Network Working Group
Request for Comments: 2018
Category: Standards Track

M. Mathis
J. Mahdavi
PSC
S. Floyd
LBNL
A. Romanow
Sun Microsystems
October 1996

TCP Selective Acknowledgment Options

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

TCP may experience poor performance when multiple packets are lost from one window of data. With the limited information available from cumulative acknowledgments, a TCP sender can only learn about a single lost packet per round trip time. An aggressive sender could choose to retransmit packets early, but such retransmitted segments may have already been successfully received.

A Selective Acknowledgment (SACK) mechanism, combined with a selective repeat retransmission policy, can help to overcome these limitations. The receiving TCP sends back SACK packets to the sender informing the sender of data that has been received. The sender can then retransmit only the missing data segments.

This memo proposes an implementation of SACK and discusses its performance and related issues.

Acknowledgements

Much of the text in this document is taken directly from RFC1072 "TCP Extensions for Long-Delay Paths" by Bob Braden and Van Jacobson. The authors would like to thank Kevin Fall (LBNL), Christian Huitema (INRIA), Van Jacobson (LBNL), Greg Miller (MITRE), Greg Minshall (Ipsilon), Lixia Zhang (XEROX PARC and UCLA), Dave Borman (BSDI), Allison Mankin (ISI) and others for their review and constructive comments.

1. Introduction

Multiple packet losses from a window of data can have a catastrophic effect on TCP throughput. TCP [Postel81] uses a cumulative acknowledgment scheme in which received segments that are not at the left edge of the receive window are not acknowledged. This forces the sender to either wait a roundtrip time to find out about each lost packet, or to unnecessarily retransmit segments which have been correctly received [Fall95]. With the cumulative acknowledgment scheme, multiple dropped segments generally cause TCP to lose its ACK-based clock, reducing overall throughput.

Selective Acknowledgment (SACK) is a strategy which corrects this behavior in the face of multiple dropped segments. With selective acknowledgments, the data receiver can inform the sender about all segments that have arrived successfully, so the sender need retransmit only the segments that have actually been lost.

Several transport protocols, including NETBLT [Clark87], XTP [Strayer92], RDP [Velten84], NADIR [Huitema81], and VMTP [Cheriton88] have used selective acknowledgment. There is some empirical evidence in favor of selective acknowledgments -- simple experiments with RDP have shown that disabling the selective acknowledgment facility greatly increases the number of retransmitted segments over a lossy, high-delay Internet path [Partridge87]. A recent simulation study by Kevin Fall and Sally Floyd [Fall95], demonstrates the strength of TCP with SACK over the non-SACK Tahoe and Reno TCP implementations.

RFC1072 [VJ88] describes one possible implementation of SACK options for TCP. Unfortunately, it has never been deployed in the Internet, as there was disagreement about how SACK options should be used in conjunction with the TCP window shift option (initially described RFC1072 and revised in [Jacobson92]).

We propose slight modifications to the SACK options as proposed in RFC1072. Specifically, sending a selective acknowledgment for the most recently received data reduces the need for long SACK options [Keshav94, Mathis95]. In addition, the SACK option now carries full 32 bit sequence numbers. These two modifications represent the only changes to the proposal in RFC1072. They make SACK easier to implement and address concerns about robustness.

The selective acknowledgment extension uses two TCP options. The first is an enabling option, "SACK-permitted", which may be sent in a SYN segment to indicate that the SACK option can be used once the connection is established. The other is the SACK option itself, which may be sent over an established connection once permission has been given by SACK-permitted.

The SACK option is to be included in a segment sent from a TCP that is receiving data to the TCP that is sending that data; we will refer to these TCP's as the data receiver and the data sender, respectively. We will consider a particular simplex data flow; any data flowing in the reverse direction over the same connection can be treated independently.

2. Sack-Permitted Option

This two-byte option may be sent in a SYN by a TCP that has been extended to receive (and presumably process) the SACK option once the connection has opened. It MUST NOT be sent on non-SYN segments.

TCP Sack-Permitted Option:

Kind: 4

```
+---------+---------+
| Kind=4  | Length=2|
+---------+---------+
```

3. Sack Option Format

The SACK option is to be used to convey extended acknowledgment information from the receiver to the sender over an established TCP connection.

