Network Working Group N. Demizu
Internet-Draft NICT
Expires: September 3, 2006 March 3, 2006

TS2 --- A Modified TCP Timestamps Mechanism
<draft-demizu-tcp-ts2-01.txt>

Status of this Memo

Copyright Notice

Abstract

This memo proposes a modified TCP Timestamps mechanism called "TS2". It uses the existing "TCP Timestamps option" specified in RFC1323 and a new TCP option called "the TCP Old Timestamps option", which is specified in this memo. As a fallback, an RFC1323-compatible mode called "TS1" is also available.

The base mechanism of TS2 includes the definitions of those two TCP Timestamps options, mode negotiation to enable TS1 or TS2, and a rule for updating internal states. The applied mechanisms of TS2 include an accurate RTT measurement mechanism that is correct even for duplicate ACK segments (RTTM/TS2), a reordering-robust mechanism to detect wrapped sequence numbers (PAWS/TS2), a lightweight mechanism to detect spoofed segments (PASA/TS2), a loss inference mechanism applicable to both original and retransmitted data segments (DLI/TS2), and a spurious loss inference detection mechanism that operates without waiting for one RTT by sending arbitrary in-window data (SLID/TS2).

Table of Contents

1. Introduction

This memo proposes a modified TCP Timestamps mechanism called "TS2". It uses the existing "TCP Timestamps option" [RFC1323] and a new TCP option called "the TCP Old Timestamps option", which is specified in this memo. The significant differences between TS2 and the TCP Timestamps option specified in [RFC1323] are the rule to determine which timestamp is echoed and the timestamp unit. In addition, TS2 solves the issues with the existing TCP Timestamps option specified in [RFC1323], as described in appendix E.

As a fallback, RFC1323-compatible mode called "TS1" is also available. The use of TS1 or TS2 is negotiated using the two options on SYN and SYN+ACK segments in the TCP three-way handshake phase.

TS2 enables several applied mechanisms, as follows. When TS2 is enabled on a TCP connection, a local node MAY enable one or more of these mechanisms on the TCP connection without additional negotiation with a remote node.

- RTTM/TS2 (Round Trip Time Measurement with TS2) enables correct RTT measurements even when a duplicate ACK segment is received.

- PAWS/TS2 (Protection Against Wrapped Sequence numbers with TS2) is a reordering-robust protection mechanism for wrapped sequence numbers.

- PASA/TS2 (Protection Against Spoofing Attacks with TS2) is a lightweight protection mechanism against spoofing attacks that inject faked SYN, data, FIN, and RST segments.

- DLI/TS2 (Data Loss Inference with TS2) infers losses of both original and retransmitted data segments.

- SLID/TS2 (Spurious Loss Inference Detection with TS2) detects spurious loss inference without waiting for one RTT by sending arbitrary in-window data.

Note:The procedures described in this memo have not been demonstrated by simulation nor by implementation.

2. Terminology

2.1 General

This memo uses the same variable names and TCP state names defined in section 3.2 of [RFC793]. In addition, it introduces the following variables and notations: SND.MAX holds the maximum value of SND.NXT;

SSEG.XXX means the XXX field on the segment being sent; and RSEG.XXX means the XXX field on the segment just received.

The memo uses a variable "SND.FACK" [MM96][MSM99], which holds the highest sequence number known to have reached the receiver, plus one. An octet is "SACKed" if the octet has been reported in a TCP SACK option [RFC2018].

The memo uses the following abbreviations defined in [RFC2581]: SMSS (Sender Maximum Segment Size), and RMSS (Receiver Maximum Segment Size).

The memo uses the following abbreviations defined in [RFC2988]: RTO (Retransmission TimeOut), RTT (Round-Trip Time), SRTT (Smoothed RTT), and RTTVAR (RTT VARiation).

According to [RFC793], SEG.LEN includes the SYN and FIN bits. Thus, segments satisfying (RSEG.LEN > 0) include data, SYN, and FIN segments. If a RST segment has data, this memo does not consider that it satisfies (RSEG.LEN > 0). For simplicity, the term "data segments" often means "data, SYN, and/or FIN segments" in this memo.

The memo refers to the initial transmission of an octet as the "original transmission", and to a subsequent transmission of the same octet as a "retransmission" [RFC3522][RFC4015]. In addition, a data segment for which part or all of its octets are sent by original transmission is referred to as an "original data segment". Other data segments are referred to as "retransmitted data segments".

All arithmetic dealing with TCP sequence numbers must be performed modulo 2^32. A sequence is called "monotonically nondecreasing" if each number is greater than or equal to its predecessor.

2.2 Requirements

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 [RFC2119].

This memo makes use of conceptual variables to describe behavior. The specific variable names, and how their values are referred to and changed, are provided here to demonstrate behavior. Implementations are not required to follow the memo exactly, as long as its external behavior is consistent with that described here.

3. Two TCP Timestamps Options

TS2 uses two TCP options: "the TCP Timestamps option" specified in [RFC1323], and a new TCP option called "the TCP Old Timestamps option", as specified in this section.

3.1 TCP Timestamps Option

Figure 3-1 shows the format of the TCP Timestamps option.
For simplicity, it is hereafter called "the TS option".

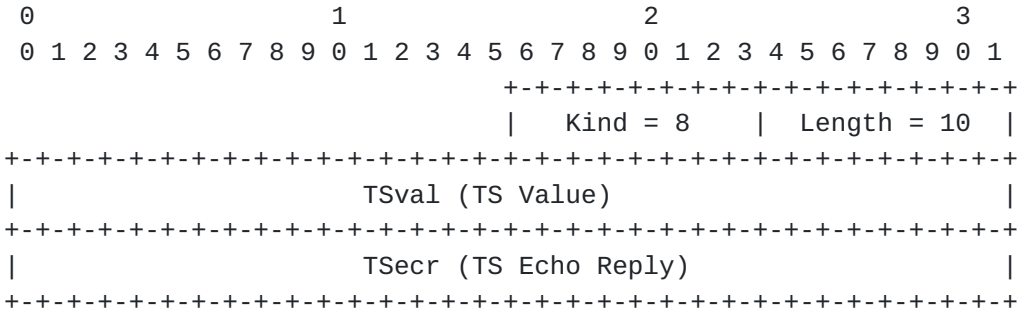


Figure 3-1: The TS option

3.2 TCP Old Timestamps Option

The format of the TCP Old Timestamps option has two forms as given below. The option-kind value is <<TBD>>.

On SYN and SYN+ACK segments, the TCP Old Timestamps option consists of only two octets (option-kind and option-length), as shown in Figure 3-2. The purpose of this form is to negotiate the use of TS2. For simplicity, this form is hereafter called "the OTS_OK option".

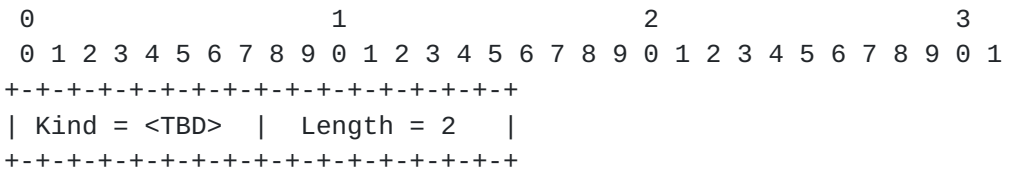


Figure 3-2: The OTS_OK option

On other segments, the format of the TCP Old Timestamps option is the same as that of the TS option, except for the option-kind value, as shown in Figure 3-3. The purpose of this form is to inform a remote node that the TSecr value is not fresh (In contrast, the TSecr value in the TS option is fresh). For simplicity, this form is hereafter called "the OTS option".

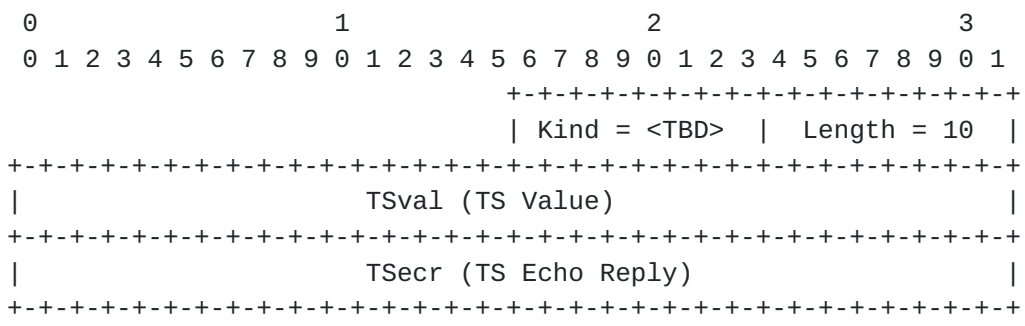


Figure 3-3: The OTS option

3.3 TSval and TSecr Fields

In both the TS option and the OTS option, the TSval field contains the current external timestamp, while the TSecr field contains the TS.Recent value, which is updated by the received TSval values, as specified in the base mechanism section.

When TS1 is enabled, the timestamp unit can be chosen between 1 second and 1 ms (10^-3), in order to be interoperable with [RFC1323]. When TS2 is enabled, the timestamp unit is fixed at 1 usec (10^-6). All arithmetic dealing with timestamps must be performed modulo 2^32.

4. Base Mechanism

This section describes how the timestamp mode (i.e., none, TS1, or TS2) is negotiated, how the timestamps option kind is chosen (i.e., the TS option or the OTS option), and how the values of SSEG.TSval and SSEG.TSecr on outgoing segments are computed.

4.1 Variables

The base mechanism uses the following variables: TS.Req (integer), TS.Mode (integer), TS.SndOff (32bit-timestamp), TS.SndAdj (32bit-timestamp), TS.Recent (32bit-timestamp), TS.RecentIsOld (boolean), and Last.Ack.Sent (32bit-sequence-number). Among these variables, only TS.Req, TS.Mode, and TS.SndOff are referred to by the applied mechanisms.

TS.Req contains the requested mode (0 = none, 1 = TS1, or 2 = TS2) of a TCP connection to be established. TS.Mode records the result of the mode negotiation. Its initial value is negative, which means "negotiation has not been completed".

TS.SndOff and TS.SndAdj help mode negotiation. See section 4.3 for more details.

TS.Recent holds the value to be echoed in the TSecr fields of both the TS option and the OTS option. Its initial value is zero. TS.RecentIsOld is accessed only when TS2 is enabled. It is true if any segment that satisfies (RSEG.LEN > 0) and carries the TS option or the OTS option has not been received after the TS.Recent value has last been echoed.

Last.Ack.Sent holds the last SSEG.ACK value sent, which is equal to the maximum SSEG.ACK value sent.

Note: TS.Recent and Last.Ack.Sent are inherited from [RFC1323].

4.2 Mode Negotiation

This subsection describes the procedure of mode negotiation using the two TCP Timestamps options on SYN and SYN+ACK segments in the TCP three-way handshake phase.

When a SYN segment is sent to establish a TCP connection, if TS2 is requested, the SYN segment SHOULD carry both the TS option and the OTS_OK option. If TS1 is requested, the SYN segment SHOULD carry the TS option, and it MUST NOT carry the OTS_OK option. If neither TS1 nor TS2 is requested, the SYN segment MUST NOT carry the TS option nor the OTS_OK option.

If a SYN (without ACK) segment is received in the LISTEN or SYN-SENT state, and if the received segment does not carry one or both of the TS option and the OTS_OK option, the SYN+ACK segment sent in reply MUST NOT carry the OTS_OK option. Similarly, if the received segment does not carry the TS option, the SYN+ACK segment sent in reply MUST NOT carry the TS option nor the OTS_OK option.

When a SYN segment is received in the LISTEN or SYN-SENT state, if TS2 is requested, the SYN+ACK segment sent in reply SHOULD carry both the TS option and the OTS_OK option as long as the above rule allows. If TS1 is requested, the SYN+ACK segment sent in reply SHOULD carry the TS option as long as the above rule allows, and it MUST NOT carry the OTS_OK option. If neither TS1 nor TS2 is requested, the SYN segment MUST NOT carry the TS option and the OTS_OK option.

On SYN and SYN+ACK segments in the TCP three-way handshake phase, if both the TS option and the OTS_OK option are exchanged, TS2 is enabled. If the TS option is exchanged but the OTS_OK option is not exchanged, TS1 is enabled. If the TS option is not exchanged, neither TS1 nor TS2 is enabled. The result is recorded in TS.Mode. When TS2 is enabled, TS.RecentIsOld is set to false.

4.3 Internal Timestamp and External Timestamp

In this memo, "internal timestamp" means a timestamp generated directly from a timestamp source such as an internal tick count or a real time clock. "External timestamp" means a timestamp exchanged in the TSval and TSecr fields. An external timestamp is calculated as an internal timestamp plus TS.SndOff.

TS.SndOff supports mode negotiation in conjunction with TS.SndAdj, which is used only during mode negotiation. Since the timestamp unit is different between TS1 and TS2, the timestamp values of TS1 and TS2 are almost always different. To save the TCP option space on SYN and SYN+ACK segments, however, only the TS option carries the TSval and TSecr fields. To be interoperable with [RFC1323], these two fields contain the timestamps of TS1. The purpose of TS.SndAdj is to adjust TS.SndOff to generate correct external timestamps for TS2 when TS2 is enabled. That is, when the first SYN segment is sent, TS.SndAdj records the difference between the timestamps of TS1 and TS2. Then, if TS2 is enabled when a SYN+ACK segment is received, TS.SndAdj is added to TS.SndOff.

The following formula shows the reason why TS.SndAdj holds the difference between the timestamps of TS1 and TS2.

```
CurTS(in TS2) = InitTS(in TS1) + ElapsedTS(in TS2)
              = InitTS(in TS1) + (CurTS(in TS2)  - InitTS(in TS2))
              = CurTS(in TS2)  + (InitTS(in TS1) - InitTS(in TS2))
              = CurTS(in TS2)  + TS.SndAdj
```

TS.SndOff is also used to avoid reusing the same range of TSval values when a TCP Control Block is reused. When a TCP Control Block is created, TS.SndOff is simply set to zero. On the other hand, when a TCP Control Block is reused, if TS2 is requested, the difference between the timestamps of TS2 and TS1 is added to TS.SndOff before the TCP three-way handshake phase begins to avoid reusing the same range of external timestamps when TS2 is enabled. If TS2 is not requested, TS.SndOff is unchanged.

In addition, TS.SndOff is used by PASA-DF/TS2 to randomize the initial timestamp values of TCP connections. See section 7.1.1 for more details.

4.4 Input Processing

This subsection describes the procedure for processing received segments.

If TS1 or TS2 is enabled, and if a received segment carrying the TS option or the OTS option satisfies at least one of inequalities (1) and (2) below, it SHOULD be processed by the base mechanism and the applied mechanisms to update their variables. Otherwise, those variables MUST NOT be updated, while the RSEG.TSval and RSEG.TSecr values on such a segment MAY be checked to test the received segment.

```
(RSEG.LEN > 0 &&
 Last.Ack.Sent - max(RCV.WND) < RSEG.SEQ + RSEG.LEN &&
 RSEG.SEQ < RCV.NXT + RCV.WND) ........................... (1)

   or

(RSEG.LEN == 0 &&
 Last.Ack.Sent - max(RCV.WND) <= RSEG.SEQ &&
 RSEG.SEQ < RCV.NXT + RCV.WND) ........................... (2)
```

When TS1 is enabled, if a received segment other than a RST segment carries the TS option and satisfies all of inequality (1), (RSEG.SEQ <= Last.ACK.sent), and (RSEG.TSval > TS.Recent), then RSEG.TSval is recorded in TS.Recent. In other words, TS.Recent holds the maximum RSEG.TSval value on in-sequence and duplicate data segments. Note that TS.Recent is not updated by out-of-order data segments, while it is updated by in-sequence and duplicate data segments. Also note that TS.Recent is monotonically nondecreasing.

When TS2 is enabled, if a received segment other than a SYN or RST segment does not carry the TS option nor the OTS option, it MUST be dropped, and an ACK segment SHOULD be sent in reply. If a received segment carries both the TS option and the OTS option, it MUST be dropped, and an ACK segment SHOULD be sent in reply. Otherwise, if a received segment other than a RST segment carries the TS option or the OTS option and satisfies inequality (1), and if TS.RecentIsOld is true or (RSEG.TSval < TS.Recent) is satisfied, then RSEG.TSval is recorded in TS.Recent, and TS.RecentIsOld is set to false. In other words, TS.Recent holds the minimum RSEG.TSval value on data segments received after a segment has last been sent. In contrast with TS1, note that TS.Recent is updated by out-of-order data segments, as well as in-sequence and duplicate data segments. Also note that TS.Recent is not monotonically nondecreasing.

Note: The reason why SYN and RST segments are handled specially is to disconnect half-open TCP connections.

4.5 Output Processing

This subsection describes the procedure for processing segments being sent.

When a segment carries the TS option or the OTS option, SSEG.TSval contains the current external timestamp value, and SSEG.TSecr contains the TS.Recent value unless otherwise specified below (i.e., <SSEG.TSval=CurrentExternalTS><SSEG.TSecr=TS.Recent>).

If TS1 is enabled, the rules below are followed.

- When a segment other than a RST segment is sent, it SHOULD carry the TS option.

- When a RST segment is sent, it SHOULD NOT carry the TS option.

Note: The reason why RST segments SHOULD NOT carry the TS option is to be interoperable with [RFC1323], which states in section 4.2 that "It is recommended that RST segments NOT carry timestamps, and that RST segments be acceptable regardless of their timestamp".

If TS2 is enabled, the rules below are followed.

- When a segment other than a RST segment is sent, if TS.RecentIsOld is false, the segment MUST carry the TS option, and TS.RecentIsOld is set to true. In contrast, if TS.RecentIsOld is true, the segment MUST carry the OTS option.

- When a RST segment is sent, it MUST carry the OTS option unless

otherwise specified below.

- When a RST segment is sent in reply to a received segment because of [RFC793], the rule below is followed.

  If the received segment carries the TS option or the OTS option, TS2 may be enabled on the remote node. Thus, to make the RST segment sent in reply acceptable to PAWS/TS2 and PASA/TS2 at the remote node, the RST segment MUST carry the OTS option, where SSEG.TSval is the RSEG.TSecr value and SSEG.TSecr is the RSEG.TSval value.
  (i.e., <SSEG.TSval=RSEG.TSecr><SSEG.TSecr=RSEG.TSval>)

  On the other hand, if the received segment does not carry the TS option nor the OTS option, TS1 may be enabled at the remote node, or neither of TS1 nor TS2 is enabled at the remote node. Thus, to make the RST segment sent in reply acceptable at the remote node in either case, the RST segment SHOULD NOT carry the TS option nor the OTS option.

Note: The reason why RST segments are handled specially is to disconnect half-open TCP connections.

For ACK segments sent in reply to invalid segments because of this memo, an ACK throttling mechanism SHOULD be implemented.

5. RTTM (Round Trip Time Measurement)

RTTM is an RTT measurement mechanism making use of the RSEG.TSecr field in the TS option on received segments. It can be enabled when either TS1 or TS2 is enabled. If a received segment does not satisfy inequality (1) nor (2), RTTM MUST NOT be performed on the segment.

The most significant difference between RTTM/TS1 (RTTM with TS1) and RTTM/TS2 (RTTM with TS2) is that RTTM/TS1 takes RTT measurements only when SND.UNA is advanced, while RTTM/TS2 takes RTT measurements whenever the TS option is received. Since both RTTM/TS1 and RTTM/TS2 take RTT measurements even if a received ACK segment was sent in reply to a retransmitted data segment, they replace Karn's algorithm [KP87].

Note: If a smoothed RTT is computed from many RTT measurements per RTT, the resulting SRTT, RTTVAR, and RTO values [RFC2988] would become short-sighted. Implementations should take care of this issue. The question of how to compute an RTO value from many measured RTTs is outside the scope of this memo.

5.1 RTTM/TS1

If TS1 is enabled, when SND.UNA is advanced, the RTT can be calculated as follows:

```
CurrentExternalTS(T1) - RSEG.TSecr + TS1_GRANULARITY
```

where TS1_GRANULARITY is the timestamp granularity of TS1.

> Note: "TS1_GRANULARITY" in the above expression SHOULD NOT be replaced with "TS1_GRANULARITY/2" unless TS1_GRANULARITY is much lower than any possible RTT.

Since there is a possibility that the RSEG.TSecr value is very old with TS1, the measured RTT may be longer than the real RTT in some corner cases. See appendix E.1 for more details.

Implementation Note: When the RSEG.TSecr value is zero, RTT SHOULD NOT be calculated. The reason is that some implementations of the TCP Timestamps option [RFC1323] send zero in the TSecr field in some scenarios where zero is obviously a bogus timestamp value.

5.2 RTTM/TS2

If TS2 is enabled, then whenever the TS option is received, the RTT can be calculated as follows:

```
CurrentExternalTS(T2) - RSEG.TSecr + TS2_GRANULARITY
```

where TS2_GRANULARITY is the timestamp granularity of TS2.

