Using TCP Selective Acknowledgement (SACK) Information to Determine Duplicate Acknowledgements for Loss Recovery Initiation

Status of this Memo

This Internet-Draft is submitted to IETF 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/ietf/1id-abstracts.txt.

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

This Internet-Draft will expire on September 2010.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with
Abstract

This document describes a TCP sender algorithm to trigger loss recovery based on the TCP Selective Acknowledgement (SACK) information gathered on a SACK scoreboard instead of simply counting the number of arriving duplicate acknowledgements (ACKs) in the traditional way. The given algorithm is more robust to ACK losses, ACK reordering, missed duplicate acknowledgements due to delayed acknowledgements, and extra duplicate acknowledgements due to duplicated segments and out-of-window segments. The algorithm allows not only a timely initiation of TCP loss recovery but also reduces false fast retransmits. It has a low implementation cost on top of the SACK scoreboard defined in RFC 3517.
Table of Contents

1. Introduction .......................................................... 5
   1.1. Conventions and Terminology ............................. 6
   1.2. Definitions ..................................................... 7
2. Algorithm Details ...................................................... 7
   2.1. Redefined IsLost (SeqNum) ............................... 7
   2.2. The Algorithm .................................................. 7
3. Discussion ............................................................. 9
   3.1. Small Segment Sender ...................................... 9
   3.2. SACK Capability Misbehavior ............................. 10
   3.3. Compatibility with Duplicate ACK based Loss Recovery Algorithms ................................. 11
4. Security Considerations ............................................... 11
5. IANA Considerations .................................................. 12
6. Acknowledgements .................................................... 12
Appendix ........................................................................... 12
   A. Scenarios ............................................................... 12
      A.1. Basic Case ..................................................... 12
      A.2. Delayed ACK ...................................................... 13
      A.3. ACK Loss ........................................................ 14
      A.4. ACK Reordering ................................................ 15
      A.5. Duplicated Packet ............................................ 16
      A.6. Mitigation of Blind Throughput Reduction Attack ................................................... 16
References ........................................................................ 16
Normative References ..................................................... 16
Informative References .................................................... 17
AUTHORS’ ADDRESSES ..................................................... 18
TO BE DELETED BY THE RFC EDITOR UPON PUBLICATION:

Changes from draft-ietf-tcpm-sack-recovery-entry-00.txt

* Mention setting of RecoveryPoint explicitly as this algorithm depends on it being valid.

* Changed definition of IsLost (SeqNum) to be less strict.

* Changed packet ordering in one of the appendix examples, now it makes more sense in the context of this algorithm. Point out in the examples which of the transmissions are due to Limited Transmit and Fast retransmit.

Changes from draft-jarvinen-tcpm-sack-recovery-entry-01.txt

* Clarified issues that based on feedback may cause confusion for the reader.

* Incorporated handling of cumulative ACKs into the algorithm

* 2581 refs -> 5681

* Added early-rexmt ID as a related one, it uses SACK information similar to this algorithm (Thanks to Anna Brunstrom).

* More cases added where this algorithm is beneficial in taking advantage of SACK block redundancy (thanks to Anna Brunstrom).

* Discuss on differences how duplicate ACK counter is managed (traditional vs. this algorithm)

* Added ref and couple of words about blind throughput reduction attack

* Wrote SACK splitting attacks. These attacks are quite close to the edge in significance. Should consider just dropping (rather insignificant).

Changes from draft-jarvinen-tcpm-sack-recovery-entry-00.txt

* TODO items embedded: Improvements with window update, clarify dupack counting

* Modified ACK reordering scenario in appendix, shows now a scenario where recovery is triggered in a more timely manner.

* IDnits
1. Introduction

The Transmission Control Protocol (TCP) [RFC793] has two methods for triggering retransmissions. First, the TCP sender relies on incoming duplicate acknowledgements (ACKs) [RFC5681], indicating receipt of out-of-order segments at the TCP receiver. After receiving a required number of duplicate ACKs (usually three), the TCP sender retransmits the first unacknowledged segment and continues with a fast recovery algorithm such as Reno [RFC5681], NewReno [RFC3782] or SACK-based loss recovery [RFC3517]. Second, the TCP sender maintains a retransmission timer that triggers retransmission of segments, if the retransmission timer expires before the segments have been acknowledged.

