TCP Maintenance and Minor Extensions (tcpm)
Internet-Draft
Updates: 1323 (if approved)
Intended status: Experimental
Expires: November 29, 2011

R. Scheffenegger
NetApp, Inc.
M. Kuehlewind
University of Stuttgart

May 28, 2011

Additional negotiation in the TCP Timestamp Option field during the TCP handshake
draft-scheffenegger-tcpm-timestamp-negotiation-02

Abstract

A number of TCP enhancements in so diverse fields as congestion control, loss recovery or side-band signaling could be improved by making the values carried in the Timestamp option transparent, and changing the receiver side processing of timestamps in the presence of selective acknowledgements.

This documents specifies a backwards compatible way of negotiating for Timestamp capabilities, and lists a number of benefits and drawbacks of this approach.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on November 29, 2011.

Copyright Notice

Table of Contents

1. Introduction

The timestamp option originally introduced in [RFC1323] was designed solely for two-way delay measurement and to support a particular TCP algorithm (Reno). It would be useful to be able to support one-way delay measurement and to take advantage of developments since TCP Reno, such as selective acknowledgements (SACK) [RFC2018].

This specification defines a protocol for the two ends of a TCP session to negotiate alternative semantics for the timestamps they will exchange during the rest of the session. It updates RFC1323 but it is backwards compatible with implementations of RFC1323 timestamp options.

The RFC1323 timestamp protocol presents the following problems when trying to extend it for alternative uses:

a. Opaque meaning for the value in a timestamp.

   * A timestamp value (TSval) as defined in [RFC1323] is deliberately only meaningful to the end that sends it. The other end is merely meant to echo the value without understanding it. This is fine if one end is trying to measure two-way delay (round trip time). However, to measure one-way delay, timestamps from both ends need to be compared by one end, which needs to relate the values in timestamps from both ends to a notion of the passage of time that both ends share.

b. No control over which timestamp to echo.

   * A host implementing [RFC1323] is meant to echo the timestamp value of the most recent in-order segment received. This was fine for TCP Reno, but it is not the best choice for TCP sessions using selective acknowledgement (SACK) [RFC2018].

   * A [RFC1323] host is meant to echo the timestamp value of the earliest unacknowledged segment, e.g. if a host delays ACKs for one segment, it echoes the first timestamp not the second. It is desirable to include delay due to ACK withholding when a host is conservatively measuring RTT. However, is not useful to include the delay due to ACK withholding when measuring one-way delay.

c. Alternative protection against wrapped sequence numbers.

   * [RFC1323] also points out that the timestamps it specifies will always strictly monotonically increase in each window so

they can be used to protect against wrapped sequence numbers (PAWS). If the endpoints negotiate an alternative timestamp scheme in which timestamps may not monotonically increase per window, then it needs to be possible to negotiate alternative protection against wrapped sequence numbers.

To solve these problems this specification changes the wire protocol of the TCP timestamp option in two main ways:

1. It updates [RFC1323] to add the ability to negotiate the semantics of timestamp options. The initiator of a TCP session starts the negotiation in the TSecr field in the first <SYN>, which is currently unused. This specification defines the semantics of the TSecr field in a segment with the SYN flag set. A version number is included to allow further extension of capability negotiation in future.

2. It updates [RFC1323] to define version 0 of timestamp capabilities to include:

   * the duration in seconds of a tick of the timestamp clock using a floating point representation

   * agreement that both ends will echo the timestamp on the most recently received segment, rather than the one that would be echoed by an [RFC1323] host. There is no specific option to request this behavior, however it is implied by successful negotiation of both SACK and timestamp capabilities.

   * an ability to mask a specified number of the lower significant bits of the timestamp values, so they are not considered for timestamp calculations, or in an algorithm to protect against wrapped sequence numbers.

With this new wire protocol, a number of new use-cases for the TCP timestamp option become possible. Section 6 gives some examples. Further extensions might be required in future. Appendix A gives an example of a further version of timestamp capability negotiation that could be defined in the future.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The reader is expected to be familiar with the definitions given in [RFC1323].

Further terminology used within this document:

Timestamp clock rate
: This document refers to clock rates for convenience. A rate is expressed in Hertz (ticks-per-second). For signaling purposes, the rate is not directly indicated in the protocol in Hertz (s^-1) but as the duration between two ticks of the timestamp clock, measured in seconds (s). The reason is to have high precision at long durations (low frequencies) available in the encoding (see Section 5 for details).

Timestamp option
: This refers to the entire TCP timestamp option, including both TSval and TSecr fields.

Timestamp capabilities
: Refers only to the values and bits carried in the TSecr field of <SYN> and <SYN,ACK> segments during a TCP handshake. For signaling purposes, the timestamp capabilities are sent in clear with the <SYN> segment, and in an encoded form (see Section 5 for details) in the <SYN,ACK> segment.