Note: "TS2_GRANULARITY" in the above expression SHOULD NOT be replaced with "TS2_GRANULARITY/2" unless TS2_GRANULARITY is much lower than any possible RTT.

When the OTS option is received, RTT SHOULD NOT be calculated, because the RSEG.TSecr value in the OTS option may be very old by definition.

6. PAWS (Protection Against Wrapped Sequence numbers)

PAWS is a mechanism for detecting old duplicate segments by making use of the RSEG.TSval field in the TS option and the OTS option on the received segment. It can be enabled when either TS1 or TS2 is enabled.

As described in appendix F, there is a possibility that a legitimate data segment could be discarded by PAWS in RFC1323 when it is delayed because of reordering. In contrast, as described in appendix E.2, PAWS/TS1 (PAWS with TS1) is slightly robust against reordering. And, PAWS/TS2 (PAWS with TS2) is robust against reordering, so that legitimate segments are unlikely to be discarded even when delayed because of reordering.

PAWS/TS1 and PAWS/TS2 use two variables: TS.RcvMin (32bit-timestamp) and TS.RcvMin_time (internal-time). TS.RcvMin holds the maximum value of the received RSEG.TSval values in both the TS option and the OTS option. TS.RcvMin_time holds the last time when a segment satisfying (RSEG.TSval >= TS.RcvMin) was received. The value of TS.RcvMin is valid for a limited amount of time depending on TS.Mode. If a received segment does not satisfy inequality (1) nor (2), these variables MUST NOT be updated.

> Note: TS.RcvMin is updated by any segment, while TS.Recent is updated only by segments satisfying (RSEG.LEN > 0). In addition, TS.RcvMin is always monotonically nondecreasing, in contrast to TS.Recent with TS2.

When the value of TS.RcvMin is valid, all received segments SHOULD be tested using TS.RcvMin, as described in the following subsections, before the acceptability test of [RFC793]. This test is called the PAWS test in this memo. To avoid discarding legitimate delayed segments due to reordering, the lower bound of acceptable RSEG.TSval values is chosen as slightly lower than TS.RcvMin, as suggested in the appendix F.5.

> Note: PAWS/TS1 and PAWS/TS2 use the dedicated variable TS.RcvMin for the PAWS test, while PAWS in [RFC1323] uses a shared variable TS.Recent.

6.1 PAWS/TS1

When TS1 is enabled, the minimum acceptable RSEG.TSval value is (TS.RcvMin - TS1_PAWS_MARGIN), where TS1_PAWS_MARGIN is a margin. Its appropriate value, such as RTO value, cannot be computed, however, because the unit of the received timestamp is unknown. Hence, this memo recommends TS1_PAWS_MARGIN = 1 simply because it is better than zero.

Thus, when the value of TS.RcvMin is valid, if a received segment other than a SYN or RST segment carries the TS option, it MUST satisfy (RSEG.TSval >= TS.RcvMin - TS1_PAWS_MARGIN). If it does not satisfy this inequality, it MUST be dropped, and an ACK segment with the TS option SHOULD be sent in reply.

Note: The reason why SYN and RST segments are not tested is to disconnect half-open TCP connections. Another reason why RST segments are not tested is to be interoperable with [RFC1323], which states in section 4.2 that "It is recommended that RST segments NOT carry timestamps, and that RST segments be acceptable regardless of their timestamp".

The value of TS.RcvMin is valid until the internal clock reaches (TS.RcvMin_time + TS1_PAWS_IDLE). TS1_PAWS_IDLE should be longer than the longest timeout, and it should be reasonably less than 2^31. The default value of TS1_PAWS_IDLE is 24 days, which is the same value specified in [RFC1323].

6.2 PAWS/TS2

6.2.1 Minimum Acceptable TSval Value

When TS2 is enabled, the minimum acceptable RSEG.TSval value is (TS.RcvMin - CurRTO), where CurRTO means the current RTO value. Thus, when the value of TS.RcvMin is valid, all received legitimate segments must satisfy (RSEG.TSval >= TS.RcvMin - CurRTO). If a received segment other than a SYN or RST segment does not satisfy this inequality, it MUST be dropped, and an ACK segment SHOULD be sent in reply. In addition, if a received RST segment with the OTS option does not satisfy this inequality, it MUST be dropped silently.

Note: The reason why RST segments without the OTS option and SYN segments are not tested here is to disconnect half-open TCP connections.

The value of TS.RcvMin is valid until the internal clock reaches (TS.RcvMin_time + TS2_PAWS_IDLE). TS2_PAWS_IDLE should be longer than the longest timeout, and it should be reasonably less than 2^31. The default value of TS2_PAWS_IDLE is 20 minutes (= 1200 seconds). (Note: 2^31 / 1000000 = 2147 seconds.)

Note 1: PAWS/TS2 assumes that RTTs measured at a local node and RTTs measured at a remote node are almost the same. Since the timestamp unit is fixed, the RTO value in the inequality of PAWS/TS2 can be evaluated under this assumption. This assumption might be wrong in some asymmetric networks. In addition, in a unidirectional data flow, a data receiver can take only one RTT measurement only when a SYN segment is exchanged. In such cases,

the robustness against reordering may be poor. Nevertheless, it would not be worse than that of PAWS in [RFC1323].

Note 2: An RTT may suddenly increase due to congestion, route changes, link-bandwidth changes, etc. Hence, the computation of RTO values should be done in a conservative manner.

6.2.2 Maximum Acceptable TSval Value

If the value of TS.RcvMin is valid, because the timestamp unit is fixed, the maximum acceptable value of RSEG.TSval can be calculated using TS.RcvMin and TS.RcvMin_time as follows.

TS.RcvMin + time2ts(CurrentTime - TS.RcvMin_time) + TS2_PAWS_DEV

where TS2_PAWS_DEV is the maximum acceptable deviation.

When a segment other than a RST segment without the OTS option or a SYN segment is received, if its RSEG.TSval value is greater than this maximum bound, the segment MAY be dropped. But if PASA-DF/TS2 is enabled, it SHOULD be dropped. If it is dropped, and if it is not a RST segment, an ACK segment SHOULD be sent in reply.

Note: The same expression with the replacement of "+ TS2_PAWS_DEV" with "- TS2_PAWS_DEV" cannot be applied to the minimum bound, because the RSEG.TSval values may be tweaked to lower values by PASA-DF/TS2 at the remote node.

7. PASA (Protection Against Spoofing Attacks)

PASA is a lightweight mechanism for protecting TCP connections against spoofing attacks injecting faked SYN, data, FIN, and RST segments. PASA can be enabled when TS2 is enabled. PASA does not work with TS1.

PASA/TS2 (PASA with TS2) consists of two parts. One is called PASA-DF/TS2 (PASA for Data and FIN segments with TS2). It detects spoofed data and FIN segments with the TS option or the OTS option by making use of received RSEG.TSecr values. It also detects spoofed RST segments with the OTS option by applying the same test. The other part is called PASA-SR/TS2 (PASA for SYN and RST segments with TS2). It enables both genuine RST segments without the OTS option and genuine SYN segments to trigger disconnection of their TCP connections, while spoofed segments are not allowed to trigger such disconnection. Since PASA-DF/TS2 and PASA-SR/TS2 are independent of each other, an implementation MAY support one or both of them.

Along with PASA, if the timestamp granularity is longer than 1 usec but methods described in appendix B are not implemented, lower bits of timestamps can be used as nonce bits to obfuscate timestamps.

7.1 PASA-DF/TS2 (PASA for Data and FIN Segments with TS2)

This subsection describes a mechanism called PASA-DF/TS2 that detects spoofed data and FIN segments with the TS option or the OTS option by making use of received RSEG.TSecr values. It also detects spoofed RST segments with the OTS option by applying the same test.

If a received segment does not satisfy inequality (1) nor (2), it MUST NOT update the variables of PASA-DF/TS2. In contrast, if PASA-DF/TS2 is enabled, any received segment SHOULD be tested by PASA-DF/TS2 before the acceptability test of [RFC793].

PASA-DF/TS2 uses four variables: TS.SndMin (32bit-timestamp), TS.SndMax (32bit-timestamp), TS.SndMax_time (internal-time), and TS.PASADF_On (boolean).

TS.SndMin holds the maximum value of the RSEG.TSecr field in the TS option and the OTS option on received segments satisfying inequality (1) or (2), while TS.SndMax holds the maximum value of the SSEG.TSval field on sent segments satisfying (SSEG.LEN > 0). For the first SYN or SYN+ACK segment sent, SSEG.TSval is copied to both TS.SndMin and TS.SndMax. Consequently, the received RSEG.TSecr values of the established TCP connection should be in the range from around TS.SndMin to TS.SndMax. The test of whether the received segments fit in this range is called the PASA-DF test in this memo. TS.SndMax_time holds the latest time when a segment satisfying

(SSEG.LEN > 0) is sent or when TS.SndOff is updated. TS.PASADF_On indicates whether the PASA-DF test can be performed. The initial value of TS.PASADF_On is true.

Note: The reason why PASA-DF/TS2 does not test RST segments without the OTS option and SYN segments is to disconnect half-open TCP connections. They are tested by PASA-SR/TS2, as described later.

When PASA-DF/TS2 is enabled, the maximum acceptable value of RSEG.TSval SHOULD be tested by PAWS/TS2. (See section 6.2.2)

7.1.1 External Timestamp Values

When PASA-DF/TS2 is enabled, TS.SndOff is utilized to randomize the initial SSEG.TSval value in order to obfuscate external timestamp values. If a TCP control block is reused by a new TCP connection, TS.SndOff MUST be increased by a random number in the range from 0 to TS2_PASADF_RNDMAX_REUSE, whose default value is 2^29 - 1 usec (about 9 minutes). In other cases, a newly generated 32-bit random number MUST be copied to TS.SndOff.

TS.SndOff is also utilized to minimize the difference between TS.SndMin and TS.SndMax after a long idle period. Since the possibility of accepting spoofed segments is the difference in 2^32, it is important to keep the difference small. Therefore, the advancement of TS.SndMax must have an upper bound. In addition, SSEG.TSval MUST be monotonically nondecreasing in order to make PAWS operable at a remote node. To satisfy these requirements, this memo proposes that the advancement of SSEG.TSval be no greater than TS2_PASADF_MAXADV, whose default value is 64 seconds. TS.SndOff MUST be tweaked as follows:

```
over_time = (CurrentTime - TS.SndMax_time) - TS2_PASADF_MAXADV;
if (over_time > 0) {
   TS.SndOff -= time2ts(over_time)
   TS.SndOff += RandomNumber(TS2_PASADF_RNDMAX_IDLE);
   TS.SndMax_time = CurrentTime;
}
```

Note 1: This memo assumes that CurrentTime is not wrapped in the lifetime of any TCP connections.

Note 2: RandomNumber(TS2_PASADF_RNDMAX_IDLE) above means a random number in the range from 0 to TS2_PASADF_RNDMAX_IDLE, whose default value is 2^26 - 1 usec (about 67 seconds).

7.1.2 Temporarily Suspension of Tests

There is a possibility that (TS.SndMax - TS.SndMin) becomes negative

when a data segment is sent after a series of sporadic transmissions that do not elicit any segments from a remote node. For example, consider a case where a TCP stack has received huge data, while its application reads the data very slowly. In this case, the TCP stack would send small window updates once in a while. During such a period, TS.SndMin and TS.SndMax are unchanged, while the SSEG.TSval values in those window updates are increasing. After a while, the difference between SSEG.TSval and TS.SndMin could be larger than 2^31. That is, (SSEG.TSval - TS.SndMin) could be negative. If a data segment was sent in that case, (TS.SndMax - TS.SndMin) also would be negative. To avoid confusion, the PASA-DF test MUST NOT be performed in such cases. Thus, this memo proposes the following procedure:

- When TS.PASADF_On is true, the PASA-DF test SHOULD be performed.

- When TS.PASADF_On is true, if (TS.SndMax - TS.SndMin) becomes negative after a segment is sent, TS.PASADF_On is set to false.

- When TS.PASADF_On is false, the PASA-DF test MUST NOT be performed.

- When TS.PASADF_On is false, if a received segment satisfies the requirement that (TS.SndMax - RSEG.TSecr) be non-negative, RSEG.TSecr is copied to TS.SndMin, and TS.PASADF_On is set to true.

This procedure is incorporated in the following two subsections.

7.1.3 Input Processing

This subsection describes the procedure when a segment is received.

When TS.PASADF_On is true, the procedure below is followed.

- In the CLOSED, LISTEN, and SYN-SENT states, RSEG.TSecr MUST NOT be tested.

- In other states, all segments other than SYN and RST segments must satisfy (TS.SndMin - CurRTO <= RSEG.TSecr <= TS.SndMax), where CurRTO means the current RTO value. When a segment other than a SYN or RST segment is received, if it does not satisfy this inequality, it MUST be dropped, and an ACK segment SHOULD be sent in reply. In addition, when a RST segment with the OTS option is received, if it does not satisfy the inequality, it MUST be dropped silently. Otherwise, when a segment other than a SYN or RST segment is received, or when a RST segment with the OTS option is received, if the received segment satisfies (RSEG.TSecr > TS.SndMin), then RSEG.TSecr is copied to

TS.SndMin.

Note: The reason why RST segments without the OTS option and SYN segments are not tested here is to disconnect half-open TCP connections.

When TS.PASADF_On is false, if a received segment satisfies the requirement that (TS.SndMax - RSEG.TSecr) be non-negative, RSEG.TSecr is copied to TS.SndMin, and TS.PASADF_On is set to true.

7.1.4 Output Processing

This subsection describes the procedure when a segment is sent.

When a segment satisfying (SSEG.LEN > 0) is sent, SSEG.TSval is copied to TS.SndMax, and the current time is recorded in TS.SndMax_time. If the segment is the first sent segment (e.g., the first SYN segment or the first SYN+ACK segment), SSEG.TSval is also copied to TS.SndMin.

When TS.PASADF_On is false, if (TS.SndMax - TS.SndMin) becomes negative after the above copying, TS.PASADF_On is set to false.

7.2 PASA-SR/TS2 (PASA for SYN and RST Segments with TS2)

This subsection describes a mechanism called PASA-SR/TS2, which enables both genuine RST segments without the OTS option and genuine SYN segments to trigger disconnection of their TCP connections, while spoofed ones are not allowed to trigger such disconnection.

If a received segment does not satisfy inequality (1) nor (2), it MUST NOT update the variables of PASA-SR/TS2. In contrast, if PASA-SR/TS2 is enabled, any received segment SHOULD be tested by PASA-SR/TS2 before the acceptability test of [RFC793].

Note: RST segments without the OTS option and SYN segments are not dropped by the base mechanism of TS2, PAWS/TS2, and PASA-DF/TS2 in order to disconnect half-open TCP connections.

7.2.1 Procedure

PASA-SR/TS2 uses the following variables: TS.PASASR_On (boolean) and TS.PASASR_time (internal-time). The initial value of TS.PASASR_On is false. TS.PASASR_time is valid only when TS.PASASR_On is true.

The procedure is as follows:

- When a SYN segment is received against an established TCP connection, regardless of whether it has the TS option or the

OTS option, it MUST be dropped, and an ACK segment without either the TS option or the OTS option SHOULD be sent in reply. The window size of the ACK segment is TS2_PASASR_WIN, which should be small enough (e.g., 1 RMSS). Then, TS.PASASR_On is set to true, and the current time is recorded in TS.PASASR_time.

- When TS.PASASR_On is true, if a received segment is not dropped by the base mechanism of TS2, the PAWS test, the PASA-DF test, and the acceptability test of [RFC793], then do the following: if (1) the received segment is not a RST segment, (2) it is a RST segment with the OTS option, or (3) a long time has been passed since the last ACK segment was sent in reply to a SYN segment (i.e., CurrentTime - TS.PASASR_time >= TS2_PASASR_TIME), then TS.PASASR_On is set to false.

- If a RST segment without the OTS option is received, and if TS.PASASR_On is false, or the segment does not satisfy (RCV.NXT <= RSEG.SEQ < RCV.NXT + TS2_PASASR_WIN), it MUST be dropped silently.

7.2.2 Examples

If a SYN segment is received against an established TCP connection, there are two possible causes. The first is that the remote node has been rebooted or disconnected silently and is trying to establish a new TCP connection with the same quadruple by chance. The second cause is that a malicious node sent a spoofed SYN segment with the same quadruple by chance.

If a RST segment without the OTS option is received against an established TCP connection, there are two possible causes. The first is that the remote node has been rebooted or disconnected silently and has sent a RST segment in reply to a segment sent by the local node. The second cause is that a malicious node sent a spoofed RST segment with the same quadruple by chance.

In any case, genuine SYN and RST segments should cause the TCP connection to disconnect, while spoofed SYN and RST segments should not cause it to disconnect.

The following four examples show how PASA-SR/TS2 would work against the four possible causes described above. Suppose that a local node and a remote node have an established TCP connection, and TS2 is enabled on it.

Case 1: The remote node has been rebooted or disconnected silently, and it sent a SYN segment to establish a new TCP connection with the same quadruple by chance.

When this genuine SYN segment is received by the local node, an ACK segment with window size = TS2_PASASR_WIN without either the TS option or the OTS option is sent in reply because of PASA-SR/TS2.

When the remote node receives this ACK segment, it sends a RST segment without the OTS option because it is in the SYN-SENT state.

Since the sequence number of this RST segment would satisfy (RCV.NXT <= RSEG.SEQ < RCV.NXT + TS2_PASASR_WIN), this RST segment would be accepted by the local node because of PASA-SR/TS2. Then, this RST segment would disconnect the existing TCP connection successfully.

After a while, another SYN segment will be retransmitted by the remote node, and a new TCP connection will be established.

Case 2: A malicious node sent a spoofed SYN segment with the same quadruple by chance.

When this spoofed SYN segment is received by the local node, an ACK segment with window size = TS2_PASASR_WIN without either the TS option or the OTS option is sent in reply because of PASA-SR/TS2. When the real remote node receives this ACK segment, since TS2 is enabled on the TCP connection, this ACK segment is dropped by the remote node, and an ACK segment is sent in reply.

The ACK segment sent in reply would be accepted by the local node. Fortunately, it would have no effect other than that the duplicate ACK counter could be falsely increased by one. Thus, the spoofed SYN segment would not disconnect the existing TCP connection.

Note: If the duplicate ACK counter is increased only by an ACK segment with the TS option, it would not be falsely increased in this case. See appendix C.

In a case where a spoofed RST segment is also sent just behind the spoofed SYN segment above, if the spoofed RST segment does not carry the OTS option, the possibility of accepting the spoofed RST segment is TS2_PASASR_WIN in 2^32. If the spoofed RST segment carries the OTS option, the possibility of accepting the spoofed RST segment is equal to the possibility of breaking PASA-DF/TS2. Both possibilities would be sufficiently small in most environments.

Case 3: The remote node sent a genuine RST segment without the OTS option to disconnect a TCP connection.

When TS2 is enabled, this case cannot happen, because a legitimate RST segment MUST carry the OTS option.

Case 4: A malicious node sent a spoofed RST segment without the OTS option with the same quadruple by chance.

Such spoofed RST segment is accepted only when TS.PASASR_On is true and (RCV.NXT <= RSEG.SEQ < RCV.NXT + TS2_PASASR_WIN) is satisfied. Thus, the possibility of accepting this spoofed RST segment is lower than TS2_PASASR_WIN in 2^32. This would be sufficiently small in most environments.

8. DLI (Data Loss Inference)

DLI is a mechanism for inferring losses of original and retransmitted data segments by making use of the RSEG.TSecr field in the TS option on received segments. It can be enabled only when TS2 is enabled. When a received segment does not satisfy inequality (1) nor (2), or it does not carry the TS option, DLI/TS2 (DLI with TS2) MUST NOT be performed on the segment.

> Note: When TS1 is enabled, the algorithms described in this section might infer losses of original data segments in limited scenarios that comprises partial acknowledgements. Since DLI with TS1 would not be able to infer losses of retransmitted data segments, however, this memo does not propose to run DLI with TS1.

DLI/TS2 improves overall throughput by reducing the number of retransmission timeouts under heavy loss rates.

8.1 Conceptual Algorithm

This subsection describes a conceptual algorithm of DLI/TS2 that infers losses of any original and retransmitted octets.

Two variables are associated with each sent octet: OC.SndTS (32bit-timestamp) and OC.SndRO (integer). OC.SndTS holds the SSEG.TSval value on the latest sent data segment containing the octet. The initial value of OC.SndRO is zero. When octets are retransmitted, the variables of these octets are reinitialized. If a segment with the TS option is received, every OC.SndRO of every octet satisfying (RSEG.TSecr > OC.SndTS) is increased by one. Then, every octet satisfying (OC.SndRO >= TS2_DLI_THRESH) is inferred lost. A real-world implementation would likely prefer to manage the retransmitted octets as sequence number ranges.

