An Implementation of the SACK-Based Conservative Loss Recovery Algorithm for TCP in ns-3

Truc Anh N Nguyen* and James P.G. Sterbenz*†§
*Information and Telecommunication Technology Center
Department of Electrical Engineering and Computer Science
The University of Kansas, Lawrence, KS 66045, USA
†School of Computing and Communications (SCC) and InfoLab21
The Lancaster University, Lancaster LA1 4WA, UK
§Department of Computing
The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong
{annguyen, jpgs}@ittc.ku.edu
www.ittc.ku.edu/resilinets

ABSTRACT

This extended abstract is a short version of our full paper, which describes our implementation and simulation analysis of the SACK option and the SACK-based conservative loss recovery algorithm for TCP in ns-3.

Categories and Subject Descriptors

I.6 [**Simulation and Modeling**]: General, Model Development, Model Validation and Analysis

General Terms

Implementation, Analysis, Testing, Verification

Keywords

TCP, SACK, ns-3, Tahoe, (New)Reno, Westwood+

1. INTRODUCTION

TCP, although having been evolved as the dominant reliable transport protocol for the Global Internet, fails to maintain its throughput when being deployed in network environments that suffer highly correlated corruption-based losses. Many mechanisms have been proposed to improve TCP loss recovery process. One of them is the SACK option extension to TCP [7], which in turns motivates the development of TCP SACK's conservative loss recovery algorithm that was originally proposed in [2] and later revised in [3]. Given the lack of SACK option and TCP SACK models in ns-3 [1], we have decided to implement these models to extend ns-3's functionality and hopefully encourage further developments and studies that are based on or make use of the models. In this extended abstract, we briefly discuss the SACK option structure and the loss recovery algorithm employed by TCP SACK in Section 2, which is then followed by a description of our implementations in Section 3. In Section 4, we discuss a few results obtained from our simulations of the models. Finally, Section 5 concludes our abstract with directions for future work. This extended abstract is a very short version of our full paper available on our ResiliNets wiki [8] [1]. We remark that although there have been studies on TCP SACK [6, 4], both of these studies implemented the previous version as discussed in [2]. To the best of our knowledge, our work is the first study on TCP SACK that incorporates the recent changes specified in [3].

Wns3 2015 March 13, Barcelona, Spain.

2. BACKGROUND

To address the limitation of traditional cumulative ACK in facing multiple segment losses per sending window, the SACK option, which was first proposed in [5] and later revised in [7] to increase its robustness, allows a TCP receiver to inform the sender isolated chunks of data existed in its buffer due to some missing segments. Each chunk is represented by a SACK block consisting of two 32-bit sequence numbers identifying the left edge and the right edge of the data chunk. The use of SACK option is negotiated by appending the SACK-PERMITTED option to the SYN message at the beginning of a connection. TCP SACK, first proposed in [2] and later refined in [3] implements another loss recovery scheme in addition to NewReno's fast recovery to exploit the information carried in SACK option to assist a TCP sender in making correct retransmission decision and better utilizing the available network's bandwidth.

3. IMPLEMENTATION

We implement the SACK-PERMITTED and SACK options by following the structures described in [7] as derived classes of `TcpOption`. The processing of these options are implemented in the base class `TcpSocketBase` to allow the use of the options by other TCP variants when desired. The construction of a SACK list, which consists of multiple SACK blocks, is assisted by an auxiliary function `TcpSocketBase::ArrangeSackBlocks` to ensure that the first block

[1] https://wiki.ittc.ku.edu/resilinets/ResiliNets_Publications

contains the triggered sequence number. After its construction, the SACK list is added to the header through a call to `TcpHeader::AppendOption`. The `AppendOption` may be recalled to ensure that the length of the SACK list together with any other options already appended to the header do not exceed the 40-byte option length limit.

