P. Hurtig
A. Brunstrom
Karlstad University
A. Petlund
Simula Research Laboratory AS
M. Welzl
University of Oslo
September 17, 2013

TCP and SCTP RTO Restart
draft-ietf-tcpm-rtorestart-01

Abstract

This document describes a modified algorithm for managing the TCP and SCTP retransmission timers that provides faster loss recovery when there is a small amount of outstanding data for a connection. The modification allows the transport to restart its retransmission timer more aggressively in situations where fast retransmit cannot be used. This enables faster loss detection and recovery for connections that are short-lived or application-limited.

1. Introduction

TCP uses two mechanisms to detect segment loss. First, if a segment is not acknowledged within a certain amount of time, a retransmission timeout (RTO) occurs, and the segment is retransmitted [RFC6298]. While the RTO is based on measured round-trip times (RTTs) between the sender and receiver, it also has a conservative lower bound of 1 second to ensure that delayed segments are not mistaken as lost. Second, when a sender receives duplicate acknowledgments, the fast retransmit algorithm infers segment loss and triggers a retransmission [RFC5681]. Duplicate acknowledgments are generated by a receiver when out-of-order segments arrive. As both segment loss and segment reordering cause out-of-order arrival, fast retransmit waits for three duplicate acknowledgments before considering the segment as lost. In some situations, however, the number of outstanding segments is not enough to trigger three duplicate acknowledgments, and the sender must rely on lengthy RTOs for loss recovery.

The number of outstanding segments can be small for several reasons:

(1) The connection is limited by the congestion control when the path has a low total capacity (bandwidth-delay product) or the connection's share of the capacity is small. It is also limited by the congestion control in the first few RTTs of a connection or after an RTO when the available capacity is probed using slow-start.

(2) The connection is limited by the receiver's available buffer space.

(3) The connection is limited by the application if the available capacity of the path is not fully utilized (e.g. interactive applications), or at the end of a transfer.

While the reasons listed above are valid for any flow, the third reason is common for applications that transmit short flows, or use a low transmission rate. Typical examples of applications that produce short flows are web servers. [RJ10] shows that 70% of all web objects, found at the top 500 sites, are too small for fast retransmit to work. [BPS98] shows that about 56% of all

retransmissions sent by a busy web server are sent after RTO expiry. While the experiments were not conducted using SACK [RFC2018], only 4% of the RTO-based retransmissions could have been avoided. Applications have a low transmission rate when data is sent in response to actions, or as a reaction to real life events. Typical examples of such applications are stock trading systems, remote computer operations and online games. What is special about this class of applications is that they are time-dependant, and extra latency can reduce the application service level [P09]. Although such applications may represent a small amount of data sent on the network, a considerable number of flows have such properties and the importance of low latency is high.

The RTO restart approach outlined in this document makes the RTO slightly more aggressive when the number of outstanding segments is small, in an attempt to enable faster loss recovery for all segments while being robust to reordering. While it still conforms to the requirement in [RFC6298] that segments must not be retransmitted earlier than RTO seconds after their original transmission, it could increase the risk of spurious timeout. Spurious timeouts typically degrade the performance of flows with multiple bursts of data, as a burst following a spurious timeout might not fit within the reduced congestion window (cwnd).

While this document focuses on TCP, the described changes are also valid for the Stream Control Transmission Protocol (SCTP) [RFC4960] which has similar loss recovery and congestion control algorithms.

1.1. Requirements Language

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 RFC 2119 [RFC2119].

2. RTO Restart Overview

The RTO management algorithm described in [RFC6298] recommends that the retransmission timer is restarted when an acknowledgment (ACK) that acknowledges new data is received and there is still outstanding data. The restart is conducted to guarantee that unacknowledged segments will be retransmitted after approximately RTO seconds. However, by restarting the timer on each incoming acknowledgment, retransmissions are not typically triggered RTO seconds after their previous transmission but rather RTO seconds after the last ACK arrived. The duration of this extra delay depends on several factors but is in most cases approximately one RTT. Hence, in most situations the time before a retransmission is triggered is equal to "RTO + RTT".

