Network Working Group
Internet-Draft
Intended Status: Standards Track
Expires: February 14, 2013

A. Sabatini
Broker Communications Inc.
.

August 15, 2012

Highly Efficient Selective Acknowledgement (SACK) for TCP
draft-sabatini-tcp-sack-01

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

This Internet-Draft will expire on January 22, 2013.

Comments are solicited and should be addressed to the author at draft-sack@tsabatini.com.

Copyright Notice

Abstract

This memo expands on the Selective Acknowledgement Protocol described in RFC2018 to improve its performance and efficiency while reducing the delay involved in recovering lost segments. This leads to very reliable and efficient communications regardless of transit delay or high levels of lost segments due to noise or congestion. It introduces a fundamentally new way of looking at Selective Acknowledgement and uses this concept to improve the performance of the RFC2018 protocol. This memo proposes an implementation of the improved SACK and discusses its performance and related issues.

Acknowledgements

Much of the text in this document is taken directly from RFC2018 "TCP Selective Acknowledgement Options" by M. Mathis, J. Mahdavi, S. Floyd and A. Romanow and RFC1072 "TCP Extensions for Long-Delay Paths" by B. Braden and V. Jacobson.

1. Introduction

This revision to the SACK protocol has its roots in a similar, HDLC based protocol I designed and implemented for secure financial transactions. That protocol, being designed for use on a worldwide basis, was born out of the need for a protocol that would handle any communications environment no matter how noisy or how much delay (including multiple satellite hops) was in the path. In later years its properties were found valuable in congestion situations where packets were dropped.

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 round-trip 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. The compatible extensions to RFC2018 proposed here enhance the protocol by changing retransmission from a worst case timer basis to a deterministic, state driven basis which responds rapidly to link conditions.

I propose modifications to the SACK options as proposed in RFC2018. Specifically, I add a transmit state to each transmitted message and return that transmit state when each acknowledgement is sent. By using the returned transmit state I can tell what messages have been transmitted after the information in the acknowledgement and thus rebuild the current state of the receiver at the transmitter. I also propose changes to the way SACK blocks are reported to insure that the oldest, and thus the most critical, are transmitted expeditiously without jeopardizing the multiple repetition of SACK information which gives the current protocol its reliability. Additionally since the space to store acknowledgements in IPv4 is limited and may not be able to accommodate all of the acknowledgement pairs, I propose a method of sending the complete receiver state by sending multiple acknowledgements when it becomes evident that transmission has stalled due to loss of multiple ACKs.

The RFC2018 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. This option is extended to both indicate that this newer version of the protocol is being used and to establish an initial value for transmit state. The other is the SACK option itself, which may be sent over an established connection once permission has been given by SACK-permitted. This has also been extended to add both the transmit state implicit in the message and the transmit state that was received at the far end (now called "Returned State").

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. Underlying concepts

In order for a sender to know how to optimally transmit messages to a receiver the sender must recreate the state of the receiver as of the last acknowledgement received (which segments have been received and acknowledged, which segments have not) and then "age" or modify that state by updating it based upon the messages transmitted since the state implicit in the acknowledgement was current. In order to do this the sender must maintain a transmission order list which contains entries for the segment ranges of each message as it is sent. We called the index into the transmission order list "Send State" and transmit this state variable with each message. The receiver, after correctly receiving the message, saves this value and

returns it (now called "Returned State") and the list of selectively acknowledged segments with each acknowledgement. When the sender receives this information it is then capable of constructing a list of missing segments by taking its unacknowledged segment range list and modifying it on the basis of the received selective acknowledgements and then removing from that list all segments that have been transmitted since the message which caused the acknowledgement which is all segments sent with indexes between the current "send state" and the "receive state" in the acknowledgement message.

To accommodate the issue of receiving segments out of order at the receiver, or those packets delayed by alternate routing, the sender does not instantly update its Current Returned State value from the incoming ACK (which could trigger a false retransmission) but rather puts it on a timer queue for a length of time ("Reordering Time") appropriate to the delay randomness in the arrival path (typically 20 to 100 ms based on media, speed and distance), which when the timer entry expires, causes the update of the Current Returned State value. If the updated Current Returned State value shows blocks that remain unacknowledged after this time out they are assumed to be lost and they are queued for retransmission.

Thus by transmitting the complete acknowledgement information through the SACK blocks from the receiver along with an indicator to the sender as to its state current at the time of the acknowledgement the sender can accurately recreate the current status of the receiver assuming all "in flight" messages were received and thus only send the unacknowledged messages starting with the oldest followed by any new messages whose transmission is requested.