3. Overview

The TCP Timestamp option (TSopt) provides timestamp echoing for round-trip time (RTT) measurements. TSopt is widely deployed and activated by default in many systems. [RFC1323] specifies TSopt the following way:

Kind: 8

Length: 10 bytes

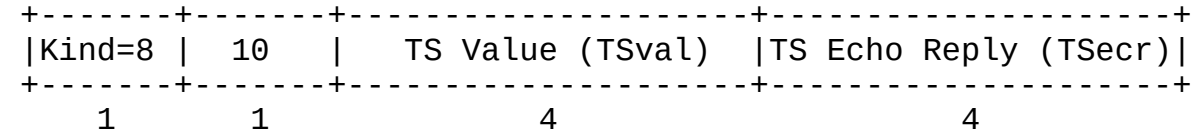

Figure 1: RFC1323 TSopt

"The Timestamps option carries two four-byte timestamp fields. The Timestamp Value field (TSval) contains the current value of the timestamp clock of the TCP sending the option.

The Timestamp Echo Reply field (TSecr) is only valid if the ACK bit is set in the TCP header; if it is valid, it echos a times- tamp value that was sent by the remote TCP in the TSval field of a Timestamps option. When TSecr is not valid, its value must be zero. The TSecr value will generally be from the most recent Timestamp option that was received; however, there are exceptions that are explained below.

A TCP may send the Timestamps option (TSopt) in an initial <SYN> segment (i.e., segment containing a SYN bit and no ACK bit), and may send a TSopt in other segments only if it re- ceived a TSopt in the initial <SYN> segment for the connection."

The comparison of the timestamp in the TSecr field to the current timestamp clock gives an estimation of the two-way delay (RTT). [RFC1323] specifies various cases when more than one timestamp is available to echo. The approach taken by [RFC1323] is not always be the best choice, i.e. when the TCP Selective Acknowledgment option (SACK) is used in conjunction. In addition there are use cases where one-way delay (OWD) measurements are needed. These mechanisms usually also rely on the TSopt to estimated the variation in OWD. Current implementations are based around certain assumptions,

* sender using one specific timestamp clock rate, or

* one specific rate from a limited set of possible timestamp clock rates, or

* the network conditions do not change for a short training period while timestamp values are sampled, and

* the sender using all bits of TSval to reflect the timestamp clock value directly with no bits used for different purposes such as covert channels.

These assumptions may not be valid in general in the public internet.

This document specifies a way of negotiating the timestamp capabilities available between the end hosts. This is enabled by using the TSecr field in the TCP <SYN> segment. In order to remain backwards compatible, a receiver capable of timestamp capability negotiation has to XOR the receivers (local) capabilities flags with the received TSval, before echoing the result back in the TSecr field. During the initial handshake, the sender has to store the sent initial TSval, in order to determine if the receiver can support this timestamp capability negotiation.

Enhancements in the area of TCP congestion control can use the measurement of the one-way delay variation as one input. However, without explicit knowledge of the partner’s timestamp clock, arriving at a good estimate requires a training phase over multiple segment exchanges. In this phase, the network conditions need remain nearly static to arrive at good measurements. In addition, the receiver has to assume that the full TSval represents the timestamp clock value of the sender, with no different use of some bits of the TSval. Covert channels or fingerprinting a timestamp value artificially increase the measurement noise, and a receiver may be lead to assume a higher timestamp clock rate than what is actually implemented by the sender. In order to assist such algorithms, explicit knowledge at an early phase of the session needs to be negotiated.

In addition, by using synergistic signaling between timestamps [RFC1323] and selective acknowledgments [RFC2018], enhancements in loss recovery are possible by removing any remaining retransmission and acknowledgment ambiguity. See Section 6 for a detailed discussion.

Receivers conforming to [RFC1323] are required to only reflect the timestamp of the last segment that was received in order, or the timestamp of the last not yet acknowledged segment in the case of delayed acknowledgments. In order to allow progressive deployment of

changed timestamp option semantics, a backwards compatible way of negotiating the semantic is required.

As the importance of the timestamp option increases by using it in more aspects of a TCP senders operation, so increases the importance of maintaining the integrity of the reflected timestamps. At the same time this must not inhibit the receiver to interpret a received timestamp in TSval.

This is achieved by indicating how many LSB bits of the timestamp value must not be interpreted by the receiver. Apart from the purpose of maintaining timestamp integrity for the use as input signal into congestion control algorithms, this also allows the use of timestamp based methods to discriminate at the earliest possible moment (within 1 RTT after the retransmission) between spurious retransmissions and genuine loss even when using slow running TCP timestamp clocks.

As an optional extension, a timestamp clock rate range negotiation is also introduced in Appendix A. This is only included as example of possible further enhancements.

4. Problem statement

Timestamp values are carried in each segment if negotiated for. However, the content of this values is to be treated as an opaque entity by the receiver. This document describes an enhancement to the timestamp negotiation, and must meet the following criteria:

- Indicate the (rough) timestamp clock rate used by the sender in a wide range. The slowest rate should be slower than 1 Hz, while the highest rate should allow unique timestamps per segment, even at extremely high link speeds. At the time of writing, the shortest meaningful duration was found to be a 64 byte packets (i.e. ACK segment) sent at a rate of 100 Gbit/s. This corresponds to a maximum timestamp clock rate of around 200 MHz, or a tick duration at about 5 ns.