The extra delay can be significant, especially for applications that use a lower RTOmin than the standard of 1 second and/or in environments with high RTTs, e.g. mobile networks. The restart approach is illustrated in Figure 1 where a TCP sender transmits three segments to a receiver. The arrival of the first and second segment triggers a delayed ACK [RFC1122], which restarts the RTO timer at the sender. RTO restart is performed approximately one RTT after the transmission of the third segment. Thus, if the third segment is lost, as indicated in Figure 1, the effective loss detection time is "RTO + RTT" seconds. In some situations, the effective loss detection time becomes even longer. Consider a scenario where only two segments are outstanding. If the second segment is lost, the time to expire the delayed ACK timer will also be included in the effective loss detection time.

```
Sender                                Receiver
              ...
DATA [SEG 1] ----------------------> (ack delayed)
DATA [SEG 2] ----------------------> (send ack)
DATA [SEG 3] ----X         /-------- ACK
(restart RTO)  <----------/
              ...
(RTO expiry)
DATA [SEG 3] ---------------------->
```

Figure 1: RTO restart example

During normal TCP bulk transfer the current RTO restart approach is not a problem. Actually, as long as enough segments arrive at a receiver to enable fast retransmit, RTO-based loss recovery should be avoided. RTOs should only be used as a last resort, as they drastically lower the congestion window compared to fast retransmit. The current approach can therefore be beneficial -- it is described in [EL04] to act as a "safety margin" that compensates for some of the problems that the authors have identified with the standard RTO calculation. Notably, the authors of [EL04] also state that "this safety margin does not exist for highly interactive applications where often only a single packet is in flight."

Although fast retransmit is preferrable there are situations where timeouts are appropriate, or the only choice. For example, if the network is severely congested and no segments arrive, RTO-based recovery should be used. In this situation, the time to recover from the loss(es) will not be the performance bottleneck. Furthermore, for connections that do not utilize enough capacity to enable fast

retransmit, RTO is the only choice. The time needed for loss detection in such scenarios can become a serious performance bottleneck.

3. RTO Restart Algorithm

To enable faster loss recovery for connections that are unable to use fast retransmit, an alternative RTO restart can be used. By resetting the timer to "RTO - T_earliest", where T_earliest is the time elapsed since the earliest outstanding segment was transmitted, retransmissions will always occur after exactly RTO seconds. This approach makes the RTO more aggressive than the standardized approach in [RFC6298] but still conforms to the requirement in [RFC6298] that segments must not be retransmitted earlier than RTO seconds after their original transmission.

This document specifies a sender-only modification to TCP and SCTP which updates step 5.3 in Section 5 of [RFC6298] (and a similar update in Section 6.3.2 of [RFC4960] for SCTP):

When an ACK is received that acknowledges new data:

(1) Set T_earliest = 0.

(2) If the following two conditions hold:

 (a) The number of outstanding segments is less than four.

 (b) There is no unsent data ready for transmission.

 set T_earliest to the time elapsed since the earliest outstanding segment was sent.

(3) Restart the retransmission timer so that it will expire after "RTO - T_earliest" seconds (for the current value of RTO).

The update requires TCP implementations to track the time elapsed since the transmission of the earliest outstanding segment (T_earliest). As the alternative restart is used only when the number of outstanding segments is less than four only four segments need to be tracked. Furthermore, some implementations of TCP (e.g. Linux TCP) already track the transmission times of all segments.

4. Discussion

In this section, we discuss the applicability and a number of issues surrounding the modified RTO restart.

4.1. Applicability

The currently standardized algorithm has been shown to add at least one RTT to the loss recovery process in TCP [LS00] and SCTP [HB08][PBP09]. For applications that have strict timing requirements (e.g. interactive web and gaming) rather than throughput requirements, the modified restart approach could be important because the RTT and also the delayed ACK timer of receivers are often large components of the effective loss recovery time. Measurements in [HB08] have shown that the total transfer time of a lost segment (including the original transmission time and the loss recovery time) can be reduced by 35% using the suggested approach. These results match those presented in [PGH06][PBP09], where the modified restart approach is shown to significantly reduce retransmission latency.

