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

Mark Allman
NASA GRC/BBN
Hari Balakrishnan
MIT
Sally Floyd
ACIRI

June, 2000
Expires: December, 2000

Enhancing TCP's Loss Recovery Using Early Duplicate Acknowledgment Response

Status of this Memo

Abstract

This document proposes two new TCP mechanisms that can be used to more effectively recover lost segments when a connection's congestion window is small, or when a large number of segments are lost in a single transmission window. The first of these mechanisms, ``Limited Transmit'', calls for sending a new data segment in response to each of the first two duplicate acknowledgments that arrive at the sender. The second mechanism is to reduce, in certain special circumstances, the number of duplicate acknowledgments required to trigger a fast retransmission.

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, for instance, because there is

only a limited amount of data to send, or because of the limit imposed by the receiver-advertised window, or because of the constraints imposed by end-to-end congestion control over a

connection with a small bandwidth-delay product [Mor97,BPS+98,Bal98,LK98,DMKM00]. When it detects a missing segment, TCP 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 [PA00]. 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].

TCP's retransmission timeout (RTO) is based on measured round-trip times (RTT) between the sender and receiver, as specified in [PA00]. 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 when 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 or if a large number of segments in a window are lost. 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 of course has to continue to disambiguate reordering from genuine loss. In contrast, using the techniques outlined in this document and in [Bal98], 25% of the RTO-based retransmissions in that dataset would have likely been avoided. In addition, [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.

The next two sections of this document outline small changes to TCP senders that will decrease the reliance on the retransmission timer, and thereby improve TCP performance when Fast Retransmit is not triggered. These changes do not adversely affect the performance of TCP nor interact adversely with other connections, in other

circumstances.

2 Modified Response to Duplicate ACKs

When a TCP sender has previously unsent data queued for transmission, new segments SHOULD use the Limited Transmit algorithm, which calls for a TCP sender to transmit new data upon the arrival of a duplicate ACK when the following conditions are satisfied:

* The receiver's advertised window allows the transmission of the segment.

* The amount of outstanding data would remain less than the congestion window plus the duplicate ACK threshold used to trigger Fast Retransmit. In other words, the sender can only send two segments beyond the congestion window (cwnd).

The congestion window (cwnd) MUST NOT be changed when these new segments are transmitted. Assuming that these new segments and the corresponding ACKs are not dropped, this procedure allows the sender to infer loss using the standard Fast Retransmit threshold of three duplicate ACKs [RFC2581]. This is more robust to reordered packets than it would be to retransmit an old packet on the first or second duplicate ACK.

Note: If the connection is using selective acknowledgments [RFC2018], the data sender MUST NOT send new segments in response to duplicate ACKs that contain no new SACK information, as a misbehaving receiver can generate such ACKs to trigger inappropriate transmission of data segments. See [SCWA99] for a discussion of attacks by misbehaving receivers.

Using Limited Transmit follows the ``conservation of packets'' congestion control principle [Jac88]. Each of the first two duplicate ACKs indicate that a segment has left the network. Furthermore, the sender has not yet decided that a segment has been dropped and therefore has no reason to assume the current congestion control state is not accurate. Therefore, transmitting segments does not deviate from the spirit of TCP's congestion control principles.

[BPS99] shows that packet reordering is not a rare network event. [RFC2581] does not provide for sending of data on the first two duplicate ACKs that arrive at the sender. This causes a burst of segments to be sent when an ACK for new data does arrive. Using Limited Transmit, data packets will be clocked out by incoming ACKs and therefore transmission will not be as bursty.

Note: Limited Transmit is implemented in the ns simulator. Researchers wishing to investigate this mechanism further can do so by enabling ``singledup_'' for the given TCP connection.

3 Reduction of the Retransmission Threshold

In some cases the TCP sender may not have new data queued and ready to be transmitted when the first two duplicate ACKs arrive. In this case, the Limited Transmit algorithm outlined in section 2 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 cwnd is small and when no unsent data segments are enqueued. In particular, if cwnd is less than 4 segments and there are no unsent segments at the sender, the duplicate ACK threshold used to trigger Fast Retransmit is reduced to cwnd-1 duplicate ACKs (where cwnd is in terms of segments). In other words, when cwnd 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 clearly 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 cwnd was smaller than 4 segments.

Note that persistent reordering of segments, coupled with an application that does not constantly send data, can result in large numbers of retransmissions. 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 when using the above algorithm. However, this would only be a problem for long-lived connections from applications that transmit in spurts.

To combat this problem, a TCP connection SHOULD 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. We note that

future research may allow this restriction to be relaxed, and refer the reader to Appendix A for a discussion of some alternate mechanisms. While not explicitly recommended by this document, we believe that these may prove useful depending on the results of further research.

4 Related Work

Deployment of Explicit Congestion Notification (ECN) [Flo94,RFC2481] may benefit connections with small congestion window sizes [SA00]. 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 [SA00].

