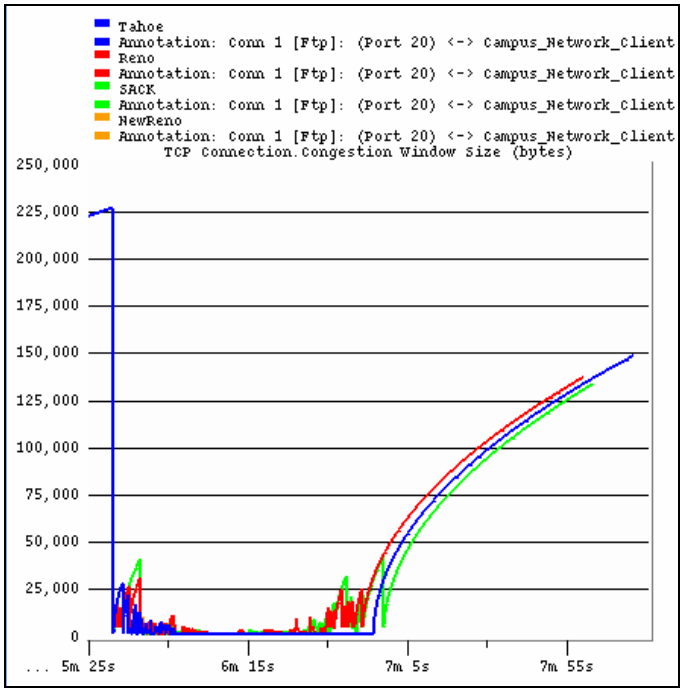
TCP parameters	Value
Slow start initial count (MSS)	1
Receive buffer size (bytes)	65,535
Maximum ACK segment	2
Initial RTO (sec)	1.0
Minimum RTO (sec)	0.5
Maximum RTO (sec)	64
RTT gain	0.125
Deviation gain	0.25
RTT deviation coefficient	4.0
Duplicate ACK threshold	3

4.1 Wireless client and server OPNET model

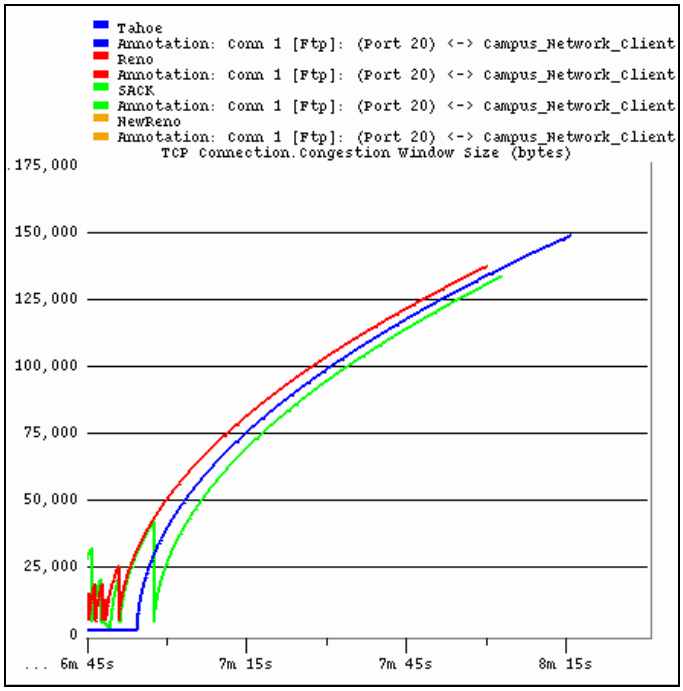
When the mobile client moves along a prescribed trajectory, signal attenuations occur at a certain distance from the base station and the packets between the client and the base station begin to drop. The network is simulated using four simulation scenarios with TCP Tahoe, Reno, Reno with SACK, and NewReno congestion control algorithms. We recorded the receiver file download response time, sender congestion window size, and TCP throughput and goodput.



(a)



(b)



(c)

Fig. 3. OPNET simulation of the wireless network: (a) Evolution of the congestion window size. (b) Detail: 325 s to 475 s. c) Detail: 405 s to 495 s.

Simulation results indicate that TCP Reno outperforms the remaining three algorithms. Fig. 3 shows the observed congestion window size for all four algorithms. It indicates that TCP Reno maintains larger congestion window size than the remaining three algorithms while TCP Reno with SACK has larger congestion window size than TCP Tahoe and TCP NewReno. (TCP Tahoe and TCP NewReno have comparable congestion window size.)

