Internet Engineering Task Force
INTERNET DRAFT
File: draft-allman-tcp-early-rexmt-00.txt

Mark Allman
NASA GRC/BBN
Konstantin Avrachenkov
INRIA
Urtzi Ayesta
France Telecom R&D
Josh Blanton
Ohio University

February, 2003
Expires: August, 2003

Early Retransmit for TCP

Status of this Memo

Abstract

This document proposes a new TCP mechanism that can be used to more effectively recover lost segments when a connection's congestion window is small. The "Early Retransmit" mechanism allows TCP to reduce, in certain special circumstances, the number of duplicate acknowledgments required to trigger a fast retransmission. This allows TCP to use fast retransmit to recover packet losses that would otherwise require a lengthy retransmission timeout.

1 Introduction

A number of researchers have pointed out that TCP's loss recovery strategies do not work well when the congestion window at a TCP sender is small. This can happen in a number of situations, such as:

(1) The TCP connection is "application limited" and has only a limited amount of data to send.

(2) The TCP connection is limited by the receiver-advertised window.

(3) The TCP connection is constrained by end-to-end congestion control when the connection's share of the path is small, the path has a small bandwidth-delay product or TCP is ascertaining the available bandwidth in the first few round-trip times of slow start.

(4) The TCP connection is "winding down" at the end of a transfer such that data is draining from the network but no new data (from the application) is available to transmit.

Many researchers have studied problems with TCP when the congestion window is small and have outlined possible mechanisms to mitigate these problems (e.g., [Mor97,BPS+98,Bal98,LK98,RFC3150,AA02]). When TCP detects a missing segment, the connection enters a loss recovery phase using one of two methods. First, if an acknowledgment (ACK) for a given segment is not received in a certain amount of time a retransmission timeout occurs and the segment is resent [RFC2988]. Second, the ''Fast Retransmit'' algorithm resends a segment when three duplicate ACKs arrive at the sender [Jac88,RFC2581]. However, because duplicate ACKs from the receiver are also triggered by packet reordering in the Internet, the TCP sender waits for three duplicate ACKs in an attempt to disambiguate segment loss from packet reordering. Once in a loss recovery phase, a number of techniques can be used to retransmit lost segments, including slow start based recovery or Fast Recovery [RFC2581], NewReno [RFC2582], and loss recovery based on selective acknowledgments (SACKs) [RFC2018,FF96,BAFW02].

TCP's retransmission timeout (RTO) is based on measured round-trip times (RTT) between the sender and receiver, as specified in [RFC2988]. To prevent spurious retransmissions of segments that are only delayed and not lost, the minimum RTO is conservatively chosen to be 1 second. Therefore, it behooves TCP senders to detect and recover from as many losses as possible without incurring a lengthy timeout during which the connection remains idle. However, if not enough duplicate ACKs arrive from the receiver, the Fast Retransmit algorithm is never triggered---this situation occurs when the congestion window is small, if a large number of segments in a window are lost or at the end of a transfer as data drains from the network. For instance, consider a congestion window (cwnd) of three segments. If one segment is dropped by the network, then at most two duplicate ACKs will arrive at the sender, assuming no ACK loss. Since three duplicate ACKs are required to trigger Fast Retransmit, a timeout will be required to resend the dropped packet.

[BPS+98] shows that roughly 56% of retransmissions sent by a busy web server are sent after the RTO expires, while only 44% are handled by Fast Retransmit. In addition, only 4% of the RTO-based retransmissions could have been avoided with SACK, which has to continue to disambiguate reordering from genuine loss. Furthermore, [All00] shows that for one particular web server the median transfer size is less than four segments, indicating that more than half of

the connections will be forced to rely on the RTO to recover from any losses that occur. Thus, loss recovery without relying on the conservative RTO is beneficial for short TCP transfers. In particular, as a consequence of points (3) and (4) above, a single segment loss will require TCP to RTO when a loss occurs in small transfers.

