On the Performance of TCP Loss Recovery Mechanisms

Michele M. de A. E. Lima
Department of Informatics
State University of West Paraná

Nelson L. S. da Fonseca
Institute of Computing
State University of Campinas

José F. de Rezende
GTA/COPPE
Federal University of Rio de Janeiro

Abstract -**This paper compares TCP Loss Recovery mechanisms that were proposed by the IETF to overcome TCP Reno lack of ability to recover efficiently from multiple losses in a single transmission window. The TCP extensions: NewReno, SACK and Limited Transmit, are compared under different traffic scenarios.**

I. Introduction

Currently, TCP is the leading transport protocol in the Internet. It is responsible for 95% of the bytes and for 90% of the packets sent. Moreover, 80% of the flows sent through the Internet are TCP flows. Among the applications that use TCP, the Web is the dominant one, comprising up to 75% of the bytes, 70% of the packets, and 75% of the flows considering both client and server traffic [1]. To optimize TCP congestion control mechanism it is necessary to take into account the characteristics of current network traffic that use TCP.

The congestion control mechanism of TCP Reno, the most popular TCP implementation, is composed of four algorithms: Slow Start, Congestion Avoidance, Fast Retransmit and Fast Recovery [2]. The first two algorithms are used by a TCP sender to control the amount of data injected into the network, while the other two are used to recover from packet losses without the need for Retransmission TimeOuts (RTOs).

Although the joint use of Fast Retransmit and Fast Recovery makes possible for a TCP Reno connection to achieve high throughput during moderate congestion, it is not effective in dealing with multiple packet losses in a single window. Each invocation of these algorithms can retransmit a single segment. At every loss, the congestion window is reduced to half of its size. Therefore, when multiple loss occur, the window size is reduced significantly. The return to its original size occurs only after a considerable delay.

If three or more packets are lost in a row, the TCP sender is forced to wait for a RTO to find out that a loss occurred, keeping the connection idle for a relatively long period. Moreover, after a RTO, the TCP sender is forced to enter in a costly Slow Start phase.

Measurement of a busy web server indicates that roughly 56% of the retransmissions happen after a RTO, while Fast Retransmit handled only 44% of those retransmissions [6]. Results also indicate that 85% of the retransmissions due to RTO could be avoided by the correct triggering of the Fast Retransmit algorithm [7].

Several IETF proposals have been presented to improve the loss recovery mechanism of TCP Reno, such as TCP SACK, TCP New Reno and TCP Limited Transmit.

In this paper, these loss recovery mechanisms are compared under different traffic scenarios. The use of TCP Limited Transmit with TCP Reno, TCP Sack, and TCP NewReno, to improve TCP performance is also investigated. Although previous works have conducted similar comparison, none of them have considered web traffic as traffic model as is done here. Moreover, no previous work compared all the mentioned TCP extensions.

This paper is organized as follows. Section II presents a brief summary of the mechanisms proposed to solve TCP Reno inefficacy. Section III presents previous research, which compare TCP Loss Recovery algorithms. In Section IV, numerical results are presented. Finally, in section V, conclusions are drawn.

II. TCP EXTENSIONS

This section briefly describes TCP SACK, TCP NewReno and TCP Limited Transmit, which were conceived to overcome TCP Reno loss recovery shortcoming.

TCP SACK ameliorates TCP Reno performance problems by providing information to the sender about the packets, which were correctly received by the TCP receiver. TCP NewReno improves TCP Reno loss recovery mechanism by the differentiation of the ACKs received. TCP Limited Transmit was specially designed to improve TCP Reno loss recovery mechanisms for connections with small congestion window size.

A. TCP Sack

The lack of information about which packets were lost leads to TCP Reno inability to recover efficiently from multiple losses in a single window [5]. The Selective ACKnowledgment (SACK) mechanism was proposed to ameliorate such limitation [3]. In SACK, the TCP receiver informs the TCP sender which packets were received correctly, allowing the TCP sender to implement a selective retransmission mechanism, so that only the missing segments are retransmitted. Since not all end systems may implement this TCP extension, its use should be negotiated.

SACK extension uses two of TCP options field, *SACK-permitted*, sent in SYN packets to indicate the willingness to use SACK. The second option field is used to convey extra acknowledgment information to TCP sender.

