TCPM
Internet-Draft
Intended status: Experimental
Expires: September 13, 2021

R. Scheffenegger
NetApp

March 12, 2021

Simple Lost Retransmission Detection with SACK TCP
draft-scheffenegger-tcpm-lrd-00

Abstract

Lost Retransmissions are a major source of latency for TCP transfers. This note specifies how selective acknowledgment (SACK) information can be used to timely recover from lost retransmissions. In addition, it codifies the congestion control reaction on lost retransmissions.

Note to Readers

Discussion of this draft takes place on the TCPM working group mailing list [1], which is archived at <https://mailarchive.ietf.org/arch/browse/tcpm/>.

Working Group information can be found at <https://datatracker.ietf.org/wg/tcpm/>;

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 13, 2021.

Table of Contents

1. Introduction

Selective Acknowledgement (SACK) is widely used to identify exactly which TCP segment was lost and only send these missing segments during a recovery episode. This helps improve the effectiveness of loss recovery and aligns with the principle of packet conservation. In addition, SACK information can also be used to infer about lost retransmissions. When this information is not used, TCP senders revert to the retransmission timeout (RTO) scheme to recover from lost retransmissions.

Current SACK implementations, with one widely deployed exception, do not perform lost retransmission detection. Lost retransmission

detection (LRD) in the one implementation that performs it was described as an emergent feature due to the way the sender is handling SACK. Therefore, LRD is handled in that stack within the current regime of loss recovery, but without any additional congestion control reaction.

This note specifies the use of SACK to detect and recover from lost retransmissions. Using this scheme, a RTO is only required to recover from excessive loss of segments, or ACKs. The intention of this note is to enhance SACK loss recovery so that most RTO events can be mitigated. Only during episodes of pathological network impediments, RTO are still necessary to achieve forward progress.

The mechanism described adheres strictly to the principle of packet conservation. It also requires the use of the forward acknowledgement (FACK) mechanism, described in more detail in [MM96a] and [TLP].

2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Overview

TCP Selective Acknowledgement [RFC2018] was designed to provide detailed information to the sender about the segements already received. Based on this information, a sender can reduce the number of unnecessary retransmissions to close to zero and also recover from a loss of multiple segments within a single round trip time (RTT), and without reverting to a retransmission timeout (RTO).

To that end, [RFC6675] describes the necessary data structures a sender has to maintain to keep track of incoming SACK information. However, no explicit attempt was made to specify how to use the information gained during the recovery episode to detect lost retransmissions.

In addition, [RFC2018] specifically stipulated up to which point a SACK enabled sender may promote segments to become eligible for retransmission under the SACK scheme. This heuristic works very well during bulk transfers, where the sender always has additional data to send. Close to the end of a stream, when there is no more data in the socket to send, current SACK implementations fail to promote still outstanding and never acknowledged segments to become eligible

for retransmission. When this happens, the performance of a TCP SACK implementation adhereing to [RFC3517] degrades and is lower than the performance of TCP NewReno [RFC3782], which can recovery this particular event without an RTO.

The introduction of a rescue retransmission, as described in [RFC6675], addresses this particular issue.

This document is concerned with the behavior of a TCP SACK sender, when after retransmission of all ourstanding segments, and the transmission of new data, the recovery state persists (SND.UNA does not advance to SND.MAX at the time of loss recovery initiation, also known as Recovery Point).

4. Definitions

This document uses the terms SND.UNA, SND.NXT, SND.MAX as defined in [RFC5681].

SND.FACK (forward acknowledgment) is used to describe the highest sequence number that has been SACKed by the receiver and subsequently seen by the sender. The full definition can be found in [MM96a] and [MM96b]. The FACK mechanism is further described in [TLP].

5. Design Considerations

The algorithm described in this document has to adhere to the principle of packet conservation. Detection and recovery from lost retransmissions is plagued with the same set of problems that can become worrysome during regular loss detection and loss recovery. Especially heavy reordering and recovery at the end-of-stream can make it hard to achive good efficiency during loss recovery.