The congestion window size decreases for all four TCP variants, as shown in Fig 3 (b) and Fig 3 (c). The decrease at 5 min 33 sec is a result of packets loss due to signal attenuation as the client moves along its trajectory rather than due to the overflow in intermediate routers. During signal attenuation, TCP Reno with SACK and TCP Reno have larger congestion window size compared to TCP Tahoe and TCP NewReno, which have similar congestion window size. TCP Tahoe and TCP NewReno are more prone to signal attenuation and decrease the congestion window size to 1 MSS until the signal strength increases at 6 min 54 sec. TCP Reno and TCP Reno with SACK maintain larger congestion window during the signal attenuation period. This leads to shorter file download response time.

The file download response time for each algorithm is shown in Table 2. TCP Reno shows shorter FTP download response time than the remaining three algorithms.

Table 2. OPNET simulation of the wireless network: File download response time.

TCP variants	Download response time (s)
TCP Tahoe	491
TCP Reno	480
TCP Reno with SACK	483
TCP NewReno	496

TCP throughput and goodput are shown in Fig. 4 and Fig. 5, respectively. They indicate that TCP Reno has the highest throughput and goodput compared to the remaining three algorithms. TCP Reno with SACK has higher throughput and goodput than TCP NewReno and Tahoe while TCP NewReno and Tahoe have comparable throughput and goodput.

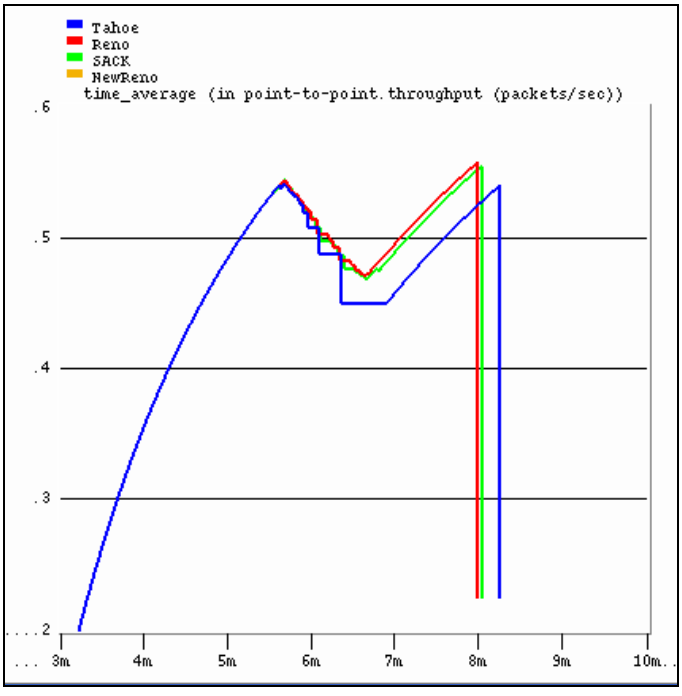


Fig. 4. OPNET simulation of the wireless network: Observed time-average throughput (packet/sec) in links from the server to the client. TCP Reno shows higher throughput than the remaining three algorithms.

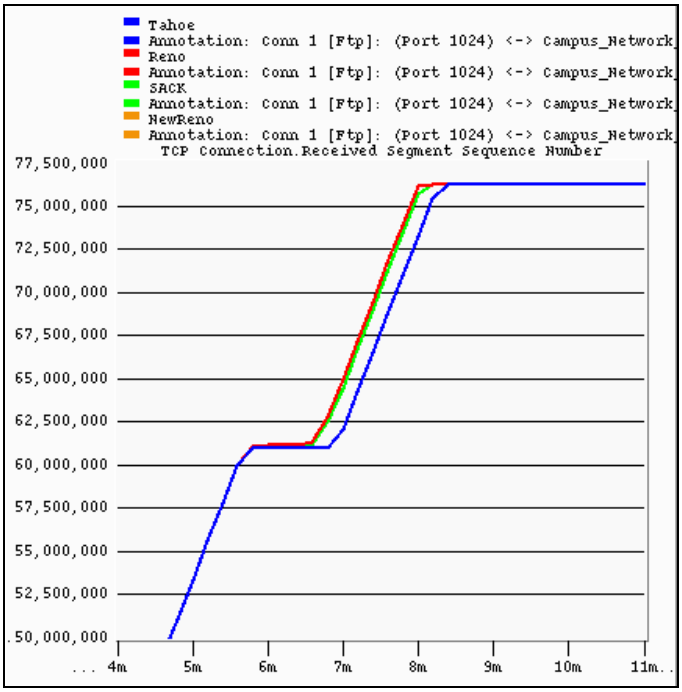


Fig. 5. OPNET simulation of the wireless network: Received segment sequence number (goodput). TCP Reno shows higher goodput than the remaining three algorithms.