A packet with *SACK* option contains a list of blocks. Each block represents contiguous bytes of segments received and queued at the TCP receiver. A block is identified by two 32-bit unsigned integers in network byte order: the left edge of block and the right edge. The former is the first sequence number of the block, and the latter is the sequence number immediately following the last sequence number of the block. A *SACK* option length is $8*n+2$ bytes, where n is the number of blocks and the two extra bytes correspond to the fields *kind* and *length* of the TCP option. A maximum of 4 SACK blocks can be specified since the *option* field of TCP header is 40 bytes length.

A TCP receiver of a connection that uses SACK should generate SACK options in all ACKs, except in those that acknowledge the enqueued segment with the highest sequence number. The SACK option is a notification of which segments were received correctly. It does not imply that the segments were transferred to the application, since the TCP receiver can discard the segment, if necessary.

When a TCP sender receives an ACK, which contains a SACK option, it uses this information to decide which packets should be retransmitted. The TCP sender adds a flag (SACKed) to each segment enqueued at the retransmission queue, which have not yet been acknowledged. When receiving an ACK with SACK option, the sender must turn on the flags of all segments that have been selectively acknowledged. The segments that should be retransmitted are the ones with the flag set to *off* and which sequence number is lower than the highest segment number. If there are enqueued segments when a timeout occurs, the TCP sender should turn off the flags of all segments in the retransmission queue, and must retransmit all segments at the left edge of the window.

B. TCP NewReno

A TCP sender knows about the correct reception of a packet only when an ACK is received. If an ACK does not acknowledge all packets already sent, the TCP sender can neither infer the number of lost segments, nor which packets should be retransmitted. In addition, a TCP sender can learn about just one lost segment per round trip time (RTT). Therefore, it is difficult to make the appropriate decision during the Fast Retransmit/Fast Recovery phase [5].

The Fast Retransmit phase begins when the TCP sender receives three duplicated ACKs, which suggests that a packet was lost. After the retransmission of a packet, an ACK can acknowledge all packets sent (total acknowledgment) or just some of them (partial acknowledgment). A partial acknowledgement is a strong indication that more than one packet were lost [4]. It is also an indication that the first unacknowledged packet should be retransmitted in order to recover more efficiently from a burst of losses, avoiding the need to wait for the expiration of the timer. In line with that, a TCP extension called TCP New Reno was proposed as a RFC with experimental status [5].

In TCP New Reno, when a packet is retransmitted, the highest sequence number is recorded in a variable called *recover*. When an ACK is received, it is verified whether the ACK is a partial or a total acknowledgment. If it is a partial acknowledgment, the packet with the lowest sequence number not yet acknowledged is retransmitted and the congestion window is deflated by the amount of acknowledged data. Moreover, the window size is incremented by one segment and a new packet is transmitted if allowed by the current congestion window value.

TCP NewReno is specially indicated in the absence of the SACK TCP extension. TCP NewReno is also recommended even when SACK is supported, since SACK option can only be used when both TCP end-systems support it.

A major drawback of TCP New Reno is that the sender retransmits only one packet per RTT. When several losses occur, the TCP sender usually recovers only after a considerable delay.

C. TCP Limited Transmit

Since both TCP SACK and TCP NewReno senders need to receive three duplicate ACKs to trigger the Fast Retransmit algorithm, they perform poorly for small window size, especially for windows smaller or equal to three segments [6][7]. When the duration of the connection is short, a TCP sender cannot probe the available bandwidth, due to the small amount of data to be sent.

The congestion window of a TCP sender is typically small during a web session since HTTP initiates separate TCP connections to transfer one or more objects of a Web page [1].

[1] HTTP 1.0 opens a TCP connection per object of a Web page. HTTP 1.1 TCP connections are persistent, which allows the transfer of more than one object per connection.

Results indicate that only 10% of RTOs occurs when the congestion window is larger than 10 segments. Moreover, only 4% of the retransmissions due to RTO can be eventually avoided by the use of SACK [6].

To improve the effectiveness of TCP for small windows, the Limited Transmit algorithm was proposed in a RFC with experimental status [8]. In the Limited Transmit algorithm when two consecutive ACKs for the same segment are received, a new segment is transmitted if the receiver advertised window allows and if the amount of outstanding data is equal or less than the congestion window plus 2 [8]. The main idea is to cause the sending of a higher number of ACKs and, consequently, the triggering of the Fast Retransmit algorithm. The size of the congestion window should not be changed during the transmission of these new segments. Moreover, if SACK is used, the TCP sender should not send new segments when duplicate ACKs do not bring new information.