4.2. Spurious Timeouts

This document describes a modified RTO restart behavior that, in some situations, reduces the loss detection time and thereby increases the risk of spurious timeouts. In theory, the retransmission timer has a lower bound of 1 second [RFC6298], which limits the risk of having spurious timeouts. However, in practice most implementations use a significantly lower value. Initial measurements, conducted by the authors, show slight increases in the number of spurious timeouts when such lower values are used. However, further experiments, in different environments and with different types of traffic, are encouraged to quantify such increases more reliably.

Does a slightly increased risk matter? Generally, spurious timeouts have a negative effect on TCP/SCTP performance as the congestion window is reduced to one segment [RFC5681], limiting an application's ability to transmit large amounts of data instantaneously. However, with respect to RTO restart spurious timeouts are only a problem for applications transmitting multiple bursts of data within a single flow. Other types of flows, e.g. long-lived bulk flows, are not affected as the algorithm is only applied when the amount of outstanding segments is less than four and no previously unsent data is available. Furthermore, short-lived and application-limited flows are typically not affected as they are too short to experience the effect of congestion control or have a transmission rate that is quickly attainable.

While a slight increase in spurious timeouts has been observed using the modified RTO restart approach, it is not clear whether the effects of this increase mandate any future algorithmic changes or not -- especially since most modern operating systems already include mechanisms to detect [RFC3522][RFC3708][RFC5682] and resolve [RFC4015] possible problems with spurious retransmissions. Further experimentation is needed to determine this and thereby move this specification from experimental to proposed standard.

5. Related Work

There are several proposals that address the problem of not having enough ACKs for loss recovery. In what follows, we explain why the mechanism described here is complementary to these approaches:

The limited transmit mechanism [RFC3042] allows a TCP sender to transmit a previously unsent segment for each of the first two duplicate acknowledgments. By transmitting new segments, the sender attempts to generate additional duplicate acknowledgments to enable fast retransmit. However, limited transmit does not help if no previously unsent data is ready for transmission or if the receiver has no buffer space. [RFC5827] specifies an early retransmit algorithm to enable fast loss recovery in such situations. By dynamically lowering the number of duplicate acknowledgments needed for fast retransmit (dupthresh), based on the number of outstanding segments, a smaller number of duplicate acknowledgments are needed to trigger a retransmission. In some situations, however, the algorithm is of no use or might not work properly. First, if a single segment is outstanding, and lost, it is impossible to use early retransmit. Second, if ACKs are lost, the early retransmit cannot help. Third, if the network path reorders segments, the algorithm might cause more unnecessary retransmissions than fast retransmit.

Following the fast retransmit mechanism standardized in [RFC5681] this draft assumes a value of 3 for dupthresh. However, by considering a dynamic value for dupthresh a tighter integration with early retransmit (or other experimental algorithms) could also be possible.

Tail Loss Probe [TLP] is a proposal to send up to two "probe segments" when a timer fires which is set to a value smaller than the RTO. A "probe segment" is a new segment if new data is available, else a retransmission. The intention is to compensate for sluggish RTO behavior in situations where the RTO greatly exceeds the RTT, which, according to measurements reported in [TLP], is not uncommon. The Probe timeout (PTO) is normally two RTTs, and a spurious PTO is less risky than a spurious RTO because it would not have the same negative effects (clearing the scoreboard and restarting with slow-

start). In contrast, RTO restart is a small sender-only modification of the RTO management algorithm and does not require an additional timer or the use of SACK.

TLP is applicable in situations where RTO restart does not apply, and it could overrule (yielding a similar general behavior, but with a lower timeout) RTO restart in cases where the number of outstanding segments is smaller than four and no new segments are available for transmission. The PTO has the same inherent restart problems as the RTO timer and could be combined with the modified restart approach to offer more consistent timeouts.

6. Acknowledgements

The authors wish to thank Godred Fairhurst, Yuchung Cheng, Mark Allman, Anantha Ramaiah and Richard Scheffenegger for commenting the draft and the ideas behind it.

All the authors are funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700). The views expressed are solely those of the author(s).

7. IANA Considerations

This memo includes no request to IANA.

8. Security Considerations

This document discusses a change in how to set the retransmission timer's value when restarted. This change does not raise any new security issues with TCP or SCTP.