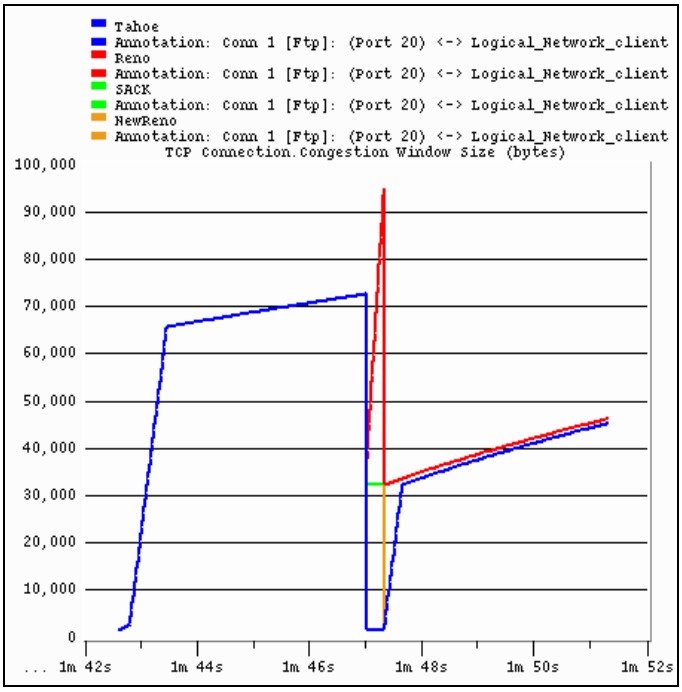
4.2 Wired client and server OPNET model

For the four TCP versions, we consider three simulation scenarios depending on the number of lost packets and analyze the behavior of TCP algorithms. We first consider the case of a single packet loss and observe the evolution of the congestion window size. In the second scenario, we consider two-packet losses in a window of data and observe the congestion window size. Finally, we consider multiple packet losses with packet latency in order to observe the congestion window size, file download response time, TCP throughput, and TCP goodput.

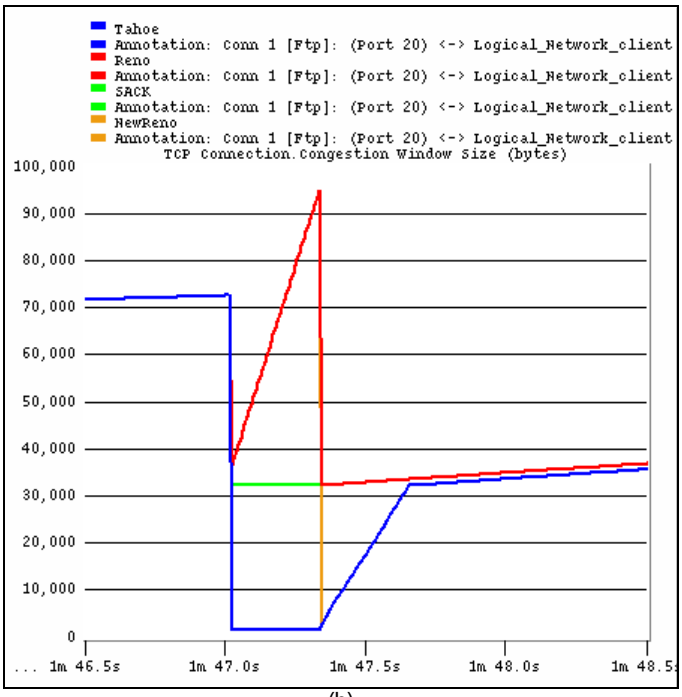
4.2.1 Scenario with single packet loss

The evolution of the congestion window size in a simulated client server network with a single packet loss is shown in Fig. 6. When a packet loss occurs at 1 min and 47 sec, TCP Tahoe reduces its congestion window size and invokes slow start mechanism after resetting its window size and the slow start threshold. However, after a packet loss occurs, TCP Reno, TCP Reno with SACK, and TCP NewReno perform fast retransmit and fast recovery.