5.1. Recovery Initiation

The algorithm outlined does not speak about the engagement of the loss recovery state by the sender TCP. It is assumed, that the methods outlined in Congestion Control [RFC5681], Early Retransmit [RFC5827] and [SRE], now incorporated into [RFC6675] are used to engage in loss recovery. This leaves only the case where all segments between SND.UNA and SND.MAX are lost to be recovered from by means of retransmission timeout.

5.2. Detection of lost retransmissions

The intuition behind the scheme is that if a retransmission succeeds, then the cumulative ack should increase one round trip time after the retransmission was sent. Otherwise, the retransmission must have

been lost. The key is to have a unambiguous signal which indicates that at least one RTT has passed after a retransmission was sent out.

As long as the sending TCP has still unsent data available, an unabigious signal can be deducted by using the FACK mechanism. After the first round of sending retransmissions, the sender MAY send previously unsent data. Once SND.FACK advances and encompasses this newly sent data, the sender can deduct with high probability, that any still outstanding packets have been dropped by the network. The sender MAY start retransmitting all still outstanding packets. If the sender chooses to do so, it MUST take an appropriate congestion control action. This action is prudent, as the loss of retransmitted packets can be a signal of persistent congestion in the network, that lasts even after the initial congestion control reaction at least one RTT before.

Note that the one popular stack performing LRD already does not react by reducing the congestion window before starting the next cycle of retransmissions. It is therefore more aggressive that the mechanism described herein. Nevertheless, no network instabilities have been reported since that stack started using LRD more than two decades ago.

5.3. Reordering

Without making use of additional information not contained in the SACK entries, only reordered ACKs can be discriminated.

If a single data segment is delayed, and later resent, it is not possible by using only information available within SACK entries to distinguish if the original or retransmitted segment was SACKed. Thus lost retransmission detection can fall victim to reordered data segments, if it were to use retransmitted segments as signal to detemine lost retransmissions.

The use of an SACK acknowledging data that was not sent at the initiation of the recovery episode prevents this issue.

On the return path, reordered ACKs may be recognized, by comparing the SACK entries contained in the ACK. The original ACK from the in-sequence, original transmission does not contain any SACK entries beyond SND.FACK, while the ACK for a retransmitted segment would likely contain SACK blocks of segments higher than the newly SACKed segment.

Also, if an ACK does not contain any newly SACKed segments than already known in the senders scoreboard, ACK reordering is likely to have occured. For example, the SACK entry may contain only a part of

an entry already in the scoreboard. However, such a simple heuristic is not enough to discriminate properly the ACK for a retransmitted data segment from the ACK of the original data segment.

5.4. Ordering of retransmitted segments

There are a number of choices when it comes to deciding which packet to transmit at what time and also in what order. With TCP SACK, the decision of what to send has been decoupled from the decision when (and how much) to send.

In the context of lost retransmission detection, there are at least four broad approaches, each of which has a different figure of merit:

- Stricty enqueue all known lost segments first in the range [SND.UNA ... SND.FACK]. Only when the last enqueued segment has been retransmitted at least once, segments which are found to be still missing may be enqueued for a 2nd cycle, again from the newer [SND.UNA ... SND.FACK]. This is the most conservative approach, and would ensure the least amount of spurious (unnecessary) retransmissions.

- A second approach would be for the sender to re-enqueue an already retransmitted segment as soon as it receives positive proof that at least 3 segments have been received, which were sent after the segment in question. This is the approach choosen by one popular stack. However, it assumes a continous data stream so that at any later time, there will still be enough data segments around that the criteria can be matched for a lost retransmission. The delay on the receiver side, before some new data can be delivered up the stack to the application can be reduced somewhat over option 1. This approach still maintains nearly optimal efficiency and very few spurious retransmissions.

- Third, a slightly more relaxed criteria for detection of lost retransmissions can be applied. As soon as any data segment is positively acknowledged (SACKed), that was sent at least dupthresh segments later, a retransmitted segment can be considered lost. Note that dupthresh is not necessarily constant in this approach, as the same guidelines as defined in Early Retransmit [RFC5827] may be applied once the retransmitted segment closes in on SND.FACK.