While the conservative loss recovery algorithm defined in [RFC3517] takes full advantage of SACK information during a loss recovery, it does not consider the very same information during the pre-recovery detection phase. Instead, it simply counts the number of arriving duplicate ACKs and leans on the number of duplicate ACKs in deciding when to enter loss recovery. However, this traditional heuristics of simply counting the number of duplicate ACKs to trigger a loss recovery fails in several cases to determine correctly the actual number of valid out-of-order segments the receiver has successfully received. First, trusting on duplicate ACKs alone utterly fails to get hold of the whole picture in case of ACK losses and ACK reordering, resulting in delayed or missed initiation of fast retransmit and fast recovery. Similarly, the delayed ACK mechanism tends to conceal the first duplicate ACK as the delayed cumulative ACK becomes combined with the first duplicate ACK when the first out-of-order segment arrives at the receiver (in case of an enlarged ACK ratio such as with ACK congestion control [RFC5690], even more significant portion is affected). Second, segment duplication or out-of-window segments increase the risk of falsely triggering loss recovery as they trigger duplicate ACKs. At worst, this legitimate behavior on out-of-window segments can be turned into a blind throughput reduction attack [CPNI09]. Third, receiver window updates or opposite direction data segments cannot be counted as duplicate ACKs with the traditional approach but can still contain
redundant SACK information that the sender could benefit from in a scenario where the actual duplicate ACKs where lost.

The algorithm specified in this document uses TCP Selective Acknowledgement Option [RFC2018] in the pre-recovery state to determine duplicate ACKs and to trigger loss recovery based on the information gathered on the SACK scoreboard [RFC3517]. It gives a more accurate heuristic for determining the number of out-of-order segments that have arrived at the TCP receiver. The information gathered on the SACK scoreboard reveals missing ACKs and allows detecting duplicate events. Therefore, the algorithm enables a timely triggering of Fast Retransmit. In addition, it allows the use of Limited Transmit [RFC3042] accurately regardless of lost ACKs and also in the cases where the SACK information is piggybacked to a cumulative ACK due to delayed ACKs. This, in turn, improves the ACK clock accuracy.

This algorithm is close to what Linux TCP implementation has used for a very long time when in conservative SACK mode. A similar approach is briefly mentioned along ACK congestion control [RFC5690] but as the usefulness of the algorithm in this document is more general and not limited to ACK congestion control we specify it separately. We also note that the definition of a duplicate acknowledgement already suggests that an incoming ACK can be considered as a duplicate ACK if it "contains previously unknown SACK information" [RFC5681]. In addition, SACK information is used, whenever available, for similar purpose by Early Retransmit [AAA+10].

This algorithm also resembles Forward Acknowledgement (FACK) [MM96] but they differ in how the quantity of data outstanding in the network is determined. FACK always assumes that every non-SACKed octet below the highest SACKed octet is lost which is only true if no reordering occurs. Thus it would simply trigger loss recovery whenever the highest SACKed octet is more than dupThresh * SMSS octets above SND.UNA.

1.1. Conventions and 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 BCP 14, RFC 2119 [RFC2119] and indicate requirement levels for protocols.
1.2. Definitions

The reader is expected to be familiar with the definitions given in [RFC5681], [RFC2018], and [RFC3517].

2. Algorithm Details

In order to use this algorithm, a TCP sender MUST have TCP Selective Acknowledgement Option [RFC2018] enabled and negotiated for the TCP connection. The TCP sender MUST maintain SACK information in an appropriate data structure such as scoreboard defined in [RFC3517]. This algorithm uses functions Update(), and SetPipe() and variables DupThresh, HighData, HighRxt, Pipe, and RecoveryPoint, as defined in [RFC3517]. Note: the definition of IsLost (SeqNum) is altered from the one specified in [RFC3517].