3. Enhanced Sack-Permitted Option

This document is designed to be an extension of RFC2018 and any implementation of it must be designed to fall back to handling RFC2018 when he other paty is not capable of handling the enhanced protocol.

Although Enhanced SACK is a compatible extension of standard SACK it is recognized that certain middleware boxes are not RFC compliant as to extensions and therefore will fail if, as would properly be done, Enhanced SACK was handled as such. Therefore Enhanced SACK is transmitted as two options both in the SYN packets as well as data and ACK packets.

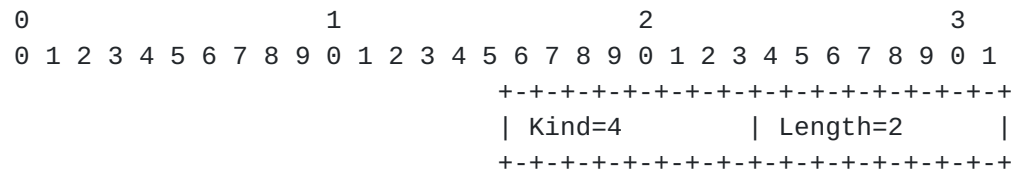
The first option of the pair MUST be the standard SACK option (Kind = 4) if this endpoint desires a SACK session of any kind. The second four or six byte option may be sent in a SYN by a TCP that has been

extended to receive (and presumably process) the Enhanced SACK option in order to indicate its willingness to enter into an Enhanced SACK session.

This option MUST NOT be transmitted on non-SYN segments in the current protocol, it is left to future study as to its use for transmitting long sequences of acknowledgements in one frame.

TCP SACK-Permitted Option:

Kind: 4



TCP Enhanced SACK Setup Option:

Kind: X1

Control Bits: 4 bits (from left to right):

PTO: Tokens Permitted, the sender supports the requirements of this extension

RTO: Request Tokens, sender requests link use the protocol outlined in this extension

PXT: Extended Tokens Supported

RXT: Request Extended Tokens: sender requests using extended

tokens.

Reserved: 12 bits

Reserved for future use, must be set to zero

If RTO is set then the Send State immediately follows, 16 bits if RXT is not set and 24 bits if it is. If necessary the option is padded with a binary zero byte so that length is an even number. In the case that one end can only support 16 bit tokens only the right most 16 bits of the extended field is used.

For brevity in this document only the lesser, 16 bit format is shown.

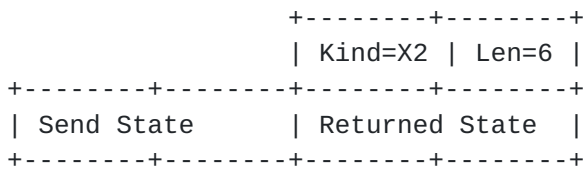
4. SACK Option Format

As with the SYN options, Enhanced SACK information must be transmitted as a seperate option in order to accomodate non RFC compliant middleware boxes. By its nature it must precede the TCP SACK Option.

TCP Enhanced SACK Option:

Kind: X2

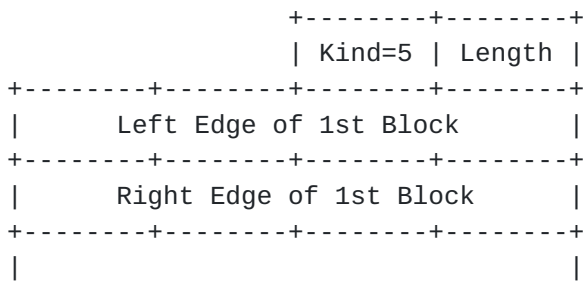
Length: 6 (or 8 if extended token)

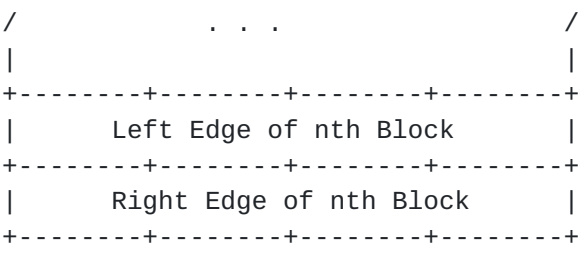


TCP SACK Option:

Kind: 5

Length: Variable





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 or use 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+6 bytes, so the 40 bytes available for TCP options can specify a maximum of 4 blocks. It is suggested that the Enhanced SACK will provide the time-stamp information used for RTTM [Jacobson92].

5. 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 MUST 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 MUST 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 MUST send an ACK for every valid segment that arrives containing new data, and each of these "duplicate" ACKs SHOULD bear a SACK option.