- Last, a sender could assume strictly in-sequence delivery of retransmitted segments. During loss recovery, the transmission rate of the sent segments is slower than just prior to the detection of the loss, in particular when PRR [RFC6937] is in use. This may reduce load-induced reordering to some extent. This

approach would allow the most timely delivery of data only blocked by a few lost segments on the receiver side, but would also have the least efficiency in terms of packet conversation.

Furthermore, the senders congestion window might not allow for many re-retransmissions before a stall. Therefore, additional steps would be necessary on the sender side, to ensure continous, paced transmission even after the ACK clock has stopped. This limits the usefulness of this approach, and addressing congestion control and timing related issues are outside the scope of this note. However, this is effectively implemented when using RACK [RFC8985].

6. Algorithm

Section 5 in [RFC2018] seems to have been interpreted as an exlusive list of which segments may become elegible for retransmission, but can also be interpreted as an inclusive list:

After the SACKed bit is turned on (as the result of processing a received SACK option), the data sender will skip that segment during any later retransmission. Any segment that has the SACKed bit turned off and is less than the highest SACKed segment is available for retransmission.

6.1. Lost Retransmission Detection

In order to track if a retransmitted segment might have been lost, the sender requires additional state while in the recovery state.

Once TCP has established that genuine loss exists in the network, it enters loss recovery. At this point, the current value of SND.MAX is stored ("Recover" in NewReno [RFC6582]). Thus it is enough to check if SND.FACK advances beyond "Recover". Once that becomes true, some previously unsent data was acknowledged by the receiver. By that time, any outstanding retransmissions should have been received as well. Thus the sender MAY retransmit the outstanding data from the SACK scoreboard again, after taking appropriate congestion control action (i.e. reducing the congestion window).

The retransmission SHOULD proceed in order of ascending sequence numbers across the unfilled holes of the SACK scoreboard, to maximize the chance that a delayed segment closes still outstanding holes.

Note that implementations tracking sequence-number ranges in their scoreboard only need to track a single sequence number per recovery episode. Multiple cycles of SACK loss recover, without leaving loss recovery in between, are possible by tracking the relevant "Recovery" in the scoreboard data structure.

Implicitly, this rule will also make sure, that all the segments which had become elegible for retransmission will have been sent at least one time, before any additional round of retransmissions is initiated. If the entire flight of data except a small number of segments at the end were lost, it takes at least one RTT for the information about successfully received segments to reach the sender. By that time, the first round of retransmissions is already completed (and additional data segments with sequence numbers higher than SND.MAX at the start of the recovery episode start may have been already been sent.)

In order to guarantee a timely delivery at end-of-stream, a TCP sender implementing LRD SHOULD also make use of the "Rescue Retransmission" as defined in [RFC6675].

6.2. LRD Algorithm Detail

1.: On entering Loss Recovery, store SND.MAX to Recover

2.: After retransmission of the last segment of a hole in the scoreboard, store Recover to Hole.Rxmit

3.: Once SND.FACK advances beyond Recover, while there are holes in the scoreboard:

3.1.: store SND.MAX to Recover

3.2.: perform adequate congestion control reaction (i.e. reduce the congestion window)

3.3.: retransmit each hole in the scoreboard, where Hole.Rxmit < Recover, when appropriate to do so.

7. Security Considerations

The algorithm presented in this paper shares security considerations with [RFC2018] and [RFC6675].

8. IANA Considerations

This document does not require any IANA actions.

9. Acknowledgements

The author would like to thank Matt Mathis for the insightful discussions about SACK and it's intended behavior and the spirit driving the design of SACK.

Dragana Damjanovic was very helpful in reviewing an earlier version of this text and point out numerous clarifications.

Furthermore, valuable feedback was received from John Heffner, Jeff Prem and Anumita Biswas.