2.1. Redefined IsLost (SeqNum)

IsLost (SeqNum) defined in [RFC3517] is stricter than necessary in counting how many segments the receiver has received past SeqNum. Instead of requiring at least three times SMSS bytes to be SACKed, it is enough to have at least two times SMSS bytes plus one byte SACKed to confirm that the receiver has received at least three segments above SeqNum (and would have generated at least three duplicate ACKs). The less strict definition is:

IsLost (SeqNum):

This routine returns whether the given sequence number is considered to be lost. The routine returns true when either DupThresh discontiguous SACKed sequences have arrived above 'SeqNum' or more than (DupThresh - 1) * SMSS bytes with sequence numbers greater than 'SeqNum' have been SACKed. Otherwise, the routine returns false.

2.2. The Algorithm

A TCP sender using this algorithm MUST take the following steps upon the receipt of any ACK containing SACK information:

1) If no previous loss event has occurred on the connection OR RecoveryPoint is less than SND.UNA (the oldest unacknowledged sequence number [RFC793]), continue with the other steps of this algorithm. Otherwise, continue the ongoing loss recovery.
2) Update the scoreboard via the Update () function as outlined in [RFC3517].

3) If ACK is a cumulative ACK, reset duplicate ACK counter to zero.

4) If ACK contains SACK blocks with previously unknown in-window SACK information (i.e., between SND.UNA and HighData, assuming SND.UNA has been updated from the acknowledgment number of the ACK), increase duplicate ACK counter.

5) Determine if a loss recovery should be initiated:

   If IsLost (SND.UNA) returns false AND the sender has received less than DupThresh duplicate ACKs, goto step 6A. Otherwise goto step 6B.

6A) Invoke optional Limited Transmit:

   Set HighRxt to SND.UNA and run SetPipe(). The TCP sender MAY transmit previously unsent data segments according the guidelines of Limited Transmit [RFC3042], with the exception that the amount of octets that can be send is determined by Pipe and cwnd.

   If cwnd - Pipe >= 1 SMSS, the TCP sender can transmit one or more segments as follows:

   Send Loop:

   a) If available unsent data exists and the receiver’s advertised window allows, transmit one segment of up to SMSS octets of previously unsent data starting with sequence number HighData+1 and update HighData to reflect the transmission of the data segment. Otherwise, exit Send Loop.

   b) Run SetPipe() to re-calculate the number of outstanding octets in the network. If cwnd - Pipe >= 1 SMSS, go to step a) of Send Loop. Otherwise, exit Send Loop.

6B) Invoke Fast Retransmit and enter loss recovery:

   Initiate a loss recovery phase, per the fast retransmit algorithm outlined in [RFC5681], and continue with a fast recovery algorithm such as the SACK-based loss recovery algorithm outlined in [RFC3517]. This includes setting RecoveryPoint to HighData as in step (1) of [RFC3517].
3.  Discussion

In scenarios where no ACK losses nor reordering occur and the first
acknowledgement with SACK information is not the ACK held due to
delayed acknowledgements mechanism, the new SACK information with
each duplicate ACK covers a single segment. Those duplicate ACKs
cause this algorithm to trigger loss recovery after three duplicate
acknowledgements and will allow transmission of new segments using
Limited Transmit on the first and second duplicate ACK. This is
identical to the behavior that would occur without this algorithm
(assuming DupThresh is 3 and that all segments are SMSS sized). This
scenario together with other typical scenarios describing the
behavior of the algorithm are depicted in Appendix A.

This algorithm SHOULD be used also with an ACK that contains a
window update or opposite direction data that could not be
considered as a duplicate ACK in the traditional algorithm. Such
behavior is safe because the SACK information can only add more
information to the current state of the sender; at worst, all
received information is just redundant.

Setting HighRxt to SND.UNA in Step 6A has no direct relation to this
algorithm. Yet it is included in the algorithm to avoid confusion in
how to implement SetPipe() correctly because it depends on having a
valid HighRxt value \[RFC3517\].

A set of potential issues to consider with the algorithm are
discussed in the following.