- Allow for timestamps that are not directly related to real time (i.e. segment counting, or use of the timestamp value as a true extension of sequence numbers).

- Provide means to prevent or at least detect tampering with the echoed timestamp value.

- Allow for future extensions that may use some of the timestamp value bits for other signaling purposes for the remainder of the session.

- Signaling must be backwards compatible with existing TCP stacks implementing basic [RFC1323] timestamps. Current methods for timestamp value generation must be supported.

- Allow to state timing information explicitly during the initial handshake, to avoid a training phase extending beyond the initial handshake.

- Possibly provide a means to disambiguate resent <SYN> segments.

Some legacy implementations exist that violate [RFC1323] in that the TSecr field in a <SYN> is not cleared (see [I-D.ietf-tcpm-tcp-security]. The protocol should have some resiliency in the presence of such misbehaving senders, and must not lead to an unfair advantage for such wrongly negotiated sessions.

As there exist some benefit to change the receiver side treatment of which timestamp value to echo, the negotiation protocol itself must also provide some backwards compatibility. Therefore, even when a sender tries to negotiate for a higher version than supported by the receiver, the receiver MUST respond with at least version 0. Also, a

future protocol enhancement MUST make sure that any extension is compatible with at least version 0.

5. Signaling

To support these design goals stated in Section 4, only the TSecr field in the initial <SYN> can be used directly. The response from the receiver has to be encoded, since no unused field is available in the <SYN,ACK>. The most straightforward encoding is a XOR with a value, known to the sender. Therefore, the receiver also uses TSecr to indicate it's capabilities, but calculates the XOR sum with the received TSval. This allows the receiver to remain stateless and functionalities like syncache (see [RFC4987]) can be maintained with no change.

During the initial TCP three-way handshake, timestamp capabilities are negotiated using the TSecr field. Timestamp capabilities MAY only be negotiated in TSecr when the SYN bit is set. A host detects the presence of timestamp capability flags when the EXO bit is set in the TSecr field of the received <SYN> segment. When receiving a session request (<SYN> segment with timestamp capabilities), a compliant TCP receiver is required to XOR the received TSval with the receivers timestamp capabilities. The resulting value is then sent in the <SYN,ACK> response.

A host initiating a TCP session must verify if the partner also supports timestamp capability negotiation and a supported version, before using enhanced algorithms. Note that this change in semantics does not necessarily change the signaling of timestamps on the wire after initial negotiation.

When selective acknowledgements [RFC2018] are also negotiated for, the immediate echoing of the last received timestamp value has to be enabled, regardless of the senders version of the timestamp capabilities.

To mitigate the effect from misbehaving TCP senders appearing to negotiate for timestamp capabilities, a receiver MUST verify that one specific bit (EXO) is set, and any reserved bits (currently 8, RES field) are cleared. This limits the chance for a receiver to mistakenly negotiate for version 0 capabilities to around 0.05%. However, as a receiver has to use changed semantics when reflecting TSval also for higher values in the version field, a misbehaving sender negotiating for SACK, but not properly clearing TSecr, may have a 37.5% chance of receiving timestamp values with modified receiver behavior. This may lead to an increased number of spurious retransmission timeouts, putting such a session to a disadvantage.

Once timestamp capabilities are successfully negotiated, the receiver must ignore an indicated number of opaque bits, before applying the heuristics defined in [RFC1323]. The monotonic increase of the

timestamp value could be violated for each newly sent segment, conflicting with the constraints imposed by PAWS.

The presented distribution of the common three fields, EXO, VER and MASK, that MUST be present regardless of which version is implemented in a compliant TCP stack, is a result of the previously mentioned design goals. The lower three octets MAY be redefined freely with subsequent versions of the timestamp capability negotiation protocol. This allows a future version to be implemented in such a way, that a receiver can still operate with the modified behavior, and a minimum amount of processing (PAWS)

The wide range of indicated timestamp clock rates (spanning 9 orders of (decimal) magnitude, or 28 binary digits, and the limitation to no more than 24 bits requires the use of a logarithmic encoding. Since the precision of the timestamp clock value is most valuable at low frequencies (long tick durations), the clock rate is encoded as a time duration. This results in full precision for common used timestamp clock tick durations, while allowing even higher frequencies at reduced precision (subnormal numbers representing very short tick durations). A format was chosen that resembles, but does not conform to, the format of an IEEE-754 binary16 representation.

The timestamp clock values a host is using must not necessarily run synchronous with the internal TCP clock. Different clock sources, such as a NTP stratum, RTC, CPU cycle counters, or other independent clocks can be used to derive the TSval. This allows the de-coupling of the coarse-grained TCP clock used for retransmission and delayed ACK timeouts, from the clock frequency indicated in the TSval itself. Since [RFC1323] timestamp clocks used to be only useful for RTT measurement, and calculation of the RTO, the straight forward use of the TCP timer directly seemed natural to minimize subsequent RTT calculations.