9. Changes from Previous Versions

9.1. Changes from draft-ietf-....-00 to -01

o Improved the wording throughout the document.

o Removed the possibility for a connection limited by the receiver's advertised window to use RTO restart, decreasing the risk of spurious retransmission timeouts.

o Added a section that discusses the applicability of and problems related to the RTO restart mechanism.

o Updated the text describing the relationship to TLP to reflect updates made in this draft.

o Added acknowledgments.

10. References

10.1. Normative References

[RFC1122] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.

[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.

[RFC3042] Allman, M., Balakrishnan, H., and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit", RFC 3042, January 2001.

[RFC3522] Ludwig, R. and M. Meyer, "The Eifel Detection Algorithm for TCP", RFC 3522, April 2003.

[RFC3708] Blanton, E. and M. Allman, "Using TCP Duplicate Selective Acknowledgement (DSACKs) and Stream Control Transmission Protocol (SCTP) Duplicate Transmission Sequence Numbers (TSNs) to Detect Spurious Retransmissions", RFC 3708, February 2004.

[RFC4015] Ludwig, R. and A. Gurtov, "The Eifel Response Algorithm for TCP", RFC 4015, February 2005.

[RFC4960] Stewart, R., "Stream Control Transmission Protocol", RFC 4960, September 2007.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.

[RFC5682] Sarolahti, P., Kojo, M., Yamamoto, K., and M. Hata, "Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP", RFC 5682, September 2009.

[RFC5827] Allman, M., Avrachenkov, K., Ayesta, U., Blanton, J., and P. Hurtig, "Early Retransmit for TCP and Stream Control Transmission Protocol (SCTP)", RFC 5827, May 2010.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.

10.2. Informative References

[BPS98] Balakrishnan, H., Padmanabhan, V., Seshan, S., Stemm, M., and R. Katz, "TCP Behavior of a Busy Web Server: Analysis and Improvements", Proc. IEEE INFOCOM Conf., March 1998.

[EL04] Ekstroem, H. and R. Ludwig, "The Peak-Hopper: A New End-to-End Retransmission Timer for Reliable Unicast Transport", IEEE INFOCOM 2004, March 2004.

[HB08] Hurtig, P. and A. Brunstrom, "SCTP: designed for timely message delivery?", Springer Telecommunication Systems, May 2010.

[LS00] Ludwig, R. and K. Sklower, "The Eifel retransmission timer", ACM SIGCOMM Comput. Commun. Rev., 30(3), July 2000.

[P09] Petlund, A., "Improving latency for interactive, thin-stream applications over reliable transport", Unipub PhD Thesis, Oct 2009.

[PBP09] Petlund, A., Beskow, P., Pedersen, J., Paaby, E., Griwodz, C., and P. Halvorsen, "Improving SCTP Retransmission Delays for Time-Dependent Thin Streams", Springer Multimedia Tools and Applications, 45(1-3), 2009.

[PGH06] Pedersen, J., Griwodz, C., and P. Halvorsen, "Considerations of SCTP Retransmission Delays for Thin Streams", IEEE LCN 2006, November 2006.

[RJ10] Ramachandran, S., "Web metrics: Size and number of resources", Google http://code.google.com/speed/articles/web-metrics.html, May 2010.

[TLP] Dukkipati, N., Cardwell, N., Cheng, Y., and M. Mathis, "TCP Loss Probe (TLP): An Algorithm for Fast Recovery of Tail Losses", Internet-draft draft-dukkipati-tcpm-tcp-loss-probe-00.txt, July 2012.

Authors' Addresses

Per Hurtig
Karlstad University
Universitetsgatan 2
Karlstad 651 88
Sweden

Phone: +46 54 700 23 35
Email: per.hurtig@kau.se

Anna Brunstrom
Karlstad University
Universitetsgatan 2
Karlstad 651 88
Sweden

Phone: +46 54 700 17 95
Email: anna.brunstrom@kau.se

Andreas Petlund
Simula Research Laboratory AS
P.O. Box 134
Lysaker 1325
Norway

Phone: +47 67 82 82 00
Email: apetlund@simula.no

Michael Welzl
University of Oslo
PO Box 1080 Blindern
Oslo N-0316
Norway

Phone: +47 22 85 24 20
Email: michawe@ifi.uio.no