Such modifications can avoid on average 25% of retransmissions due to RTO. Moreover bursts of retransmissions are prevented, since they are clocked out by incoming ACKs.

III. Related Work

In [9], Fall and Floyd presented a comparison study of TCP Tahoe, TCP Reno, TCP New Reno and TCP SACK. They illustrated the benefits of adding SACK extension to TCP. The comparison was based in simulated scenarios ranging from one to four packets losses. This work did not consider Internet traffic characteristics. Moreover, it did not use multiple metrics for comparison.

Balakrishnan, Padmanabhan, Seshan, Stemm and Katz [6] analyzed the use of TCP connections by Web browsers accessing a busy Web server. They discovered that short Web transfers lead to poor loss recovery performance. They presented an enhancement to TCP loss recovery mechanism, called Enhanced Recovery, which is very similar to the joint use of Limited Transmit and TCP NewReno. In this study, only the aspect of congestion control mechanism of Enhanced Recovery and TCP SACK are compared.

Lin and Kung [7] proposed a new version of TCP congestion control mechanism, called TCP NetReno, which was optimized to improve TCP Fast Recovery phase and to avoid RTOs. One of the enhancements proposed in this work is similar to the Limited Transmit proposal. TCP NetReno was compared to TCP SACK and to TCP NewReno, by considering the number of dropped packets, the number of RTOs and the goodput. The simulation experiments used backlogged FTP traffic.

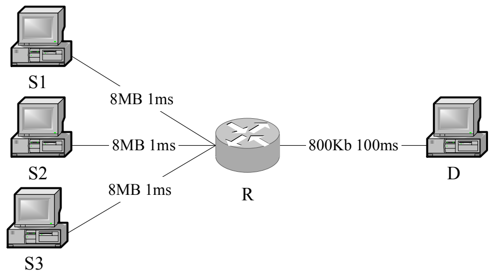


Fig1. Topology used in the simulation experiments

IV. Numerical Results

Simulation experiments were performed using ns-2.1b9a [10]. The topology used is presented in Fig 1. The link between node *R* and node *D* is the bottleneck link. RED was the buffer management mechanisms used in the bottleneck link.

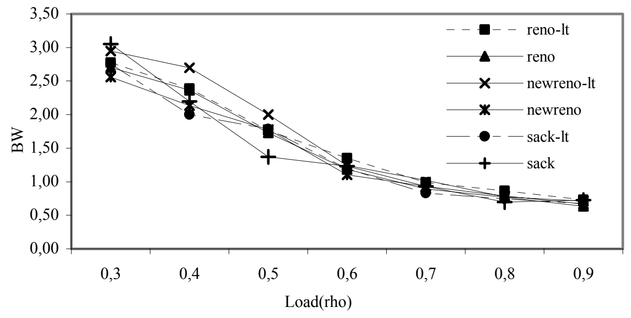
The simulations used a traffic generator, called TrafficGen, to generate specific traffic loads [11]. The Web traffic is modeled by a hybrid Lognormal/Pareto distribution. The body of the distribution corresponding to an area of 0.88 is modeled by a Lognormal distribution, with mean of 7247 bytes and the tail is modeled by a Pareto distribution with mean of 10558 bytes [13]. FTP traffic is generated using an exponential distribution with mean of 512 KBytes.

Both FTP and HTTP traffic is generated from node S_1 to node *D*. Additional CBR/UDP traffic is generated from nodes S_2 and S_3 to node *D* to create interfering traffic.

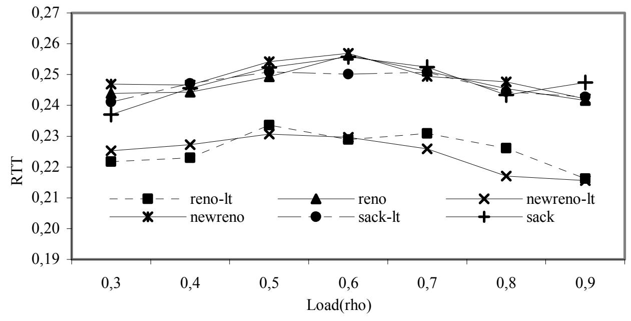
TCP Reno, TCP NewReno, and TCP Sack were compared under different load conditions. The total load was varied from 0,3 to 0,9. The joint use of these protocols with Limited Transmit, denoted by reno-lt, newreno-lt and sack-lt, were simulated under the same traffic scenario. The average obtained bandwidth (BW), the average round trip time (RTT) and the average number of timeouts (RTO) as a function of network load are plotted in Fig. 2 and in Fig. 3. Confidence intervals were generated by the independent replication method and are not shown for the sake of visual interpretation.