Most stacks will at first not be able to dynamically adjust their timestamp clock rate. Therefore, the indicated clock duration can be a static, compile time value. To use the indicated clock duration, for example to perform one-way delay variation calculations, simple integer operations can be used after an initial conversion of the wire presentation to longer (i.e. 32 or 64 bit) integer values.

5.1. Capability Flags

In order to signal the supported capabilities, the TSecr value is overloaded with the following flags and fields during the initial <SYN> and <SYN,ACK> segments. The initiating host of a session with timestamp capability negotiation has to keep minimal state to decode the returned capabilities XOR’ed with the sent TSval.

Kind: 8

Length: 10 bytes

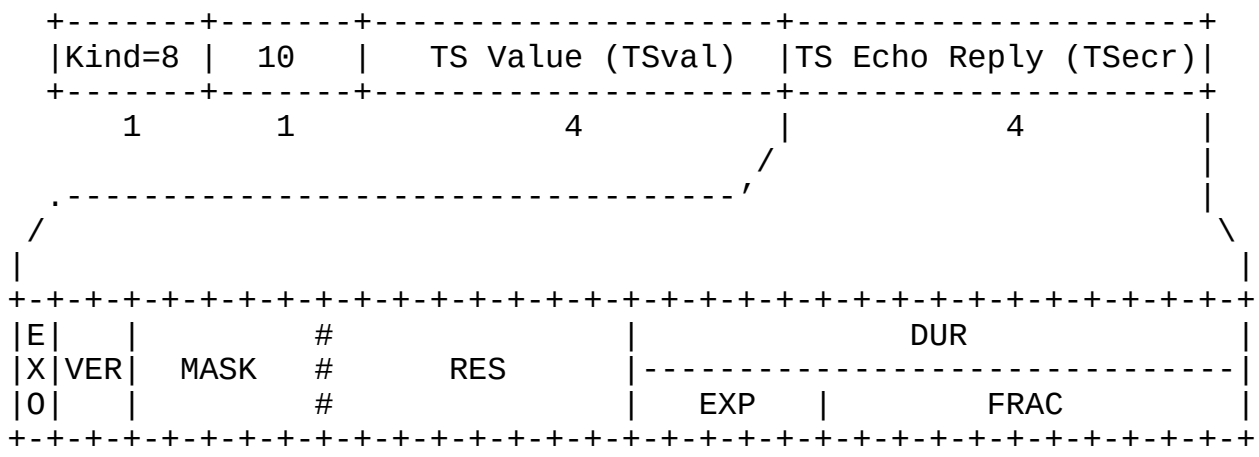

Figure 2: Timestamp Capability flags

Common fields to all versions:

EXO - Extended Options (1 bit)
: Indicates that the sender supports extended timestamp capabilities as defined by this document, and MUST be set to one by a compliant implementation. This flag also enables the immediate echoing of the TSval with the next ACK, if both timestamp capabilities and selective acknowledgement [RFC2018] are successful negotiated during the initial handshake. This change in semantics is independent of the version in the signaled timestamp capabilities.

VER - Version (2 bits)
: Version of the capabilities fields definition. This document specifies codepoint 0. With the exception of the immediate mirroring - simplifying the receiver side processing - and the masking of some LSB bits before performing the Protection Against Wrapped Sequence Numbers (PAWS) test, hosts must treat received timestamps as opaque entity and not use them as inputs into advanced heuristics, if the version is not supported. The lower 3 octets of the timestamp capability flags MUST be ignored if an unsupported version is received. It is expected, that a host will implement at least version 0. A receiver MUST respond with the appropriate (equal or version 0) version when responding to a new session request.

MASK - Mask Timestamps (5 bits)
The MASK field indicates how many least significant bits should be excluded by the receiver, before further processing the timestamp (i.e. PAWS, of for timing purposes). The unmasked portion of a TSval has to comply with the constraints imposed by [RFC1323] on the generation of valid timestamps, e.g. must be monotonic increasing between segments, and strict monotonic increasing for each window. Note that this does not impact the reflected timestamp in any way - TSecr will always be equal to an appropriate TSval. This field MUST be present in all future version of timestamp capability fields. A value of 31 (all bits set) MUST be interpreted by a receiver that the full TSval is opaque. For PAWS to be effective, at least 2 bits are required to discriminate between an increase (and roll-over) versus outdated segments.

Version 0 specific fields:

RES - Reserved (8 bits)
Reserved for future use, and MUST be zero ("0") with version 0. If timestamp capabilities are received with version set to 0, but some of these bits set, the receiver MUST ignore the extended options field and react as if the TSecr was zero (compatibility mode).

DUR - Duration (16 bits)
The timestamp clock tick duration, measured in seconds. This is a binary floating point value, indicating the length between two timestamp clock ticks. A value of zero (both exponent and fraction set to zero) is supported and indicates, that the timestamp values are NOT linear related to wall-clock time (i.e. the sender may perform some form of segment counting or sequence number extension instead). A host receiving a duration of zero from the other end host MUST NOT perform time-based heuristics which take the received TSval into account. The special floating point numbers infinity and not-a-number (NaN), where all exponent bits are set, are not supported.
Timestamp clock periods faster than 1 ms SHOULD be implemented by inserting the timestamp "late" before transmitting a segment to avoid unnecessary timing jitter. Shortest clock periods, with periods of only a few microseconds or less, are provided for hardware-assisted implementations.
The range of possible values runs from 15.99 s to 7.45 ns with highest precision, and down to 3.64 ps with reducing precision, which is also the shortest difference in tick

duration, that could be resolved. This equates to clock frequencies of 0.06 Hz, 134 MHz and 275 GHz respectively. Despite the provision of such a large dynamic range, a receiver should consider, that a timestamp clock may deviate from the indicated rate by a large fraction.

