Internet Engineering Task Force
INTERNET DRAFT
File: draft-allman-tcp-sack-05.txt

Ethan Blanton
Ohio University
Mark Allman
BBN/NASA GRC

June, 2001
Expires: December, 2001

A Conservative SACK-based Loss Recovery Algorithm for TCP

Status of this Memo

Abstract

This document presents a conservative loss recovery algorithm for TCP that is based on the use of the selective acknowledgment TCP option. The algorithm presented in this document conforms to the spirit of the current congestion control specification, but allows TCP senders to recover more effectively when multiple segments are lost from a single flight of data.

1 Introduction

This document presents a conservative loss recovery algorithm for TCP that is based on the use of the selective acknowledgment TCP option. While the TCP selective acknowledgment (SACK) option [RFC2018] is being steadily deployed in the Internet [All00] there is evidence that hosts are not using the SACK information when making retransmission and congestion control decisions [PF00]. The goal of this document is to outline one straightforward method for TCP implementations to use SACK information to increase performance.

[RFC2581] allows advanced loss recovery algorithms to be used by TCP

[RFC793] provided that they follow the spirit of TCP's congestion control algorithms [RFC2581,RFC2914]. [RFC2582] outlines one such

advanced recovery algorithm called NewReno. This document outlines a loss recovery algorithm that uses the selective acknowledgment (SACK) [RFC2018] TCP option to enhance TCP's loss recovery. The algorithm outlined in this document, heavily based on the algorithm detailed in [FF96], is a conservative replacement of the fast recovery algorithm [Jac90,RFC2581]. The algorithm specified in this document is a straightforward SACK-based loss recovery strategy that follows the guidelines set in [RFC2581] and can safely be used in TCP implementations. Alternate SACK-based loss recovery methods can be used in TCP as implementers see fit (as long as the alternate algorithms follow the guidelines provided in [RFC2581]). Please note, however, that the SACK-based decisions in this document (such as what segments are to be sent at what time) are largely decoupled from the congestion control algorithms, and as such can be treated as separate issues if so desired.

2 Definitions

The reader is expected to be familiar with the definitions given in [RFC2581].

For the purposes of explaining the SACK-based loss recovery algorithm we define two variables that a TCP sender stores:

``HighACK'' is the sequence number of the highest cumulative ACK received at a given point.

``HighData'' is the highest sequence number transmitted at a given point.

For the purposes of this specification we define a ``duplicate acknowledgment'' as an acknowledgment (ACK) whose cumulative ACK number is equal to the current value of HighACK, as described in [RFC2581].

We define a variable ``DupThresh'' that holds the number of duplicate acknowledgments required to trigger a retransmission. Per [RFC2581] this threshold is defined to be 3 duplicate acknowledgments. However, implementers should consult any updates to [RFC2581] to determine the current value for DupThresh (or method for determining its value).

3 Keeping Track of SACK Information

For a TCP sender to implement the algorithm defined in the next section it must keep a data structure to store incoming selective acknowledgment information on a per connection basis. Such a data structure is commonly called the ``scoreboard''. For the purposes of the algorithm defined in this document the scoreboard MUST implement the following functions:

Update ():

Each octet that is cumulatively ACKed or SACKed should be marked

accordingly in the scoreboard data structure, and the total number of octets SACKed should be recorded. For each octet that has not been either cumulatively acknowledged or SACKed, a ``DupSACK'' counter is kept and incremented for each SACK block which newly SACKs an octet of greater sequence number.

Note: SACK information is advisory and therefore SACKed data MUST NOT be removed from TCP's retransmission buffer until the data is cumulatively acknowledged [RFC2018].

MarkRetran ():

When a retransmission is sent, the scoreboard MUST be updated with this information so that data is not repeatedly retransmitted by the SACK-based algorithm outlined in this document. Note: If a retransmission is lost it will be repaired using TCP's retransmission timer.

NextSeg ():

This routine MUST return the sequence number range of the oldest segment that has not been cumulatively ACKed or SACKed and has not been retransmitted, per the following rules:

(1) Look for the lowest sequence number that is not ACKed or SACKed, but has a DupSACK counter of at least DupThresh. If such a sequence number ``S'' exists, this routine MUST return a sequence number range starting at octet S.