From Fig. 2, it can be noticed that for loads greater than 0,6 there is no significant difference on the obtained bandwidth by different TCP extensions.

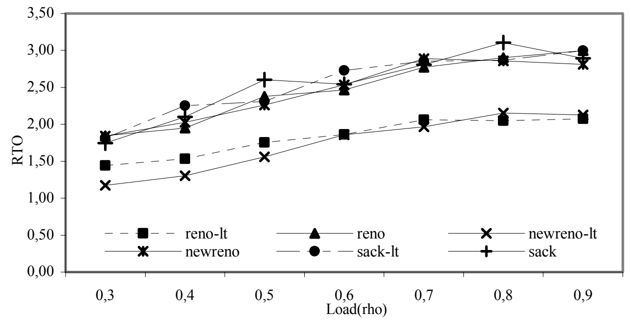
It can be verified in Fig 2.a that the use of any extension do not influence so much in the obtained bandwidth when the network load is greater of 60%. TCP Reno and TCP NewReno with Limited Transmit obtained slightly higher share bandwidth.



(a)



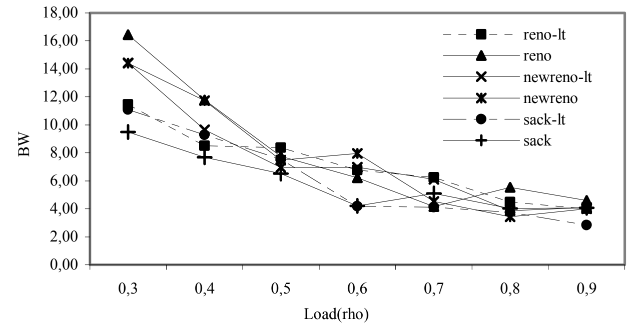
(b)



(c)

Fig 2. Web Experiments

The joint use of Limited Transmit with TCP Reno (reno-lt) and the joint use of Limited Transmit with TCP NewReno (newreno-lt) decrease the duration of RTT's as well as the number of RTO's. The decrease of RTT given by reno-lt is 9% of the RTT value given by Reno, and 26% of the number of RTO's given by Reno. The decrease of the duration of RTT and of the number of RTO's given by new-reno-lt when compared to the corresponding values given by NewReno is respectively 10% and 30%. Newreno-lt provides better results because it can also avoids timeout when multiple segments are lost in a single window.



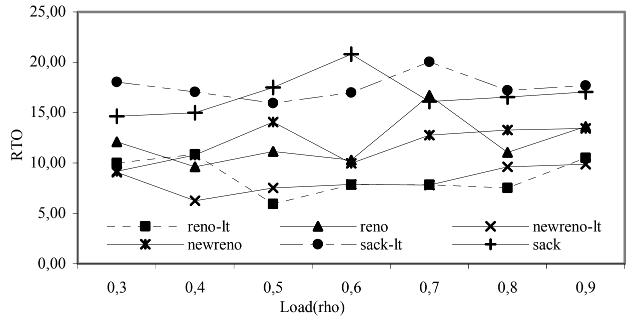


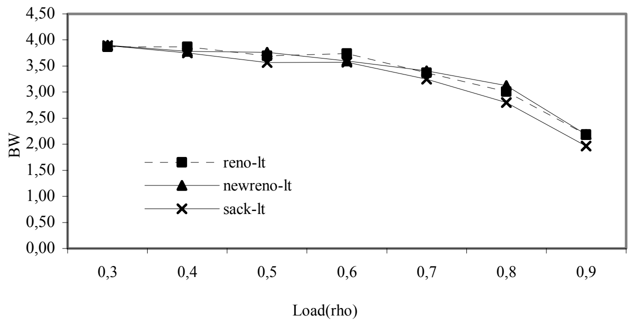
Fig 3. FTP Experiments

These results were already expected, since the Limited Transmit was designed to improve TCP performance for small congestion windows, typical of web traffic. In the experiments the average congestion window size was 6 segments for loads less than 0,6 and 3 segments for higher loads.

