Network Working Group
Request for Comments: 3708
Category: Experimental

E. Blanton
Purdue University
M. Allman
ICIR
February 2004

Using TCP Duplicate Selective Acknowledgement (DSACKs) and Stream Control Transmission Protocol (SCTP) Duplicate Transmission Sequence Numbers (TSNs) to Detect Spurious Retransmissions

Status of this Memo

Copyright Notice

Abstract

TCP and Stream Control Transmission Protocol (SCTP) provide notification of duplicate segment receipt through Duplicate Selective Acknowledgement (DSACKs) and Duplicate Transmission Sequence Number (TSN) notification, respectively. This document presents conservative methods of using this information to identify unnecessary retransmissions for various applications.

1. Introduction

TCP [RFC793] and SCTP [RFC2960] provide notification of duplicate segment receipt through duplicate selective acknowledgment (DSACK) [RFC2883] and Duplicate TSN notifications, respectively. Using this information, a TCP or SCTP sender can generally determine when a retransmission was sent in error. This document presents two methods for using duplicate notifications. The first method is simple and can be used for accounting applications. The second method is a conservative algorithm to disambiguate unnecessary retransmissions from loss events for the purpose of undoing unnecessary congestion control changes.

This document is intended to outline reasonable and safe algorithms for detecting spurious retransmissions and discuss some of the considerations involved. It is not intended to describe the only possible method for achieving the goal, although the guidelines in this document should be taken into consideration when designing alternate algorithms. Additionally, this document does not outline what a TCP or SCTP sender may do after a spurious retransmission is detected. A number of proposals have been developed (e.g., [RFC3522], [SK03], [BDA03]), but it is not yet clear which of these proposals are appropriate. In addition, they all rely on detecting spurious retransmits and so can share the algorithm specified in this document.

Finally, we note that to simplify the text much of the following discussion is in terms of TCP DSACKs, while applying to both TCP and SCTP.

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 RFC 2119 [RFC2119].

2. Counting Duplicate Notifications

For certain applications a straight count of duplicate notifications will suffice. For instance, if a stack simply wants to know (for some reason) the number of spuriously retransmitted segments, counting all duplicate notifications for retransmitted segments should work well. Another application of this strategy is to monitor and adapt transport algorithms so that the transport is not sending large amounts of spurious data into the network. For instance, monitoring duplicate notifications could be used by the Early Retransmit [AAAB03] algorithm to determine whether fast retransmitting [RFC2581] segments with a lower than normal duplicate ACK threshold is working, or if segment reordering is causing spurious retransmits.

More speculatively, duplicate notification has been proposed as an integral part of estimating TCP's total loss rate [AEO03] for the purposes of mitigating the impact of corruption-based losses on transport protocol performance. [EOA03] proposes altering the transport's congestion response to the fraction of losses that are actually due to congestion by requiring the network to provide the corruption-based loss rate and making the transport sender estimate the total loss rate. Duplicate notifications are a key part of estimating the total loss rate accurately [AEO03].

3. Congestion/Duplicate Disambiguation Algorithm

When the purpose of detecting spurious retransmissions is to "undo" unnecessary changes made to the congestion control state, as suggested in [RFC2883], the data sender ideally needs to determine:

(a) That spurious retransmissions in a particular window of data do not mask real segment loss (congestion).

For example, assume segments N and N+1 are retransmitted even though only segment N was dropped by the network (thus, segment N+1 was needlessly retransmitted). When the sender receives the notification that segment N+1 arrived more than once it can conclude that segment N+1 was needlessly resent. However, it cannot conclude that it is appropriate to revert the congestion control state because the window of data contained at least one valid congestion indication (i.e., segment N was lost).

(b) That network duplication is not the cause of the duplicate notification.

Determining whether a duplicate notification is caused by network duplication of a packet or a spurious retransmit is a nearly impossible task in theory. Since [Pax97] shows that packet duplication by the network is rare, the algorithm in this section simply ceases to function when network duplication is detected (by receiving a duplication notification for a segment that was not retransmitted by the sender).

The algorithm specified below gives reasonable, but not complete, protection against both of these cases.