During the fast retransmit and fast recovery phase, TCP Reno with SACK maintains constant congestion window size because of the selective acknowledgment mechanism, as shown in Fig. 6 (b). TCP Reno and TCP NewReno have variable window size due to different fast retransmit and fast recovery mechanisms. The four algorithms have identical file download response time in case of single packet loss. The algorithms also have identical throughput and comparable goodput.



(a)



(b)

Fig. 6. OPNET simulation of the wired network with a single packet loss: (a) Evolution of the congestion window size. (b) Detail: 106.7 s to 108.6 s.

4.2.2 Scenario with two-packet losses

In case of two-packet losses, TCP Reno with SACK has the largest congestion window size and the shortest file download response time, as shown in Fig. 7. TCP NewReno and TCP Tahoe have similar congestion window size (except during the fast retransmit and fast recovery phase). TCP NewReno has shorter download response time than TCP Tahoe. TCP Reno has the worst download response time due to two-packet losses

within a window of data. TCP Tahoe, TPC Reno with SACK and TCP NewReno have comparable throughput and goodput while TCP Reno has the worst performance.

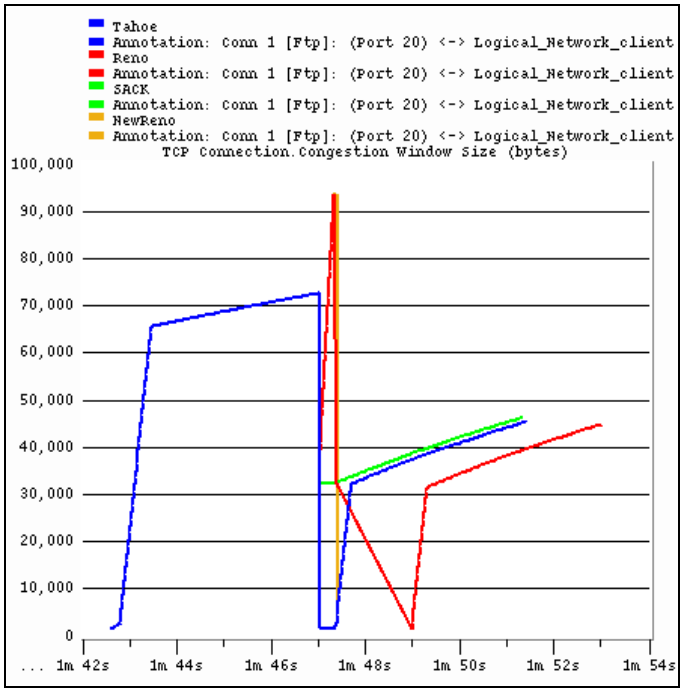


Fig. 7. OPNET simulation of the wired network with two-packet losses: Evolution of the congestion window size.

4.2.3 Scenario with 0.05 % packet loss

We simulate wired client server model with 0.05 % packet loss and packet latency of 1 ms with four TCP congestion control algorithms. Simulation results show that TCP Reno has larger file download response time than TCP Reno with SACK and NewReno, which have comparable file download response time. TCP Tahoe has a sharp increase in file download response time and, hence, it performs the worst compared to the remaining three algorithms. The file download response time for all four algorithms in case of 0.05 % packet loss is shown in Table 3.

Table 3. OPNET simulation of the wired network with 0.05 % packet loss: File download response time. TCP Reno with SACK shows shorter FTP download response time than the remaining three algorithms.

TCP variants	Download response time (s)
TCP Tahoe	168
TCP Reno	167
TCP Reno with SACK	164
TCP NewReno	165

The evolution of the congestion window size for the four TCP algorithms is shown in Fig 8. TCP NewReno frequently reduces its congestion window size whereas TCP Reno, Reno with SACK, and Tahoe maintain higher congestion window size. However, TCP Reno with SACK has shorter file download response time than the remaining three algorithms.