EXP - Exponent (5 bits)
The exponent component of the binary floating point number indicating the timestamp tick duration. The exponent bias is 28. Subnormal numbers (lower precision), where the exponent is set to zero, extend the lowest possible value representation to 2^-39 s (or 3.64 ps) at reduced precision. An exponent value of 31 MUST be treated as normal exponent. This allows timestamp clock ticks of up to 15.99 s. Note that this representation is not identical to the binary16 definition in IEEE 754-2008, and can not be processed as-is in a standard floating point library. See Section 6.1 for details.

FRAC - Fraction (11 bits)
The fraction component of a binary floating point number indicating the timestamp tick duration. The range with the highest resolution, excluding subnormal numbers, covers clock periods between 7.45 ns (or 134 MHz clock frequency) and 15.99 s (0.06 Hz). The field has an implicit lead bit with value 1 unless the exponent field is stored with all zeros.

Example for an timestamp capability negotiation, to indicate that the senders timestamp clock (tcp clock) is running with 1 ms per tick:

SYN, TSopt=<X>, TSecr=EXO|MASK|EXP=18|FRAC=0x031

The clock rate calculates as 2^(18-28)*1.00000110001b, thus indicates an actual clock rate of 999.93 us

5.2. Implicit extended negotiation

If both Timestamp capabilities and Selective Acknowledgement options [RFC2018] are negotiated (both hosts send these options in their respective segments), both hosts MUST echo the timestamp value of the last received segment, irrespective of the order of delivery. Note that this is in conflict with [RFC1323], where only the timestamp of the last segment received in sequence is mirrored. As SACK allows discrimination of reordered or lost segments, the reflected timestamps are not required to convey the most conservative information. If SACK indicates lost or reordered packets at the receiver, the sender MUST take appropriate action such as ignoring the received timestamps for calculating the round trip time, or

assuming a delayed packet (with appropriate handling). The exact implications are beyond the scope of this document.

The immediate echoing of the last received timestamp value allowed by the synergistic use of the timestamp option with the SACK option enables enhancements to improve loss recovery, round trip time (RTT) and one-way delay (OWD) variation measurements (see Section 6) even during loss or reordering episodes. This is enabled by removing any retransmission ambiguity using unique timestamps for every retransmission, while simultaneously the SACK option indicates the ordering of received segments even in the presence of ACK loss or reordering.

6. Possible use cases

6.1. One-way delay variation measurement

New congestion control algorithms are currently proposed, that react on the measured one-way delay variation (i.e. [I-D.ietf-ledbat-congestion], [Chirp]). This control variable is updated after each received ACK:

$$C(t) = TSval(t) - TSecr(t)$$

$$V(t) = C(t) - C(t-1)$$

provided that the timestamp clock rates at both ends are running at roughly the same rate. Without prior knowledge of the timestamp clock rate used by the partner, a sender can try to learn this rate by observing the exchanged segments for a duration of a few RTTs. However, such a scheme fails if the partner uses some form of implicit integrity check of the timestamp values, which would appear as either random scrambling of LSB bits in the timestamp, or give the impression of a much higher clock rate than what is actually used. If the partner uses some form of segment counting as timestamp value, without any direct relationship to the wall-clock time, the above formula will fail to yield meaningful results. Finally the network conditions need to remain stable during any such training phase, so that the sender can arrive at reasonable estimates of the partners timestamp clock rate.

This note addresses these concerns by providing a means by which both host are required to use a timestamp clock that is closely related to the wall-clock time, with known clock rate, and also provides means by which a host can signal the use of a few LSB bits for timestamp value integrity checks. To arrive at a valid one-way delay (OWD) variation, first the timestamp received from the partner has to be right-shifted by a known amount of bits as defined by the mask field. Next the local and remote timestamp values need to be normalized to a common base clock rate (typically, the local clock rate):

$$C_t = (TSecr >> \text{local mask}) - (TSval >> \text{remote mask}) * \frac{\text{remote clock rate}}{\text{local clock rate}}$$

$$V(t) = C(t) - C(t-1)$$

The adjustment factor can be calculated once during the timestamp capability negotiation phase, and pure integer arithmetic can be used during per-segment processing:

```
EXP.min = min(EXP.loc, EXP.rem)

EXP.rem -= EXP.min

EXP.loc -= EXP.min

FRAC.rem = (0x800 | FRAC.rem) << EXP.rem

FRAC.loc = (0x800 | FRAC.loc) << EXP.loc
```

and assuming that the local clock rate (tick duration) is lower

```
ADJ = FRAC.rem / FRAC.loc
```

with ADJ being a integer variable. For higher precision, two appropriately calculated integers can be used.