10. References

10.1. Normative References

[RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, DOI 10.17487/RFC2018, October 1996, <https://www.rfc-editor.org/info/rfc2018>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <https://www.rfc-editor.org/info/rfc5681>.

[RFC5827] Allman, M., Avrachenkov, K., Ayesta, U., Blanton, J., and P. Hurtig, "Early Retransmit for TCP and Stream Control Transmission Protocol (SCTP)", RFC 5827, DOI 10.17487/RFC5827, May 2010, <https://www.rfc-editor.org/info/rfc5827>.

[RFC6582] Henderson, T., Floyd, S., Gurtov, A., and Y. Nishida, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 6582, DOI 10.17487/RFC6582, April 2012, <https://www.rfc-editor.org/info/rfc6582>.

[RFC6675] Blanton, E., Allman, M., Wang, L., Jarvinen, I., Kojo, M., and Y. Nishida, "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP", RFC 6675, DOI 10.17487/RFC6675, August 2012, <https://www.rfc-editor.org/info/rfc6675>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

10.2. Informative References

[DAC] Beomjoon Kim, . and . Jaiyong Lee, "Retransmission Loss Recovery by Duplicate Acknowledgement Counting", IEEE Communications Letters, vol. 8, no. 1, pp. 69-71 , January 2004.

[LRD] Beomjoon Kim, ., Dongmin Kim, ., and . Jaiyong Lee, "Lost Retransmission Detection for TCP SACK", IEEE Communications Letters, vol. 8, no. 9, pp. 600-602 , September 2004.

[LRD2] Beomjoon Kim, ., Yong-Hoon Choi, ., Jaiyong Lee, ., Min-Seok Oh, ., and . Jin-Sung Choi, "["Lost Retransmission Detection for TCP Part 2", "TCP using SACK option"]", Proceedings of IFIP-TC6 Networking 2004, LNCS 3042, Springer-Verlag, vol. 3042, pp. 88-99 , May 2004.

[LRSF] Hurtig, P, ., Garcia, J, ., and A. Brunstrom, "Loss Recovery in Short TCP/SCTP Flows", Karlstad University Studies 2006:71 , December 2006.

[MM96a] Mathis, M, . and J. Mahdavi, "["Forward Acknowledgment", "Refining TCP Congestion Control"]", Proceedings of SIGCOMM 1996 , August 1996.

[MM96b] Mathis, M, . and J. Mahdavi, "TCP Rate-Halving with Bounding Parameters", September 2004, <http://www.psc.edu/networking/papers/FACKnotes/current>.

[RFC3517] Blanton, E., Allman, M., Fall, K., and L. Wang, "A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP", RFC 3517, DOI 10.17487/RFC3517, April 2003, <https://www.rfc-editor.org/info/rfc3517>.

[RFC3782] Floyd, S., Henderson, T., and A. Gurtov, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 3782, DOI 10.17487/RFC3782, April 2004, <https://www.rfc-editor.org/info/rfc3782>.

[RFC6937] Mathis, M., Dukkipati, N., and Y. Cheng, "Proportional Rate Reduction for TCP", RFC 6937, DOI 10.17487/RFC6937, May 2013, <https://www.rfc-editor.org/info/rfc6937>.

[RFC7323] Borman, D., Braden, B., Jacobson, V., and R. Scheffenegger, Ed., "TCP Extensions for High Performance", RFC 7323, DOI 10.17487/RFC7323, September 2014, <https://www.rfc-editor.org/info/rfc7323>.

[RFC8312] Rhee, I., Xu, L., Ha, S., Zimmermann, A., Eggert, L., and R. Scheffenegger, "CUBIC for Fast Long-Distance Networks", RFC 8312, DOI 10.17487/RFC8312, February 2018, <https://www.rfc-editor.org/info/rfc8312>.

[RFC8985] Cheng, Y., Cardwell, N., Dukkipati, N., and P. Jha, "The RACK-TLP Loss Detection Algorithm for TCP", RFC 8985, DOI 10.17487/RFC8985, February 2021, <https://www.rfc-editor.org/info/rfc8985>.