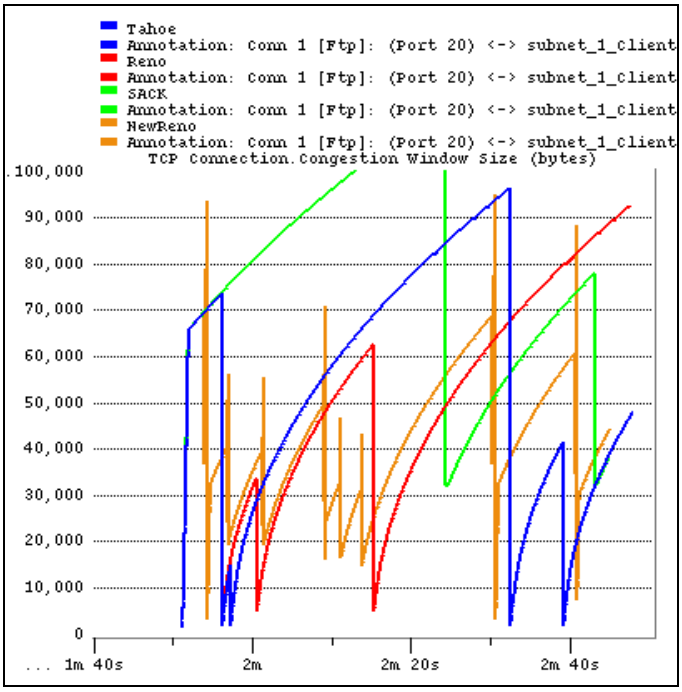


Fig. 8. OPNET simulation of the wired network with 0.05 % packet loss: Evolution of the congestion window size.

TCP Reno with SACK shows higher throughput and goodput than the remaining three algorithms. TCP throughput and goodput for the four TCP variants are shown in Fig. 9 and Fig.10, respectively.

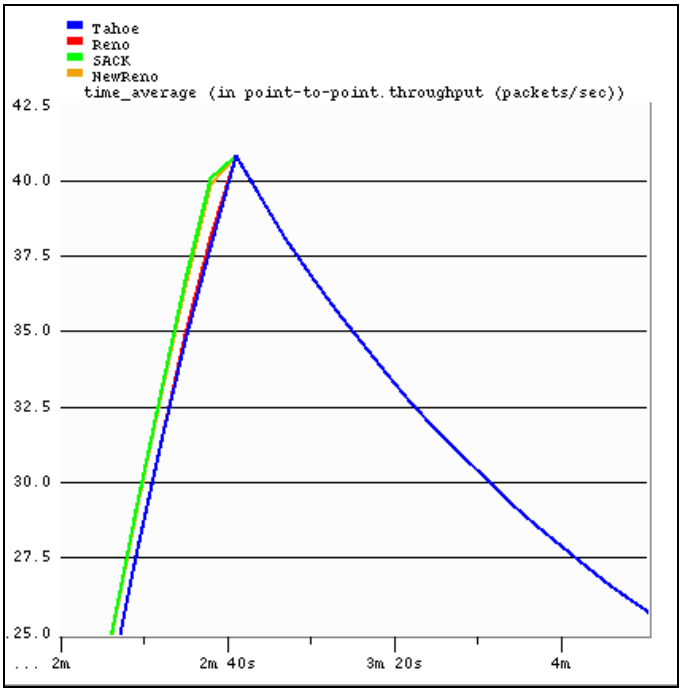


Fig. 9. OPNET simulation of the wired network with 0.05% packet loss: Observed time-average throughput from server to client. TCP Reno with SACK shows higher throughput than the remaining three algorithms.

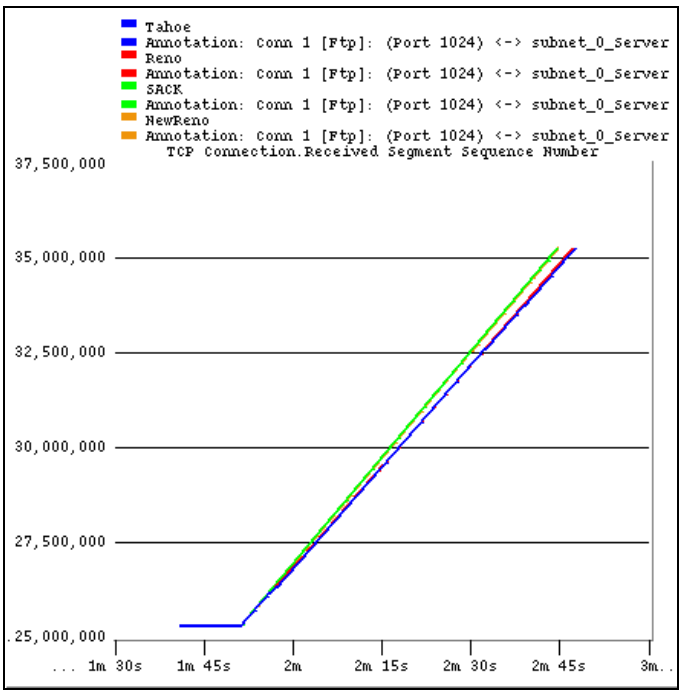


Fig. 10. OPNET simulation of the wired network with 0.05 % packet loss: Received segment sequence number (goodput). TCP Reno with SACK shows higher goodput than the remaining three algorithms.