We assume the TCP sender has a data structure to hold selective acknowledgment information (e.g., as outlined in [RFC3517]). The following steps require an extension of such a 'scoreboard' to incorporate a slightly longer history of retransmissions than called for in [RFC3517]. The following steps MUST be taken upon the receipt of each DSACK or duplicate TSN notification:

(A) Check the corresponding sequence range or TSN to determine whether the segment has been retransmitted.

(A.1) If the SACK scoreboard is empty (i.e., the TCP sender has received no SACK information from the receiver) and the left edge of the incoming DSACK is equal to SND.UNA, processing of this DSACK MUST be terminated and the congestion control state MUST NOT be reverted during the current window of data. This clause intends to cover the

case when an entire window of acknowledgments have been dropped by the network. In such a case, the reverse path seems to be in a congested state and so reducing TCP's sending rate is the conservative approach.

(A.2) If the segment was retransmitted exactly one time, mark it as a duplicate.

(A.3) If the segment was retransmitted more than once processing of this DSACK MUST be terminated and the congestion control state MUST NOT be reverted to its previous state during the current window of data.

(A.4) If the segment was not retransmitted the incoming DSACK indicates that the network duplicated the segment in question. Processing of this DSACK MUST be terminated. In addition, the algorithm specified in this document MUST NOT be used for the remainder of the connection, as future DSACK reports may be indicating network duplication rather than unnecessary retransmission. Note that some techniques to further disambiguate network duplication from unnecessary retransmission (e.g., the TCP timestamp option [RFC1323]) may be used to refine the algorithm in this document further. Using such a technique in conjunction with an algorithm similar to the one presented herein may allow for the continued use of the algorithm in the face of duplicated segments. We do not delve into such an algorithm in this document due the current rarity of network duplication. However, future work should include tackling this problem.

(B) Assuming processing is allowed to continue (per the (A) rules), check all retransmitted segments in the previous window of data.

(B.1) If all segments or chunks marked as retransmitted have also been marked as acknowledged and duplicated, we conclude that all retransmissions in the previous window of data were spurious and no loss occurred.

(B.2) If any segment or chunk is still marked as retransmitted but not marked as duplicate, there are outstanding retransmissions that could indicate loss within this window of data. We can make no conclusions based on this particular DSACK/duplicate TSN notification.

In addition to keeping the state mentioned in [RFC3517] (for TCP) and [RFC2960] (for SCTP), an implementation of this algorithm must track

all sequence numbers or TSNs that have been acknowledged as duplicates.

4. Related Work

In addition to the mechanism for detecting spurious retransmits outlined in this document, several other proposals for finding needless retransmits have been developed.

[BA02] uses the algorithm outlined in this document as the basis for investigating several methods to make TCP more robust to reordered packets.

The Eifel detection algorithm [RFC3522] uses the TCP timestamp option [RFC1323] to determine whether the ACK for a given retransmit is for the original transmission or a retransmission. More generally, [LK00] outlines the benefits of detecting spurious retransmits and reverting from needless congestion control changes using the timestamp-based scheme or a mechanism that uses a "retransmit bit" to flag retransmits (and ACKs of retransmits). The Eifel detection algorithm can detect spurious retransmits more rapidly than a DSACK-based scheme. However, the tradeoff is that the overhead of the 12-byte timestamp option must be incurred in every packet transmitted for Eifel to function.

The F-RTO scheme [SK03] slightly alters TCP's sending pattern immediately following a retransmission timeout and then observes the pattern of the returning ACKs. This pattern can indicate whether the retransmitted segment was needed. The advantage of F-RTO is that the algorithm only needs to be implemented on the sender side of the TCP connection and that nothing extra needs to cross the network (e.g., DSACKs, timestamps, special flags, etc.). The downside is that the algorithm is a heuristic that can be confused by network pathologies (e.g., duplication or reordering of key packets). Finally, note that F-RTO only works for spurious retransmits triggered by the transport's retransmission timer.