The Limited Transmit mechanism introduced in [RFC3042] allows a TCP sender to send previously unsent data upon the reception of each of the two duplicate ACKs that precede a fast retransmit. By sending these two new segments the TCP sender is attempting to induce additional duplicate ACKs (if appropriate) so that Fast Retransmit will be triggered before the retransmission timeout expires. The "Early Retransmit" mechanism outlined in this document covers the case when previously unsent data is not available for transmission.

The next section of this document outlines a small change to TCP senders that will decrease the reliance on the retransmission timer, and thereby improve TCP performance when Fast Retransmit would not otherwise be triggered.

2 Reduction of the Retransmission Threshold

Limited Transmit [RFC3042] allows the sender to attempt to induce enough duplicate ACKs to trigger Fast Retransmit. However, in some cases the TCP sender may not have new data queued and ready to be transmitted or may be limited by the advertised window when the first two duplicate ACKs arrive. In these cases, the Limited Transmit algorithm cannot be utilized. If there is a large amount of outstanding data in the network, not being able to transmit new segments when the first two duplicate ACKs arrive is not a problem, as Fast Retransmit will be triggered naturally. However, when the amount of outstanding data is small the sender will have to rely on the RTO to repair any lost segments.

As an example, consider the case when cwnd is three segments and one of these segments is dropped by the network. If the other two segments arrive at the receiver and the corresponding ACKs are not dropped by the network the sender will receive two duplicate ACKs, which is not enough to trigger the Fast Retransmit algorithm. The loss can therefore be repaired only after an RTO. However, the sender has enough information to infer that it cannot expect three duplicate ACKs when one segment is dropped.

The first mitigation of the above problem involves lowering the duplicate ACK threshold when the amount of outstanding data is small and when no unsent data segments are enqueued. In particular, if the amount of outstanding data (ownd) is less than 4 segments and there are no unsent segments ready for transmission at the sender, the duplicate ACK threshold used to trigger Fast Retransmit MAY be reduced to ownd-1 duplicate ACKs (where ownd is in terms of segments). In other words, when ownd is small enough that losing one segment would not trigger Fast Retransmit, we reduce the duplicate ACK threshold to the number of duplicate ACKs expected if

one segment is lost. This mitigation is less robust in the face of reordered segments than the standard Fast Retransmit threshold of three duplicate ACKs. Research shows that a general reduction in the number of duplicate ACKs required to trigger fast retransmission of a segment to two (rather than three) leads to a reduction in the ratio of good to bad retransmits by a factor of three [Pax97]. However, this analysis did not include the additional conditioning on the event that the ownd was smaller than 4 segments.

We note two "worst case" scenarios for Early Retransmit:

(1) Persistent reordering of segments, coupled with an application that does not constantly send data, can result in large numbers of needless retransmissions when using Early Retransmit. For instance, consider an application that sends data two segments at a time, followed by an idle period when no data is queued for delivery by TCP. If the network consistently reorders the two segments, the TCP sender will needlessly retransmit one out of every two unique segments transmitted (and one-third of all segments) when using the above algorithm. However, this would only be a problem for long-lived connections from applications that transmit in spurts.

(2) Similar to the above, consider the case of 2 segment transfers that always experience reordering. Just as in (1) above, one out of every two unique data segments will be retransmitted needlessly, therefore one-third of the traffic will be spurious.

Currently this document offers no suggestion on how to mitigate the above problems. Appendix A offers a survey of possible mitigations. However, the authors would like further input before choosing one of these options (or, deciding that the worst case scenarios listed above are sufficiently rare that Early Retransmit can be used without modification).

3 Related Work

Deployment of Explicit Congestion Notification (ECN) [Flo94,RFC2481] may benefit connections with small congestion window sizes [RFC2884]. ECN provides a method for indicating congestion to the end-host without dropping segments. While some segment drops may still occur, ECN may allow TCP to perform better with small cwnd sizes because the sender will be required to detect less segment loss [RFC2884].

4 Security Considerations

The security considerations found in [RFC2581] apply to this document. No additional security problems have been identified with Early Retransmit at this time.