DLI/TS2 uses the RSEG.TSecr field in the TS option only, because, when out-of-order data exist in the receive buffer, all segments sent in reply to data segments carry the TS option, while other segments carry the OTS option. That is, the number of data segments arriving at the remote node is equal to the number of segments with the TS option departing from the remote node. To count the number of observed possible reorders precisely, any segments with the TS option (including data segments, window updates, etc.) SHOULD be counted, while any segments with the OTS option (including apparently pure ACK segment, etc.) MUST NOT be counted.

If the granularity of timestamps is coarser than the mean time between each data transmission, multiple data segments may carry the same SSEG.TSval value, and DLI/TS2 would be less effective. Some ideas to mitigate this problem is shown in appendix B.

TS2_DLI_THRESH indicates the number of observed possible reorders required to infer a loss. The default value is 3, which is the same value as the so-called duplicate acknowledgement threshold specified in [RFC2581]. TS2_DLI_THRESH might be implemented as an adaptive variable in the future.

8.2 Space-Optimized Algorithms

The conceptual algorithm, however, would not be easy to implement because of memory limitations. Therefore, this memo proposes the following space-optimized algorithms that do not require a huge memory space.

(1) DLI-SEG/TS2 infers losses of any original and retransmitted data segments. It uses two variables for each data segment.

(2) DLI-SACK/TS2 infers losses of original and retransmitted data in SACK holes, which exist between SND.UNA and SND.FACK. It uses two variables for each SACK hole.

(3) DLI-UNA/TS2 infers losses of original and retransmitted data at SND.UNA. It uses two variables for each TCP connection.

(4) DLI-NXT/TS2 infers losses of original and retransmitted data at SND.NXT minus one. When (SND.FACK < SND.NXT) is true, it infers losses of data between SND.FACK and SND.NXT. It uses two variables for each TCP connection.

(5) DLI-MAX/TS2 infers losses of original and retransmitted data at SND.MAX minus one. When (SND.NXT < SND.FACK) is true, it infers losses of data between SND.FACK and SND.MAX. It uses two variables for each TCP connection.

These algorithms can be implemented independently. Since DLI-SEG/TS2 is sufficiently powerful, however, if it is implemented, other algorithms (2)-(5) need not be implemented.

Note 1: Fast Retransmit [RFC2581] and SACK [RFC2018][RFC3517] are helpful for inferring losses of original data segments, while they cannot infer losses of retransmitted data segments in contrast to DLI/TS2. They are helpful, however, if DLI-UNA/TS2 is implemented but DLI-SEG/TS2 and DLI-SACK/TS2 are not implemented.
See appendix C.2 for more details.

Note 2: If some data have been retransmitted, losses of data between SND.UNA and the highest retransmitted sequence number cannot be inferred using IsLost() [RFC3517], while such losses can be inferred by DLI/TS2. See appendix C.2 for more details.

8.2.1 DLI-SEG/TS2 (DLI for Data Segments with TS2)

This subsection describes an algorithm called DLI-SEG/TS2 that infers losses of any original and retransmitted data segments. It is useful when each data segment is already managed by an internal data structure and is not repacketized.

DLI-SEG/TS2 uses two variables for each sent or retransmitted data segment: DS.SndTS (32bit-timestamp) and DS.SndRO (integer). DS.SndTS holds the SSEG.TSval value on the latest sent data segment. DS.SndRO counts the number of received segments with the TS option satisfying (RSEG.TSecr > DS.SndTS). In other words, it counts the number of observed possible reorders from the point of the view of the data segment.

The procedure is as follows.

When a data segment is sent or retransmitted, its SSEG.TSval value is recorded in DS.SndTS, and DS.SndRO is cleared.

When a segment with the TS option is received, every DS.SndRO of every data segment satisfying (RSEG.TSecr > DS.SndTS) is increased by one. Then, all data segments satisfying (DS.SndRO >= TS2_DLI_THRESH) are inferred lost.

> Implementation Hint: Prepare a chain of structures for data segments, sorted by DS.SndTS. When a new data segment is sent, a new structure for the data segment is allocated, SSEG.TSval is copied to DS.SndTS, and DS.SndRO is cleared; then the structure is inserted at the tail of the chain. When a data segment is retransmitted, SSEG.TSval is copied to DS.SndTS, and DS.SndRO is cleared; then the structure is moved to the tail of the chain. When a segment with the TS option is received, traverse the chain from the head while (RSEG.TSecr > DS.SndTS). For each structure satisfying this inequality, DS.SndRO is increased by one. Then, every structure satisfying (DS.SndRO >= TS2_DLI_THRESH) is inferred lost.

8.2.2 DLI-SACK/TS2 (DLI for Data in SACK Holes with TS2)

This subsection describes an algorithm called DLI-SACK/TS2 that infers losses of original and retransmitted data in SACK holes, which exist between SND.UNA and SND.FACK. It can be enabled when SACK is enabled.

DLI-SACK/TS2 uses two variables for each SACK hole: SH.SndTS (32bit-timestamp) and SH.SndRO (integer). SH.SndTS holds the SSEG.TSval value on the latest sent data segment containing data in the SACK hole. SH.SndRO counts the number of received segments with

the TS option satisfying (RSEG.TSecr > SH.SndTS). In other words, it counts the number of observed possible reorders from the point of the view of the data in the SACK hole since part or all of the data in the SACK hole were sent or retransmitted last time.

The procedure is as follows.

When a segment with the TCP SACK option is received, if a new SACK hole is created, then a new data structure is allocated, the received RSEG.TSecr value is copied to SH.SndTS, and SH.SndRO is cleared.

> Note: The SSEG.TSval values on the un-SACKed data segments in the new SACK hole likely would be no greater than the received RSEG.TSecr value. Thus, it would be safe to use the received RSEG.TSecr value for the initial value of SH.SndTS here.

If an existing SACK hole is split by a received SACK block, SH.SndTS and updated SH.SndRO are inherited to the split SACK holes. If an existing SACK hole is shrunken or expanded, SH.SndTS and SH.SndRO are unchanged.

When part or all of data in a SACK hole is retransmitted, the SSEG.TSval value on the data segment is copied to SH.SndTS, and SH.SndRO is cleared.

When a segment with the TS option is received, SH.SndRO of every SACK hole satisfying (RSEG.TSecr > SH.SndTS) is increased by one. Then, whole data in all SACK holes satisfying (SH.SndRO >= TS2_DLI_THRESH) are inferred lost.

> Implementation Hint: Prepare a chain of SACK holes, sorted by SH.SndTS. When a new SACK hole is created, RSEG.TSecr is copied to SH.SndTS, SH.SndRO is cleared, and the SACK hole is inserted at the tail of the chain. When data in a SACK hole is retransmitted, SSEG.TSval is copied to SH.SndTS, and SH.SndRO is cleared; then the SACK hole is moved to the tail of the chain. When a segment with the TS option is received, traverse the chain from the head while (RSEG.TSecr > SH.SndTS). For each SACK hole satisfying this inequality, SH.SndRO is increased by one. Then, every SACK hole satisfying (SH.SndRO >= TS2_DLI_THRESH) is inferred lost. If a SACK hole is split by a received SACK block, the split SACK holes inherit SH.SndTS, SH.SndRO, and the position in the chain.

8.2.3 DLI-UNA/TS2 (DLI for Data at SND.UNA with TS2)

This subsection describes an algorithm called DLI-UNA/TS2 that infers losses of original and retransmitted data at SND.UNA. It works only when (SND.UNA < SND.MAX) is true.

DLI-UNA/TS2 uses two variables: TS.UNA.SndTS (32bit-timestamp) and TS.UNA.SndRO (integer). TS.UNA.SndTS holds the SSEG.TSval value on the latest sent data segment containing data at SND.UNA. TS.UNA.SndRO counts the number of received segments with the TS option satisfying (RSEG.TSecr > TS.UNA.SndTS). In other words, it counts the number of observed possible reorders from the point of the view of data at SND.UNA since last sent. The variables have valid values only when (SND.UNA < SND.MAX) is true.

The procedure is as follows.

When data at SND.UNA is sent or retransmitted, the SSEG.TSval value on the data segment is recorded in TS.UNA.SndTS, and TS.UNA.SndRO is cleared.

When (SND.UNA < SND.MAX) is true, if a received segment does not advance SND.UNA (i.e., RSEG.ACK <= SND.UNA), and if it carries the TS option and satisfies (RSEG.TSecr > TS.UNA.SndTS), then TS.UNA.SndRO is increased by one. After that, if (TS.UNA.SndRO >= TS2_DLI_THRESH) is true, data at SND.UNA is inferred lost. Otherwise, if a received segment advances SND.UNA (i.e., old SND.UNA < RSEG.ACK <= SND.MAX), the current external timestamp value is copied to TS.UNA.SndTS, and TS.UNA.SndRO is cleared.

> Implementation Note: If PASA-DF/TS2 is enabled, then when SND.UNA is advanced, for time-optimization TS.SndMax can be copied to TS.UNA.SndTS, instead of the current external timestamp value.

If DLI-SACK/TS2 is enabled, when SND.UNA is advanced by a received segment (i.e., old SND.UNA < RSEG.ACK <= SND.MAX), and if the new SND.UNA is in an existing SACK hole, SH.SndTS and SH.SndRO of the SACK hole are copied to TS.UNA.SndTS and TS.UNA.SndRO, respectively.

8.2.4 DLI-NXT/TS2 (DLI for Data at SND.NXT minus one with TS2)

This subsection describes an algorithm called DLI-NXT/TS2 that infers losses of data at SND.NXT-1. It works only when (SND.UNA < SND.NXT) is true. When (SND.FACK < SND.NXT) is true, it infers losses of original and retransmitted data between SND.FACK and SND.NXT.

DLI-NXT/TS2 uses two variables: TS.NXT.SndTS (32bit-timestamp) and TS.NXT.SndRO (integer). TS.NXT.SndTS holds the SSEG.TSval value on the latest sent data segment containing the data at SND.NXT-1. TS.NXT.SndRO counts the number of received segments with the TS option satisfying (RSEG.TSecr > TS.NXT.SndTS). In other words, it counts the number of observed possible reorders from the point of the view of the data at SND.NXT-1 since last sent. The variables have valid values when the data at SND.NXT-1 have not been acknowledged nor SACKed.

The procedure is as follows.

When the data at SND.NXT is sent, or when (SND.FACK < SND.NXT) is true and part or all of data between SND.FACK and SND.NXT is retransmitted, if previously received window size is not zero (i.e., TCP persist timer is off), then the SSEG.TSval value on the data segment is recorded in TS.NXT.SndTS, and TS.NXT.SndRO is cleared.

When the data at SND.NXT-1 have not been acknowledged nor SACKed, if a segment with the TS option satisfying (RSEG.TSecr > TS.NXT.SndTS) is received, then TS.NXT.SndRO is increased by one. After that, if (TS.NXT.SndRO >= TS2_DLI_THRESH), the data at SND.NXT-1 is inferred lost. In this case, if (SND.FACK < SND.NXT) is also true, the data between SND.FACK and SND.NXT is inferred lost.

8.2.5 DLI-MAX/TS2 (DLI for Data at SND.MAX minus one with TS2)

This subsection describes an algorithm called DLI-MAX/TS2 that infers losses of data at SND.MAX-1. It works only when (SND.UNA < SND.MAX) is true. When (SND.NXT < SND.FACK) is true, it infers losses of original and retransmitted data between SND.FACK and SND.MAX.

DLI-MAX/TS2 uses two variables: TS.MAX.SndTS (32bit-timestamp) and TS.MAX.SndRO (integer). TS.MAX.SndTS holds the SSEG.TSval value on the latest sent data segment containing the data at SND.MAX-1. TS.MAX.SndRO counts the number of received segments with the TS option satisfying (RSEG.TSecr > TS.MAX.SndTS). In other words, it counts the number of observed possible reorders from the point of the view of the data at SND.MAX-1 since last sent. The variables have valid values when the data at SND.MAX-1 have not been acknowledged nor SACKed.

The procedure is as follows.

When the data at SND.MAX is sent, or when (SND.NXT < SND.FACK) is true and part or all of data between SND.FACK and SND.MAX is retransmitted, if previously received window size is not zero (i.e., TCP persist timer is off), then the SSEG.TSval value on the data segment is recorded in TS.MAX.SndTS, and TS.MAX.SndRO is cleared.

When the data at SND.MAX-1 have not been acknowledged nor SACKed, if a segment with the TS option satisfying (RSEG.TSecr > TS.MAX.SndTS) is received, then TS.MAX.SndRO is increased by one. After that, if (TS.MAX.SndRO >= TS2_DLI_THRESH), the data at SND.MAX-1 is inferred lost. In this case, if (SND.NXT < SND.FACK) is also true, the data between SND.FACK and SND.MAX is inferred lost.

9. SLID (Spurious Loss Inference Detection)

SLID is a mechanism for detecting spurious loss inference by making use of the received RSEG.TSecr value in the TS option and the OTS option. SLID can be enabled when TS2 is enabled. When a received segment does not satisfy inequality (1) nor (2), or it does not carry the TS option nor the OTS option, then SLID/TS2 (SLID with TS2) MUST NOT be performed on the segment.

To decide whether a loss inference is genuine or spurious, when a loss is inferred, SLID/TS2 first sends an arbitrary in-window data segment as a probe, then examines incoming segments until a decision is made. Probe data segment may not contain the data being probed. Therefore, it may consist of unsent data, decidedly lost data, or inferredly lost data.

SLID/TS2 makes it possible to postpone the retransmission of data that has been inferred lost until the loss inference is decided to be genuine, in order to avoid wasting bandwidth and transmission battery power on unnecessary retransmissions. It also makes it possible to postpone the reduction of congestion window until the loss inference is decided to be genuine. In addition, SLID/TS2 can be applied to detect a posteriori spurious retransmissions in order to alleviate unnecessary data retransmissions and duplicate acknowledgements. Response algorithms are outside the scope of this memo.

9.1 Conceptual Algorithm

This subsection describes a conceptual algorithm of SLID/TS2 that detects spurious loss inference of any octet.

Two variables are associated with each sent octet: OC.TgtTS (32bit-timestamp) and OC.PrbTS (32bit-timestamp). OC.TgtTS (target timestamp) holds the latest SSEG.TSval value on the original or retransmitted decidedly-lost segments containing the octet. OC.PrbTS (probe timestamp) holds the SSEG.TSval value on the first data segment that is sent since the octet has been inferred lost. This sent segment is called "probe segment" in this memo. It may not contain the octet being probed.

After sending a probe segment, every incoming segment is examined until all necessary decisions are made for every octet that is inferred lost. More specifically, the RSEG.TSecr values of every incoming segment is compared with the "border timestamp" of each octet that has been inferred lost in order to decide whether each loss inference is genuine or spurious.

The border timestamp is calculated as follows:

TS2_SLID_BTS(TgtTS, PrbTS) = (k * TgtTS + (1 - k) * PrbTS),

where k = 1/2.

If a received RSEG.TSecr value is greater than the border timestamps of some octets, the loss inferences of those octets are decided to be genuine. Otherwise, if a received segment acknowledges or SACKs some octets, the loss inferences of those octets are decided to be spurious. In other cases, another segment is awaited. The reason why received RSEG.TSecr values are compared with the border timestamp of each octet instead of OC.TgtTS is shown in appendix G.4.

The precise procedure is as follows.

To reset the target timestamp, when an octet is first sent, or when a decidedly lost octet is retransmitted, the SSEG.TSval value on the sent data segment is recorded in both OC.TgtTS and OC.PrbTS.

To set the probe timestamp, when any data segment is sent, the SSEG.TSval value on the segment is copied to OC.PrbTS of all octets that have been inferred lost and satisfy (OC.TgtTS == OC.PrbTS). The sent data segment may carry these octets.

To make a decision, when a segment with the TS option or the OTS option is received, for any octet satisfying (OC.TgtTS < OC.PrbTS), if (RSEG.TSecr > TS2_SLID_BTS(OC.TgtTS, OC.PrbTS)) is true, then its loss inference is decided to be genuine. Otherwise, if such an octet is acknowledged or SACKed by the segment, then its loss inference is decided to be spurious. In other cases, another segment is awaited.

To avoid incorrect decision due to underflow, when any segment is sent, for any octet that is inferred lost, if (SSEG.TSval - OC.TgtTS) is negative, then its loss inference is decided to be spurious.

When a decision is made, an appropriate response algorithm such as [RFC4015] is executed, then OC.PrbTS is copied to OC.TgtTS to terminate this procedure.

Note: (OC.TgtTS < OC.PrbTS) is true only when the octet has been inferred lost and has been probed. After a decision is made, (OC.TgtTS == OC.PrbTS) becomes true.

A real-world implementation would likely prefer to manage the octets as sequence number ranges.

9.2 Space-Optimized Algorithms

The conceptual algorithm, however, would not be easy to implement because of memory limitations. Therefore, this memo proposes the

following space-optimized algorithms that do not require a huge memory space.

(1) SLID-SEG/TS2 detects spurious loss inference of any data segments. It uses two variables for each data segment.

(2) SLID-SACK/TS2 detects spurious loss inference of data in SACK holes, which exist between SND.UNA and SND.FACK. It uses two variables for each SACK hole.

(3) SLID-UNA/TS2 detects spurious loss inference of data at SND.UNA. It uses two variables for each TCP connection.

(4) SLID-NXT/TS2 detects spurious loss inference of data at SND.NXT minus one. When (SND.FACK < SND.NXT) is true, it detects spurious loss inference of data between SND.FACK and SND.NXT. It uses two variables for each TCP connection.

(5) SLID-MAX/TS2 detects spurious loss inference of data at SND.MAX minus one. When (SND.NXT < SND.FACK) is true, it detects spurious loss inference of data between SND.FACK and SND.MAX. It uses two variables for each TCP connection.

These algorithms can be implemented independently. Since SLID-SEG/TS2 is sufficiently powerful, however, if it is implemented, other algorithms (2)-(5) need not be implemented.

9.2.1 SLID-SEG/TS2 (SLID for Data Segments with TS2)

This subsection describes an algorithm called SLID-SEG/TS2 that detects spurious loss inference of any sent data segments. It is useful when each data segment is already managed by an internal data structure and is not repacketized.

SLID-SEG/TS2 uses two variables for each sent or retransmitted data segment: DS.TgtTS (32bit-timestamp) and DS.PrbTS (32bit-timestamp). DS.TgtTS (target timestamp) holds the latest SSEG.TSval value on the original or retransmitted decidedly-lost data segments. DS.PrbTS (probe timestamp) holds the SSEG.TSval value on the first data segment that is sent since the data segment has been inferred lost. The sent segment may be different from the data segment being probed.

The procedure is as follows.

To reset the target timestamp, when a data segment is first sent, or when a decidedly lost data segment is retransmitted, the SSEG.TSval value on the sent data segment is recorded in both DS.TgtTS and DS.PrbTS.

To set the probe timestamp, when any data segment is sent, the SSEG.TSval value on the segment is copied to DS.PrbTS of all data segment structures that have been inferred lost, and satisfy (DS.TgtTS == DS.PrbTS). The sent data segment may be the same as one of these data structures.

To make a decision, when a segment with the TS option or the OTS option is received, for any data segment structure satisfying (DS.TgtTS < DS.PrbTS), if the received RSEG.TSecr value is greater than TS2_SLID_BTS(DS.TgtTS, DS.PrbTS), then its loss inference is decided to be genuine. Otherwise, if such data segment structure is acknowledged or SACKed by the segment, then its loss inference is decided to be spurious. In other cases, another segment is awaited.

To avoid incorrect decision due to underflow, when any segment is sent, for any data segment structure that is inferred lost, if (SSEG.TSval - DS.TgtTS) is negative, then its loss inference is decided to be spurious.

When a decision is made, an appropriate response algorithm is executed, then DS.PrbTS is copied to DS.TgtTS to terminate this procedure.

Note: (DS.TgtTS < DS.PrbTS) is true only when the data segment has been inferred lost and has been probed. After a decision is made, (DS.TgtTS == DS.PrbTS) becomes true.

The implementation hint for DLI-SEG/TS2 might be of help in implementing SLID-SEG/TS2.

9.2.2 SLID-SACK/TS2 (SLID for Data in SACK Holes with TS2)

This subsection describes an algorithm called SLID-SACK/TS2 that detects spurious loss inference of data segments containing data in SACK holes, which exist between SND.UNA and SND.FACK. The implementation hint for DLI-SACK/TS2 might be of help in implementing SLID-SACK/TS2.

SLID-SACK/TS2 uses two variables for each SACK hole: SH.TgtTS (32bit-timestamp) and SH.PrbTS (32bit-timestamp). SH.TgtTS (target timestamp) holds the RSEG.TSecr value on the segment which created the SACK hole. SH.PrbTS (probe timestamp) holds the SSEG.TSval value on the first data segment that is sent since the data in the SACK hole has been inferred lost. The sent segment may not carry the data being probed.

The procedure is as follows.

To reset the target timestamp, when a segment with the TS option or

the OTS option is received, if a SACK hole is created by the SACK blocks on the segment, its RSEG.TSecr value is recorded in both SH.TgtTS and SH.PrbTS of the created SACK hole. If a SACK hole is expanded by the SACK blocks on the segment, the RSEG.TSecr value is recorded in both SH.TgtTS and SH.PrbTS of the expanded SACK hole.