The purpose of the SACK blocks is to recreate the status of the receiver at the transmitter. To that end the most important information is (1) new or changed blocks, (2) the second transmission of new or changed blocks, (3) a complete enumeration of all received blocks starting from the oldest first.

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

* The data receiver first fills in "Send State" in the option from the current value of its "Send State". The data receiver then fills in "Returned State" from its "Saved Send State" which was set by either the SYN option or the SACK option of the last TCP packet that contained a value which was logically greater than the current saved value.

* SACK blocks representing all discontiguous segment ranges received where those ranges are logically over the Acknowledgement Number in the TCP header are kept in logically ascending by segment range list. Additionally a count (a four byte binary for safety) is maintained in each block which represents the number of times it has been transmitted. Each time a SACK block is added or changes (normally by being merged with another entry) the count is set to zero(0).

* All SACK block slots SHOULD be filled on each each normal ACK transmitted, starting with those that have the lowest count (acknowledging the most recently received segments), followed by those with the next lowest count, and so on until all SACK block slots are filled. As each SACK block is moved to a slot its count is incremented by one(1) (thus care needs to be taken on the

second and subsequent passes to skip those entries currently in SACK blocks slots). By always starting from the oldest we insure the most critical have the first chance at receiving a SACK block slot. SACK blocks are empty ONLY if there are less SACK blocks outstanding than there are available slots. This methodology assures a fair number of transmissions to all SACK blocks.

* The receiver receives a Send State from the sender that is logically greater than any previously seen the receiver must generate an ACK regardless of whether any SACK blocks have changed. Note that such a Send State change can come from an ACK produced by the sender as well as a message.

* The definitions in RFC1022 are changed such that if there are entries on the SACK block list an ACK ALWAYS goes out in response to a received data segment.

* To insure that the last added or changed SACK block is transmitted a second time, if the link goes idle for 2*Reordering Time the receiver SHOULD send another ACK following the rules above.

* A timer is maintained based on the timestamp of the oldest SACK block and is set to RTT*1.25, it is reset each time a SACK block with a different segment start becomes the oldest SACK block. At the expiration of this timer, since this and probably other segments have not been retransmitted to the receiver, the receiver resets the timer to .25*RTT and again sends sufficient acknowledgements to completely transmit all current SACK blocks starting from the one with the logically lowest segment start and proceeding in ascending sequence. Note that this process is aborted by any action that changes the oldest SACK block. This timer is used to assure that in case of a burst error the sender has enough information to restart properly.

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

As each transmission request from the calling program is processed and entry is made into the segment queue to handle the request and its buffering. Entries are removed from the segment queue when the segment is completely acknowledged either through the Acknowledgement Number Field of the TCP header passing through the end of that segment or by a SACK block completing the acknowledgement of that segment. The segment is also appended to the end of the retransmission queue and transmission restarted from that segment if transmission has stopped.

Before processing the SACK information the Acknowledgement Number Field of the TCP header is used to eliminate outdated entries from the segment queue, saved list and retransmission queue before new information is added. The acknowledgement may split the first entry in either the segment queue or the retransmission queue in which case a pseudo entry is created in that queue for the unacknowledged remainder which additionally points to the saved original entry with an additional field which is the count of pseudo segments derived from it, set to one in this case. The Acknowledgement Number Field of the TCP header may end up eliminating a pseudo entry in which case the pseudo segment count of the original saved entry is decremented and if zero the segment is then entirely removed from both the segment queue and the saved list.

In processing selective acknowledgements the transmitter applies each SACK block to first the segment queue and then the retransmission queue. If the SACK block completely acknowledges a segment it is removed from the segment queue and moved to the saved list with a count of zero or completely if from the retransmission queue. If the SACK block completely acknowledges a pseudo segment that segment is removed and if from the segment queue the pseudo segment count in the related saved entry is decremented. If the SACK block acknowledges the beginning or end of a segment in either queue a pseudo entry is created with the adjusted unacknowledged remainder, if the segment was on the segment list the original segment is moved to the saved list and the pseudo count is set to one. If the entry was already a pseudo segment and this SACK acknowledges the beginning or end of the segment, the segment limits are adjusted but no other action occurs. If this entry is already a pseudo entry and the SACK block splits the segment in two a second pseudo entry is created to handle the right hand side of the range and, if on the segment list, the pseudo segment count in the related save list entry is incremented by one. The original pseudo entry is modified to represent the left hand range created by the SACK.

The sender maintains a Transmission List which an array of structures into which the segment start and end addresses of each transmitted block (be it a primary transmission or a retransmission) is placed. This list is, for optimal processing, a power of 2 in size and is, at a minimum, four times as large as the Maximum Number of Segments Outstanding. As each segment is transmitted the current Transmit Token modulus the Transmission List size is used as an index into this structure to store the segment start and end and then the Transmit Token is incremented by one.