5. Conclusion

In this paper, we simulated and compared performance of various TCP algorithms in wireless and wired networks. Simulation results indicated that in wireless networks with signal attenuation, fading, and multipath, TCP Reno outperforms other congestion control algorithms in terms of congestion window size and file download response time. The throughput and goodput in the case of TCP Reno is also higher than the remaining three algorithms. TCP Reno with SACK performs better than TCP Tahoe and TCP NewReno while TCP NewReno and TCP Tahoe algorithms show similar performance. In case of wired networks, TCP Reno shows significant performance degradation in case of multiple packet losses. Even though TCP Reno with SACK does not exhibit sharp decrease in congestion window as TCP NewReno when a packet loss occurs, the overall performance of TCP Reno with SACK and TCP NewReno is comparable in terms of file download response time. However, TCP Reno with SACK shows higher throughput and goodput than the remaining three algorithms.

References

[1] J. Postel, "Transmission control protocol," RFC 793, Sept. 1981.

[2] 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.

[3] G. Holland and N. Vaidya, "Analysis of TCP performance over mobile ad hoc networks," in *Proc. ACM/IEEE Int. Conf. on Mobile Computing*, Seattle, WA, USA, Sept. 1999, pp. 219–230.

[4] 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.

[5] 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.

[6] C.-H. Ng, J. Chow, and Lj. Trajkovic, "Performance evaluation of the TCP over WLAN 802.11 with the Snoop performance enhancing proxy," OPNETWORK 2002, Washington, DC, Aug. 2002.

[7] W. G. Zeng, M. Zhan, Z. Li, and Lj. Trajkovic, "Improving TCP performance with periodic disconnections over wireless links," *OPNETWORK 2003*, Washington, DC, Aug. 2003.

[8] W. G. Zeng and Lj. Trajkovic, "TCP packet control for wireless networks," in *Proc. IEEE Int. Conf. on Wireless and Mobile Computing, Networking and Communications* (*WiMob 2005*), Montreal, Canada, Aug. 2005, vol. 2, pp. 196–203.

[9] M. Omueti and Lj. Trajkovic, "M-TCP+: using disconnection feedback to improve performance of TCP in wired/wireless networks," in *Proc. SPECTS 2007*, San Diego, CA, USA, July 2007, pp. 443–450.

[10] OPNET Modeler software [Online]. Available:http://www.opnet.com/university_program/teaching_with_opnet.

[11] 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.

[12] W. Stevens, "TCP slow start, congestion avoidance, fast retransmit and fast recovery algorithms," *RFC 2001*, Jan. 1997.

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

[14] 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.

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

[16] M. N. Akhtar, M. A. O. Barry, and H. S. Al-Raweshidy, "Modified Tahoe TCP for wireless networks using OPNET simulator," in *Proc of the London Communications Symposium (LCS2003)*, London, UK, Sept. 2003.

[17] S. Floyd and K. Fall, "Simulation based comparisons of Tahoe, Reno, and SACK TCP," *ACM Computer Communication Review*, vol. 26, no. 3, pp. 5–21, July 1996.

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

[19] H. Lee, S. Lee, and Y. Choi, "The influence of the large bandwidth-delay product on TCP Reno, NewReno, and SACK," in *Proc. Information Networking Conference*, Oita, Japan, Feb. 2001, pp. 327–334.

[20] S. Floyd and T. Henderson, "The NewReno modification to TCP's fast recovery algorithm," *RFC 2582*, Apr. 1999.

[21] R. Paul and Lj. Trajkovic, "Selective-TCP for wired/wireless networks," in *Proc. SPECTS 2006*, Calgary, AL, Canada, Aug. 2006, pp. 339–346.

[22] F. Anjum and L. Tassiulas, "Comparative study of various TCP versions over a wireless link with correlated losses," *IEEE/ACM Transactions on Networking*, vol. 11, no. 3, pp. 370–383, June 2003.