Internet Engineering Task Force
INTERNET DRAFT
File: draft-allman-tcp-sack-11.txt

Ethan Blanton
Ohio University
Mark Allman
BBN/NASA GRC
Kevin Fall
Intel Research

July, 2002
Expires: January, 2003

# 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 [RFC2581], but allows TCP senders to recover more effectively when multiple segments are lost from a single flight of data.

## 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].

## 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 [PF01]. 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].

The reader is assumed to be familiar with selective acknowledgments as specified in [RFC2018].

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

``HighACK'' is the sequence number of the highest byte of data that has been cumulatively ACKed at a given point.

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

``HighRxt'' is the highest sequence number which has been retransmitted during the current loss recovery phase.

``Pipe'' is a sender's estimate of the number of bytes outstanding in the network. This is used during recovery for limiting the sender's sending rate. The pipe variable allows TCP to use a fundamentally different congestion control than specified in [RFC2581]. The algorithm is often referred to as the ``pipe algorithm''.

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).

Finally, a range of sequence numbers [A,B] is said to ``cover'' sequence number S if A <= S <= B.

## 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''. The specifics of the scoreboard data structure are out of scope for this document (as long as the implementation can perform all functions required by this specification).

Note that while this document speaks of marking and keeping track of octets, a real world implementation would probably want to keep track of octet ranges or otherwise collapse the data while ensuring that arbitrary ranges are still markable.

## 4 Processing and Acting Upon SACK Information

For the purposes of the algorithm defined in this document the

scoreboard SHOULD implement the following functions:

Update ():

Given the information provided in an ACK, 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.

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].

LeftNetwork ():

This routine traverses the scoreboard from HighACK to HighData and returns the total number of octets that have left the network. Octets are considered to have left the network when a SACK has arrived that covers the octet or when the sender has determined that the octet has been dropped by the network.

The sender considers an octet 'S1' to have been dropped in the network (and not replaced with a retransmission) when the following conditions are met:





(a) 'S1' has not been ACKed or SACKed.

(b) 'S1' is greater than HighRxt.

(c) 'S1' is less than the highest octet covered by any received SACK.

(d) Either DupThresh discontiguous SACKed sequences have arrived above 'S1' or DupThresh * SMSS bytes with sequence numbers greater than 'S1' have been SACKed.

SetPipe ():

This routine MUST set the ``pipe'' variable to an estimate of the number of octets that are currently in transit between the TCP sender and the TCP receiver using the following equation:

pipe = HighData - HighACK - LeftNetwork ()

NextSeg ():

This routine uses the scoreboard data structure maintained by the Update() function to determine what to transmit based on the SACK information that has arrived from the data receiver (and hence been marked in the scoreboard). NextSeg () MUST return the sequence number range of the next segment that is to be transmitted, per the following rules:

(1) If there exists a smallest unSACKed sequence number 'S2' that meets the criteria for determining loss given in steps (a)-(d) in LeftNetwork () above the sequence range of one segment of up to SMSS octets starting with S2 MUST be returned.

(2) If no sequence number 'S2' per rule (1) exists but there exists available unsent data and the receiver's advertised window allows, the sequence range of one segment of up to SMSS octets of previously unsent data starting with sequence number HighData+1 MUST be returned.

(3) If the conditions for rules (1) and (2) fail, but there exists an unSACKed sequence number 'S3' that meets the criteria for detecting loss given in steps (a)-(c) in LeftNetwork () (specifically excluding step (d)) then one segment of up to SMSS octets starting with S2 MUST be returned.

(4) If the conditions for each of (1), (2), and (3) are not met, then NextSeg () MUST indicate failure, and no segment is returned.

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.

## 5 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 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 algorithm [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 in doing so the following steps MUST be taken:

(1) RecoveryPoint = HighData

When the TCP sender receives a cumulative ACK for this data octet the loss recovery phase is terminated.

(2) ssthresh = cwnd = (FlightSize / 2)

The congestion window (cwnd) and slow start threshold (sstrhesh) are reduced to half of FlightSize per [RFC2581].

(3) Retransmit the first data segment presumed dropped -- the segment starting with sequence number HighACK + 1. To prevent repeated retransmission of the same data, set HighRxt to the highest sequence number in the retransmitted segment.

(4) Run SetPipe

Set a ``pipe'' variable to the number of outstanding octets currently ``in the pipe''; this is the data which has been sent by the TCP sender but for which no cumulative or selective acknowledgment has been received and the data has not been determined to have been dropped in the network. This data is assumed to be still traversing the network path.

(5) In order to take advantage of potential additional available cwnd, proceed to step (C) 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

RecoveryPoint signals the end of loss recovery and the loss recovery phase MUST be terminated. Any information contained in the scoreboard for sequence numbers greater than the new value of HighACK SHOULD NOT be cleared when leaving the loss recovery phase.

(B) Upon receipt of an ACK that does not cover RecoveryPoint the following actions MUST be taken:

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

(B.2) Use SetPipe () to re-calculate the number of octets still in the network.

(C) If cwnd - pipe >= 1 SMSS the sender SHOULD transmit one or more segments as follows:

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

(C.2) If any of the data octets sent in (C.1) are below HighData, HighRxt MUST be set to the highest sequence number of the segment retransmitted.

(C.3) If any of the data octets sent in (C.1) are above HighData, HighData must be updated to reflect the transmission of previously unsent data.

(C.4) The estimate of the amount of data outstanding in the network must be updated by incrementing pipe by the number of octets transmitted in (C.1).

(C.5) If cwnd - pipe >= 1 SMSS, return to (C.1)

## 5.1 Retransmission Timeouts

In order to avoid memory deadlocks, the TCP receiver is allowed to discard data that has already been acknowledged with a selective acknowledgment. As a result [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 4 and 5, Update () MAY continue to be

used appropriately upon receipt of ACKs. 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.

## 6 Managing the RTO Timer

The standard TCP RTO estimator is defined in [RFC2988]. Due to the fact that the SACK algorithm in this document can have an impact on the behavior of the estimator, implementers may wish to consider how the timer is managed. [RFC2988] calls for the RTO timer to be re-armed each time an ACK arrives that advances the cumulative ACK point. Because the algorithm presented in this document can keep the ACK clock going through a fairly significant loss event, (comparatively longer than the algorithm described in [RFC2581]), on some networks the loss event could last longer than the RTO. In this case the RTO timer would expire prematurely and a segment that need not be retransmitted would be resent.

Therefore we give implementers the latitude to use the standard [RFC2988] style RTO management or, optionally, a more careful variant that re-arms the RTO timer on each retransmission that is sent during recovery MAY be used. This provides a more conservative timer than specified in [RFC2988], and so may not always be an attractive alternative. However, in some cases it may prevent needless retransmissions, go-back-N transmission and further reduction of the congestion window.

## 7 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.

## 8 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 early drafts. 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, Ken Calvert, Reiner Ludwig, Jamshid Mahdavi, Matt Mathis, Shawn Ostermann, Vern Paxson, Venkat Venkatsubra and Lili Wang 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.

## Normative References

[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

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

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

## Non-Normative 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.

[PF01] Jitendra Padhye, Sally Floyd. Identifying the TCP Behavior of Web Servers, ACM SIGCOMM, August 2001.

[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.



---



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

Kevin Fall
Intel Research
2150 Shattuck Ave., PH Suite
Berkeley, CA 94704
kfall@intel-research.net