The joint use of Sack with Limited Transmit does not imply in significant improvement. It performs worst than do reno-lt and newreno-lt. One possible explanation for that is that multiple losses do not contribute to more than 10% of the timeouts. Since Sack was design to improve TCP when multiple losses occur in a single window, it is not effective in avoiding timeouts [7]. Sack is more effective when connections have large congestion windows, which is common in satellite networks.

Fig. 3 presents results for FTP traffic. RTT results are not shown since they slightly differ for different TCP extensions.

It can be seen that the mean number of RTO produced under FTP traffic can be seven times greater than the number of RTO under web traffic. However, the use of reno-lt and newreno-lt also reduces the number of RTO by 26% and 29% when compared to results given by Reno and NewReno, respectively.



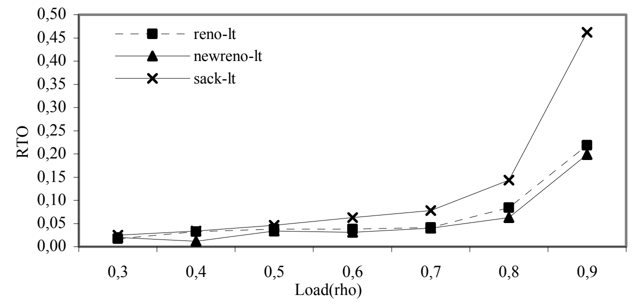


Fig 4. Fairness Experiments

Fig. 4 presents fairness results for reno-lt, newreno-lt and sack-lt. Web traffic was used and the load was varied from 0,3 to 0,9.

RTT results are not presented since they did not significantly differ. As can be seen in the Fig. 4, the obtained bandwidth value is very similar, which means that they share fairly the available bandwidth. However, newreno-lt presents better results for the mean number of RTO. Again, Sack performance was the worst among them.

V. Conclusions

In this paper, the joint use of Limited Transmit with TCP Reno, with TCP NewReno and with TCP SACK were investigated. Simulation experiments using current network traffic were utilized in the comparison of different schemes. It was shown that the joint use of TCP NewReno with Limited Transmit provides better improvement on the loss recovery mechanism of TCP Reno when compared to the improvement given by the other schemes since TCP NewReno with Limited Transmit reduces the total number of timeouts and consequently reduces the delay of the connections.

References

[1] K. Claffy, G. Miller, and K. Thompson. “The Nature of the Beast: Recent Traffic Measurements from an Internet Backbone”. In Proceedings of INET'98.

[2] M. Allman, V. Paxson, W. Stevens. “TCP Congestion Control”. RFC 2581, April 1999.

[3] M. Mathis, J. Mahdavi, S. Floyd and A. Romanow. “TCP Selective Acknowledgment Options”. RFC 2018, October 1996.

[4] J. Hoe. “Improving the Startup Behavior of a Congestion Control Scheme for TCP”. In ACM SIGCOMM, August 1996.

[5] S. Floyd and T. Henderson. “The NewReno Modification to TCP's Fast Recovery Algorithm”. RFC 2582, April 1999.

[6] H. Balakrishnan, V. Padmanabhan, S. Seshan, S. Stemm and R. Katz. “TCP Behavior of a Busy Internet Server: Analysis and Improvements”. In Proceedings of IEEE Infocom, March 1998.

[7] Dong Lin and H. T. Kung. “TCP Fast Recovery Strategies: Analysis and Improvements”. In Proceedings of Infocom 98, March 1998.

[8] M. Allman, H. Balakrishnan and S. Floyd. “Enhancing TCP's Loss Recovery Using Limited Transmit”. RFC 3042, January 2001.

[9] Kevin Fall and Sally Floyd. “Simulation-based Comparisons of Tahoe, Reno and SACK TCP”. Computer Communication Review, July 1996.

[10] http://www.isi.edu/nsnam/ns

[11] Kleber V. Cardoso and José. F. de Rezende. “HTTP Traffic Modeling: Development and Application”. International Telecommunications Symposium – ITS2002, Natal Brazil. “in press”.

[12] Maurizio Molina, Paolo Casteli, and Gianluca Foddis. “Web Traffic Modelling Exploiting TCP connections – Temporal Clustering through HTML-Reduce”. IEEE Network Magazine, vol. 14, no 3, pp 46-55, 2000.

[13] Pauls Barford, Azer Bestravos, Adam Bradley and Mark Crovella. “Changes in Web Client Access patterns: Characteristics and Caching Implications”. In Special Issue on Characterization and Performance Evaluations, 1999.