3.1.  Small Segment Sender

If a TCP sender is sending small segments (usually intentionally
overriding Nagle algorithm \[RFC896\]), the IsLost (SND.UNA) used in
step 5 of the algorithm might fail to detect the need for loss
recovery on the third duplicate acknowledgement because not enough
octets have been SACKed to cover more than (DupThresh - 1) * SMSS
bytes above SND.UNA. Therefore, an adapted duplicate ACK algorithm
is needed as a fallback. Steps 3, 4 and the latter condition of step
5 implement the adapted duplicate ACK algorithm in parallel to the
SACK block based detection.

The number of duplicate ACKs is an artificial metric to estimate the
number of segments the receiver has already in its receive buffer.
How accurately they match depends on the scenario. Because of that,
the goal of the duplicate ACK counter included into this algorithm
is not to achieve bug-to-bug compatibility with the plain duplicate
ACK counter but to estimate how many out-of-order segments the
receiver has already queued in a more accurate way. Therefore, the
duplicate ACK counter used as a fallback mechanism in this algorithm
differs from the plain duplicate ACK counter. However, such
differences indicate a scenario where the plain counter was not able
to accurately keep track of the receiver state.

While the fallback algorithm itself does not look into
acknowledgment field in order to make a decision whether ACK is a
"duplicate ACK", the duplicate ACK counter is not renamed in this
document as in practice most of ACKs that increment the counter
would still contain a duplicate acknowledgment number. In contrast
to the traditional approach, only condition that must be satisfied
to increment the duplicate ACK counter with this algorithm is that
the acknowledgement MUST contain at least one in-window SACK block
that covers octets that were not previously SACKed [RFC5681]. In
cases with ACK losses or delayed ACKs this condition can also match
to cumulative ACKs, receiver window updates and opposite direction
data segments but still the counter can safely be incremented.

Alternatively to the fallback algorithm, a TCP sender that is able
to discern segment boundaries accurately can consider full segments
in IsLost (SeqNum) regardless of segment size. Therefore, such a
TCP sender can avoid the problem with small segments using IsLost
(SND.UNA) check alone which means that Steps 3, 4 and the latter
condition of step 5 are redundant and not required to be
implemented.

Note: the small segments problem is not unique to this algorithm but
also the SACK-based loss recovery [RFC3517] encounters it because of
how IsLost (SeqNum) is defined.

3.2. SACK Capability Misbehavior

If the receiver represents such a SACK misbehavior that it
advertises SACK capability but never sends any SACK blocks when it
should, this algorithm fails to enter loss recovery and
retransmission timeout is required for recovery. However, such
misbehavior does not allow SACK-based loss recovery [RFC3517] to
work either, and a TCP sender will anyway require a timeout to
recover if there was more than one lost data segment within the
window.
3.3. Compatibility with Duplicate ACK based Loss Recovery Algorithms

This algorithm SHOULD NOT be used together with a fast recovery algorithm that determines the segments that have left the network based on the number of arriving duplicate acknowledgements (e.g., NewReno [RFC3782]), instead of the actual segments reported by SACK. In presence of ACK reordering such an algorithm will count the delayed duplicate acknowledgements during the fast recovery algorithm as extra while determining the number of packets that have left the network.

In general there should be very little reason to combine this algorithm with a loss recovery algorithm that is based on inferior, non-SACK based information only.

4. Security Considerations

A malicious TCP receiver may send false SACK information for sequence number ranges which it has not received in order to trigger Fast Retransmit sooner. Such behavior would only be useful when out-of-order segments have arrived because otherwise the flow undergoes a loss recovery with a window reduction. This kind of lying involves guessing which segments will arrive later. In case the guess was wrong, the performance of the flow is ruined because the TCP sender will need a retransmission timeout as it will not retransmit the segments until it assumes SACK reneging. On a successful guess the attacker is able to trigger the recovery slightly earlier. The later segments would have allowed reporting the very same regions with SACK anyway. Therefore, the gain from this attack is small, hardly justifiable considering the drastic effect of a misguess. Furthermore, a similar attack can be made with the duplicate acknowledgment based algorithm (even if the new SACK information rule is applied) by sending false duplicate acknowledgements with false SACK ranges, and trivially without the new SACK information rule.