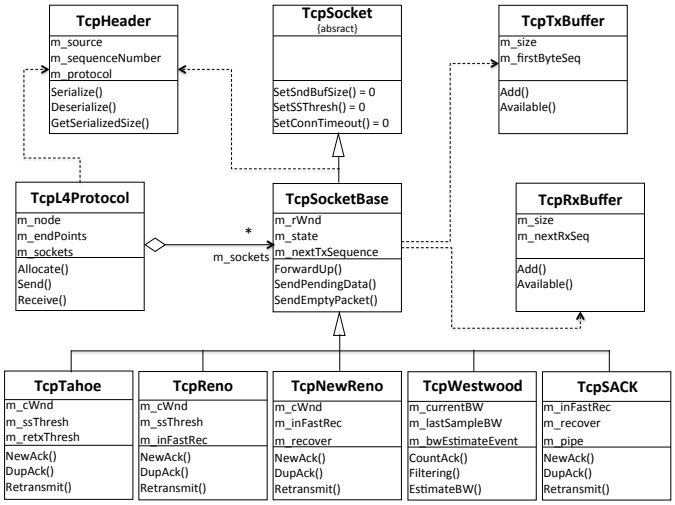

Figure 1: Class diagram of TCP module in ns-3

Figure 1 illustrates the interaction between different TCP classes in ns-3. The implementation of TCP SACK consists of 2 main files: `TcpScoreBoard` and `TcpSack`. `TcpScoreBoard` implements the scoreboard structure, which is a list of scoreboard entries (`TcpScoreBoardEntry`) where each containing the sequence number of a transmitted segment m_sequenceNumber and a SACK flag m_isSacked. The scoreboard is used by a TCP SACK sender to keep track of SACK information received from the other side. While in this work, we are not concerned with the efficiency of the list data structure, our implementation ensures that it can correctly perform all the scoreboard functions outlined in the specification, including `AddEntry`, which creates an entry when a new data segment is transmitted using `TcpSack::SendDataPacket`, `Update`, which scans through the scoreboard to discard entries that are cumulatively ACKed and update SACK flags of those that are SACKed upon the arrival of an ACK, `IsLost`, which determines if a sequence number is considered to be lost, `SetPipe`, which calculates the number of outstanding segments m_pipe, and `IsDupACK`, which determines if a received segment that carries SACK information is a duplicate ACK as defined in the new specification [3]. The `NextSeq` functionality described in [3] is accomplished through `NextSeqPerRule1`, `TcpSocketBase::SendPendingDate` (in place of `NextSeqPerRule2`), `NextSeqPerlRule3`, and `NextSeqPerRule4` methods.

The implementation of the SACK-based loss recovery algorithm is contained in `TcpSack`, which is inherited from the base class `TcpSocketBase` and covers the 3 main methods: `SackDupAck`, `NewAck`, and `StartPipe`. Given the new definition of duplicate ACK in the revised TCP SACK specification, we give our method a different name called `SackDupAck` to distinguish it from `TcpSocketBase::DupAck`. This method is called when `TcpScoreBoard::IsDupAck` returns true. In addition to determining if the TCP needs to enter loss recovery like `DupAck`, `SackDupAck` also uses TCP SACK's pipe algorithm by calling `StartPipe` to take advantage of additional available congestion window for additional data transmission. Implemented in `NewAck`, TCP SACK's behavior on the receipt of a new ACK is very similar to NewReno's except the employment of the pipe algorithm when a partial ACK is received. Finally, `StartPipe` implements the TCP SACK's pipe algorithm, which is the heart of TCP SACK allowing a better utilization of available network's bandwidth by exploiting information conveyed in SACK blocks. This method calculates the amount of additional data that can be transmitted as the difference between the congestion window m_cWnd and the number of outstanding octets m_pipe.

4. SIMULATION RESULTS

In our work, we perform simulations to study TCP SACK's behavior and compare its performance with the existing TCP variants in ns-3 including Tahoe, Reno, NewReno, and Westwood+. Figure 2 illustrates our simulation topology, and Table 1 summarizes all the simulation parameters. Due to the page constraint in this extended abstract, we present only a plot in Figure 3, which shows that TCP SACK performs as well as NewReno and outperforms both Tahoe and Reno in the presence of correlated losses with high error rate. TCP SACK performs worse than Westwood+ at the error rate of 10^{-3} because even though TCP SACK tries to send more data after a loss, it still reduces its congestion window when losses occur. Westwood+, on the other hand, tries to estimate the actual bandwidth to make decision on its sending rate following a packet loss.