(2) If we fail to find a segment per rule 1, but the connection has unsent data available to be transmitted, NextSeg () MUST return a sequence number range corresponding to one segment of this new data.

(3) If rules 1 and 2 fail, this routine MUST return a segment that has not been ACKed or SACKed but may not meet the DupThresh requirement in 1.

(4) Finally, if rules 1-3 fail, NextSeg () MUST indicate this and no data will be sent.

AmountSACKed ():

This routine MUST return the total number of octets which fall between HighACK and HighData that have been selectively acknowledged by the receiver.

LeftNetwork ():

This function MUST return the number of octets in the given

sequence number range that have left the network. The algorithm checks each octet in the given range and separately keeps track of the number of retransmitted octets and the number of octets that are cumulatively ACKed but were not SACKed whose DupSACK

counter is less than DupThresh. Note: it is possible to have octets that fit both categories. In this case, the octets MUST be counted in both categories. After checking the sequence number range given, this routine returns the sum of the two counters.

Note: The SACK-based loss recovery algorithm outlined in this document requires more computational resources than previous TCP loss recovery strategies. However, we believe the scoreboard data structure can be implemented in a reasonably efficient manner (both in terms of computation complexity and memory usage) in most TCP implementations.

4 Algorithm Details

Upon the receipt of any ACK containing SACK information, the scoreboard MUST be updated via the Update () routine.

Upon the receipt of the first (DupThresh - 1) duplicate ACKs, the scoreboard is also to be updated as normal. Note: The first and second duplicate ACKs can also be used to trigger the transmission of previously unsent segments using the Limited Transmit mechanism [RFC3042].

When a TCP sender receives the duplicate ACK corresponding to DupThresh ACKs, the scoreboard MUST be updated with the new SACK information (via Update ()) and a loss recovery phase SHOULD be initiated, per the fast retransmit algorithm outlined in [RFC2581], and the following steps MUST be taken:

(1) Set a ``pipe'' variable to the number of outstanding octets (i.e., octets that have been sent but not yet acknowledged), per the following equation:

    pipe = HighData - HighACK - AmountSACKed ()

    This variable represents the amount of data currently ``in the pipe''; this is the data which has been sent by the TCP sender but not acknowledged by the TCP receiver. This data can be assumed to still be traversing the network path.

(2) Set a ``RecoveryPoint'' variable to HighData. When the TCP sender receives a cumulative ACK for this data octet the loss recovery phase is terminated.

(3) The congestion window (cwnd) is reduced to half of FlightSize per [RFC2581]. The value of the slow start threshold (ssthresh) is set to the halved value of cwnd.

(4) Retransmit the first data segment not covered by HighACK. Use

the MarkRetran () function to mark the sequence number range as having been retransmitted in the scoreboard. In order to take advantage of potential additional available cwnd, proceed to step (D) below.

Once a TCP is in the loss recovery phase the following procedure MUST be used for each arriving ACK:

(A) An incoming cumulative ACK for a sequence number greater than or equal to RecoveryPoint signals the end of loss recovery and the loss recovery phase MUST be terminated. The scoreboard SHOULD NOT be cleared when leaving the loss recovery phase.

(B) Upon receipt of a duplicate ACK the following actions MUST be taken:

(B.1) Use Update () to record the new SACK information conveyed by the incoming ACK.

(B.2) The pipe variable is decremented by the number of newly SACKed data octets conveyed in the incoming ACK plus the number of octets whose DupSACK counter exceeded DupThresh, as that is the amount of new data presumed to have left the network.

(C) When a ``partial ACK'' (an ACK that increases the HighACK point, but does not terminate loss recovery) arrives, the following actions MUST be performed:

(C.1) Before updating HighACK based on the received cumulative ACK, save HighACK as OldHighACK.

(C.2) The scoreboard MUST be updated based on the cumulative ACK and any new SACK information that is included in the ACK via the Update () routine.

(C.3) The value of pipe MUST be decremented by the number of octets returned by the LeftNetwork () routine when given the sequence number range OldHighACK-HighACK.

(D) While pipe is less than cwnd and the receiver's advertised window permits, the TCP sender SHOULD transmit one or more segments as follows:

(D.1) The scoreboard MUST be queried via NextSeg () for the sequence number range of the next segment to transmit, and the given segment is sent.