Note: The SSEG.TSval values on the lost data segments in a new SACK hole likely would be less than the received RSEG.TSecr value. Therefore, it would be safe to use the RSEG.TSecr value of the received segment which created the SACK hole as initial values of SH.TgtTS and SH.PrbTS.

To set the probe timestamp, when any data segment is sent, the SSEG.TSval value on the segment is copied to SH.PrbTS of all SACK holes that are inferred lost and satisfy (SH.TgtTS == SH.PrbTS). The sent data segment may carry data in one of these SACK holes.

To make a decision, when a segment with the TS option or the OTS option is received, for any SACK hole with (SH.TgtTS < SH.PrbTS), if (RSEG.TSecr > TS2_SLID_BTS(SH.TgtTS, SH.PrbTS)) is true, then its loss inference is decided to be genuine. Otherwise, if part or all of data in such SACK hole is acknowledged or SACKed by the segment, then its loss inference is decided to be spurious. In other cases, another segment is awaited.

To avoid incorrect decision due to underflow, when any segment is sent, for any SACK hole that is inferred lost, if (SSEG.TSval - SH.TgtTS) is negative, then its loss inference is decided to be spurious.

When a decision is made, an appropriate response algorithm is executed, then SH.PrbTS is copied to SH.TgtTS to terminate this procedure.

Note: (SH.TgtTS < SH.PrbTS) is true only when the data in the SACK hole has been inferred lost and has been probed. After a decision is made, (SH.TgtTS == SH.PrbTS) becomes true.

9.2.3 SLID-UNA/TS2 (SLID for Data at SND.UNA with TS2)

This subsection describes an algorithm called SLID-UNA/TS2 that detects spurious loss inference of data segment containing data at SND.UNA. It works only when (SND.UNA < SND.MAX) is true.

SLID-UNA/TS2 uses two variables: TS.UNA.TgtTS (32bit-timestamp) and TS.UNA.PrbTS (32bit-timestamp). TS.UNA.TgtTS (target timestamp) holds the closest subsequent timestamp value to the latest SSEG.TSval value on original or retransmitted decidedly-lost data segments containing data at SND.UNA. TS.UNA.PrbTS (probe timestamp) holds the

SSEG.TSval value on the first data segment that is sent since the data at SND.UNA has been inferred lost. The sent segment may not contain the data at SND.UNA. The initial values of both TS.UNA.TgtTS and TS.UNA.PrbTS are equal to the SSEG.TSval value on the first sent SYN segment. The variables have valid values if (SND.UNA < SND.MAX) is true.

The procedure is as follows.

To reset the target timestamp, when an original or decidedly lost data segment containing data at SND.UNA is sent, the SSEG.TSval value is recorded in both TS.UNA.TgtTS and TS.UNA.PrbTS. In addition, when a received segment with the TS option or the OTS option advances SND.UNA (i.e., old SND.UNA < RSEG.ACK <= SND.MAX), RSEG.TSecr is copied to TS.UNA.PrbTS if (RSEG.TSecr > TS.UNA.PrbTS) is true, then TS.UNA.PrbTS is copied to TS.UNA.TgtTS.

To set the probe timestamp, when any data segment is sent, if data at SND.UNA is inferred lost and (TS.UNA.TgtTS == TS.UNA.PrbTS) is true, then the SSEG.TSval value on the sent data segment is recorded in TS.UNA.PrbTS. The sent data segment may carry data at SND.UNA.

To make a decision, when a segment with the TS option or the OTS option is received, if (TS.UNA.TgtTS < TS.UNA.PrbTS) is true, and (RSEG.TSecr > TS2_SLID_BTS(TS.UNA.TgtTS, TS.UNA.PrbTS)) is true, then the loss inference is decided to be genuine. Otherwise, if SND.UNA is advanced by the segment (i.e., old SND.UNA < RSEG.ACK <= SND.MAX), the loss inference is decided to be spurious. In other cases, another segment is awaited.

To avoid incorrect decision due to underflow, when any segment is sent, if the data at SND.UNA is inferred lost, and if (SSEG.TSval - TS.UNA.TgtTS) is negative, then the loss inference is decided to be spurious.

When a decision is made, an appropriate response algorithm is executed. Then, if (RSEG.TSecr > TS.UNA.PrbTS) is satisfied, RSEG.TSecr is copied to TS.UNA.PrbTS. After that, TS.UNA.PrbTS is copied to TS.UNA.TgtTS to terminate this procedure.

Note: (TS.UNA.TgtTS < TS.UNA.PrbTS) is true only when data at SND.UNA has been inferred lost and has been probed. After a decision is made, (TS.UNA.TgtTS == TS.UNA.PrbTS) becomes true.

If SLID-SACK/TS2 is enabled, when SND.UNA is advanced by a received segment (i.e., old SND.UNA < RSEG.ACK <= SND.MAX), if the new SND.UNA is in an existing SACK hole, SH.TgtTS and SH.PrbTS of the SACK hole are copied to TS.UNA.TgtTS and TS.UNA.PrbTS, respectively.

9.2.4 SLID-NXT/TS2 (SLID for Data at SND.NXT minus one with TS2)

This subsection describes an algorithm called SLID-NXT/TS2 that detects spurious loss inference of data segment containing data at SND.NXT-1. It works only when (SND.UNA < SND.NXT) is true. When (SND.FACK < SND.NXT) is true, it detects spurious loss inference of data segment containing data between SND.FACK and SND.NXT.

SLID-NXT/TS2 uses two variables: TS.NXT.TgtTS (32bit-timestamp) and TS.NXT.PrbTS (32bit-timestamp). TS.NXT.TgtTS (target timestamp) holds the latest SSEG.TSval value on the original or retransmitted decidedly-lost data segments containing data at SND.NXT-1. TS.NXT.PrbTS (probe timestamp) holds the SSEG.TSval value on the first data segment that is sent since the data at SND.NXT-1 has been inferred lost. The sent segment may not contain the data at SND.NXT-1. The initial values of both TS.NXT.TgtTS and TS.NXT.PrbTS are equal to the SSEG.TSval value on the first sent SYN segment. The variables have valid values if (SND.UNA < SND.NXT) is true.

The procedure is as follows.

To reset the target timestamp, when data at SND.NXT is sent, the SSEG.TSval value on the data segment is recorded in both TS.NXT.TgtTS and TS.NXT.PrbTS. Note that SND.NXT is increased by SSEG.LEN after the data segment is sent.

To set the probe timestamp, when any data segment is sent, if it does not contain data at SND.NXT, data at SND.NXT-1 is inferred lost, (TS.NXT.TgtTS == TS.NXT.PrbTS) is true, and previously received window size is not zero (i.e., TCP persist timer is off), then the SSEG.TSval value on the sent data segment is recorded in TS.NXT.PrbTS. The sent data segment may carry data at SND.NXT-1.

To make a decision, when a segment with the TS option or the OTS option is received, if (TS.NXT.TgtTS < TS.NXT.PrbTS) is true, and (RSEG.TSecr > TS2_SLID_BTS(TS.NXT.TgtTS, TS.NXT.PrbTS)) is true, then the loss inference is decided to be genuine. Otherwise, if the data is acknowledged or SACKed by the segment, then the loss inference is decided to be spurious. In other cases, another segment is awaited.

To avoid incorrect decision due to underflow, when any segment is sent, if the data at SND.NXT-1 is inferred lost, and if (SSEG.TSval - TS.NXT.TgtTS) is negative, then the loss inference is decided to be spurious.

When a decision is made, an appropriate response algorithm is executed, and TS.NXT.PrbTS is copied to TS.NXT.TgtTS to terminate this procedure. When (SND.FACK < SND.NXT) is true, the decision is applied to all data segments containing data between SND.FACK and

SND.NXT.

Note: (TS.NXT.TgtTS < TS.NXT.PrbTS) is true only when data at SND.NXT-1 has been inferred lost and has been probed. After a decision is made, (TS.NXT.TgtTS == TS.NXT.PrbTS) becomes true.

9.2.5 SLID-MAX/TS2 (SLID for Data at SND.MAX minus one with TS2)

This subsection describes an algorithm called SLID-MAX/TS2 that detects spurious loss inference of data segment containing data at SND.MAX-1. It works only when (SND.UNA < SND.MAX) is true. When (SND.NXT < SND.FACK) is true, it detects spurious loss inference of data segment containing data between SND.FACK and SND.MAX.

SLID-MAX/TS2 uses two variables: TS.MAX.TgtTS (32bit-timestamp) and TS.MAX.PrbTS (32bit-timestamp). TS.MAX.TgtTS (target timestamp) holds the latest SSEG.TSval value on the original or retransmitted decidedly-lost data segments containing data at SND.MAX-1. TS.MAX.PrbTS (probe timestamp) holds the SSEG.TSval value on the first data segment that is sent since the data at SND.MAX-1 has been inferred lost. The sent segment may not contain the data at SND.MAX-1. The initial values of both TS.MAX.TgtTS and TS.MAX.PrbTS are equal to the SSEG.TSval value on the first sent SYN segment. The variables have valid values if (SND.UNA < SND.MAX) is true.

The procedure is as follows.

To reset the target timestamp, when data at SND.MAX is sent, the SSEG.TSval value on the data segment is recorded in both TS.MAX.TgtTS and TS.MAX.PrbTS. Note that SND.MAX is increased by SSEG.LEN after the data segment is sent.

To set the probe timestamp, when any data segment is sent, if it does not contain data at SND.MAX, data at SND.MAX-1 is inferred lost, (TS.MAX.TgtTS == TS.MAX.PrbTS) is true, and previously received window size is not zero (i.e., TCP persist timer is off), then the SSEG.TSval value on the sent data segment is recorded in TS.MAX.PrbTS. The sent data segment may carry data at SND.MAX-1.

To make a decision, when a segment with the TS option or the OTS option is received, if (TS.MAX.TgtTS < TS.MAX.PrbTS) is true, and (RSEG.TSecr > TS2_SLID_BTS(TS.MAX.TgtTS, TS.MAX.PrbTS)) is true, then the loss inference is decided to be genuine. Otherwise, if the data is acknowledged or SACKed by the segment, then the loss inference is decided to be spurious. In other cases, another segment is awaited.

To avoid incorrect decision due to underflow, when any segment is sent, if the data at SND.MAX-1 is inferred lost, and if (SSEG.TSval - TS.MAX.TgtTS) is negative, then the loss inference is

decided to be spurious.

When a decision is made, an appropriate response algorithm is executed, and TS.MAX.PrbTS is copied to TS.MAX.TgtTS to terminate this procedure. When (SND.NXT < SND.FACK) is true, the decision is applied to all data segments containing data between SND.FACK and SND.MAX.

Note: (TS.MAX.TgtTS < TS.MAX.PrbTS) is true only when data at SND.MAX-1 has been inferred lost and has been probed. After a decision is made, (TS.MAX.TgtTS == TS.MAX.PrbTS) becomes true.

10. Security Considerations

PASA/TS2 is a lightweight protection mechanism against spoofing attacks injecting faked SYN, data, FIN, and RST segments.

The vulnerability described in [CVE05] and [CERT05] is mitigated in PAWS/TS1 and PAWS/TS2 because of inequalities (1) and (2). When TS2 is enabled, it is also mitigated by PASA/TS2.

11. IANA Considerations

The option-kind value of the TCP Old Timestamps option needs to be assigned.

12. Acknowledgements

The TCP Timestamps option was originally specified in [RFC1323] by Van Jacobson, Bob Braden, and Dave Borman. Many ideas in this memo are thus inherited from it.

The TS.Recent update rule of TS2 was inspired by Reiner Ludwig [Lud03a][Lud03b]. The idea of detecting spurious loss inference by making use of the TSecr field was inspired by the Eifel Detection Algorithm [RFC3522] proposed by Reiner Ludwig.

The idea of detecting spoofed segments by making use of the TSecr field was proposed by Kacheong Poon. He has given the author invaluable insights, ideas, and comments on timestamp handling through discussions on [PD04], etc.

13. References

13.1 Normative References

[RFC793] J. Postel, "Transmission Control Protocol", STD7, RFC793, September 1981.

[RFC1323] V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance", RFC1323, May 1992.

[RFC2119] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", BCP14, RFC2119, March 1997.

[RFC2581] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control", RFC2581, April 1999.

[RFC2988] V. Paxson and M. Allman, "Computing TCP's Retransmission Timer", RFC2988, November 2000.

[RFC3522] R. Ludwig and M. Meyer, "The Eifel Detection Algorithm for TCP", RFC3522, April 2003.

13.2 Informative References

[RFC1122] R. Braden, Editor, "Requirements for Internet Hosts - Communication Layers", RFC1122, October 1989.

[RFC2018] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgement Options", RFC2018, October 1996.

[RFC2385] A. Heffernan, "Protection of BGP Sessions via the TCP MD5 Signature Option", RFC2385, August 1998.

[RFC2883] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP", RFC2883, July 2000.

[RFC3042] M. Allman, H. Balakrishnan, and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit", RFC3042, January 2001.

[RFC3465] M. Allman, "TCP Congestion Control with Appropriate Byte Counting (ABC)", RFC3465, February 2003.

[RFC3517] E. Blanton, M. Allman, K. Fall, and L. Wang, "A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP", RFC3517, April 2003.

[RFC3708] E. Blanton and M. Allman, "Using TCP Duplicate Selective Acknowledgement (DSACKs) and Stream Control Transmission Protocol (SCTP) Duplicate Transmission Sequence Numbers (TSNs) to Detect Spurious Retransmissions", RFC3708, February 2004.

[RFC3782] S. Floyd, T. Henderson, and A. Gurtov, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC3582, April 2004.

[RFC4015] R. Ludwig and A. Gurtov, "The Eifel Response Algorithm for TCP", RFC4015, February 2005.

[All04] M. Allman, "Re: [tcpm] long options draft revision", the IETF TCPM WG mailing list, September 2004. URL "http://www1.ietf.org/mail-archive/web/tcpm/current/

msg00748.html"

[Bra93] R. Braden, "TCP Extensions for High Performance: An Update", (work in progress), Internet-Draft, June 1993. URL "http://www.kohala.com/start/tcplw-extensions.txt"

[CERT05] US-CERT, "Vulnerability Note VU#637934", May 2005. URL "http://www.kb.cert.org/vuls/id/637934"

[CVE05] CVE-2005-0356, May 2005. URL "http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=2005-0356"

[Duk03a] M. Duke, "[Tsvwg] Updating timestamps (ts_recent) in Linux", the IETF TSVWG WG mailing list, August 2003. URL "http://www1.ietf.org/mail-archive/web/tsvwg/current/msg04379.html"

[Duk03b] M. Duke, "RE: [Tsvwg] Updating timestamps (ts_recent) in Linux", the IETF TSVWG WG mailing list, August 2003. URL "http://www1.ietf.org/mail-archive/web/tsvwg/current/msg04391.html"

[JBB97] V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance", (work in progress), Internet-Draft <draft-ietf-tcplw-high-performance-00.txt>, February 1997.

[JBB03] V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance", (work in progress), Internet-Draft <draft-jacobson-tsvwg-1323bis-00.txt>, August 2003.

[KP87] P. Karn and C. Partridge, "Estimating Round-Trip Times in Reliable Transport Protocols", Proceedings of SIGCOMM'87, August 1987.

[Lud03a] R. Ludwig, "RE: [Tsvwg] Updating timestamps (ts_recent) in Linux", the IETF TSVWG WG mailing list, August 2003. URL "http://www1.ietf.org/mail-archive/web/tsvwg/current/msg04389.html"

[Lud03b] R. Ludwig, "[Tsvwg] RFC1323.bis [was: Updating timestamps (ts_recent)]", the IETF TSVWG WG mailing list, August 2003. URL "http://www1.ietf.org/mail-archive/web/tsvwg/current/msg04397.html"

[Mil98] D. Miller, "possible bug in PAWS", the IETF TCP-IMPL WG mailing list, March 1998. URL "http://tcp-impl.grc.nasa.gov/tcp-impl/list/archive/1035.html"

[MM96] M. Mathis and J. Mahdavi, "Forward Acknowledgment:

Refining TCP Congestion Control", Proceedings of SIGCOMM'96, August 1996.

[MSM99] M. Mathis, J. Semke, and J. Mahdavi, "The Rate-Halving Algorithm for TCP Congestion Control", (work in progress), Internet-Draft <draft-mathis-tcp-ratehalving-00.txt>, August 1999.

[PD04] K. Poon and N. Demizu, "Use of TCP timestamp option to defend against blind spoofing attack", (work in progress), Internet-Draft <draft-poon-tcp-tstamp-mod-01.txt>, October 2004.

Author's Address

Noritoshi Demizu
National Institute of Information and Communications Technology
4-2-1 Nukui-Kitamachi, Koganei, Tokyo 184-8795, Japan
Phone: +81-42-327-7432 (Ex.5813)
E-mail: demizu@nict.go.jp

Appendix A: TS2 Reference

This appendix gives the formal description of TS1 and TS2 by using C-like pseudocode as a reference.

An implementation MAY support TS2. If an implementation supports TS2, it MAY implement one or more of RTTM, PAWS, PASA, DLI and SLID.

A.1 TCP Options

TS2 uses "the TCP Timestamps option" (see section 3.1), and "the TCP Old Timestamps option" (see section 3.2).

For simplicity, the TCP Timestamps option is called "the TS option". The TCP Old Timestamps option with option-length=10 is called "the OTS option", and that with option-length=2 is called "the OTS_OK option".

A.2 Types

The following types are used in this appendix: boolean, integer, 32bit-sequence-number, 32bit-timestamp, and internal-time.

All arithmetic dealing with the 32bit-sequence-number and 32bit-timestamp types must be performed modulo 2^32.

The format of internal-time depends on the implementation: for example, it may be an integer with unit = 1 second, or an OS-dependent structure.

The following type conversion function is used:
time2ts() converts an internal-time value to a 32bit-timestamp value.

A.3 Functions

The following boolean functions are defined.

To compare sequence numbers:

```
SEQ_GT(a, b)   True if (a >  b) in modulo 2^32.  False, otherwise.
SEQ_GE(a, b)   True if (a >= b) in modulo 2^32.  False, otherwise.
SEQ_LT(a, b)   True if (a <  b) in modulo 2^32.  False, otherwise.
SEQ_LE(a, b)   True if (a <= b) in modulo 2^32.  False, otherwise.
```

To compare timestamps:

```
TS_GT(a, b)    True if (a >  b) in modulo 2^32.  False, otherwise.
TS_GE(a, b)    True if (a >= b) in modulo 2^32.  False, otherwise.
TS_LT(a, b)    True if (a <  b) in modulo 2^32.  False, otherwise.
```

```
   TS_LE(a, b)    True if (a <= b) in modulo 2^32.  False, otherwise.
```

To compare times:

```
   TIME_GE(a, b)  True if a is no earlier than b.   False, otherwise.
   TIME_LT(a, b)  True if a is earlier than b.      False, otherwise.
```

To check octets:

```
   IsSACKed(seq)   True if the octet at sequence number "seq" has
                   been ACKed or SACKed.  False, otherwise.
   IsSACKed(range) True if part or all of the octets in "range" have
                   been ACKed or SACKed.  False, otherwise.

   IsInferredLost(seq)   True if the octet at sequence number "seq"
                         has been inferred lost.  False, otherwise.
   IsInferredLost(range) True if part or all of the octets in "range"
                         have been inferred lost.  False, otherwise.
```

The following function which returns a timestamp is defined.

To calculate border timestamp:

```
   TS2_SLID_BTS(TgtTS, PrbTS) = (k * TgtTS + (1 - k) * PrbTS),
   where k = 1/2.
```

A.4 Inequalities

The following inequalities are defined for testing received segments.

- Inequality (1) --- for data, SYN and FIN segments:

```
  (RSEG.LEN > 0 &&
   SEQ_LT(Last.Ack.Sent - max(RCV.WND), RSEG.SEQ + RSEG.LEN) &&
   SEQ_LT(RSEG.SEQ, RCV.NXT + RCV.WND))
```

- Inequality (2) --- for ACK segments:

```
  (RSEG.LEN == 0 &&
   SEQ_LE(Last.Ack.Sent - max(RCV.WND), RSEG.SEQ) &&
   SEQ_LT(RSEG.SEQ, RCV.NXT + RCV.WND))
```

The following boolean function is defined for evaluating these inequalities with respect to a received segment.

```
   TS_ISLEG()     True if (1) or (2) is satisfied.  False, otherwise.
```

A.5 Variables

If a received segment does not satisfy inequality (1) nor (2), variables below MUST NOT be updated.

A.5.1 Variables for Base Mechanism

The following variables are defined for the base mechanism. (See section 4)

TS.Req (integer)
- This variable represents a user's request:
  0 if neither TS1 nor TS2 is requested,
  1 if TS1 is requested, or
  2 if TS2 is requested.
- The initial value is given by the user.

TS.Mode (integer)
- This variable represents the result of the mode negotiation:
  negative if negotiation has not been completed,
  0 if both TS1 and TS2 are disabled,
  1 if TS1 is enabled, or
  2 if TS2 is enabled.
- The initial value is negative.