Any previously required training on the remote clock rate can be removed, resulting in a simpler and more dependable algorithm. Furthermore, transient network effects during the training phase which may result in a wrong inference of the remote clock rate are eliminated completely.

6.2. Early spurious retransmit detection

Using the provided timestamp negotiation scheme, clients utilizing slow running timestamp clocks can set aside a small number of least significant bits in the timestamps. These bits can be used to differentiate between original and retransmitted segments, even within the same timestamp clock tick (i.e. when RTT is smaller than the TCP timestamp clock rate). It is recommended to use only a single bit (mask = 1), unless the sender can also perform lost retransmission detection. Using more than 2 bits for this purpose is discouraged due to the diminishing probability of loosing retransmitted packets more than one time. A simple scheme could send out normal data segments with the so masked bits all cleared. Each advance of the timestamp clock also clears those bits again. When a segment is retransmitted without the timestamp clock increasing, these bits increased by one for each consecutive retry of the same segment, until the maximum value is reached. Newly sent segments (during the same clock interval) should maintain these bits, in order to maintain monotonically increasing values, even though compliant end hosts do not require this property. This scheme maintains monotonically increasing timestamp values - including the masked bits. Even without negotiating the immediate mirroring of timestamps (done by simultaneously doing timestamp capabilities negotiation, and selective acknowledgments), this extends the use of the Eifel Detection [RFC3522] and Eifel Response [RFC4015] algorithm to detect

and react to spurious retransmissions under all circumstances. Also, currently experimental schemes such as ER-SRTO [Cho08] could be deployed without requiring the receiver to explicitly support that capability.

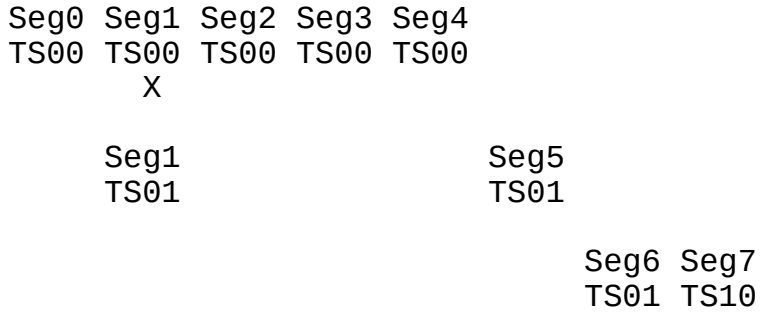

Figure 3: timestamp for spurious retranmit detection

Masked bits are the 2nd digit, the timestamp value is represented by the first digit. The timestamp clock "ticks" between segment 6 and 7.

6.3. Early lost retransmission detection

During phases where multiple segments in short succession (but not necessarily successive segments) are lost, there is a high likelihood that at least one segment is retransmitted, while the cause of loss (i.e. congestion, fading) is still persisting. The best current algorithms can recover such a lost retransmission with a few constraints, for example, that the session has to have at least DupThresh more segments to send beyond the current recovery phase. During loss recovery, when a retransmission is lost again, currently the timestamp can also not be used as means of conveying additional information, to allow more rapid loss recovery while maintaining packet conservation principles. Only the timestamp of the last segment preceding the continuous loss will be reflected. Using the extended timestamp option negotiation together with selective acknowledgements, the receiver will immediately reflect the timestamp of the last seen segment. Using both SACK and TS information synergistically, a sender can infer the exact order in which original and retransmitted segments are received. This allows a slightly less conservative and faster approach to retransmit lost retransmitted segments.

This can be implemented in combination with the masked bit approach described in the previous paragraph, or without. However, if the timestamp clock rate is lower than 1/2 RTT, both the original and the retransmitted segment may carry an identical timestamp. If the

sender cannot discriminate between the original and the retransmitted segments, is MUST refrain from taking any action before such a determination can be made.

In this example, masked bits are used, with a simple marking method. As the timestamp value of the retransmission itself is already different from the original segments, such an additional discrimination would not strictly be required here. The timestamp clock ticks in the first digit and the dupthresh value is 3.