(D.2) The pipe variable MUST be incremented by the number of data octets sent in (D.1).

(D.3) If any of the data octets sent in (D.1) are below HighData, they MUST be marked as retransmitted via Update ().

(D.4) If cwnd - pipe is greater than 1 SMSS, return to (D.1)

4.1 Retransmission Timeouts

Keeping track of SACK information depends on the TCP sender having an accurate measure of the current state of the network, the conditions of this connection, and the state of the receiver's buffer. Due to these limitations, [RFC2018] suggests that a TCP sender SHOULD expunge the SACK information gathered from a receiver upon a retransmission timeout ``since the timeout might indicate that the data receiver has reneged.'' Additionally, a TCP sender MUST ``ignore prior SACK information in determining which data to retransmit.'' However, a SACK TCP sender SHOULD still use all SACK information made available during the slow start phase of loss recovery following an RTO.

As described in Sections 3 and 4, Update () and MarkRetran () SHOULD continue to be used appropriately upon receipt of ACKs and retransmissions, respectively. This will allow the slow start recovery period to benefit from all available information provided by the receiver, despite the fact that SACK information was expunged due to the RTO.

If there are segments missing from the receiver's buffer following processing of the retransmitted segment, the corresponding ACK will contain SACK information. In this case, a TCP sender SHOULD use this SACK information by using the NextSeg () routine to determine what data should be sent in each segment of the slow start.

5 Research

The algorithm specified in this document is analyzed in [FF96], which shows that the above algorithm is effective in reducing transfer time over standard TCP Reno [RFC2581] when multiple segments are dropped from a window of data (especially as the number of drops increases). [AHKO97] shows that the algorithm defined in this document can greatly improve throughput in connections traversing satellite channels.

6 Security Considerations

The algorithm presented in this paper shares security considerations with [RFC2581]. A key difference is that an algorithm based on SACKs is more robust against attackers forging duplicate ACKs to force the TCP sender to reduce cwnd. With SACKs, TCP senders have an additional check on whether or not a particular ACK is legitimate. While not fool-proof, SACK does provide some amount of protection in this area.

Acknowledgments

The authors wish to thank Sally Floyd for encouraging this document and commenting on an early draft. The algorithm described in this

document is largely based on an algorithm outlined by Kevin Fall and Sally Floyd in [FF96], although the authors of this document assume responsibility for any mistakes in the above text. Murali Bashyam, Jamshid Mahdavi, Matt Mathis, Vern Paxson, Venkat Venkatsubra, Reiner Ludwig and Shawn Ostermann provided valuable feedback on

earlier versions of this document. Finally, we thank Matt Mathis and Jamshid Mahdavi for implementing the scoreboard in ns and hence guiding our thinking in keeping track of SACK state.

References

[AHKO97] Mark Allman, Chris Hayes, Hans Kruse, Shawn Ostermann. TCP Performance Over Satellite Links. Proceedings of the Fifth International Conference on Telecommunications Systems, Nashville, TN, March, 1997.

[All00] Mark Allman. A Web Server's View of the Transport Layer. ACM Computer Communication Review, 30(5), October 2000.

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

[Jac90] Van Jacobson. Modified TCP Congestion Avoidance Algorithm. Technical Report, LBL, April 1990.

[PF00] Jitendra Padhye, Sally Floyd. TBIT, the TCP Behavior Inference Tool, October 2000. http://www.aciri.org/tbit/.

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

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

[RFC2026] Scott Bradner. The Internet Standards Process -- Revision 3, RFC 2026, October 1996

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

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

[RFC2914] Sally Floyd. Congestion Control Principles, RFC 2914, September 2000.

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

Author's Addresses:

Ethan Blanton
Ohio University Internetworking Research Lab
Stocker Center
Athens, OH 45701

eblanton@irg.cs.ohiou.edu

Mark Allman
BBN Technologies/NASA Glenn Research Center
Lewis Field
21000 Brookpark Rd. MS 54-5
Cleveland, OH 44135
Phone: 216-433-6586
Fax: 216-433-8705
mallman@bbn.com
http://roland.grc.nasa.gov/~mallman