TS.Recent (32bit-timestamp)
- This variable holds the value to be echoed in SSEG.TSecr. The initial value is zero.
  - If TS1 is enabled, this variable holds the maximum RSEG.TSval value received on segments satisfying SEQ_LE(RSEG.SEQ, Last.ACK.sent).
  - If TS2 is enabled, this variable holds the minimum RSEG.TSval value on segments satisfying (RSEG.LEN > 0) received after a segment has last been sent.
- It is similar as TS.Recent as defined in [RFC1323].

TS.RecentIsOld (boolean)
- This variable is true when TS.Mode == 2 and no segment is received after the last segment has been sent. Therefore, this variable indicates whether the value in TS.Recent has been echoed to a remote node when TS.Mode == 2.
- Its value is valid only when TS.Mode is 2. Its value is undefined in other modes.

Last.Ack.Sent (32bit-sequence-number)
- This variable holds the last SSEG.ACK value sent.
- It is the same as Last.Ack.Sent as defined in [RFC1323].

TS.SndOff (32bit-timestamp)

- This variable holds an offset to convert an internal timestamp value to an external timestamp value.

TS.SndAdj (32bit-timestamp)
- This variable holds the difference between the return values of GetIntTS1() and GetIntTS2() when the first SYN was sent.
- It adjusts TS.SndOff in the TCP three-way handshake phase when a TCP connection is established with TS2 by a local node.

A.5.2 Variables for PAWS

The following variables are defined for PAWS. (See section 6)

TS.RcvMin (32bit-timestamp)
- This variable holds the maximum received RSEG.TSval value in both the TS option and the OTS option.

TS.RcvMin_time (internal-time)
- This variable holds the time when TS.RcvMin was last updated.

A.5.3 Variables for PASA

A.5.3.1 Variables for PASA-DF/TS2

The following variables are defined for PASA-DF/TS2.
(See section 7.1)

TS.SndMin (32bit-timestamp)
- This variable holds the maximum received RSEG.TSecr value in both the TS option and the OTS option.

TS.SndMax (32bit-timestamp)
- This variable holds the maximum SSEG.TSval value on sent segments satisfying (SSEG.LEN > 0).

TS.SndMax_time (internal-time)
- This variable holds the time when TS.SndMax was last updated.
- This variable is referred to in order to determine whether TS.SndOff MUST be tweaked. If there is another, simpler way to determine it, this variable can be omitted.

TS.PASADF_On (boolean)
- This variable indicates whether the PASA-DF test can be performed.
- The initial value is true.

A.5.3.2 Variables for PASA-SR/TS2

The following variables are defined for PASA-SR/TS2.
(See section 7.2)

TS.PASASR_On (boolean)
- This variable indicates whether PASA-SR/TS2 should be performed.
- The initial value is false. It is set to true when a SYN segment is received against an established TCP connection.

TS.PASASR_time (internal-time)
- This variable holds the time when the last SYN segment was received.
- Its value is valid only when TS.PASASR_On is true.

A.5.4 Variables for DLI

A.5.4.1 Variables for DLI-SEG/TS2

The following variables are defined for DLI-SEG/TS2. They are associated with each data segment. (See section 8.2.1)

DS.SndTS (32bit-timestamp) for each data segment
- This variable holds the SSEG.TSval value on the latest sent data segment.

DS.SndRO (integer) for each data segment
- This variable counts the number of received segments with the TS option satisfying TS_GT(RSEG.TSecr, DS.SndTS), which means the number of observed possible reorders.

The following variables are associated with each data segment. They might have been implemented.

DS.Start (32bit-sequence-number) for each data segment
- This variable holds the lowest sequence number of the data segment.

DS.End (32bit-sequence-number) for each data segment
- This variable holds the highest sequence number in the data segment, plus one.

Note: Since DLI-SEG/TS2 is very powerful, if it is implemented, other algorithms need not be implemented.

A.5.4.2 Variables for DLI-SACK/TS2

The following variables are defined for DLI-SACK/TS2. They are

associated with each SACK hole. (See section 8.2.2)

SH.SndTS (32bit-timestamp) for each SACK hole
- This variable holds the SSEG.TSval value on the latest sent data segment containing data in this SACK hole.
- When a SACK hole is allocated, it holds the RSEG.TSecr value.

SH.SndRO (integer) for each SACK hole
- This variable counts the number of received segments with the TS option satisfying TS_GT(RSEG.TSecr, SH.SndTS), which means the number of observed possible reorders.

The following variables are associated with each SACK hole. They would have been implemented in typical SACK implementations.

SH.Start (32bit-sequence-number) for each SACK hole
- This variable holds the lowest sequence number in the SACK hole.

SH.End (32bit-sequence-number) for each SACK hole
- This variable holds the highest sequence number in the SACK hole, plus one.

A.5.4.3 Variables for DLI-UNA/TS2

The following variables are defined for DLI-UNA/TS2.
(See section 8.2.3)

TS.UNA.SndTS (32bit-timestamp)
- This variable holds the SSEG.TSval value on the latest sent data segment containing data at SND.UNA.
- Its value is valid only when SEQ_LT(SND.UNA, SND.MAX).

TS.UNA.SndRO (integer)
- This variable counts the number of received segments with the TS option satisfying TS_GT(RSEG.TSecr, TS.UNA.SndTS), which means the number of observed possible reorders. It is cleared when TS.UNA.SndTS is updated.
- Its value is valid only when SEQ_LT(SND.UNA, SND.MAX).

A.5.4.4 Variables for DLI-NXT/TS2

The following variables are defined for DLI-NXT/TS2.
(See section 8.2.4)

TS.NXT.SndTS (32bit-timestamp)
- This variable holds the SSEG.TSval value on the latest sent data segment containing data at SND.NXT-1.
- Its value is valid only when data at SND.NXT-1 have not been

acknowledged nor SACKed.

TS.NXT.SndRO (integer)
- This variable counts the number of received segments with the TS option satisfying TS_GT(RSEG.TSecr, TS.NXT.SndTS), which means the number of observed possible reorders. It is cleared when TS.NXT.SndTS is updated.
- Its value is valid only when data at SND.NXT-1 have not been acknowledged nor SACKed.

A.5.4.5 Variables for DLI-MAX/TS2

The following variables are defined for DLI-MAX/TS2.
(See section 8.2.5)

TS.MAX.SndTS (32bit-timestamp)
- This variable holds the SSEG.TSval value on the latest sent data segment containing data at SND.MAX-1.
- Its value is valid only when data at SND.MAX-1 have not been acknowledged nor SACKed.

TS.MAX.SndRO (integer)
- This variable counts the number of received segments with the TS option satisfying TS_GT(RSEG.TSecr, TS.MAX.SndTS), which means the number of observed possible reorders. It is cleared when TS.MAX.SndTS is updated.
- Its value is valid only when data at SND.MAX-1 have not been acknowledged nor SACKed.

A.5.5 Variables for SLID

The following variables are defined for SLID. (See section 9)

A.5.5.1 Variables for SLID-SEG/TS2

The following variables are defined for SLID-SEG/TS2. They are associated with each data segment. (See section 9.2.1)

DS.TgtTS (32bit-timestamp) for each data segment
- This variable holds the SSEG.TSval value on the original or retransmitted decidedly-lost data segments.

DS.PrbTS (32bit-timestamp) for each data segment
- This variable holds the SSEG.TSval value on the first data segment that is sent since the data segment has been inferred lost.

A.5.5.2 Variables for SLID-SACK/TS2

The following variables are defined for SLID-SACK/TS2. They are associated with each SACK hole. (See section 9.2.2)

SH.TgtTS (32bit-timestamp) for each SACK hole
- This variable holds the RSEG.TSecr value on the segment that created the SACK hole.

SH.PrbTS (32bit-timestamp) for each SACK hole
- This variable holds the SSEG.TSval value on the first data segment that is sent since the data in the SACK hole has been inferred lost.

A.5.5.3 Variables for SLID-UNA/TS2

The following variables are defined for SLID-UNA/TS2.
(See section 9.2.3)

TS.UNA.TgtTS (32bit-timestamp)
- This variable holds the closest subsequent timestamp value to the SSEG.TSval value on original or retransmitted decidedly-lost data segments containing data at SND.UNA.

TS.UNA.PrbTS (32bit-timestamp)
- This variable holds the SSEG.TSval value on the first data segment that is sent since the data at SND.UNA has been inferred lost.

A.5.5.4 Variables for SLID-NXT/TS2

The following variables are defined for SLID-NXT/TS2.
(See section 9.2.4)

TS.NXT.TgtTS (32bit-timestamp)
- This variable holds the SSEG.TSval value on the original or retransmitted decidedly-lost data segments containing data at SND.NXT-1.

TS.NXT.PrbTS (32bit-timestamp)
- This variable holds the SSEG.TSval value on the first data segment that is sent since the data at SND.NXT-1 has been inferred lost.

A.5.5.5 Variables for SLID-MAX/TS2

The following variables are defined for SLID-MAX/TS2.
(See section 9.2.5)

TS.MAX.TgtTS (32bit-timestamp)
- This variable holds the SSEG.TSval value on the original or retransmitted decidedly-lost data segments containing data at SND.MAX-1.

TS.MAX.PrbTS (32bit-timestamp)
- This variable holds the SSEG.TSval value on the first data segment that is sent since the data at SND.MAX-1 has been inferred lost.

A.6 Current Time

The following pseudocode functions are defined for getting the current time or current timestamp.

GetTime() --- Get Current Time (internal-time)
- Get the current time in an internal time format.
- The time returned by this function MUST NOT be wrapped in the lifetime of any TCP connections.

GetIntTS1() --- Get Current Internal TS for TS1 (32bit-timestamp)
- Get the current time in a 32-bit unsigned integer in the unit of the TS1 timestamp. Note that the actual SSEG.TSval value is calculated as GetIntTS1() + TS.SndOff.
- The timestamp unit for TS1 is in the range of 1 sec to 1 ms.

GetIntTS2() --- Get Current Internal TS for TS2 (32bit-timestamp)
- Get the current time in a 32-bit unsigned integer in the unit of the TS2 timestamp. Note that the actual SSEG.TSval value is calculated as GetIntTS2() + TS.SndOff.
- The timestamp unit for TS2 is fixed at 1 usec.

GetRTO2() --- Get Current RTO value for TS2 (32bit-timestamp)
- Get the current RTO value in a 32-bit unsigned integer in the unit of the TS2 timestamp so that it can be used in PAWS/TS2 and PASA-DF/TS2.

A.7 Random Number Generator

The following pseudocode function is defined for getting random numbers.

RandomNumber(max)
- It returns a random number between 0 and max.

A.8 Constants

The following constants are defined.

TS1_GRANULARITY (32bit-timestamp) for TS1
- This constant represents the granularity of GetIntTS1() in the unit of the TS1 timestamp.

TS2_GRANULARITY (32bit-timestamp) for TS2
- This constant represents the granularity of GetIntTS2() in the unit of the TS2 timestamp.

TS1_PAWS_MARGIN (32bit-timestamp) for PAWS/TS1
- When PAWS/TS1 is enabled, the minimum acceptable RSEG.TSval is calculated as (TS.RcvMin - TS1_PAWS_MARGIN).
- The default value is 1.

TS1_PAWS_IDLE (internal-time) for PAWS/TS1
- The value of TS.RcvMin is valid for this amount of time if TS1 is enabled.
- The default value is 24 days.

TS2_PAWS_IDLE (internal-time) for PAWS/TS2
- The value of TS.RcvMin is valid for this amount of time if TS2 is enabled.
- The default value is 20 minutes. (This value should be longer than the longest timeout.)

TS2_PAWS_DEV (32bit-timestamp) for PAWS/TS2
- This constant represents the acceptable deviation of received RSEG.TSval.
- The default value is 1 minute.

TS2_PASADF_RNDMAX_REUSE (32bit-timestamp) for PASA-DF/TS2
- When a TCP control block is reused by a new TCP connection, TS.SndOff is increased by a random number in the range from 0 to this value.
- The default value is 2^29 - 1 usec (about 9 minutes).

TS2_PASADF_RNDMAX_IDLE (32bit-timestamp) for PASA-DF/TS2
- After an idle of TS2_PASADF_MAXADV, TS.SndOff is increased by TS2_PASADF_MAXADV plus a random number in the range from 0 to this value.
- The default value is 2^26 - 1 usec (about 67 seconds).

TS2_PASADF_MAXADV (internal-time) for PASA-DF/TS2
- This constant represents the maximum increase in SSEG.TSval.
- The default value is 64 seconds. (This value SHOULD be greater than the maximum RTO value.)

TS2_PASASR_WIN (integer) for PASA-SR/TS2
- This constant represents the window size of the ACK segments sent in reply to SYN segments.

- The default value is the maximum default SMSS value on the node. The value MAY be RMSS on the TCP connection (In this case, it is not constant, though).

TS2_PASASR_TIME (internal-time) for PASA-SR/TS2
- This constant represents the time in which received RST segments should be specially handled.
- The default value is 10 seconds.

TS2_DLI_THRESH (integer) for DLI/TS2
- If the number of observed possible reorders from the point of the view of a target octet is greater than or equal to this value, the target segment is inferred lost.
- The default value is 3 (The same value as the so-called duplicate acknowledgement threshold specified in [RFC2581]).
- It might be implemented as an adaptive variable in the future.

A.9 Attributes

A.9.1 Attributes of Received Segments

The following flags represent attributes of the received segment.

isSYN	True only if it is a SYN segment.
isSYNACK	True only if it is a SYN+ACK segment.
isRST	True only if it is a RST segment.
isFirstSYN	True only if it is the first SYN segment.
isFirstSYNACK	True only if it is the first SYN+ACK segment.
withTS	True only if it carries the TS option.
withOTS	True only if it carries the OTS option.
withOTS_OK	True only if it carries the OTS_OK option.

A.9.2 Attributes of TCP Connection

The following variable indicates an attribute of TCP connection.

TCP.State	The current TCP State (See section 3.2 of [RFC793])

A.10 Procedures

A.10.1 Initialization

When a TCP Control Block is created or reused, the procedure below is followed to initialize the variables.

```
/* Step 1: Base(TS1&TS2) */
TS.Req = 0, 1 or 2;     /* Requested by user */
TS.Mode = -1;           /* Not negotiated yet */
```

```
        TS.SndAdj = 0;
        if (<TCP Control Block is reused>) {
                if (TS.Req == 2) {
                        /*
                         * To avoid reusing the same range of SSEG.TSval
                         */
                        TS.SndOff += (GetIntTS2() - GetIntTS1());
                } else {
                        /* No need to change TS.SndOff */
                }
        } else {
                TS.SndOff = 0;
        }

        if (TS.Req == 2) {
                /* Step 2-1: PASA-DF/TS2 */
                TS.PASADF_On = true;
                if (<TCP Control Block is reused>) {
                        TS.SndOff +=
                                RandomNumber(TS2_PASADF_RNDMAX_REUSE);
                } else {
                        /* Randomize the initial timestamp */
                        TS.SndOff = RandomNumber(2^32);
                }
                /* Step 2-2: PASA-SR/TS2 */
                TS.PASASR_On = false;
        }
```

When a SACK hole is allocated, the procedure below is followed to initialize the variables.

```
        /* Step 1: DLI-SACK/TS2 */
        if (TS.Mode == 2) {
                SH.SndTS = RSEG.TSecr;
                SH.SndRO = 0;
        }

        /* Step 2: SLID-SACK/TS2 */
        if (TS.Mode == 2) {
                SH.TgtTS = RSEG.TSecr;
                SH.PrbTS = RSEG.TSecr;
        }
```

A.10.2 Input Processing

A.10.2.1 Input Processing in LISTEN and SYN-SENT States

When a SYN segment is received in the LISTEN state, or a SYN or

SYN+ACK segment is received in the SYN-SENT state, the procedure

below is followed.

```
/* Step 1: Base(TS1&TS2) */
TS.Mode = (!withTS ? 0 : !withOTS_OK ? 1 : 2);
if (TS.Mode > TS.Req) {
        TS.Mode = TS.Req;
}
if (TS.Mode > 0) {
        TS.Recent = RSEG.TSval;
        if (TS.Mode == 2) {
                TS.RecentIsOld = false;
                if (TCP.State == "SYN-SENT") {
                        TS.SndOff += TS.SndAdj;
                }
        }
}

/* Step 2: RTTM(TS1&TS2) */
if (TCP.State == "SYN-SENT" && TS.Mode > 0) {
        if (TS.Mode == 1 && withTS && RSEG.TSecr != 0) {
                Measured_RTT = ((GetIntTS1() + TS.SndOff)
                                - RSEG.TSecr + TS1_GRANULARITY);
        } else if (TS.Mode == 2 && withTS) {
                Measured_RTT = ((GetIntTS2() + TS.SndOff)
                                - RSEG.TSecr + TS2_GRANULARITY);
        }
}

/* Step 3: PAWS(TS1&TS2) */
if (TS.Mode > 0) {
        TS.RcvMin = RSEG.TSval;
        TS.RcvMin_time = GetTime();
}
```

