Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 17, 2011

Y. Nishida
WIDE Project

April 15, 2011

Rescue Retransmission for SACK-based Loss Recovery Algorithm
draft-nishida-tcpm-rescue-retransmission-00

Abstract

This memo describes an issue in the recovery algorithm in RFC3517 and proposes a simple modification to avoid unnecessary timeouts for performance improvement.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

RFC3517 [RFC3517] defines conservative loss recovery algorithm based on the use of the selective acknowledgment (SACK) TCP option [RFC2018]. It is designed to follows the guidelines set in RFC2581 [RFC2581] in order to be used safely in TCP implementations. However, in some situations, the loss recovery algorithm in RFC3517 fails to retransmit segments even though there are available pipe size for the connection. This failure of the retransmission can causes unnecessary timeouts which can lead performance degradation. This document describes the issue and propose a simple modification to solve this problem. The proposed solution allows SACK-based TCP to attain the same performance as NewReno [RFC3782].

2. Conventions and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Problem Description

In RFC3517, when a sender receives the duplicate ACK corresponding to DupThresh ACKs, it enters loss recovery phase. In the loss recovery phase, whenever sender receives ACK segments, it re-calculate the size of pipes by calling Update() and SetPipe(). and determines which segments should be sent by calling NextSeg(). However, there are some situations where NextSeg() returns no segment although the size of pipes is not zero. This behavior results from the following logic in the NextSeg(). When NextSeg() tries to find segments to be retransmitted, it uses the IsLost() that returns segments which are most likely lost. In order to increase the accuracy, IsLost() determines that the packet with 'SeqNum' is lost when DupThresh discontiguous SACKed sequences have arrived above 'SeqNum' or (DupThresh * SMSS) bytes with sequence numbers greater than 'SeqNum' have been SACKed. If IsLost() returns no packet, NextSeg() uses new segments for the next transmission.

In this logic, a problem can arise when a sender does not have new segments to be sent. In this case, if IsLost() returns no packet, NextSeg() cannot find a packet for the next transmission and packet transmissions will be delayed until one of the following events happens.

- ACKs have arrived and IsLost() finds new lost segments

- Application feeds data to TCP

- Retransmission timer expires

However, in some situations, such as where window size is small, the number of arrived ACKs might not be enough to identify lost segments. In addition, applications might feed data intermittently or might not have no more data to feed. In this case, TCP will need timer expiration to retransmit segments even though there are enough pipe size to send a packet.

4. Possible Scenario

This section describe a possible scenario where the issue described in the document happens.

The following is a virtual tcpdump log.

```
  1  10:41:00.000001 A > B: . 1000:2000(1000) ack 1 win 32768
  2  10:41:00.001001 A > B: . 2000:3000(1000) ack 1 win 32768
  3  10:41:00.002001 A > B: . 3000:4000(1000) ack 1 win 32768
  4  10:41:00.003001 A > B: . 4000:5000(1000) ack 1 win 32768
  5  10:41:00.004001 A > B: . 5000:6000(1000) ack 1 win 32768
  6  10:41:00.010001 B > A: . ack 1000 win 16384 < sack {2000:3000} >
  7  10:41:00.011001 B > A: . ack 1000 win 16384 < sack {2000:4000} >
  8  10:41:00.012001 B > A: . ack 1000 win 16384 < sack {2000:5000} >
  9  10:41:00.015001 A > B: . 1000:2000(1000) ack 1 win 32768
 10  10:41:00.018001 B > A: . ack 5000 win 16384
```

In this example, A sends data segments to B. At the beginning of the log, the cwnd of A is 5 SMSS (SMSS=1000 octets), hence A sends 5 segments to B (line 1-5). Here, if the segment sent in line 1 (segment 1000:2000) and line 5 (segment 5000:6000) are lost, B sends 3 duplicated ACKs for the lost segment (line 6-8) to ask retransmission for the segment 1000:2000. At line 8, A receives DupThresh ACKs and retransmits the lost segment (at line 9). At this time, A enters loss recovery phase and set pipe size to 2.5 SMSS. At line 10, A receives the ACK triggered by the arrival of the segment 1000:2000. Upon the reception of the ACK at line 10, A performs the following steps to determine if there are segments can be sent.