TCP SACK Option:

Kind: 5

Length: Variable

```
                  +--------+--------+
                  | Kind=5 | Length |
+--------+--------+--------+--------+
|      Left Edge of 1st Block       |
+--------+--------+--------+--------+
|      Right Edge of 1st Block      |
+--------+--------+--------+--------+
|                                   |
/            . . .                  /
|                                   |
+--------+--------+--------+--------+
|      Left Edge of nth Block       |
+--------+--------+--------+--------+
|      Right Edge of nth Block      |
+--------+--------+--------+--------+
```

The SACK option is to be sent by a data receiver to inform the data sender of non-contiguous blocks of data that have been received and queued. The data receiver awaits the receipt of data (perhaps by means of retransmissions) to fill the gaps in sequence space between received blocks. When missing segments are received, the data receiver acknowledges the data normally by advancing the left window edge in the Acknowledgement Number Field of the TCP header. The SACK option does not change the meaning of the Acknowledgement Number field.

This option contains a list of some of the blocks of contiguous sequence space occupied by data that has been received and queued within the window.

Each contiguous block of data queued at the data receiver is defined in the SACK option by two 32-bit unsigned integers in network byte order:

* Left Edge of Block

  This is the first sequence number of this block.

* Right Edge of Block

  This is the sequence number immediately following the last sequence number of this block.

Each block represents received bytes of data that are contiguous and isolated; that is, the bytes just below the block, (Left Edge of Block - 1), and just above the block, (Right Edge of Block), have not been received.

A SACK option that specifies n blocks will have a length of 8*n+2 bytes, so the 40 bytes available for TCP options can specify a maximum of 4 blocks. It is expected that SACK will often be used in conjunction with the Timestamp option used for RTTM [Jacobson92], which takes an additional 10 bytes (plus two bytes of padding); thus a maximum of 3 SACK blocks will be allowed in this case.

The SACK option is advisory, in that, while it notifies the data sender that the data receiver has received the indicated segments, the data receiver is permitted to later discard data which have been reported in a SACK option. A discussion appears below in Section 8 of the consequences of advisory SACK, in particular that the data receiver may renege, or drop already SACKed data.

4. Generating Sack Options: Data Receiver Behavior

If the data receiver has received a SACK-Permitted option on the SYN for this connection, the data receiver MAY elect to generate SACK options as described below. If the data receiver generates SACK options under any circumstance, it SHOULD generate them under all permitted circumstances. If the data receiver has not received a SACK-Permitted option for a given connection, it MUST NOT send SACK options on that connection.

If sent at all, SACK options SHOULD be included in all ACKs which do not ACK the highest sequence number in the data receiver's queue. In this situation the network has lost or mis-ordered data, such that the receiver holds non-contiguous data in its queue. RFC 1122, Section 4.2.2.21, discusses the reasons for the receiver to send ACKs in response to additional segments received in this state. The receiver SHOULD send an ACK for every valid segment that arrives containing new data, and each of these "duplicate" ACKs SHOULD bear a SACK option.

If the data receiver chooses to send a SACK option, the following rules apply:

* The first SACK block (i.e., the one immediately following the kind and length fields in the option) MUST specify the contiguous block of data containing the segment which triggered this ACK, unless that segment advanced the Acknowledgment Number field in the header. This assures that the ACK with the SACK option reflects the most recent change in the data receiver's buffer queue.

* The data receiver SHOULD include as many distinct SACK blocks as possible in the SACK option. Note that the maximum available option space may not be sufficient to report all blocks present in the receiver's queue.

* The SACK option SHOULD be filled out by repeating the most recently reported SACK blocks (based on first SACK blocks in previous SACK options) that are not subsets of a SACK block already included in the SACK option being constructed. This assures that in normal operation, any segment remaining part of a non-contiguous block of data held by the data receiver is reported in at least three successive SACK options, even for large-window TCP implementations [RFC1323]). After the first SACK block, the following SACK blocks in the SACK option may be listed in arbitrary order.

It is very important that the SACK option always reports the block containing the most recently received segment, because this provides the sender with the most up-to-date information about the state of the network and the data receiver's queue.

5. Interpreting the Sack Option and Retransmission Strategy: Data Sender Behavior

When receiving an ACK containing a SACK option, the data sender SHOULD record the selective acknowledgment for future reference. The data sender is assumed to have a retransmission queue that contains the segments that have been transmitted but not yet acknowledged, in sequence-number order. If the data sender performs re-packetization before retransmission, the block boundaries in a SACK option that it receives may not fall on boundaries of segments in the retransmission queue; however, this does not pose a serious difficulty for the sender.

One possible implementation of the sender's behavior is as follows. Let us suppose that for each segment in the retransmission queue there is a (new) flag bit "SACKed", to be used to indicate that this particular segment has been reported in a SACK option.

When an acknowledgment segment arrives containing a SACK option, the data sender will turn on the SACKed bits for segments that have been selectively acknowledged. More specifically, for each block in the SACK option, the data sender will turn on the SACKed flags for all segments in the retransmission queue that are wholly contained within that block. This requires straightforward sequence number comparisons.