A.10.2.2 Input Processing in Other States

When a segment is received in the SYN-RECEIVED state, the ESTABLISHED state, or a later state, the procedure below is followed. Note that TS.Mode is determined when the first SYN or SYN+ACK segment is received, as described in appendix A.10.2.1.

```
/*
 * Step 1: Check received segment.
 */
if (TS.Mode == 1) {
        /* Step 1-1-1: PAWS/TS1 */
        idle_time = GetTime() - TS.RcvMin_time;
        if (!isSYN && && !isRST &&
```

```
                    TIME_LT(idle_time, TS1_PAWS_IDLE) &&
```

```
                TS_LT(RSEG.TSval, TS.RcvMin - TS1_PAWS_MARGIN)) {
                    /* This segment MUST be dropped. */
                    /* An ACK SHOULD be sent in reply. */
            }
        } else if (TS.Mode == 2) {
            /* Step 1-2-1: Base(TS2) */
            if ((!isSYN && !isRST && (!withTS && !withOTS)) ||
                (withTS && withOTS)) {
                    /* This segment MUST be dropped. */
                    /* An ACK SHOULD be sent in reply. */
            }
            /* Step 1-2-2: PAWS/TS2 */
            idle_time = GetTime() - TS.RcvMin_time;
            if (!isSYN && (!isRST || withOTS) &&
                TIME_LT(idle_time, TS2_PAWS_IDLE)) {
                    if (TS_LT(RSEG.TSval, TS.RcvMin - GetRTO2())) {
                            /* This segment MUST be dropped. */
                            /* An ACK SHOULD be sent in reply. */
                    }
                    mean_ts = (TS.RcvMin + time2ts(idle_time));
                    if (TS_GT(RSEG.TSval, mean_ts + TS2_PAWS_DEV)) {
                            /*
                             * This segment MAY be dropped.
                             * If PASA-DF/TS2 is enabled,
                             * it SHOULD be dropped.
                             * If it is dropped,
                             * an ACK SHOULD be sent in reply.
                             */
                    }
            }
            /* Step 1-2-3-1: PASA-DF/TS2 */
            if (!isSYN &&  (!isRST || withOTS) && TS.PASADF_On) {
                    if (TS_LT(RSEG.TSecr, TS.SndMin - GetRTO2()) ||
                        TS_GT(RSEG.TSecr, TS.SndMax)) {
                            /* This segment MUST be dropped. */
                            /* An ACK SHOULD be sent in reply. */
                    }
            }
            /* Step 1-2-3-2: PASA-SR/TS2 */
            if (isSYN) {
                    TS.PASASR_On = true;
                    TS.PASASR_time = GetTime();
                    /*
                     * This segment MUST be dropped.
                     * An ACK with win=TS2_PASASR_WIN
                     * without TS nor OTS SHOULD be sent in reply.
                     */
            }
```

```
                if (isRST && !withOTS && TS.PASASR_On) {
```

```
                    if (TIME_GE(GetTime() - TS.PASASR_time,
                                TS2_PASASR_TIME)) {
                            TS.PASASR_On = false;
                    }
                    if (TS.PASASR_On &&
                        !(SEQ_LE(RCV.NXT, RSEG.SEQ) &&
                          SEQ_LT(RSEG.SEQ,
                                 RCV.NXT + TS2_PASASR_WIN))) {
                            /* This segment MUST be dropped. */
                            /* ACK MUST NOT be sent. */
                    }
            }
        }

        /*
         * Step 2: Check acceptability by [RFC793].
         */

        /*
         * Step 3: Process received segment.
         *
         * Note: (RSEG.LEN > 0 && TS_ISLEG()) is equal to inequality (1)
         */
        if (TS.Mode == 1 && TS_ISLEG()) {
                /* Step 3-1-1: Base(TS1) */
                if (RSEG.LEN > 0 && SEQ_LE(RSEG.SEQ, Last.ACK.sent) &&
                    TS_LT(TS.Recent, RSEG.TSval)) {
                        TS.Recent = RSEG.TSval;
                }
                /* Step 3-1-2: RTTM/TS1 */
                if (withTS && RSEG.TSecr != 0) {
                        Measured_RTT = ((GetIntTS1() + TS.SndOff)
                                        - RSEG.TSecr + TS1_GRANULARITY);
                }
                /* Step 3-1-3: PAWS/TS1 */
                if (TIME_GE(GetTime() - TS.RcvMin_time, TS1_PAWS_IDLE)||
                    TS_LT(TS.RcvMin, RSEG.TSval)) {
                        TS.RcvMin = RSEG.TSval;
                        TS.RcvMin_time = GetTime();
                }
        } else if (TS.Mode == 2 && TS_ISLEG()) {
                /* Step 3-2-1: Base(TS2) */
                if (RSEG.LEN > 0 &&
                    (TS.RecentIsOld || TS_GT(TS.Recent, RSEG.TSval))) {
                        TS.Recent = RSEG.TSval;
                        TS.RecentIsOld = false;
                }
                /* Step 3-2-2: RTTM/TS2 */
```

```
                if (withTS) {
```

```
                Measured_RTT = ((GetIntTS2() + TS.SndOff)
                                - RSEG.TSecr + TS2_GRANULARITY);
        }
        /* Step 3-2-3: PAWS/TS2 */
        if (TIME_GE(GetTime() - TS.RcvMin_time, TS2_PAWS_IDLE)||
            TS_LT(TS.RcvMin, RSEG.TSval)) {
                TS.RcvMin = RSEG.TSval;
                TS.RcvMin_time = GetTime();
        }
        /* Step 3-2-4-1: PASA-DF/TS2 */
        if (TS.PASADF_On) {
                if (TS_LT(TS.SndMin, RSEG.TSecr)) {
                        TS.SndMin = RSEG.TSecr;
                }
        } else {
                if (TS_GE(TS.SndMax, RSEG.TSecr)) {
                        /* Restart the PASA-DF test. */
                        TS.SndMin = RSEG.TSecr;
                        TS.PASADF_On = true;
                }
        }
        /* Step 3-2-4-2: PASA-SR/TS2 */
        if (TS.PASASR_On && (!isRST || withOTS)) {
                TS.PASASR_On = false;
        }
        /* Step 3-2-5-1: DLI-SEG/TS2 */
        foreach data segment {
                if (withTS && TS_GT(RSEG.TSecr, DS.SndTS) &&
                    ++DS.SndRO >= TS2_DLI_THRESH) {
                        /*
                         * This data segment is inferred lost.
                         */
                }
        }
        /* Step 3-2-5-2: DLI-SACK/TS2 */
        foreach SACK hole {
                if (withTS && TS_GT(RSEG.TSecr, SH.SndTS) &&
                    ++SH.SndRO >= TS2_DLI_THRESH) {
                        /*
                         * Whole data in this SACK hole
                         * is inferred lost.
                         */
                }
                /* To help DLI-UNA/TS2 */
                if (SEQ_LE(SH.Start, new SND.UNA) &&
                    SEQ_LT(new SND.UNA, SH.End)) {
                        /* New SND.UNA is in this SACK hole. */
                        TS.UNA.SndTS = SH.SndTS;
```

```
TS.UNA.SndRO = SH.SndRO;
```

```
                }
        }
        /* Step 3-2-5-3: DLI-UNA/TS2 */
        if (SEQ_LT(old SND.UNA, SND.MAX)) {
                if (SEQ_LE(RSEG.ACK, old SND.UNA)) {
                        if (withTS &&
                            TS_GT(RSEG.TSecr, TS.UNA.SndTS) &&
                            ++TS.UNA.SndRO >= TS2_DLI_THRESH) {
                                /*
                                 * Data at SND.UNA
                                 * is inferred lost.
                                 */
                        }
                } else {
                        TS.UNA.SndTS = GetIntTS2() + TS.SndOff;
                        TS.UNA.SndRO = 0;
                }
        }
        /* Step 3-2-5-4: DLI-NXT/TS2 */
        if (!IsSACKed(SND.NXT-1)) {
                if (withTS && TS_GT(RSEG.TSecr, TS.NXT.SndTS) &&
                    ++TS.NXT.SndRO >= TS2_DLI_THRESH) {
                        if (SEQ_LT(new SND.FACK, SND.NXT)) {
                                /*
                                 * Data between SND.FACK and
                                 * SND.NXT is inferred lost.
                                 */
                        } else {
                                /*
                                 * Data at SND.NXT-1
                                 * is inferred lost.
                                 */
                        }
                }
        }
        /* Step 3-2-5-5: DLI-MAX/TS2 */
        if (!IsSACKed(SND.MAX-1)) {
                if (withTS && TS_GT(RSEG.TSecr, TS.MAX.SndTS) &&
                    ++TS.MAX.SndRO >= TS2_DLI_THRESH) {
                        if (SEQ_LT(SND.NXT, new SND.FACK)) {
                                /*
                                 * Data between SND.FACK and
                                 * SND.MAX is inferred lost.
                                 */
                        } else {
                                /*
                                 * Data at SND.MAX-1
                                 * is inferred lost.
```

```
*/
```

```
                        }
                }
        }
        /* Step 3-2-6-1: SLID-SEG/TS2 */
        foreach data segment {
                if (TS_LT(DS.TgtTS, DS.PrbTS)) {
                        b_ts = TS2_SLID_BTS(DS.TgtTS, DS.PrbTS);
                        if (TS_GT(RSEG.TSecr, b_ts)) {
                                /*
                                 * The loss inference is
                                 * GENUINE.
                                 */
                                DS.TgtTS = DS.PrbTS;
                        } else if (IsSACKed(DS)) {
                                /*
                                 * The loss inference is
                                 * SPURIOUS.  Execute a response
                                 * algorithm if necessary.
                                 */
                                DS.TgtTS = DS.PrbTS;
                        }
                }
        }
        /* Step 3-2-6-2: SLID-SACK/TS2 */
        foreach SACK hole {
                if (TS_LT(SH.TgtTS, SH.PrbTS)) {
                        b_ts = TS2_SLID_BTS(SH.TgtTS, SH.PrbTS);
                        if (TS_GT(RSEG.TSecr, b_ts)) {
                                /*
                                 * The loss inference is
                                 * GENUINE.
                                 */
                                SH.TgtTS = SH.PrbTS;
                        } else if (IsSACKed(SH)) {
                                /*
                                 * The loss inference is
                                 * SPURIOUS.  Execute a response
                                 * algorithm if necessary.
                                 */
                                SH.TgtTS = SH.PrbTS;
                        }
                }
                /* To help SLID-UNA/TS2 */
                if (SEQ_LE(SH.Start, new SND.UNA) &&
                    SEQ_LT(new SND.UNA, SH.End)) {
                        /* New SND.UNA is in this SACK hole. */
                        TS.UNA.TgtTS = SH.TgtTS;
                        TS.UNA.PrbTS = SH.PrbTS;
```

```
                        }
```

```
        }
        /* Step 3-2-6-3: SLID-UNA/TS2 */
        if (TS_LT(TS.UNA.TgtTS, TS.UNA.PrbTS)) {
                b_ts = TS2_SLID_BTS(TS.UNA.TgtTS, TS.UNA.PrbTS);
                if (TS_GT(RSEG.TSecr, b_ts)) {
                        /*
                         * The loss inference is GENUINE.
                         */
                        TS.UNA.TgtTS = TS.UNA.PrbTS;
                } else if (SEQ_LT(old SND.UNA, new SND.UNA)) {
                        /*
                         * The loss inference is SPURIOUS.
                         * Execute a response algorithm
                         * if necessary.
                         */
                        if (TS_GT(RSEG.TSecr, TS.UNA.PrbTS)) {
                                TS.UNA.PrbTS = RSEG.TSecr;
                        }
                        TS.UNA.TgtTS = TS.UNA.PrbTS;
                }
        } else {
                if (SEQ_LT(old SND.UNA, new SND.UNA)) {
                        if (TS_GT(RSEG.TSecr, TS.UNA.PrbTS)) {
                                TS.UNA.PrbTS = RSEG.TSecr;
                        }
                        TS.UNA.TgtTS = TS.UNA.PrbTS;
                }
        }
        /* Step 3-2-6-4: SLID-NXT/TS2 */
        if (TS_LT(TS.NXT.TgtTS, TS.NXT.PrbTS)) {
                b_ts = TS2_SLID_BTS(TS.NXT.TgtTS, TS.NXT.PrbTS);
                if (TS_GT(RSEG.TSecr, b_ts)) {
                        /*
                         * The loss inference is GENUINE.
                         */
                        TS.NXT.TgtTS = TS.NXT.PrbTS;
                } else if (IsSACKed(SND.NXT-1)) {
                        /*
                         * The loss inference is SPURIOUS.
                         * Execute a response algorithm
                         * if necessary.
                         */
                        TS.NXT.TgtTS = TS.NXT.PrbTS;
                }
        }
        /* Step 3-2-6-5: SLID-MAX/TS2 */
        if (TS_LT(TS.MAX.TgtTS, TS.MAX.PrbTS)) {
                b_ts = TS2_SLID_BTS(TS.MAX.TgtTS, TS.MAX.PrbTS);
```

```
                    if (TS_GT(RSEG.TSecr, b_ts)) {
```

```
                        /*
                         * The loss inference is GENUINE.
                         */
                        TS.MAX.TgtTS = TS.MAX.PrbTS;
                } else if (IsSACKed(SND.MAX-1)) {
                        /*
                         * The loss inference is SPURIOUS.
                         * Execute a response algorithm
                         * if necessary.
                         */
                        TS.MAX.TgtTS = TS.MAX.PrbTS;
                }
        }
}
```

A.10.3 Output Processing

When a RST segment is sent in reply to a received segment because of [RFC793], the following processing is utilized.

- If the received segment carries the TS option or the OTS option, the RST segment MUST carry the OTS option with <SSEG.TSval=RSEG.TSecr><SSEG.TSecr=RSEG.TSval>. Otherwise, the RST segment SHOULD NOT carry the TS option nor the OTS option.

When an ACK segment is sent in reply to a received SYN or SYN+ACK segment because of [RFC793], the following procedure is performed:

```
/* Step 1: PASA-SR/TS2 */
if (TS.Mode == 2) {
        TS.PASASR_On = true;
        TS.PASASR_time = GetTime();
        /*
         * This segment MUST be dropped.
         * An ACK with win=TS2_PASASR_WIN
         * without TS nor OTS SHOULD be sent in reply.
         */
}
```

In other cases, when a segment is sent, the procedure below is followed:

```
/* Step 1: PASA-DF/TS2 */
/* To avoid advancing SSEG.TSval too much after an idle. */
if (TS.Mode == 2) {
        over_time = ((GetTime() - TS.SndMax_time)
                     - TS2_PASADF_MAXADV);
        if (over_time > 0) {
```

```
                    TS.SndOff -= time2ts(over_time);
```

```
                        TS.SndOff +=
                                RandomNumber(TS2_PASADF_RNDMAX_IDLE);
                        TS.SndMax_time = GetTime();
                }
        }

        /* Step 2: Base(TS1&TS2) */
        if (isSYN ? (TS.Req > 0) : (TS.Mode > 0)) {
                if (isSYN && TS.Req == 2) {
                        /* Put the OTS_OK option. */
                }
                if (TS.Mode == 2) {
                        if (isRST || TS.RecentIsOld) {
                                SSEG.TSkind = OTS;
                        } else {
                                SSEG.TSkind = TS;
                                TS.RecentIsOld = true;
                        }
                        SSEG.TSval = GetIntTS2() + TS.SndOff;
                        SSEG.TSecr = TS.Recent;
                } else {
                        SSEG.TSkind = TS;
                        SSEG.TSval = GetIntTS1() + TS.SndOff;
                        SSEG.TSecr = TS.Recent;
                }
        }
        if (isFirstSYN && TS.Req == 2 && TS.Mode < 0) {
                TS.SndAdj = GetIntTS1() - GetIntTS2();
        }
        LAST.Ack.Sent = SSEG.ACK;

        /* Step 3: PASA-DF/TS2 */
        if ((isSYN ? (TS.Req == 2) : (TS.Mode == 2)) && SSEG.LEN > 0) {
                TS.SndMax = SSEG.TSval;
                TS.SndMax_time = GetTime();
                if (TS_GT(TS.SndMin, TS.SndMax)) {
                        /* Stop the PASA-DF test */
                        TS.PASADF_On = false;
                }
        }
        if ((isFirstSYN || isFirstSYNACK) && TS.Req == 2) {
                TS.SndMin = SSEG.TSval;
        }

        /* Step 4-1: DLI-SEG/TS2 */
        if (TS.Mode == 2 && SSEG.LEN > 0) {
                /* A data segment is sent. */
                DS.SndTS = SSEG.TSval;
```

```
                DS.SndRO = 0;
```

```
        }

        /* Step 4-2: DLI-SACK/TS2 */
        if (TS.Mode == 2 && SSEG.LEN > 0) {
                /* Data in a SACK hole is retransmitted. */
                if (SEQ_LT(SSEG.SEQ, SH.End) &&
                    SEQ_LT(SH.Start, SSEG.SEQ + SSEG.LEN)) {
                        SH.SndTS = SSEG.TSval;
                        SH.SndRO = 0;
                }
        }

        /* Step 4-3: DLI-UNA/TS2 */
        if ((isSYN ? (TS.Req == 2) : (TS.Mode == 2)) && SSEG.LEN > 0) {
                /* Data at SND.UNA is sent. */
                if (SEQ_LE(SSEG.SEQ, SND.UNA) &&
                    SEQ_LT(SND.UNA, SSEG.SEQ + SSEG.LEN)) {
                        TS.UNA.SndTS = SSEG.TSval;
                        TS.UNA.SndRO = 0;
                }
        }

        /* Step 4-4: DLI-NXT/TS2 */
        if ((isSYN ? (TS.Req == 2) : (TS.Mode == 2)) && SSEG.LEN > 0) {
                /* Data at SND.NXT or between SND.FACK and SND.NXT */
                if ((SSEQ.SEQ == old SND.NXT && SND.WND > 0) ||
                    (SEQ_LT(SND.FACK, old SND.NXT) &&
                     SEQ_LT(SSEG.SEQ, old SND.NXT) &&
                     SEQ_LT(SND.FACK, SSEG.SEQ + SSEG.LEN))) {
                        TS.NXT.SndTS = SSEG.TSval;
                        TS.NXT.SndRO = 0;
                }
        }

        /* Step 4-5: DLI-MAX/TS2 */
        if ((isSYN ? (TS.Req == 2) : (TS.Mode == 2)) && SSEG.LEN > 0) {
                /* Data at SND.MAX or between SND.FACK and SND.MAX */
                if ((SSEQ.SEQ == old SND.MAX && SND.WND > 0) ||
                    (SEQ_LT(old SND.NXT, SND.FACK) &&
                     SEQ_LT(SSEG.SEQ, old SND.MAX) &&
                     SEQ_LT(SND.FACK, SSEG.SEQ + SSEG.LEN))) {
                        TS.MAX.SndTS = SSEG.TSval;
                        TS.MAX.SndRO = 0;
                }
        }

        /* Step 5-1: SLID-SEG/TS2 */
        if (TS.Mode == 2) {
```

```
                foreach data segment {
```

```
                        if (IsInferredLost(DS)) {
                                if (TS_GT(DS.TgtTS, SSEG.TSval)) {
                                        /*
                                         * The loss inference is assumed
                                         * SPURIOUS.  Execute a response
                                         * algorithm if necessary.
                                         */
                                } else {
                                        /* An probe segment is sent. */
                                        DS.PrbTS = SSEG.TSval;
                                }
                        } else {
                                /* Assert(DS.TgtTS == DS.PrbTS) */
                                if (SEQ_LT(SSEG.SEQ, DS.End) &&
                                          SEQ_LT(DS.Start,
                                                 SSEG.SEQ + SSEG.LEN)){
                                        /*
                                         * The sent segment is original
                                         * or decidedly lost.
                                         */
                                        DS.TgtTS = SSEG.TSval;
                                        DS.PrbTS = SSEG.TSval;
                                }
                        }
                }
        }

        /* Step 5-2: SLID-SACK/TS2 */
        if (TS.Mode == 2) {
                foreach SACK hole {
                        if (IsInferredLost(SH)) {
                                if (TS_GT(SH.TgtTS, SSEG.TSval)) {
                                        /*
                                         * The loss inference is assumed
                                         * SPURIOUS.  Execute a response
                                         * algorithm if necessary.
                                         */
                                } else {
                                        /* An probe segment is sent. */
                                        SH.PrbTS = SSEG.TSval;
                                }
                        } else {
                                /* Assert(SH.TgtTS == SH.PrbTS) */
                                if (SEQ_LT(SSEG.SEQ, SH.End) &&
                                          SEQ_LT(SH.Start,
                                                 SSEG.SEQ + SSEG.LEN)){
                                        /*
                                         * The sent segment is original
```

* or decidedly lost.

```
                                        */
                                       SH.TgtTS = SSEG.TSval;
                                       SH.PrbTS = SSEG.TSval;
                                }
                        }
                }
        }

        /* Step 5-3: SLID-UNA/TS2 */
        if ((isFirstSYN || isFirstSYNACK) && TS.Req == 2) {
                TS.UNA.TgtTS = SSEG.TSval;
                TS.UNA.PrbTS = SSEG.TSval;
        }
        if (TS.Mode == 2) {
                if (IsInferredLost(SND.UNA)) {
                        if (TS_GT(TS.UNA.TgtTS, SSEG.TSval)) {
                                /*
                                 * The loss inference is assumed
                                 * SPURIOUS.  Execute a response
                                 * algorithm if necessary.
                                 */
                        } else {
                                /* An probe segment is sent. */
                                TS.UNA.PrbTS = SSEG.TSval;
                        }
                } else {
                        /* Assert(TS.UNA.TgtTS == TS.UNA.PrbTS) */
                        if (SEQ_LE(SSEG.SEQ, SND.UNA) &&
                                   SEQ_LT(SND.UNA,
                                          SSEG.SEQ + SSEG.LEN)) {
                                /*
                                 * The sent segment is original
                                 * or decidedly lost.
                                 */
                                TS.UNA.TgtTS = SSEG.TSval;
                                TS.UNA.PrbTS = SSEG.TSval;
                        }
                }
        }

        /* Step 5-4: SLID-NXT/TS2 */
        if ((isFirstSYN || isFirstSYNACK) && TS.Req == 2) {
                /* Initialize TS.NXT.TgtTS and TS.NXT.PrbTS */
                TS.NXT.TgtTS = SSEG.TSval;
                TS.NXT.PrbTS = SSEG.TSval;
        }
        if (TS.Mode == 2) {
                if (IsInferredLost(new SND.NXT-1)) {
```

```
                    if (TS_GT(TS.NXT.TgtTS, SSEG.TSval)) {
```

```
                        /*
                         * The loss inference is assumed
                         * SPURIOUS.  Execute a response
                         * algorithm if necessary.
                         */
                } else {
                        /* An probe segment is sent. */
                        TS.NXT.PrbTS = SSEG.TSval;
                }
        } else {
                /* Assert(TS.NXT.TgtTS == TS.NXT.PrbTS) */
                if (SEQ_LE(SEG.SEQ, new SND.NXT-1) &&
                           SEQ_LT(new SND.NXT-1,
                                  SSEQ.SEQ + SSEG.LEN)) {
                        /*
                         * The sent segment is original
                         * or decidedly lost.
                         */
                        TS.NXT.TgtTS = SSEG.TSval;
                        TS.NXT.PrbTS = SSEG.TSval;
                }
        }
}

/* Step 5-5: SLID-MAX/TS2 */
if ((isFirstSYN || isFirstSYNACK) && TS.Req == 2) {
        /* Initialize TS.MAX.TgtTS and TS.MAX.PrbTS */
        TS.MAX.TgtTS = SSEG.TSval;
        TS.MAX.PrbTS = SSEG.TSval;
}
if (TS.Mode == 2) {
        if (IsInferredLost(new SND.MAX-1)) {
                if (TS_GT(TS.MAX.TgtTS, SSEG.TSval)) {
                        /*
                         * The loss inference is assumed
                         * SPURIOUS.  Execute a response
                         * algorithm if necessary.
                         */
                } else {
                        /* An probe segment is sent. */
                        TS.MAX.PrbTS = SSEG.TSval;
                }
        } else {
                /* Assert(TS.MAX.TgtTS == TS.MAX.PrbTS) */
                if (SEQ_LE(SEG.SEQ, new SND.MAX-1) &&
                           SEQ_LT(new SND.MAX-1,
                                  SSEQ.SEQ + SSEG.LEN)) {
                        /*
```

* The sent segment is original

```
                            * or decidedly lost.
                            */
                           TS.MAX.TgtTS = SSEG.TSval;
                           TS.MAX.PrbTS = SSEG.TSval;
                       }
                   }
               }
```

A.11 Layouts of TCP Options

When both the OTS_OK option and the TS option are sent on SYN or SYN+ACK segments, the following format is recommended.