A variation of the lying attack discards reliability of the flow but as soon as the reliability is not a concern of the receiver, a number of simpler ways exist to attack TCP independently of this algorithm. Thus this algorithm is not considered to weaken TCP security properties against false information.

Splitting SACK blocks into a smaller than the received segment sized chunks allows the receiver to enable recovery to start sooner because of IsLost (SeqNum) discontiguous check. However, by doing so the receiver neglects the possibility of reordering for a little gain. If the segment was just reordered, the sender performs
unnecessary window reduction and unnecessary retransmission of the reordered segment. Another variant of SACK block splitting simply tries to increase consumption of bandwidth by triggering a burst of retransmissions falsely. However, the difference between sending three duplicate ACKs (traditional algorithm) and a single ACK with SACK blocks will not offer significant benefits to make such an attack practical with a small DupThresh value such as three. In case the sender keeps track of segment boundaries and applies them in IsLost (SeqNum), such attack will not succeed as the sender cannot be misled to believe that a segment was split into multiple chunks.

5. IANA Considerations

This document has no actions for IANA.

6. Acknowledgements

The authors would like to thank Alexander Zimmermann and Anna Brunstrom for the comments on this document.

Appendix

A. Scenarios

A.1. Basic Case

In this scenario no Delayed ACK, ACK losses, reordering or other "abnormal" behavior happens. For simplicity all the segments are SMSS sized.

Once the TCP receiver gets first out-of-order segment, it sends a duplicate ACK with SACK information about the received octets. The following two out-of-order segments trigger a duplicate ACK each, with the corresponding range SACKed in addition to the previously known information. The sender gets those duplicate ACKs in-order, each of them will SACK a new previously unknown segment.

This algorithm triggers loss recovery on third duplicate ACK because IsLost (SeqNum) returns true as more than (DupThresh - 1) * SMSS bytes become SACKed on the same acknowledgement, thus the behavior is identical to that of a sender which is using duplicate acknowledgments. If Limited Transmit is in use, two first duplicate
ACKs allow a single segment to be sent with either of the algorithms (Pipe is decremented by SMSS by the SACKed octets per ACK allowing SMSS worth of new octets).

<table>
<thead>
<tr>
<th>ACK Received</th>
<th>Transmitted Segment</th>
<th>Received Segment</th>
<th>ACK Sent (Including SACK Blocks)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>3000-3499</td>
<td>3000-3499</td>
<td>(delayed ACK)</td>
</tr>
<tr>
<td></td>
<td>3500-3999</td>
<td>3500-3999</td>
<td>4000</td>
</tr>
<tr>
<td>2000</td>
<td>4000-4499</td>
<td>(dropped)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4500-4999</td>
<td>4500-4999</td>
<td>4000, SACK=4500-5000</td>
</tr>
<tr>
<td>3000</td>
<td>5000-5499</td>
<td>5000-5499</td>
<td>4000, SACK=4500-5500</td>
</tr>
<tr>
<td></td>
<td>5500-5999</td>
<td>5500-5999</td>
<td>4000, SACK=4500-6000</td>
</tr>
<tr>
<td>4000</td>
<td>6000-6499</td>
<td>6000-6499</td>
<td>4000, SACK=4500-6500</td>
</tr>
<tr>
<td></td>
<td>6500-6999</td>
<td>6500-6999</td>
<td>4000, SACK=4500-7000</td>
</tr>
<tr>
<td>4000, SACK=4500-5000</td>
<td>(lim. tr.) 7000-7499</td>
<td>7000-7499</td>
<td>4000, SACK=4500-7500</td>
</tr>
<tr>
<td>4000, SACK=4500-5500</td>
<td>(lim. tr.) 7500-7999</td>
<td>7500-7999</td>
<td>4000, SACK=4500-8000</td>
</tr>
<tr>
<td>4000, SACK=4500-6000</td>
<td>(fast retr.) 4000-4499</td>
<td>4000-4499</td>
<td>8000</td>
</tr>
</tbody>
</table>

A.2. Delayed ACK

The case with delayed ACK occurs when the receiver sends the first ACK with SACK information but since the previous ACK was sent with a lower sequence number because an acknowledgment is held by delayed ACK, the sender will not considered it as duplicate ACK. Because the segment contains SACK information that is identical to the basic case, the sender can use Limited Transmit with the same segments as in the basic case and will start loss recovery at the third acknowledgment, i.e., with the second duplicate acknowledgment. In the same situation the duplicate ACK based sender will have to wait for one more duplicate ACK to arrive to do the same as the first acknowledgment is fully "wasted".