Acknowledgments

We thank Sally Floyd for her feedback in discussions about Early

Retransmit. We also thank Sally Floyd and Hari Balakrishnan who helped with a large portion of the text of this document when it was part of a seperate effort.

References

[AA02] Urtzi Ayesta, Konstantin Avrachenkov, "The Effect of the Initial Window Size and Limited Transmit Algorithm on the Transient Behavior of TCP Transfers", In Proc. of the 15th ITC Internet Specialist Seminar, Wurzburg, July 2002.

[All00] Mark Allman. A Server-Side View of WWW Characteristics. ACM Computer Communications Review, October 2000.

[AP99] Mark Allman, Vern Paxson. On Estimating End-to-End Network Path Properties. ACM SIGCOMM, September 1999.

[BAFW02] Ethan Blanton, Mark Allman, Kevin Fall, Lili Wang. A Conservative SACK-based Loss Recovery Algorithm for TCP, October 2002. Internet-Draft draft-allman-tcp-sack-13.txt (work in progress).

[Bal98] Hari Balakrishnan. Challenges to Reliable Data Transport over Heterogeneous Wireless Networks. Ph.D. Thesis, University of California at Berkeley, August 1998.

[BPS+98] Hari Balakrishnan, Venkata Padmanabhan, Srinivasan Seshan, Mark Stemm, and Randy Katz. TCP Behavior of a Busy Web Server: Analysis and Improvements. Proc. IEEE INFOCOM Conf., San Francisco, CA, March 1998.

[BPS99] Jon Bennett, Craig Partridge, Nicholas Shectman. Packet Reordering is Not Pathological Network Behavior. IEEE/ACM Transactions on Networking, December 1999.

[FF96] Kevin Fall, Sally Floyd. Simulation-based Comparisons of Tahoe, Reno, and SACK TCP. ACM Computer Communication Review, July 1996.

[Flo94] Sally Floyd. TCP and Explicit Congestion Notification. ACM Computer Communication Review, October 1994.

[Jac88] Van Jacobson. Congestion Avoidance and Control. ACM SIGCOMM 1988.

[LK98] Dong Lin, H.T. Kung. TCP Fast Recovery Strategies: Analysis and Improvements. Proceedings of InfoCom, March 1998.

[Mor97] Robert Morris. TCP Behavior with Many Flows. Proceedings of the Fifth IEEE International Conference on Network Protocols. October 1997.

[Pax97] Vern Paxson. End-to-End Internet Packet Dynamics. ACM SIGCOMM, September 1997.

[SCWA99] Stefan Savage, Neal Cardwell, David Wetherall, Tom Anderson. TCP Congestion Control with a Misbehaving Receiver. ACM Computer Communications Review, October 1999.

[RFC2018] Matt Mathis, Jamshid Mahdavi, Sally Floyd, Allyn Romanow. TCP Selective Acknowledgement Options. RFC 2018, October 1996.

[RFC2481] K. K. Ramakrishnan, Sally Floyd. A Proposal to Add Explicit Congestion Notification (ECN) to IP. RFC 2481, January 1999.

[RFC2581] Mark Allman, Vern Paxson, W. Richard Stevens. TCP Congestion Control. RFC 2581, April 1999.

[RFC2582] Sally Floyd, Tom Henderson. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 2582, April 1999.

[RFC2883] Sally Floyd, Jamshid Mahdavi, Matt Mathis, Matt Podolsky. An Extension to the Selective Acknowledgement (SACK) Option for TCP. RFC 2883, July 2000.

[RFC2884] Jamal Hadi Salim and Uvaiz Ahmed. Performance Evaluation of Explicit Congestion Notification (ECN) in IP Networks. RFC 2884, July 2000.

[RFC2988] Vern Paxson, Mark Allman. Computing TCP's Retransmission Timer. RFC 2988, April 2000.