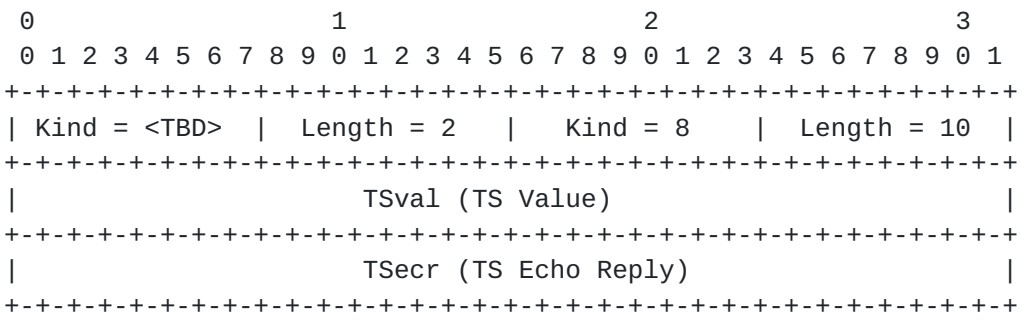


Figure A-1: The OTS_OK option and the TS option

When either the TS option or the OTS option is sent, the following format is recommended. Two NOPs may be replaced with another 2-octet option.

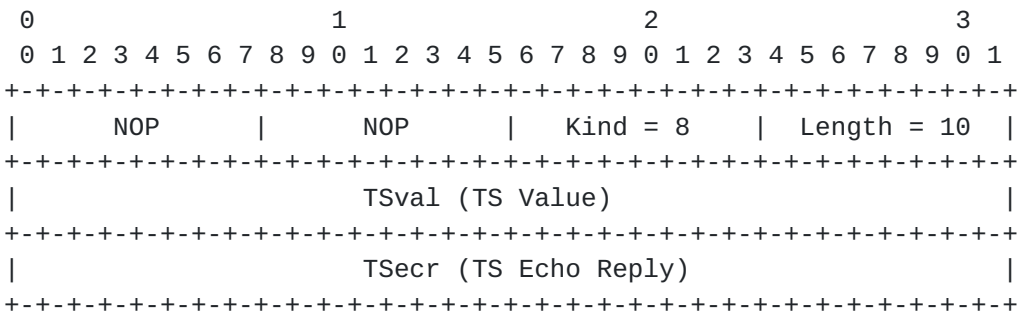


Figure A-2: The TS option and NOPs

Appendix B: Granularity of Timestamps

The granularity of a timestamp is a different concept from the unit of a timestamp. Timestamps are internally generated from an internal tick count or a real time clock. The unit of a timestamp means the time length of 1 in the timestamp, while the granularity of a timestamp means the interval time for updating a timestamp source so that the resulting timestamp is changed.

> Note: In many cases, the granularity of a timestamp would not be shorter than the unit of the timestamp. With TS1, since the timestamp unit can be chosen between 1 second and 1 ms, it would be simplest to make it the same as the granularity of a timestamp. In contrast, with TS2, since the timestamp unit is fixed at 1 usec, the granularity will be much longer than the unit in most cases.

If the granularity of timestamps is coarser than the mean time between each data transmission, multiple data segments may carry the same SSEG.TSval value, and DLI/TS2 and SLID/TS2 would be less effective. If the granularity of timestamp is not fine enough, the following idea might improve it.

The first idea uses segment counter. It is associated with each TCP connection. It is increased by one when a segment is sent. Its maximum value is TS2_GRANULARITY - 1 to keep timestamps monotonically nondecreasing. It is cleared when timestamp source is changed, or TS.SndOff is changed. Whenever an internal timestamp is generated, it is added to the timestamp.

The second idea uses a CPU's embedded cycle counter. Each time when an internal tick count is increased, or a real time clock is read, the value of CPU's embedded cycle counter is also recorded. Then, when an internal timestamp is generated, add the difference between the recorded cycle counter value and the current cycle counter value to the generated timestamp.

In any case, since the timestamp unit for TS2 is fine (i.e., 1 usec) compared to today's possible RTTs, some lower bits of internal timestamps might be usable as nonce to obfuscate timestamps.

Appendix C: Loss Inference With SACK and DLI/TS2

This appendix discusses a loss inference procedure making use of SACK [RFC2018][RFC3517] and DLI/TS2.

It does not discuss which data should be transmitted when more than one data segment can be sent or retransmitted. This topic is outside the scope of this appendix.

C.1 Highest Sequence Number of Retransmitted Data

This appendix uses one variable: SND.RTX (32bit-sequence-number). RTX stands for retransmission. It holds the maximum sequence number of acknowledged or retransmitted data, plus one octet.

The initial value of SND.RTX is SND.UNA. When an acceptable segment is received, if SND.UNA is advanced and (SND.RTX < new SND.UNA) is true, then the new SND.UNA is copied to SND.RTX. When a segment is sent, if (SSEG.SEQ < SND.MAX && SND.RTX < SSEG.SEQ + SSEG.LEN) is satisfied, then SSEG.SEQ + SSEG.LEN is copied to SND.RTX. Note that SND.RTX is not rewound upon a retransmission timeout.

As a result, all retransmitted but unacknowledged data satisfies (SND.UNA <= data < SND.RTX), while all transmitted but unacknowledged data satisfies (SND.UNA <= data < SND.MAX). By the definition given in the terminology section, at least one octet in an "original data segment" satisfies (SND.RTX <= octet < SND.MAX), and all octets in a "retransmitted data segment" satisfy (SND.UNA <= octet < SND.RTX).

C.2 Loss Inference

This subsection describes how losses are inferred with or without SACK and DLI/TS2.

When DLI/TS2 is enabled, it infers losses of both original and retransmitted data segments in the range from SND.UNA to SND.MAX. Since DLI/TS2 counts the number of received segments with the TS option for inferring losses, it is not very robust against losses of segments sent by a remote node.

When SACK is enabled, IsLost(), as specified in [RFC3517], infers losses of original data segments. In other words, it can infer losses of data satisfying (SND.RTX <= data < SND.MAX), but it cannot infer losses of data satisfying (SND.UNA <= data < SND.RTX). Nevertheless, it is more robust against losses of ACK segments than DLI/TS2, because multiple SACK blocks can be sent on each segment. Therefore, in spite of its limitations, IsLost() is helpful even when DLI/TS2 is enabled.

When DLI/TS2 is disabled, if SACK is disabled or (SND.UNA < SND.RTX) is true, duplicate ACK segments need to be counted to trigger a Fast Retransmit (i.e., to infer the loss of data at SND.UNA), as follows:

- After a retransmission timeout, duplicate ACK segments SHOULD NOT be counted until the retransmitted data is acknowledged.

  The purpose is to avoid counting duplicate ACK segments sent in reply to data segments that were sent before the timeout. Such duplicate ACK segments are often observed when a retransmission timeout is triggered because of the loss of the data segment sent by a Fast Retransmit.

- Duplicate ACK segments for data below SND.UNA SHOULD NOT be counted. That is, if SACK is enabled, ACK segments with D-SACK [RFC2883] below RSEG.ACK and ACK segments without SACK blocks SHOULD NOT be counted.

- Duplicate ACK segments for data above SND.UNA SHOULD be counted. That is, if SACK is enabled, ACK segments with D-SACK above RSEG.ACK and ACK segments with SACK blocks but without D-SACK SHOULD be counted. If TS2 is enabled, however, segments without the TS option SHOULD NOT be counted for accuracy.

- When SND.UNA is advanced in the loss recovery phase, regardless of the number of received duplicate ACK segments, data starting at the new SND.UNA SHOULD be inferred lost as with NewReno [RFC3782]. This is especially helpful when SACK is enabled and (new SND.UNA < SND.RTX) is true.

C.3 SACK Scoreboard

A data sender SHOULD maintain a SACK scoreboard carefully so that it can effectively recover losses and transmit new data.

According to section 5.1 of [RFC2018], "When a retransmit timeout occurs the data sender MUST ignore prior SACK information in determining which data to retransmit". When TS2 is enabled, however, this appendix recommends that the SACK scoreboard not be discarded upon a retransmission timeout. Instead, it recommends that existing SACK blocks in the SACK scoreboard be updated by newly received SACK blocks if there are conflicts, as follows.

- One variable is associated with each SACK block: SB.RcvTS (32bit-timestamp). It holds the RSEG.TSval value on the segment that last updated this SACK block.

- When a received SACK block other than a D-SACK block satisfies (RSEG.TSval > SB.RcvTS), where RSEG.TSval represents the

RSEG.TSval value on the segment that carried the received SACK block, the corresponding existing SACK block SHOULD be overwritten by the received SACK block in order to avoid possible conflicts. Otherwise, if (RSEG.TSval == SB.RcvTS) is true, the corresponding existing SACK block MAY be expanded by the received SACK block.

- If RSEG.ACK points to the middle of an existing SACK block, the start sequence number of the existing SACK block is changed to new SND.UNA + 1 SMSS without updating SB.RcvTS. Then, the data at SND.UNA are inferred lost.

In the Slow Start and Congestion Avoidance phase [RFC2581], when (SND.NXT < SND.MAX) is true (i.e., when SND.NXT has been rewound because of a retransmission timeout), SND.NXT SHOULD skip the SACKed data so as not to retransmit it. In addition, skipped SACKed data SHOULD NOT be calculated as part of the flight size.

If ABC [RFC3465] is enabled, then when an ACK segment is received, the number of octets acknowledged by the ACK segment needs to be calculated. In this calculation, already SACKed data SHOULD be omitted. Since the SACK information may not be fully synchronized with the data receiver, the number of octets acknowledged by each ACK segment SHOULD NOT exceed some upper bound (e.g., 2 SMSS).

Note: According to the fourth paragraph of section 2.3 in [RFC3465], TCP stacks need to determine whether a TCP connection is "during a slow start phase that follows a retransmission timeout". This appendix recommends that (SND.NXT < SND.MAX) be used to determine this.

C.4 SACK-LF (SACK Lowest First)

A data receiver SHOULD inform its data sender of appropriate SACK information so that the sender can recover lost data effectively.

A data receiver maintains a queue of SACK blocks to be sent in the TCP SACK option to the data sender. To comply with section 4 of [RFC2018], when a SACK block is updated, it is typically moved to the head of the queue. As a result, the most recently updated SACK blocks are informed to the data sender using the TCP SACK option.

Suppose that some data segments are lost within an RTT. In this case, a data receiver typically receives the out-of-order data segments in ascending order. Therefore, SACK blocks sent in reply within the same RTT (or the first RTT) are typically sorted in descending order. In contrast, within the next RTT (or the second RTT), if the data receiver receives all the lost data, the same SACK blocks (which would be the highest SACK blocks) on the last ACK

segment within the first RTT, excluding cumulatively acknowledged SACK blocks, are sent in reply, while RSEG.ACK is gradually advancing. In general, by considering the possibility that some retransmitted data segments are lost, the most recently updated SACK blocks (which would be located far from SND.NXT) will be sent in reply within the second or later RTTs, while the data sender would want to confirm the SACK blocks just above SND.NXT.

In the current standard TCP, whenever a retransmitted data segment is lost, a retransmission timeout is triggered in order to re-retransmit the lost data. According to section 5.1 of [RFC2018], "When a retransmit timeout occurs the data sender MUST ignore prior SACK information in determining which data to retransmit". Thus, for the same reason discussed in the previous paragraph, the data receiver keeps sending the same SACK blocks, which likely would be the highest SACK blocks. As a result, the data sender will retransmit all data between SND.UNA and the lowest reported SACK block. This retransmitted data will include data that was SACKed before the retransmission timeout. That is, bandwidth might be wasted if the data sender complies with section 5.1 of [RFC2018].

To mitigate this problem, this subsection proposes SACK-LF, as follows:

When RCV.NXT is advanced at a data receiver, a certain number of the lowest SACK blocks are moved to the head of the queue. The number of SACK blocks to be moved is chosen so that all SACK blocks are sent the same number of times, so as to make the SACK information robust against losses of ACK segments.

This memo proposes that the number of SACK blocks to be moved to the head of the queue be the sum of the following two numbers plus one (i.e., num_removed + num_lowest + 1).

- num_removed:

  The number of SACK blocks that were sent in the previous TCP SACK option and are removed by the received RSEG.ACK.

- num_lowest:

  The number of SACK blocks that were sent in the previous TCP SACK option and are in the current lowest N SACK blocks, where N is the number of SACK blocks sent in the previous TCP SACK option.

If a data sender discards the SACK scoreboard upon a retransmission timeout, SACK-LF that is performed at a data receiver will mitigate the number of unnecessary retransmissions. If D-SACK is not

supported by the data sender, SACK-LF will also mitigate the number of spurious Fast Retransmits. If the SACK information has not been fully synchronized with the data receiver, SACK-LF will suppress unnecessary retransmissions.

In addition to SACK-LF, this subsection proposes the following:

- If a data receiver discards part of an out-of-order consecutive data block that has been informed to the data sender by using the TCP SACK option, the shrunken SACK block SHOULD be moved to the head of the queue in order to inform of the change.

- When a data receiver receives a data segment, if it discards part or all of the data, the SACK blocks on the segment sent in reply SHOULD NOT include the discarded part of the data. Note that section 8 of [RFC2018] says "MUST" instead of "SHOULD NOT".

Appendix D: Summary of TCP Timestamps Option in RFC1323

The TCP Timestamps option [RFC1323] is currently deployed widely. There is also a variant of the TCP Timestamps option, which probably is more prevalent than the option described in [RFC1323]. The variant is called "rfc1323bis" [JBB03] (see also [Bra93] and [JBB97]) in this appendix. For simplicity, the TCP Timestamps option is called "the TS option" here.

This appendix describes the behaviors of the TCP Timestamps option specified in RFC1323 and rfc1323bis, by using C-like pseudocode. Some definitions are borrowed from the TS2 reference given in appendix A.

D.1 Types

The following types are borrowed from the TS2 reference: boolean, integer, 32bit-sequence-number, 32bit-timestamp, and internal-time.

D.2 Functions

The following functions are borrowed from the TS2 reference: SEQ_LE(), SEQ_LT(), TS_LT(), TS_LE(), and TIME_LT().

D.3 Inequalities

The following inequalities are defined.

- Inequality (A) ... RFC1323

  ```
  (SEQ_LE(RSEG.SEQ, Last.ACK.sent) &&
   SEQ_LT(Last.ACK.sent, RSEG.SEQ + RSEG.LEN))
  ```

- Inequality (B) ... rfc1323bis

  ```
  SEQ_LE(RSEG.SEQ, Last.ACK.sent)
  ```

  Note: (RSEG.TSval >= TS.Recent) is omitted in this inequality because it is part of the PAWS test.

Only one of (A) or (B) SHOULD be implemented. A boolean function called TS_ISLEG() returns true if the selected inequality is satisfied. Otherwise, it returns false.

Note: In addition to the inequalities given above, this memo recommends that (Last.Ack.Sent - max(RCV.WND) <= RSEG.SEQ) also be checked, in addition to (A) or (B).

D.4 Variables

The following variables are defined in [RFC1323].

TS.Recent (32bit-timestamp)
- This variable records the maximum RSEG.TSval value on the received segments satisfying TS_ISLEG(). It is echoed in SSEG.TSecr.

Last.Ack.Sent (32bit-sequence-number)
- This variable holds the last SSEG.ACK value sent.

The following variables are defined here to describe the behaviors.

TS.Req (boolean)
- This variable represents a user's request:
  True if the TCP Timestamps option is requested.
  False, otherwise.
- The initial value is given by the user.

TS.OK (boolean)
- This variable is true if the TS option is enabled. The initial value is false. It is set to true if the TS option is exchanged on SYN and SYN+ACK segments in the TCP three-way handshake phase.

TS.Recent_time (internal-time)
- This variable holds the time when TS.Recent was last updated.

D.5 Current Time

The following pseudocode functions are defined here for getting the current time or current timestamp.

GetTime() --- Get Current Time (internal-time)
- Get the current time in an internal time format.
- The time returned by this function MUST NOT be wrapped in the lifetime of any TCP connections.

GetTS() --- Get Current Timestamp (32bit-timestamp)
- Get the current time in a 32bit unsigned integer so that it can be sent in the TS option.
- The timestamp unit MUST be in the range of 1 sec to 1 ms.

D.6 Constants

The following constants are defined here to describe the behaviors.

TS_GRANULARITY (32bit-timestamp)

- This constant represents the granularity of GetTS() in the unit of the return value of GetTS().

TS_PAWS_IDLE (internal-time) for PAWS
- The value of TS.Recent is valid for TS_PAWS_IDLE if TS.OK is true.
- The default value is 24 days.

D.7 Attributes of Received Segments

The following flags are borrowed from the TS2 reference: isSYN, isRST, isFirstSYN, isFirstSYNACK, and withTS.

D.8 Procedures

D.8.1 Initialization

When a TCP Control Block is created or reused, the procedure below is followed.

```
TS.Req = true or false;         /* Requested by user */
TS.OK = false;
```

D.8.2 Input Processing

When a segment is received, the procedure below is followed.

```
if (isFirstSYN || isFirstSYNACK) {
        if (TS.Req && withTS) {
                TS.OK = true;
                TS.Recent = RSEG.TSval;
                TS.Recent_time = GetTime();
        }
} else if (TS.OK) {
        /* (R1) PAWS */
        if (!isSYN && !isRST &&
            TIME_LT(GetTime() - TS.Recent_time, TS_PAWS_IDLE) &&
            TS_LT(RSEG.TSval, TS.Recent)) {
                /* This segment MUST be dropped. */
                /* An ACK with TS SHOULD be sent. */
        }
        /* (R2) If it is outside the window, reject it. */
        /* (R3) Update TS.Recent */
        if (TS_ISLEG() && TS_LE(TS.Recent, RSEG.TSval)) {
                TS.Recent = RSEG.TSVal;
                TS.Recent_time = GetTime();
        }
        /* RTTM: If it advances SND.UNA, do RTTM. */
        Measured_RTT = GetTS() - RSEG.TSecr + TS_GRANULARITY;
```

```
        }
```

D.8.3 Output Processing

When a segment is sent, the procedure below is followed.

```
        if (TS.OK) {
                /* Put the TCP Timestamps option */
                SSEG.TSval = GetTS();
                SSEG.TSecr = TS.Recent;
        }
        LAST.Ack.Sent = SSEG.ACK;
```

Appendix E: Issues with TCP Timestamps Option in RFC1323

This appendix discusses the issues with both the TCP Timestamps option in [RFC1323] and rfc1323bis [JBB03]. It also discusses how these issues are handled in TS1 and TS2.

E.1 RTTM

This subsection discusses the issues of RTT measurements.

Since the RTTMs in RFC1323, rfc1323bis, and TS1 take RTT measurements only when SND.UNA is advanced, they cannot take RTT measurements during the loss recovery phase, except when partial or full acknowledgement is received. In contrast, RTTM/TS2 can take RTT measurements whenever it receives the TS option, even when SND.UNA is not advanced.

When a remote node is compliant with RFC1323, RTTM overestimates RTTs in the following scenario.

> Assume that all data segments sent within an RTT arrive at the remote node but all ACK segments sent in reply are lost. Upon a retransmission timeout, the lowest lost data is retransmitted, and an ACK segment sent in reply is received.
>
> In this case, the TSecr field on the received ACK segment has the TSval value on the last original data segment that arrived at the remote node. Therefore, RTTM at the local node measures the time from when the last original data segment was sent until when an ACK segment sent in reply to the retransmitted data segment is received. Thus, the measured RTT is much longer than the real RTT and nearly equal to the RTO value [Duk03a].
>
> In contrast, if the remote node complies with RTTM in rfc1323bis, RTTM/TS1, or RTTM/TS2, then the received ACK segment carries the TSval value on the retransmitted data segment. Therefore, RTTM at the local node takes a correct RTT measurement, because it measures the time from when the lowest lost data is retransmitted until when its ACK segment is received.

When a remote node is compliant with RFC1323, rfc1323bis or TS1, RTTM overestimates RTTs in the following scenario.

> Assume that the local node sends a data segment but an ACK segment sent in reply is lost. Before the retransmission timeout at the local node, the remote node sends a data segment, which acknowledges the data sent by the local node.
>
> In this case, the TSecr field on the received data segment has the

TSval value on the data segment sent by the local node. Since SND.UNA at the local node is advanced by the received data segment, RTTM at the local node measures the time from when the data segment was sent by the local node until when the data segment is received. Thus, the measured RTT is longer than the real RTT [Duk03b].

In contrast, RTTM/TS2 will not take an RTT measurement because the data segment sent by the remote node carries the OTS option.

E.2 PAWS and Reordering

As described in appendix F, there is a possibility that a legitimate data segment could be discarded by PAWSs in RFC1323 and rfc1323bis when it is delayed because of reordering.

In addition, there is a possibility that a legitimate ACK segment in a unidirectional data flow could be discarded by PAWS in rfc1323bis when it is delayed because of reordering [Mil98].

In contrast, PAWS/TS1 is slightly more robust against reordering than PAWS in RFC1323 and rfc1323bis, because of TS1_PAWS_MARGIN. PAWS/TS2 is robust against reordering, and legitimate segments are unlikely to be discarded even when they are delayed because of reordering.