Technically an acknowledgement with a sequence number higher than what was previously acknowledged is not a duplicate acknowledgement but a presence of the SACK block tells another story revealing the receiver which used delayed ACK, and thus the missing duplicate acknowledgement in between. The response of a TCP sender taking advantage of such inferred duplicate acknowledgements is well within
the guidelines of packet conservation principle [Jac88] as it still sends only when segments have left the network.

<table>
<thead>
<tr>
<th>ACK</th>
<th>Transmitted</th>
<th>Received</th>
<th>ACK Sent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Segment</td>
<td>Segment</td>
<td>(Including SACK Blocks)</td>
</tr>
<tr>
<td>1500</td>
<td>3000-3499</td>
<td>3000-3499</td>
<td>3500</td>
</tr>
<tr>
<td></td>
<td>3500-3999</td>
<td>3500-3999</td>
<td></td>
</tr>
<tr>
<td>2500</td>
<td>4000-4499</td>
<td>(dropped)</td>
<td>4000, SACK=4500-5000</td>
</tr>
<tr>
<td></td>
<td>4500-4999</td>
<td>4500-4999</td>
<td></td>
</tr>
<tr>
<td>3500</td>
<td>5000-5499</td>
<td>5000-5499</td>
<td>4000, SACK=4500-5500</td>
</tr>
<tr>
<td></td>
<td>5500-5999</td>
<td>5500-5999</td>
<td>4000, SACK=4500-6000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4000, SACK=4500-5000 (two segments left the network)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6000-6499</td>
<td>6000-6499</td>
<td>4000, SACK=4500-6500</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(lim. tr.) 8500-6999</td>
<td>8500-6999</td>
<td>4000, SACK=4500-7000</td>
</tr>
<tr>
<td>4000, SACK=4500-5500</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(lim. tr.) 7000-7499</td>
<td>7000-7499</td>
<td>4000, SACK=4500-7500</td>
</tr>
<tr>
<td>4000, SACK=4500-6000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(fast retr.) 4000-4499</td>
<td>4000-4499</td>
<td>7500</td>
</tr>
<tr>
<td>4000, SACK=4500-6500</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### A.3. ACK Loss

This case with ACK loss shares much behavior with the case with delayed ACK. If hole at RCV.NXT is filled, the sender will notice that cumulative ACK advanced. In case of out-of-order segments the first ACK which gets through to the sender includes SACK blocks up to the quantity the SACK block redundancy is able to cover. With this algorithm the sender immediately takes use of all the information that is made available by the incoming ACK.

<table>
<thead>
<tr>
<th>ACK</th>
<th>Transmitted</th>
<th>Received</th>
<th>ACK Sent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Segment</td>
<td>Segment</td>
<td>(Including SACK Blocks)</td>
</tr>
<tr>
<td>1000</td>
<td>3000-3499</td>
<td>3000-3499</td>
<td>(delayed ACK)</td>
</tr>
<tr>
<td></td>
<td>3500-3999</td>
<td>3500-3999</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>4000-4499</td>
<td>(dropped)</td>
<td>4000, SACK=4500-5000</td>
</tr>
<tr>
<td></td>
<td>4500-4999</td>
<td>4500-4999</td>
<td></td>
</tr>
<tr>
<td>3000</td>
<td>5000-5499</td>
<td>5000-5499</td>
<td>4000, SACK=4500-5500</td>
</tr>
<tr>
<td></td>
<td>5500-5999</td>
<td>5500-5999</td>
<td>4000, SACK=4500-6000</td>
</tr>
</tbody>
</table>
4000
6000-6499  6000-6499   4000, SACK=4500-6500
6500-6999  6500-6999   4000, SACK=4500-7000
4000, SACK=4500-5500 (two segments left the network)
(lim. tr.)  7000-7499  7000-7499   4000, SACK=4500-7500
(lim. tr.)  7500-7999  7500-7999   4000, SACK=4500-8000
4000, SACK=4500-6000
(fast retr.) 4000-4499  4000-4499   8000
4000, SACK=4500-6500