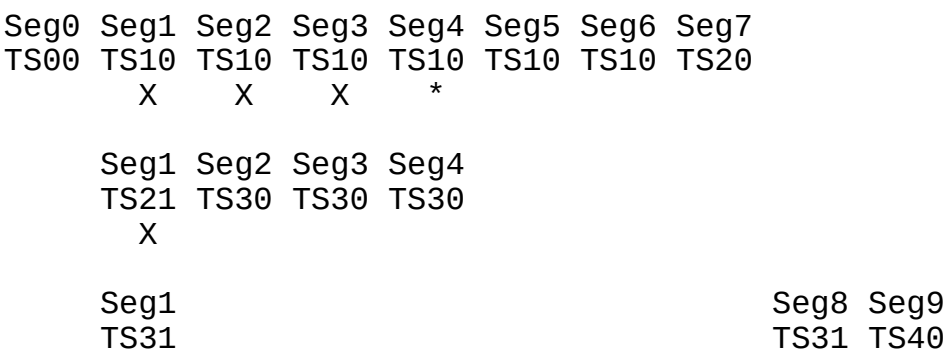

Figure 4: timestamp under loss

If Seg1,TS00 is lost twice, and Seg4,TS10 is also lost, the sender could resend Seg1 once more after seeing dupthresh number of segments sent after the first retransmission of Seg1 being received (ie, when Seg4 is SACKed). However, there is a ambiguity between retransmitted segments and original segments, as the sender cannot know, if a SACK for one particular segment was due to the retransmitted segment, or a delayed original segment. The timestamp value will not help in this case, as per RFC1323 it will be held at TS00 for the entire loss recovery episode. Therefore, currently a sender has to assume that any SACKed segments may be due to delayed original sent segments, and can only resolve this conflict by injecting additional, previously unsent segments. Once dupthresh newly injected segments are SACKed, continuous loss (and not further delay) of Seg1 can safely be assumed, and that segment be resent. This approach is conservative but constrained by the requirement that additional segments can be sent, and thereby delayed in the response.

With the synergistic use of timestamp extended options together with selective acknowledgments, the receiver would immediately reflect back the timestamp of the last received segment. This allows the sender to discriminate between a SACK due to a delayed Seg4,TS10, or a SACK because of Seg4,TS30. Therefore, the appropriate decision (retransmission of Seg1 once more, or addressing the observed reordering/delay accordingly [I-D.blanton-tcp-reordering] can be taken with high confidence.

6.4. Integrity of the Timestamp value

If the timestamp is used for congestion control purposes, an incentive exists for malicious receivers to reflect tampered timestamps, as demonstrated with some exploits [CUBIC].

One way to address this is to not use timestamp information directly, but to keep state in the sender for each sent segment, and track the round trip time independent of sent timestamps. Such an approach has the drawback, that it is not straightforward to make it work during loss recovery phases for those segments possibly lost (or reordered). In addition there is processing and memory overhead to maintain possibly extensive lists in the sender that need to be consulted with each ACK. Despite these drawbacks, this approach is currently implemented due to lack of alternatives (see [Linux], and [BSD10]).

The preferred approach is that the sender MAY choose to protect timestamps from such modifications by including a fingerprint (secure hash of some kind) in some of the least significant bits. However, doing so prevents a receiver from using the timestamp for other purposes, unless the receiver has prior knowledge about this use of some bits in the timestamp value. Furthermore, strict monotonic increasing values are still to be maintained. That constraint restricts this approach somewhat and limits or inhibits the use of timestamp values for direct use by the receiver (i.e. for one-way delay variation measurement, as the hash bits would look like random noise in the delay measurement).

6.5. Disambiguation with slow Timestamp clock

In addition, but somewhat orthogonal to maintaining timestamp value integrity, there is a use case when the sender does not support a timestamp clock rate that can guarantee unique timestamps for retransmitted segments. This may happen whenever the TCP timestamp clock rate is slower than the round-trip time of the path. For unambiguously identifying regular from retransmitted segments, the timestamp must be unique for otherwise identical segments. Reserving the least significant bits for this purpose allows senders with slow running timestamp clocks to make use of this feature. However, without modifying the receiver behavior, only limited benefits can be extracted from such an approach. Furthermore the use of this option has implications in the protection against wrapped sequence numbers (PAWS - [RFC1323]), as the more bits are set aside for tamper prevention, the faster the timestamp number space cycles.

Using Timestamp capabilities to explicitly negotiate mask bits, and set aside a (low) number of least significant bits for the above listed purposes, allows a sender to use more reliable integrity

checks. These masked bits are not to be considered part of the timestamp value, for the purposes described in [RFC1323] (i.e. PAWS) and subsequent heuristics using timestamp values (i.e. Eifel Detection), thereby lifting the strict requirement of always monotonically increasing timestamp values. However, care should be taken to not mask too many bits, for the reasons outlined in [RFC1323]. Using a mask value higher than 8 is therefore discouraged.

The reason for having 5 bits for the mask field nevertheless is to allow the implementation of this protocol in conjunction with TCP cookie transaction (TCPCT) extended timestamps [RFC6013]. That allows for nearly a quarter of a 128 bit timestamp to be set aside.

6.6. Opaque timestamps as segment digest

After making TCP alternate checksums historic ([RFC6247]), there still remains a need to address increased corruption probabilities when segment sizes are increased (see [I-D.ietf-tcpm-anumita-tcp-stronger-checksum]).

Utilizing an all-opaque TSval field allows the sender to include a stronger CRC32, with semantics independent of the fixed TCP header fields. However, such a use would again exclude the use of PAWS on the receiver side, and a receiver would need to know the specifics of the digest for processing. It is assumed, that such a digest would only cover the data payload of a TCP segment. In order to allow disambiguation of retransmissions, a special TSval can be defined (e.g. TSval=0) which bypasses regular CRC processing but allows the identification of retransmitted segments.