1. Update the pipe size by calling update() and SetPipe(). Since HighACK = 5000, HighData is 6000 and IsLost(5000) returns false, the value of pipe is set to 1000.

2. Because cwnd - pipe >= 1 SMSS, it decides to send one or more segments.

3. Call NextSeg() to determine what segments to be sent.

Now, if A has no unsent data, only available packet can be sent is segment 5000:6000. NextSeg() checks if this segment can be sent by applying the following logics, however none of them can be applied.

1. rule (1) cannot be applied to this segment. Because (1.b) and (1.c) return false,

2. rule (2) cannot be applied since there is no available unsent data.

3. rule (3) cannot be applied to this segment. Because (1.b) returns false.

Hence NextSeg() returns no segment in this case, which means TCP has no segment to be sent until timeout happens. In case where there are multiple packet loss in a window and TCP has no data to send at the moment, it will be possible that TCP falls into this situation.

5. Proposed Fix

To solve the problem mentioned above, we propose to introduce one variable: RescueRxt for TCP sender and add the following logic as the fourth rule.

(4) If the conditions for rules (1), (2) and (3) fail, but there exists unSACKed data, one segment of up to SMSS octets MAY be returned if RescueRxt is not set. The returned segment MUST include the highest unSACKed sequence number.

When a segment is returned by this rule, RescueRxt MUST be set to the highest octets of the segment. Also, HighRxt MUST NOT be updated.

In addition to this rule, TCP sender MUST reset RescueRxt when it receives cumulative ACK for a sequence number greater than RescueRxt.

6. Discussion

The simple approach to address this issue is to send unSACKed data when the conditions for rules (1), (2) and (3) failed as long as there is available pipe size. A similar approach is also proposed in [I-D.scheffenegger-tcpm-sack-loss-recovery]. However, this approach can cause lots of unnecessary retransmissions where segments are reordered but not lost.

The proposed fix in the document allows TCP to retransmit one segment per RTT where all available data TCP has is unSACKed and not sure if it is lost. Since the objective of this algorithm is to avoid retransmission timeout and maintain ack clocking, but not to utilize unused pipe, sending one segment per RTT is enough for this purpose. By sending this one packet, the sender TCP will have a good chance to receive additional ACKs from the receiver, which can trigger another retransmissions in the next RTT. The variable RescueRxt ensures that the retransmission by this algorithm happens only once in a RTT. This logic can drastically suppress amount of unnecessary retransmissions in case of reordering.

7. Acknowledgements

The authors gratefully acknowledge Richard Scheffenegger who originally identified the issue described in the document and gave insightful comments. The authors also would like to appreciate Mark Allman and Ethan Blanton for their careful reviewing on the initial idea of the logic and their valuable feedbacks.

8. Security Considerations

This document only propose simple modification in RFC3782. There are no known additional security concerns for this algorithm.

9. IANA Considerations

This document does not create any new registries or modify the rules for any existing registries managed by IANA.

10. References

10.1. Normative References

[RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2581] Allman, M., Paxson, V., and W. Stevens, "TCP Congestion Control", RFC 2581, April 1999.

[RFC3517] Blanton, E., Allman, M., Fall, K., and L. Wang, "A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP", RFC 3517, April 2003.

[RFC3782] Floyd, S., Henderson, T., and A. Gurtov, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 3782, April 2004.

10.2. Informative References

[I-D.scheffenegger-tcpm-sack-loss-recovery]
Scheffenegger, R., "Improving SACK-based loss recovery for TCP", draft-scheffenegger-tcpm-sack-loss-recovery-00 (work in progress), November 2010.

Author's Address

Yoshifumi Nishida
WIDE Project
Endo 5322
Fujisawa, Kanagawa 252-8520
Japan

Email: nishida@wide.ad.jp