A.4. ACK Reordering

With ACK reordering an ACK is postponed. Due to redundancy the next
ACK after postponed one contains not only its own information but
also the information of the reordered ACK (similar to the ACK losses
case). When the reordered ACK arrives later, the sender already
knows the information it provides and therefore no actions are taken
with this algorithm.

<table>
<thead>
<tr>
<th>ACK Received</th>
<th>Transmitted Segment</th>
<th>Received Segment</th>
<th>ACK Sent (Including SACK Blocks)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>3000-3499</td>
<td>3000-3499</td>
<td>(delayed ACK)</td>
</tr>
<tr>
<td></td>
<td>3500-3999</td>
<td>3500-3999</td>
<td>4000</td>
</tr>
<tr>
<td>2000</td>
<td>4000-4499</td>
<td>(dropped)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4500-4999</td>
<td>4500-4999</td>
<td>4000, SACK=4500-5000 (delayed)</td>
</tr>
<tr>
<td>3000</td>
<td>5000-5499</td>
<td>5000-5499</td>
<td>4000, SACK=4500-5500</td>
</tr>
<tr>
<td></td>
<td>5500-5999</td>
<td>5500-5999</td>
<td>4000, SACK=4500-6000</td>
</tr>
<tr>
<td>4000</td>
<td>6000-6499</td>
<td>6000-6499</td>
<td>4000, SACK=4500-6500</td>
</tr>
<tr>
<td></td>
<td>6500-6999</td>
<td>6500-6999</td>
<td>4000, SACK=4500-7000</td>
</tr>
</tbody>
</table>
| 4000, SACK=4500-5500 (two segments left the network)
(lim. tr.)  7000-7499  7000-7499   4000, SACK=4500-7500
(lim. tr.)  7500-7999  7500-7999   4000, SACK=4500-8000
4000, SACK=4500-5000 (has only redundant information)
4000, SACK=4500-6000
(fast retr.) 4000-4499  4000-4499   8000
4000, SACK=4500-6500
A.5. Duplicated Packet

A duplicate packet is received either due to unnecessary retransmission or hardware duplication. It adds a redundant ACK which has only redundant information or a data segment to the stream which will trigger a redundant duplicate ACK (possibly with SACK and/or DSACK [RFC2883] information). Because neither adds any new SACKed octets at the TCP sender, this algorithm will not do anything whereas a duplicate ACK based receiver would falsely consider it as a duplicate ACK.

If one of the redundant ACKs is lost, the effect of duplication is just cancelled.

It would be possible for the sender to detect this case using DSACK alone.

A.6. Mitigation of Blind Throughput Reduction Attack

In case an attacker knows or is able to guess 4-tuple of a TCP connection, it may apply a blind throughput reduction attack [CPNI09]. In this attack TCP is tricked to send duplicate ACKs to the other endpoint using segments likely residing out-of-window that is considerably easier to achieve than a match with sequence numbers. If more than dupThresh duplicate ACKs can be triggered in a row without any legitimate segment that advances acknowledged sequence number, the other end acts according to the false congestion signal and halves the window.

With this algorithm such duplicate ACKs are filtered because they do not have any new in-window SACK blocks (DSACK [RFC2883] might be present though, but it does not cover in-window octets).

References

Normative References


Informative References


AUTHORS’ ADDRESSES

Ilpo Jarvinen  
University of Helsinki  
P.O. Box 68  
FI-00014 UNIVERSITY OF HELSINKI  
Finland  
Email: ilpo.jarvinen@helsinki.fi

Markku Kojo  
University of Helsinki  
P.O. Box 68  
FI-00014 UNIVERSITY OF HELSINKI  
Finland  
Email: kojo@cs.helsinki.fi