[RFC3042] Mark Allman, Hari Balakrishnan, Sally Floyd. Enhancing TCP's Loss Recovery Using Limited Transmit. RFC 3042, January 2001.

[RFC3150] Spencer Dawkins, Gabriel Montenegro, Markku Kojo, Vincent Magret. End-to-end Performance Implications of Slow Links. RFC 3150, July 2001.

Author's Addresses:

Mark Allman
NASA Glenn Research Center/BBN Technologies
Lewis Field
21000 Brookpark Rd. MS 54-2
Cleveland, OH 44135
Phone: 216-433-6586
Fax: 216-433-8705
mallman@bbn.com
http://roland.grc.nasa.gov/~mallman

Konstantin Avrachenkov
INRIA
2004 route des Lucioles, B.P.93
06902, Sophia Antipolis
France

Phone: 00 33 492 38 7751
Email: k.avrachenkov@inria.fr

Urtzi Ayesta
France Telecom R&D
905 rue Albert Einstein
06921 Sophia Antipolis
France
Email: Urtzi.Ayesta@francetelecom.com

Josh Blanton
Ohio University
301 Stocker Center
Athens, OH 45701
jblanton@irg.cs.ohiou.edu

Appendix A: Research Issues in Adjusting the Duplicate ACK Threshold

Decreasing the number of duplicate ACKs required to trigger Fast Retransmit, as suggested in section 2, has the drawback of making Fast Retransmit less robust in the face of minor network reordering. Two egregious examples of problems caused by reordering are given in section 2. This appendix outlines several schemes that have been suggested to mitigate the problems caused to Early Retransmit by reordering. These methods need further research before they are suggested for use in shared networks.

One possible mitigation for the damge of spurious retransmits is to allow a TCP connection to only send one retransmission using a duplicate ACK threshold of less than three. This allows for enhanced recovery for short connections and protects the network from longer connections that could possibly use this algorithm to send many needless retransmissions.

Using information provided by the DSACK option [RFC2883], a TCP sender can determine when its Fast Retransmit threshold is too low, causing needless retransmissions due to reordering in the network. Coupling the information provided by DSACKs with the algorithm outlined in section 2 may provide a further enhancement. Specifically, the proposed reduction in the duplicate ACK threshold would not be taken if the network path is known to be reordering segments.

The next method is to detect needless retransmits based on the time between the retransmission and the next ACK received. As outlined in [AP99] if this time is less than half of the minimum RTT observed thus far the retransmission was likely unnecessary. When using less than three duplicate ACKs as the threshold to trigger Fast Retransmit, a TCP sender could attempt to determine whether the retransmission was needed or not. In the case when it was unnecessary, the sender could refrain from further use of Fast Retransmit with a threshold of less than three duplicate ACKs. This method of detecting bad retransmits is not as robust as using DSACKs. However, the advantage is that this mechanism only requires

sender-side implementation changes.

A TCP sender can take measures to avoid the case where a large percentage of the unique segments transmitted are being needlessly retransmitted due to the use of a low duplicate ACK threshold (such as the one outlined in section 2). Specifically, the sender can limit the percentage of retransmits based on a duplicate ACK threshold of less than three. This allows the mechanism to be used throughout a long lived connection, but at the same time protecting the network from potentially wasteful needless retransmissions. However, this solution does not attempt to address the underlying problem, but rather limits the damage the algorithm can cause.

Finally, [Bal98] outlines another solution to the problem of having no new segments to transmit into the network when the first two duplicate ACKs arrive. In response to these duplicate ACKs, a TCP sender transmits zero-byte segments to induce additional duplicate ACKs [Bal98]. This method preserves the robustness of the standard Fast Retransmit algorithm at the cost of injecting segments into the network that do not deliver any data (and, therefore are potentially wasting network resources).

Even with the introduction of the Early Retransmit mechanism, the loss of the last segment of a transfer will lead to a timeout. To overcome this TCP can send an extra segment at the end of the session containing no data. One may expect this would introduce less aditional load than the proposal of [Bal98], but requires more research before such a mechanism can be recommended.