Note: Linux seems to comply with RFC1323, instead of rfc1323bis, and it appears to have implemented measures including the same idea as TS1_PAWS_MARGIN.

E.3 Spoofed Segment Detection

[PD04] proposes to detect spoofed segments by making use of the TSecr field. To achieve this goal, when an ACK segment is sent, its TSval value is the same value as the TSval value on the last data segment. Unfortunately, this mechanism makes it impossible to apply PAWS for ACK segments. In addition, there could be other unknown problems.

In contrast, PASA/TS2 detects spoofed segments without tweaking the TSval values. Thus, it does not have such problems.

E.4 Retransmitted Data Loss Inference

It has been said that if the TSval values on out-of-order data segments were echoed by a data receiver, the data sender would be able to infer losses of retransmitted data segments. The TCP Timestamps options in RFC1323, rfc1323bis, and TS1 cannot infer such losses.

In contrast, TS2 enables DLI/TS2 to infer losses of both

retransmitted data segments and original data segments.

E.5 Corner Case of Eifel

According to section 3.3 of [RFC3522], if a remote node supports the TCP Timestamps option in RFC1323 and does not support D-SACK [RFC2883], then when all ACK segments within an RTT are lost, the Eifel Detection Algorithm [RFC3522] will misinterpret the consequent retransmission timeout as a spurious timeout.

In contrast, if a remote node supports the TCP Timestamps option in rfc1323bis or TS1, there is no such problem.

E.6 Vulnerability

If an implementation that complies with rfc1323bis overwrites TS.Recent with RSEG.TSval whenever it receives a segment satisfying (RSEG.TSval >= TS.Recent && RSEG.SEQ <= Last.ACK.sent), it has a vulnerability [CVE05][CERT05].

In contrast, implementations complying with RFC1323, TS1, and TS2 do not have such a vulnerability when the window size is not very large. If TS2 is enabled, PASA/TS2 combined with PAWS/TS2 will detect spoofed segments even when the window size is very large.

E.7 Summary

The table below summarizes the issues discussed in this appendix.

	RFC1323	rfc1323bis	TS1	TS2
RTTM: Dup-ACKs	NG	NG	NG	OK
RTTM: Overestimation	NG	Fair	Fair	OK
PAWS: Reordering	NG	NG	Fair	OK
PASA: PAWS for ACKs	NG	NG	NG	OK
DLI: Retransmitted Data	NG	NG	NG	OK
Eifel: A Corner Case	NG	OK	OK	OK
Vulnerability	Fair	NG	Fair	OK

Table E-1: Summary of Issues with TCP Timestamps Option

Appendix F: Problem of PAWS in RFC1323 and Reordering

There is a possibility that legitimate data segments could be discarded by PAWS in [RFC1323] when those segments are delayed because of reordering. This appendix shows some examples of this problem, and describes a generic scenario and possible negative effects, then proposes a possible solution.

F.1 Example 1: Reordering and Fast Retransmit with Limited Transmit

In this example, suppose that TCP A is sending data to TCP B. Assume that TCP A supports the TCP Timestamps option in [RFC1323], TCP Congestion Control [RFC2581], and Limited Transmit [RFC3042], and that TCP B supports the TCP Timestamps option with PAWS in [RFC1323].

Suppose that the data segment sequence W.1, X.2, Y.3, Z.4, S.5 is sent by TCP A, where the letter indicates the sequence number and the digit represents the timestamp in the TSval field. In this data segment sequence, suppose that W.1 and X.2 are sent in the Congestion Avoidance phase, Y.3 and Z.4 are sent by Limited Transmit, and S.5 is sent by Fast Retransmit.

Figure F-1 illustrates the data segment sequence observed at TCP A. The x-axis represents time, and the y-axis represents the sequence number. W.1 through Z.4 and S.5 indicate the data segments sent. Each 'o' mark indicates a received ACK segment. Lines are drawn to connect the symbols between data segments and between ACK segments.

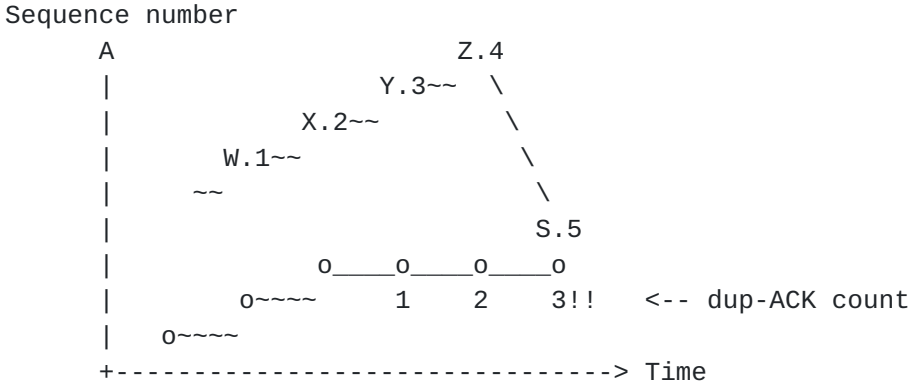


Figure F-1: Time vs. sequence number at TCP A

Now, suppose that the data segment sequence W.1, X.2, Y.3, Z.4, S.5 sent by TCP A is reordered as W.1, X.2, Y.3, S.5, Z.4 (i.e., Z.4 and S.5 are exchanged) on the path to TCP B. Figure F-2 illustrates the resulting data segment sequence observed at TCP B.

What happens at TCP B is described below.

0. Assume TS.Recent is valid and TS.Recent == 0.
   Assume RCV.NXT == S.

1. W.1 is received. PAWS accepts it because TS.Recent < 1.
   TS.Recent is not updated because RCV.NXT < W.

2. X.2 is received. PAWS accepts it because TS.Recent < 2.
   TS.Recent is not updated because RCV.NXT < X.

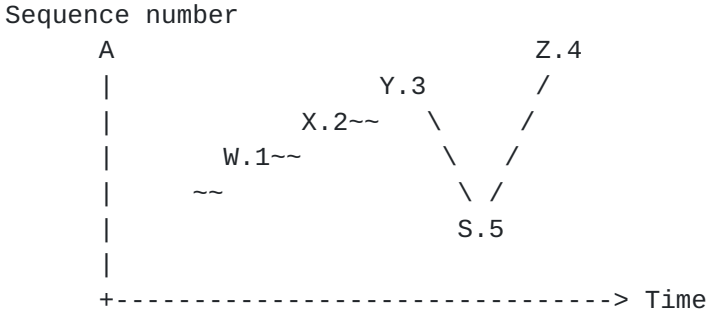
3. Y.3 is received. PAWS accepts it because TS.Recent < 3.
   TS.Recent is not updated because RCV.NXT < Y.

4. S.5 is received. PAWS accepts it because TS.Recent < 5.
   TS.Recent is updated because RCV.NXT == S and S.5 has data.

   Now, TS.Recent == 5 and RCV.NXT >= S + the data length of S.5.
   (The actual new value of RCV.NXT depends on the out-of-order
   data queue in TCP B.)

5. Z.4 is received. PAWS discards it because TS.Recent > 4.

In this example, the legitimate segment Z.4 is discarded by PAWS in step 5. Figure F-2 illustrates this scenario.



Segment	(prev)	W.1	X.2	Y.3	S.5	Z.4
PAWS	-	Pass	Pass	Pass	Pass	Fail
TS.Recent	0	0	0	0	5	5
RCV.NXT	S	S	S	S	>S	>S

Figure F-2: Time vs. sequence number at TCP B

Even in the case where TCP A does not support Limited Transmit (i.e., the case where Y.3 and Z.4 are not sent in the example above), if the data segment sequence W.1, X.2, S.5 sent by TCP A is reordered as W.1, S.5, X.2 (i.e., X.2 and S.5 are exchanged) on the path to TCP B, X.2 could be discarded by PAWS. Since there would be a small gap

between the time when X.2 is sent and the time when S.5 is sent, the possibility of this problem occurring would be less than in the example above.

F.2 Example 2: Reordering and NewReno

In this example, suppose that TCP A is sending data to TCP B. Assume that TCP A supports the TCP Timestamps option in [RFC1323], TCP Congestion Control [RFC2581], and NewReno [RFC3782], and that TCP B supports the TCP Timestamps option with PAWS in [RFC1323].

Suppose that the data segment sequence W.1, X.2, Y.3, Z.4, S.5 is sent by TCP A, where the letter indicates the sequence number and the digit represents the timestamp in the TSval field. In the data segment sequence, suppose that W.1 through Z.4 are sent by Fast Recovery at each time when a duplicate ACK segment is received, and that S.5 is sent by NewReno.

Figure F-3 illustrates the data segment sequence observed at TCP A. This figure uses the same notation that in Figure F-1.

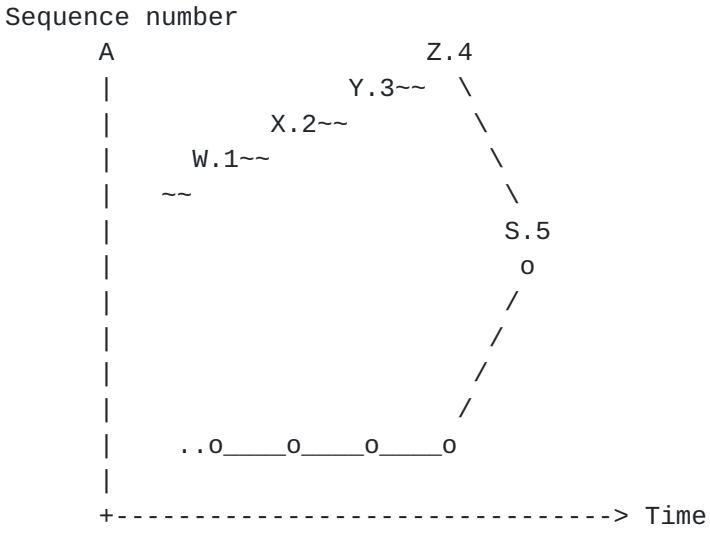


Figure F-3: Time vs. sequence number at TCP A

Now, suppose that the data segment sequence W.1, X.2, Y.3, Z.4, S.5 sent by TCP A is reordered as W.1, X.2, Y.3, S.5, Z.4 (i.e., Z.4 and S.5 are exchanged) on the path to TCP B.

The resulting data segment sequence observed at TCP B is the same as that shown in Figure F-2. What happens at TCP B is also the same as in Example 1 above. Consequently, the legitimate segment Z.4 is discarded by PAWS.

F.3 Generic Scenario

In general, this problem occurs in the following scenario.

Suppose that TCP A is sending data to TCP B, and data segments Z.4 and S.5 are sent by TCP A, where the letter indicates the sequence number and the digit represents the timestamp in the TSval field.

1. Data segment Z.4 is sent by the sender (TCP A).

2. Data segment S.5 is sent by the sender (TCP A).

   Note: Segment S.5 would be a retransmitted segment sent by Fast Retransmit, NewReno, SACK [RFC2018][RFC3517], or another mechanism that infers a segment loss and retransmits the lost data quickly. The sequence number of segment S.5 would be less than SND.NXT.

3. Segment S.5 arrives at the receiver earlier than segment Z.4.

   Suppose that segment S.5 satisfies (RSEG.SEQ <= RCV.NXT < RSEG.SEQ + RSEG.LEN), and that the TSval value on segment S.5 is not older than the TS.Recent value at the receiver (TCB B).

   Segment S.5 is accepted by PAWS at the receiver. TS.Recent at the receiver is updated with the TSval value on segment S.5 (i.e., TS.Recent = 5). RCV.NXT is also updated.

4. Segment Z.4 arrives at the receiver (TCP B).

   Segment Z.4 is discarded by PAWS because the TSval value (= 4) on segment Z.4 is older than the TS.Recent value (= 5) at the receiver.

In this scenario, the gap between the time when segment Z.4 is sent and the time when segment S.5 is sent should be small, so that reordering could exchange segments Z.4 and S.5.

F.4 Negative effects

This problem would cause some negative effects on TCP performance.

A data sender would spend additional time detecting a loss and recovering from it. Moreover, the sender would consider the loss to be a congestion indication, and the congestion window would needlessly be further reduced.

In addition, discarding legitimate segments at a data receiver is a waste of bandwidth.

F.5 Possible Solution

A straightforward way to solve this problem would be to modify the rules of PAWS so that valid delayed segments are accepted.

The new rule would be as follows:

- Change the inequality in R1) in section 4.2.1 of [RFC1323] as shown below:

    Current: RSEG.TSval < TS.Recent
    Proposal: RSEG.TSval < TS.Recent - T1, where T1 = RTO value.

- In addition, to keep TS.Recent be monotonically nondecreasing, in R3) in section 4.2.1 of [RFC1323], TS.Recent should be updated only when RSEG.TSval >= TS.Recent.

With this new rule, it would be very important to choose the value of T1 appropriately. This would be difficult for a data receiver, however, because it does not know the unit of the TSval values on the received segments.

Appendix G: Alternative Ideas

G.1 TCP Feature Array Option

Since the purpose of the OTS_OK option (i.e., the OTS option with option-length=2) is to negotiate the enabling of a feature, it could be replaced with a bit in something like a "binary option negotiation option" [All04]. The format would be like the following:

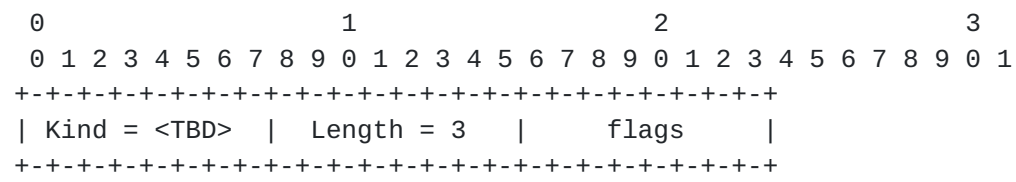


Figure B-1: The TCP Feature Array option

This idea is not employed because it requires additional TCP option space (i.e., at least 3 octets) and a new option-kind value.

Nevertheless, this new 3-octet option can be carried on a SYN segment even in the following combination (40 octets total).

- 4 octets: TCP MSS option [RFC793]
- 3 octets: TCP Feature Array option
- 3 octets: TCP Window Scale option [RFC1323]
- 10 octets: TCP Timestamps option [RFC1323]
- 2 octets: TCP SACK-PERMITTED option [RFC2018]
- 18 octets: TCP MD5 Signature option [RFC2385]

Normally, for alignment at a 32-bit boundary, one NOP is put after the TCP Window Scale option, and two NOPs are put before the TCP Timestamps option, as described in appendix A of [RFC1323]. If these three NOPs are removed, the TCP Feature Array option can be inserted as above without breaking the 32-bit timestamps alignment.

G.2 Timestamp Unit

In this memo, the timestamp unit for TS2 is fixed at 1 usec (10^-6). This value is advantageous for inferring losses of data and detecting spurious loss inference quickly, especially in highspeed networks, and taking finer RTT measurements in LAN environments. In addition, some lower bits of timestamps can be used as nonce to obfuscate timestamps.

An alternative idea would be to fix the unit at 1 ms (10^-3). Since [RFC1323] specifies that the unit is in the range of 1 second to 1 ms, the unit of 1 ms is interoperable with [RFC1323]. In addition, if the timestamp unit for TS2 is changed to 1 ms, PASA-DF/TS2 would

become more powerful, because the difference between TS.SndMax and TS.SndMin becomes thousand times smaller. This idea is not employed here, however, in order to obtain the advantages given in the previous paragraph.

Note: If the timestamp unit is changed to 1 ms, the variable TS.SndAdj can be removed. The default value of TS2_PAWS_IDLE is changed from 20 minutes to 24 days. TS.PASADF_On of PASA-DF/TS2 can be removed.

Another alternative idea would be to negotiate the timestamp unit by using SYN segments within the range between, e.g., 1 sec and 1 nsec. In this case, TS2_PAWS_IDLE should be replaced with a variable. This idea is not employed here because such negotiation would not be simple and would require additional TCP option space.

G.3 TS option without TSecr field

Since the value in the TSecr field in the OTS option may be very old and useless, an alternative idea would be to replace the OTS option (of option-length=10) with the TS option without the TSecr field (i.e., option-length=6) [Duk03b].

This idea is not employed here because the TSecr field in the OTS option is referred to by PASA-DF/TS2 and SLID/TS2. In addition, some new mechanisms might use this field in the future.

G.4 Eifel Detection Algorithm and TS2

This subsection shows the reason why this memo proposes not to apply the Eifel Detection Algorithm [RFC3522], which detects a posteriori spurious retransmissions by making use of the TCP Timestamps option [RFC1322], to TS2 by illustrating a case where the Eifel Detection Algorithm is not robust against reordering with TS2.

Suppose that TCP A is sending certain amount of data to TCP B, where TS2 is enabled on the TCP connection between them, and TCP A supports the Eifel Detection Algorithm. Assume that TCP A now sends two data segments: an original data segment S.1 and a retransmitted data segment R.2, where the letter indicates the sequence number and the digit represents the timestamp in the TSval field. The following scenario shows the problem.

1. First, TCP A sends an original data segment S.1 to TCP B.

2. Then, TCP A sends a retransmitted data segment R.2 to TCP B. The TSval value on data segment R.2 is recorded in RetransmitTS in [RFC3522]. Assume that this retransmission is genuine. Note that the sequence number R.2 is less than S.1.

3. Assume that the data segments sequence S.1, R.2 sent by TCP A is reordered as R.2, S.1 (i.e., they are exchanged) on the path to TCP B.

4. First, TCP B receives retransmitted data segment R.2. TCP B sends an ACK segment with TSecr=2 sent in reply to R.2. Assume that this ACK segment is lost.

5. Then, TCP B receives original data segment S.1. TCP B sends an ACK segment with TSecr=1 sent in reply to S.1.

6. TCP A receives an ACK segment with TSecr=1 only. Since the received TSecr value (=1) is less than RetransmitTS (=2) in [RFC3522], TCP A falsely infers the retransmission of data segment R.2 spurious.

As illustrated in this scenario, when TS2 is enabled, the Eifel Detection Algorithm is not robust against the combination of the reordering of data segments and the losses of ACK segments. The cause of this problem is that the Eifel Detection Algorithm assumes that TS.Recent is monotonically nondecreasing, while TS.Recent with TS2 is not monotonically nondecreasing (See section 4.4).

To solve this problem, SLID/TS2 is proposed. It compares RSEG.TSecr with the border timestamp calculated by TS2_SLID_BTS() specified in section 9.1, instead of comparing with either target timestamp or probe timestamp. With the current definition of TS2_SLID_BTS(), SLID/TS2 is robust against reordering if delays are less than RTT/2. Unfortunately, the permissible delays of SLID-SACK/TS2 may be less than RTT/TS2 depending on SACK hole sizes.

Appendix H: Changes from -00 version.

- Base Mechanism
  - Former sections 4.3 and 4.4 are renumbered to 4.4 and 4.5 in order to introduce new section 4.3 "Internal Timestamp and External Timestamp".

- RTTM
  - TS1_RTTM_G is renamed to TS1_GRANULARITY.
  - TS2_RTTM_G is renamed to TS2_GRANULARITY.
  - TS_RTTM_G is renamed to TS_GRANULARITY.

- PAWS
  - Section 6.2 is split into 6.2.1 and 6.2.2.

- PASA
  - In section 7.1.3 and appendix A.10.2.1: When a SYN+ACK segment is received in the SYN-SENT state, RSEG.TSecr is not tested now.
  - TS2_PASADF_RNDMAX_REUSE is introduced.
  - TS2_PASADF_RNDMAX_IDLE is introduced.

- DLI
  - DLD (Data Loss Detection) is renamed to DLI (Data Loss Inference).
  - Sections 8.1 to 8.3 are renumbered to 8.2.1 to 8.2.3.
    Sections 8.1 and 8.2 are added.
    Sections for DLI-UNA/TS2 and DLI-SACK/TS2 are exchanged.
    (Sections 8.2.2 and 8.2.3 are exchanged, and
     appendices A.5.4.2 and A.5.4.3 are exchanged.)
  - DS.Start and DS.End are added to appendix A.5.4.1.
  - TS.SndUnaTS is renamed to TS.UNA.SndTS.
  - TS.SndUnaRO is renamed to TS.UNA.SndRO.
  - Section 8.2.4 "DLI-NXT/TS2" is added.
    Appendix A.5.4.4 is also added.
  - Section 8.2.5 "DLI-MAX/TS2" is added.
    Appendix A.5.4.5 is also added.

- SLID
  - SRD (Spurious Retransmission Detection) is replaced with SLID (Spurious Loss Inference Detection).

- Appendices
  - isSYNACK is introduced in appendix A.
  - Former Appendices A.1 to A.10 are renumbered to A.2 to A.11 in order to introduce new appendix A.1 "TCP Options".
  - Former appendix B is moved to appendix G in order to introduce new appendix B "Granularity of Timestamps".
  - Appendix H "Changes from -00 version" is added.

Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.