[sack-recovery-entry] Jarvinen, I, . and M. Kojo, "Using TCP Selective Acknowledgement (SACK) Information to Determine Duplicate Acknowledgements for Loss Recovery Initiation", March 2010, <http://tools.ietf.org/html/draft-ietf-tcpm-sack-recovery-entry-01>.

[SACKPerf] Kodama, Y, ., Takano, R, ., Okazaki, F, ., and T. Kudoh, "Improvement of Communication Performance of Linux TCP/IP by Fixing a Problem in Detection of Loss of Retransmission", March 2008, <http://projects.itri.aist.go.jp/gnet/sack-bug.html>.

[SRE] Jarvinen, I. and M. Kojo, "Using TCP Selective Acknowledgement (SACK) Information to Determine Duplicate Acknowledgements for Loss Recovery Initiation", draft-ietf-tcpm-sack-recovery-entry-01 (work in progress), March 2010.

[TCPLat] Cardwell, N, ., Savage, S, ., and T. Anderson, "Modeling TCP Latency", Proceedings IEEE INFOCOM , March 2000.

[TLP] Dukkipati, N., Cardwell, N., Cheng, Y., and M. Mathis, "Tail Loss Probe (TLP): An Algorithm for Fast Recovery of Tail Losses", draft-dukkipati-tcpm-tcp-loss-probe-01 (work in progress), February 2013.

10.3. URIs

[1] mailto:tcpm@ietf.org

Appendix A. Lost Retransmission Detection Example

The following lengthy graph shows the intended behavior under pathological packet loss, where every third segment is lost. Note that SACK LRD will not be able to recover, if the loss ratio during recovery is higher than about 50%, due to the congestion window reduction.

For clarity, each segment is denoted only via a single number. Note that the ACKs are also given with the segement they ack, not the next sequence number.

A.1. Lost Retransmission, Mid-Stream

ACK Received	Transmitted Segment	Received Segment	ACK Sent (Including SACK Blocks)
1000			
	5000-5499	5000-5499	(delayed ACK)
	5500-5999	5500-5999	
			6000
2000			
	6000-6499	(dropped)	
	6500-6999	(dropped)	
3000			
	7000-7499	(dropped)	
	7500-7999	(dropped)	
4000			
	8000-8499	(dropped)	
	8500-8999	(dropped)	
5000			
	9000-9499	9000-9499	
			6000; 9000-9500
	9500-9999	9500-9999	
			6000; 9000-10000
6000			
	10000-10499	10000-10499	
			6000; 9000-10500
	10500-10999	10000-10999	
			6000; 9000-11000
6000; 9000-9500 (lim. tr.)	11000-11499	11000-11499	
			6000; 9000-11500
6000; 9000-10000 (lim. tr.)	11500-11999	11500-11999	(end-of-stream)
			6000; 9000-12000

```
       6000; 9000-10500
       (fast retr.) 6000-6499    (dropped)   [^1]

       6000; 9000-11000

       6000; 9000-11500
                    6500-6999    6500-6999   [^2]
                                             6000; 6500-7000,9000-12000
       6000; 9000-12000

       6000; 6500-7000,9000-12000
                    7000-7499    7000-7499   [^2]
                                             6000; 6500-7500,9000-12000
       6000; 6500-7500,9000-12000
                    7500-7999    7500-7999   \[mark 8999*\]
                                             6000; 6500-8000,9000-12000
       6000; 6500-8000,9000-12000            (trigger 6000-6499)
                    6000-6499    6000-6499   \[mark 8999*\]
                                             8000; 9000-12000
       8000; 9000-12000
                    8000-8499    8000-8499   \[mark 8999*\]
                                             8500; 9000-12000
       8500; 9000-12000
                    8500-8999    8500-8999   \[mark 12499**\]
                                             12000
       12000                                 (exit loss recovery)
```

Figure 1

Author's Address

Richard Scheffenegger
NetApp
Am Europlatz 2
Vienna 1120
AT

Email: rs.ietf@gmx.at