After the SACKed bit is turned on (as the result of processing a received SACK option), the data sender will skip that segment during any later retransmission. Any segment that has the SACKed bit turned off and is less than the highest SACKed segment is available for retransmission.

After a retransmit timeout the data sender SHOULD turn off all of the SACKed bits, since the timeout might indicate that the data receiver has reneged. The data sender MUST retransmit the segment at the left edge of the window after a retransmit timeout, whether or not the SACKed bit is on for that segment. A segment will not be dequeued and its buffer freed until the left window edge is advanced over it.

5.1 Congestion Control Issues

This document does not attempt to specify in detail the congestion control algorithms for implementations of TCP with SACK. However, the congestion control algorithms present in the de facto standard TCP implementations MUST be preserved [Stevens94]. In particular, to preserve robustness in the presence of packets reordered by the network, recovery is not triggered by a single ACK reporting out-of-order packets at the receiver. Further, during recovery the data sender limits the number of segments sent in response to each ACK. Existing implementations limit the data sender to sending one segment during Reno-style fast recovery, or to two segments during slow-start [Jacobson88]. Other aspects of congestion control, such as reducing the congestion window in response to congestion, must similarly be preserved.

The use of time-outs as a fall-back mechanism for detecting dropped packets is unchanged by the SACK option. Because the data receiver is allowed to discard SACKed data, when a retransmit timeout occurs the data sender MUST ignore prior SACK information in determining which data to retransmit.

Future research into congestion control algorithms may take advantage of the additional information provided by SACK. One such area for future research concerns modifications to TCP for a wireless or satellite environment where packet loss is not necessarily an indication of congestion.

6. Efficiency and Worst Case Behavior

If the return path carrying ACKs and SACK options were lossless, one block per SACK option packet would always be sufficient. Every segment arriving while the data receiver holds discontinuous data would cause the data receiver to send an ACK with a SACK option containing the one altered block in the receiver's queue. The data sender is thus able to construct a precise replica of the receiver's queue by taking the union of all the first SACK blocks.

Since the return path is not lossless, the SACK option is defined to include more than one SACK block in a single packet. The redundant blocks in the SACK option packet increase the robustness of SACK delivery in the presence of lost ACKs. For a receiver that is also using the time stamp option [Jacobson92], the SACK option has room to include three SACK blocks. Thus each SACK block will generally be repeated at least three times, if necessary, once in each of three successive ACK packets. However, if all of the ACK packets reporting a particular SACK block are dropped, then the sender might assume that the data in that SACK block has not been received, and unnecessarily retransmit those segments.

The deployment of other TCP options may reduce the number of available SACK blocks to 2 or even to 1. This will reduce the redundancy of SACK delivery in the presence of lost ACKs. Even so, the exposure of TCP SACK in regard to the unnecessary retransmission of packets is strictly less than the exposure of current implementations of TCP. The worst-case conditions necessary for the sender to needlessly retransmit data is discussed in more detail in a separate document [Floyd96].

Older TCP implementations which do not have the SACK option will not be unfairly disadvantaged when competing against SACK-capable TCPs. This issue is discussed in more detail in [Floyd96].

7. Sack Option Examples

The following examples attempt to demonstrate the proper behavior of SACK generation by the data receiver.

Assume the left window edge is 5000 and that the data transmitter sends a burst of 8 segments, each containing 500 data bytes.

Case 1: The first 4 segments are received but the last 4 are dropped.

The data receiver will return a normal TCP ACK segment acknowledging sequence number 7000, with no SACK option.

Case 2: The first segment is dropped but the remaining 7 are received.

Upon receiving each of the last seven packets, the data receiver will return a TCP ACK segment that acknowledges sequence number 5000 and contains a SACK option specifying one block of queued data:

Triggering Segment	ACK	Left Edge	Right Edge
5000	(lost)		
5500	5000	5500	6000
6000	5000	5500	6500
6500	5000	5500	7000
7000	5000	5500	7500
7500	5000	5500	8000
8000	5000	5500	8500
8500	5000	5500	9000

Case 3: The 2nd, 4th, 6th, and 8th (last) segments are dropped.