Finally, [AP99] briefly investigates using the time between retransmitting a segment via the retransmission timeout and the arrival of the next ACK as an indicator of whether the retransmit was needed. The scheme compares this time delta with a fraction (f) of the minimum RTT observed thus far on the connection. If the time delta is less than f*minRTT then the retransmit is labeled spurious. When f=1/2 the algorithm identifies roughly 59% of the needless retransmission timeouts and identifies needed retransmits only 2.5% of the time. As with F-RTO, this scheme only detects spurious retransmits sent by the transport's retransmission timer.

5. Security Considerations

It is possible for the receiver to falsely indicate spurious retransmissions in the case of actual loss, potentially causing a TCP or SCTP sender to inaccurately conclude that no loss took place (and possibly cause inappropriate changes to the senders congestion control state).

Consider the following scenario: A receiver watches every segment or chunk that arrives and acknowledges any segment that arrives out of order by more than some threshold amount as a duplicate, assuming that it is a retransmission. A sender using the above algorithm will assume that the retransmission was spurious.

The ECN nonce sum proposal [RFC3540] could possibly help mitigate the ability of the receiver to hide real losses from the sender with modest extension. In the common case of receiving an original transmission and a spurious retransmit a receiver will have received the nonce from the original transmission and therefore can "prove" to the sender that the duplication notification is valid. In the case when the receiver did not receive the original and is trying to improperly induce the sender into transmitting at an inappropriately high rate, the receiver will not know the ECN nonce from the original segment and therefore will probabilistically not be able to fool the sender for long. [RFC3540] calls for disabling nonce sums on duplicate ACKs, which means that the nonce sum is not directly suitable for use as a mitigation to the problem of receivers lying about DSACK information. However, future efforts may be able to use [RFC3540] as a starting point for building protection should it be needed.

6. Acknowledgments

Sourabh Ladha and Reiner Ludwig made several useful comments on an earlier version of this document. The second author thanks BBN Technologies and NASA's Glenn Research Center for supporting this work.

7. References

7.1. Normative References

[RFC793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

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

[RFC2883] Floyd, S., Mahdavi, J., Mathis, M. and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP", RFC 2883, July 2000.

[RFC2960] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L. and V. Paxson, "Stream Control Transmission Protocol", RFC 2960, October 2000.

7.2. Informative References

[AAAB03] Allman, M., Avrachenkov, K., Ayesta, U. and J. Blanton, "Early Retransmit for TCP", Work in Progress, June 2003.

[AEO03] Allman, M., Eddy, E. and S. Ostermann, "Estimating Loss Rates With TCP", Work in Progress, August 2003.

[AP99] Allman, M. and V. Paxson, "On Estimating End-to-End Network Path Properties", SIGCOMM 99.

[BA02] Blanton, E. and M. Allman. On Making TCP More Robust to Packet Reordering. ACM Computer Communication Review, 32(1), January 2002.

[BDA03] Blanton, E., Dimond, R. and M. Allman, "Practices for TCP Senders in the Face of Segment Reordering", Work in Progress, February 2003.

[EOA03] Eddy, W., Ostermann, S. and M. Allman, "New Techniques for Making Transport Protocols Robust to Corruption-Based Loss", Work in Progress, July 2003.

[LK00] R. Ludwig, R. H. Katz. The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions. ACM Computer Communication Review, 30(1), January 2000.

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

[RFC1323] Jacobson, V., Braden, R. and D. Borman, "TCP Extensions for High Performance", RFC 1323, May 1992.

[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.

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

[RFC3540] Spring, N., Wetherall, D. and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces", RFC 3540, June 2003.

[SK03] Sarolahti, P. and M. Kojo, "F-RTO: An Algorithm for Detecting Spurious Retransmission Timeouts with TCP and SCTP", Work in Progress, June 2003.

8. Authors' Addresses

Ethan Blanton
Purdue University Computer Sciences
1398 Computer Science Building
West Lafayette, IN 47907

EMail: eblanton@cs.purdue.edu

Mark Allman
ICSI Center for Internet Research
1947 Center Street, Suite 600
Berkeley, CA 94704-1198
Phone: 216-243-7361

EMail: mallman@icir.org
http://www.icir.org/mallman/

9. Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.