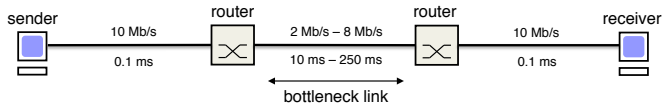

Figure 2: Single flow topology

Parameter	Values
Access link bandwidth	10 Mb/s
Bottleneck link bandwidth	4 Mb/s to 8 Mb/s
Access link propagation delay	0.1 ms
Bottleneck link propagation delay	50 ms to 250 ms
Packet MTU size	1500 B
Delayed ACK count	2 segments
Delayed ACK timeout	200 ms
Error model	BurstErrorModel
Burst error rate	10^{-7} to 10^{-3}
Burst size	2 to 5
Application type	Bulk send application
Simulation time	600 s
Number of runs	10

Table 1: Simulation parameters

5. CONCLUSIONS AND FUTURE WORK

In this extended abstract, we briefly discuss our work on implementing and simulating the SACK option and the newest version of the SACK-based conservative loss recovery algorithm for TCP in ns-3. We acknowledge that in this

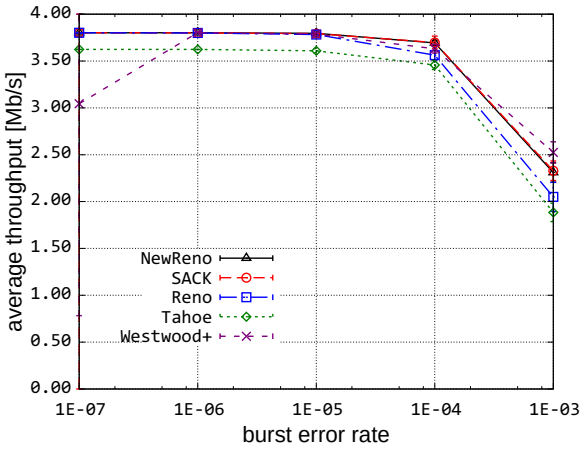

Figure 3: Average throughput vs. burst error rate

work, we use a very simple simulation topology. Hence, for the future work, we plan to extend our study of TCP SACK using a more complicated topology.

6. REFERENCES

[1] The ns-3 network simulator. `http://www.nsnam.org`, July 2009.

[2] E. Blanton, M. Allman, K. Fall, and L. Wang. A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP. RFC 3517 (Proposed Standard), Apr. 2003.

[3] E. Blanton, M. Allman, L. Wang, I.Jarvinen, M. Kojo, and Y.Nishida. A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP. RFC 6675 (Standard), 2012.

[4] J. L. Gil. Performance analysis of the tcp sack-based loss recovery mechanism (rfc 3517) under correlated losses. In *Proceedings of the ACM International Workshop on Performance Monitoring, Measurement, and Evaluation of Heterogeneous Wireless and Wired Networks*, PM2HW2N '06, pages 64–73, New York, NY, USA, 2006. ACM.

[5] V. Jacobson and R. Braden. TCP extensions for long-delay paths. RFC 1072, Oct. 1988. Obsoleted by RFCs 1323, 2018.

[6] S. Ladha, P. Amer, J. Caro, A., and J. Iyengar. On the prevalence and evaluation of recent TCP enhancements. In *Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE*, volume 3, pages 1301–1307 Vol.3, Nov 2004.

[7] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. RFC 2018 (Proposed Standard), Oct. 1996.

[8] T. A. N. Nguyen and J. P. G. Sterbenz. An Implementation of the SACK-Based Conservative Loss Recovery Algorithm for TCP in ns-3. Technical Report ITTC-FY2015-TR-69221-02, Information Telecommunication and Technology Center, University of Kansas, Lawrence, KS, 2015.