5 Security Considerations

The security implications of the changes proposed in this document are minimal. The potential security issues come from the subversion of end-to-end congestion control from "false" duplicate ACKs, where a "false" duplicate ACK is a duplicate ACK that does not actually acknowledge new data received at the TCP receiver. False duplicate ACKs could result from duplicate ACKs that are themselves duplicated in the network, or from misbehaving TCP receivers that send false duplicate ACKs to subvert end-to-end congestion control [SCWA99,RFC2581].

When the TCP data receiver has agreed to use the SACK option, the TCP data sender has fairly strong protection against false duplicate ACKs. In particular, with SACK, a duplicate ACK that acknowledges new data arriving at the receiver reports the sequence numbers of that new data. Thus, with SACK, the TCP sender can verify that an arriving duplicate ACK acknowledges data that the TCP sender has actually sent, and for which no previous acknowledgment has been received, before sending new data as a result of that acknowledgment. For further protection, the TCP sender could keep a record of packet boundaries for transmitted data packets, and recognize at most one valid acknowledgment for each packet (e.g., the first acknowledgment acknowledging the receipt of all of the sequence numbers in that packet).

One could imagine some limited protection against false duplicate ACKs for a non-SACK TCP connection, where the TCP sender keeps a record of the number of packets transmitted, and recognizes at most one acknowledgment per packet to be used for triggering the sending of new data. However, this accounting of packets transmitted and acknowledged would require additional state and extra complexity at the TCP sender, and does not seem necessary.

The most important protection against false duplicate ACKs comes from the limited potential of duplicate ACKs in subverting end-to-end congestion control. There are two separate cases to consider, when the TCP sender receives less than a threshold number

of duplicate ACKs, and when the TCP sender receives at least a threshold number of duplicate ACKs.

First we consider the case when the TCP sender receives less than a threshold number of duplicate ACKs. For example, the TCP receiver could send two duplicate ACKs after each regular ACK. One might imagine that the TCP sender would send at three times its allowed sending rate. However, using Limited Transmit as outlined in section 2 the sender is only allowed to exceed the congestion window by less than the duplicate ACK threshold, and thus would not send a new packet for each duplicate ACK received.

We next consider the case when the TCP sender receives at least the threshold number of duplicate ACKs. This is an increased possibility with the reduction of the duplicate ACK threshold for the special case proposed in Section 3. However, in addition to retransmitting a packet when a threshold number of duplicate ACKs is received, the TCP sender also halves its congestion window, thus reinforcing the role of end-to-end congestion control. If the retransmitted packet is itself dropped, then it will only be retransmitted again after the retransmit timer expires. Thus, the potential drawback of a reduced threshold is not one of congestion collapse for the network. Instead, the potential drawback would be that of a single unnecessary retransmission, and an accompanying unnecessary reduction of the congestion window, for the TCP connection itself. This is not a security consideration, but a performance consideration for the TCP connection itself. We note that the reduced threshold would only apply when the TCP sender does not have additional data ready to transmit, so the performance penalty would be small.

References

[All00] Mark Allman. A Server-Side View of WWW Characteristics. May, 2000. In preparation.

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

[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.

[DMKM00] Spencer Dawkins, Gabriel Montenegro, Markku Kojo, Vincent

Magret. End-to-end Performance Implications of Slow Links, Internet-Draft draft-ietf-pilc-slow-03.txt, March 2000 (work in progress).

[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.

[FMM+99] Sally Floyd, Jamshid Mahdavi, Matt Mathis, Matt Podolsky, Allyn Romanow, An Extension to the Selective Acknowledgement (SACK) Option for TCP, Internet-Draft draft-floyd-sack-00.txt, August 1999.

[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.

[PA00] Vern Paxson, Mark Allman. Computing TCP's Retransmission Timer, April 2000. Internet-Draft draft-paxson-tcp-rto-01.txt (work in progress).

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

[SA00] Jamal Hadi Salim and Uvaiz Ahmed, Performance Evaluation of Explicit Congestion Notification (ECN) in IP Networks, draft-hadi-jhsua-ecnperf-01.txt, March 2000 (work in progress).

[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.

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@grc.nasa.gov
http://roland.grc.nasa.gov/~mallman

Hari Balakrishnan
Laboratory for Computer Science
545 Technology Square
Massachusetts Institute of Technology
Cambridge, MA 02139
hari@lcs.mit.edu
http://nms.lcs.mit.edu/~hari/

Sally Floyd
AT&T Center for Internet Research at ICSI (ACIRI)
Phone: +1 (510) 666-2989
floyd@aciri.org
http://www.aciri.org/floyd/

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 3, has the drawback of making Fast Retransmit less robust in the face of minor network reordering. As outlined in section 3, this document only allows a TCP to use Fast Retransmit one time when the number of duplicate ACKs is less than three. This appendix suggests several methods by which this restriction may be removed. However, these methods need further research before they are suggested for use in shared networks.

Using information provided by the DSACK option [FMM+99], 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 3 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 a 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 3). 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 just limit 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).