The data receiver ACKs the first packet normally. The third, fifth, and seventh packets trigger SACK options as follows:

Triggering Segment	ACK	First Block Left Edge	First Block Right Edge	2nd Block Left Edge	2nd Block Right Edge	3rd Block Left Edge	3rd Block Right Edge
5000	5500						
5500	(lost)						
6000	5500	6000	6500				
6500	(lost)						
7000	5500	7000	7500	6000	6500		
7500	(lost)						
8000	5500	8000	8500	7000	7500	6000	6500
8500	(lost)						

Suppose at this point, the 4th packet is received out of order. (This could either be because the data was badly misordered in the network, or because the 2nd packet was retransmitted and lost, and then the 4th packet was retransmitted). At this point the data receiver has only two SACK blocks to report. The data receiver replies with the following Selective Acknowledgment:

Triggering Segment	ACK	First Block Left Edge	First Block Right Edge	2nd Block Left Edge	2nd Block Right Edge	3rd Block Left Edge	3rd Block Right Edge
6500	5500	6000	7500	8000	8500		

Suppose at this point, the 2nd segment is received. The data receiver then replies with the following Selective Acknowledgment:

Triggering Segment	ACK	First Block Left Edge	First Block Right Edge	2nd Block Left Edge	2nd Block Right Edge	3rd Block Left Edge	3rd Block Right Edge
5500	7500	8000	8500				

8. Data Receiver Reneging

Note that the data receiver is permitted to discard data in its queue that has not been acknowledged to the data sender, even if the data has already been reported in a SACK option. Such discarding of SACKed packets is discouraged, but may be used if the receiver runs out of buffer space.

The data receiver MAY elect not to keep data which it has reported in a SACK option. In this case, the receiver SACK generation is additionally qualified:

* The first SACK block MUST reflect the newest segment. Even if the newest segment is going to be discarded and the receiver has already discarded adjacent segments, the first SACK block MUST report, at a minimum, the left and right edges of the newest segment.

* Except for the newest segment, all SACK blocks MUST NOT report any old data which is no longer actually held by the receiver.

Since the data receiver may later discard data reported in a SACK option, the sender MUST NOT discard data before it is acknowledged by the Acknowledgment Number field in the TCP header.

9. Security Considerations

This document neither strengthens nor weakens TCP's current security properties.

10. References

[Cheriton88] Cheriton, D., "VMTP: Versatile Message Transaction Protocol", RFC 1045, Stanford University, February 1988.

[Clark87] Clark, D., Lambert, M., and L. Zhang, "NETBLT: A Bulk Data Transfer Protocol", RFC 998, MIT, March 1987.

[Fall95] Fall, K. and Floyd, S., "Comparisons of Tahoe, Reno, and Sack TCP", ftp://ftp.ee.lbl.gov/papers/sacks.ps.Z, December 1995.

[Floyd96] Floyd, S., "Issues of TCP with SACK", ftp://ftp.ee.lbl.gov/papers/issues_sa.ps.Z, January 1996.

[Huitema81] Huitema, C., and Valet, I., An Experiment on High Speed File Transfer using Satellite Links, 7th Data Communication Symposium, Mexico, October 1981.

[Jacobson88] Jacobson, V., "Congestion Avoidance and Control", Proceedings of SIGCOMM '88, Stanford, CA., August 1988.

[Jacobson88}, Jacobson, V. and R. Braden, "TCP Extensions for Long-Delay Paths", RFC 1072, October 1988.

[Jacobson92] Jacobson, V., Braden, R., and D. Borman, "TCP Extensions for High Performance", RFC 1323, May 1992.

[Keshav94] Keshav, presentation to the Internet End-to-End Research Group, November 1994.

[Mathis95] Mathis, M., and Mahdavi, J., TCP Forward Acknowledgment Option, presentation to the Internet End-to-End Research Group, June 1995.

[Partridge87] Partridge, C., "Private Communication", February 1987.

[Postel81] Postel, J., "Transmission Control Protocol - DARPA Internet Program Protocol Specification", RFC 793, DARPA, September 1981.

[Stevens94] Stevens, W., TCP/IP Illustrated, Volume 1: The Protocols, Addison-Wesley, 1994.

[Strayer92] Strayer, T., Dempsey, B., and Weaver, A., XTP -- the xpress transfer protocol. Addison-Wesley Publishing Company, 1992.

[Velten84] Velten, D., Hinden, R., and J. Sax, "Reliable Data Protocol", RFC 908, BBN, July 1984.

11. Authors' Addresses

Matt Mathis and Jamshid Mahdavi
Pittsburgh Supercomputing Center
4400 Fifth Ave
Pittsburgh, PA 15213
mathis@psc.edu
mahdavi@psc.edu

Sally Floyd
Lawrence Berkeley National Laboratory
One Cyclotron Road
Berkeley, CA 94720
floyd@ee.lbl.gov

Allyn Romanow
Sun Microsystems, Inc.
2550 Garcia Ave., MPK17-202
Mountain View, CA 94043
allyn@eng.sun.com