When each each new SACK option is processed, its Returned Token is checked against the Current Returned Token and the Future Returned Token List, if logically greater than any of the above, it is

inserted into the Future Returned Token list in logical order along with the current time-stamp incremented by the Reordering Time. If it is the first entry on the list a timer is started for the value token time stamp minus current time stamp. When the timer expires the first entry on the Future Returned Token list set as the Current Returned Token and then it is removed from the list. If there are more members on the Future Returned Token list the timer is restarted with a value of the time-stamp in that entry minus the current time-stamp. A change in the Current Returned Token causes a recreation of the Retransmission Queue by first copying the Segment Queue and then removing from it all segments that have been transmitted subsequently, in a process identical to processing a SACK block starting at the segment identified by the Current Returned Token from the Transmission List and continuing through the Transmission List up to the (but not including) the Transmit Token. The Transmission Pointer is then set to first incomplete entry in the Retransmission Queue and transmission restarted if it has stopped.

Another method of implementation which allows quicker retransmission response at the expense of building the Retransmission Queue a second time is to retrieve the time stamp of the just receieved Returned Token if that Returned Token has not previously been seen (by comparing it with the Current Returned Token and the Future Returned Token list) and then subtracting from that timestamp the Reordering Time to allow for out of order messages. This value is then used to select any entry on the Transmission List with an equal or greater timestamp. The first version of the Retransmission Queue is created by copying the Segment Queue and then removing the segments that have since been retransmitted based on the adjusted timestamp. When the timer on the Future Returned Token List expires the retransmission queue is recreated a second time as in the preceeding paragraph.

Note: For processing efficiency we believe most people will implement the Retransmission Queue as additional fields in the Segment Queue.

6.1 Handling last segment problem

If the sender side of the link goes idle for 2*Reordering Time and there are still unacknowledged segments the sender SHOULD send an ACK (which would have the updated Send State) so that the receiver may ultimately detect if the last message is missing and cause it to be transmitted (the receiver would pass the Send State back as Returned State and the sender would realize the segment is still outstanding).

6.2 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]. This algorithm eliminates much unnecessary retransmission so is likely to lessen overall congestion.

Note that the enhanced protocol does not suffer from traditional congestion collapse even though it is more robust, since it does not use timers and is rate limited by the tokens. Delayed and lost messages and ACKS make it slower but do not increase the traffic it sends significantly.

The use of time-outs as a fall-back mechanism for detecting dropped packets is unchanged by the SACK option. Because in normal operation acknowledgements will prevent retransmit timeout, when a retransmit timeout occurs the data sender SHOULD 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.

7. Efficiency and Worst Case Behavior

Although this high efficiency improved SACK option sends more and larger SACK blocks and more acknowledgements than the previous version, with an active bi-directional link additional acknowledgements are often associated with data transmission and thus not a penalty. If the SACK option needs to be used due to segment loss then the improved efficiency afforded with this protocol more than justifies the additional SACK blocks.

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].

8. Time-stamping

One pleasant benefit of having a token which is returned by the far end on a deterministic basis is the easy calculation of round trip delay. We can save a time stamp along with the segment information in our transmission order array. This allows us to calculate round trip delay when we receive our "Returned State" value and use it to access the time-stamp. Since more than one received message might have the same "Returned State" value we mark the time-stamp after use to indicate that the value should not be used again. Note that if an acknowledgement is lost we will calculate a longer delay than is accurate therefore we must smooth the returned values, typically returning the sma llest out of the last N where N is typically four.

9. Data Receiver Reneging

Since the Sender is recreating the state of the Receiver, the data Receiver MUST NOT discard data in its queue once that data has been reported in a SACK option. The Receiver is responsible for allocating enough buffers so that the missing segments within the window may be properly received and processed. Since enhanced SACK is event driven the lack of a new event for 2.50*RTT SHOULD trigger a connection reset to guard agains denial of service attacks.

10. Security Considerations

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

11. References

[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.

[Mathis96] Mathis, M., Mahdavi, J., Floyd, S., Romanow, J. "TCP Selective Acknowleddgement Options", RFC 2018, October 1996.

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

Author's Address

Anthony Sabatini
Broker Communications Inc.
200 West 20th Street

Suite 1216
New York, NY 10011
Email: draft-sack@tsabatini.com

The author is currently a master's degree candidate at -

Hofstra University
Hempstead, N.Y.

His adviser is Dr. Xiang Fu