The full semantics of such a data-only CRC scheme are beyond the scope of this document, but would require a different version of the timestamp capability. Nevertheless, allowing the full TSval to remain unprocessed by the receiver for the purpose of PAWS even in version 0 could still allow the successful negotiation of sender-side enhancements such as loss recovery improvements (see Section 6.2, and Section 6.3).

In effect, the masked portion of the timestamp values represent an unreliable out of band signal channel, that could also be used for other purposes than solely performing timestamp integrity checks (for example, this would allow ER-SRTO algorithms [Cho08]).

6.7. Timestamp value as covert channel

Covert channels SHOULD NOT be implemented by using the mask field, as the explicit masking clearly points to such a channel. As the

regular operation of the timestamp clock is still maintained, covert channels working by artificially delaying data segments in an application (and thereby influencing the timestamp inserted into the segment) work unaffected. The received TSval would need to be shifted by the appropriate number of bits, before extracting the data from the covert channel by the receiver.

7. Discussion

RTT and OWD variation during loss episodes is not deeply researched. Current heuristics ([RFC1122], [RFC1323], Karn's algorithm [RFC2988]) explicitly exclude (and prevent) the use of RTT samples when loss occurs. However, solving the retransmission ambiguity problem - and the related reliable ACK delivery problem - would enable new functionality to improve TCP processing. Also, having an immediate echo of the last received timestamp value would enable new research to distinguish between corruption loss (assumed to have no RTT / OWD impact) and congestion loss (assumed to have RTT / OWD impact). Research into this field appears to be rather neglected, especially when it comes to large scale, public internet investigations. Due to the very nature of this, passive investigations without signals contained within the headers are only of limited use in empirical research.

Retransmission ambiguity detection during loss recovery would allow an additional level of loss recovery control without reverting to timer-based methods. As with the deployment of SACK, separating "what" to send from "when" to send it could be driven one step further. In particular, less conservative loss recovery schemes which do not trade principles of packet conservation against timeliness, require a reliable way of prompt and best possible feedback from the receiver about any delivered segment and their ordering. [RFC2018] SACK alone goes quite a long way, but using timestamp information in addition could remove any ambiguity. However, the current specs in [RFC1323] make that use impossible, thus a modified semantic (receiver behavior) is a necessity.

A synergistic signaling with immediate timestamp value echoes would however break legacy, per-packet RTT measurements. The reason is, that delayed ACKs would not be covered. Research has shown, that per-packet updates of the RTT estimation (for the purpose of calculating a reasonable RTO value) are only of limited benefit (see [Path99], and [PH04]). This is the most serious implication of the proposed synergistic signaling scheme with directly echoing the timestamp value of the segment triggering the ACK. Even when using the directly reflected timestamp values in an unmodified RTT estimator, the immediate impact would be limited to causing premature RTOs when the sending rate suddenly drops below two segments per RTT. That is, assuming the receiver implements delayed ACK and sending one ACK for every other data segment received. If the receiver has D-SACK [RFC2883] enabled, such premature RTOs can be detected and mitigated by the sender (for example, by increasing minRTO for low bandwidth flows).

8. Acknowledgements

The authors would like to thank Dragana Damjanovic for some initial thoughts around Timestamps and their extended potential use.

The editor would like to thank Bob Briscoe for his insightful comments, and the gratuitous donation of text, that have resulted in a substantially improved document.

9. Updates to Existing RFCs

Care has been taken to make sure the updates in this specification can be deployed incrementally.

Updates to existing [RFC1323] implementations are only REQUIRED if they do not clear the TSecr value in the initial <SYN> segment. This is a misinterpretation of [RFC1323] and may leak data anyway (see [I-D.ietf-tcpm-tcp-security]). Otherwise, there will be no need to update an RFC1323-compliant TCP stack unless the timestamp capabilities negotiation is to be used.

Implementations compliant with the definitions in this document shall be prepared to encounter misbehaving senders, that don't clear TSecr in their initial <SYN>. It is believed, that checking the reserved bits to be all zero provides enough protection against misbehaving senders.

10. IANA Considerations

With this document, the IANA is requested to establish a new registry to record the timestamp capability flags defined with future versions (codepoints 1, 2 and 3).

The lower 24 bits (3 octets) of the timestamp capabilities field may be freely assigned in future versions. The first octet must always contain the EXO, VER and MASK fields for compatibility, and the MASK field MUST be set to allow interoperation with a version 0 receiver.

This document specifies version 0 and the use of the last three octets to signal the senders timestamp clock rate to the receiver.

11. Security Considerations

The algorithm presented in this paper shares security considerations with [RFC1323] (see [I-D.ietf-tcpm-tcp-security]).

Some implementations address the vulnerabilities of [RFC1323], by dedicating a few low-order bits of the timestamp fields for use with a (secure) hash, that protects against malicious modification of TSecr value by the receiver. A MASK field has been provided to transparently notify the receiver about that alternate use of low-order bits. This allows the use of timestamps for purposes requiring higher integrity and security while maintaining transparency to the receiver.

12. References

12.1. Normative References

[RFC1323] Jacobson, V., Braden, B., and D. Borman, "TCP Extensions for High Performance", RFC 1323, May 1992.

[RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.

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

12.2. Informative References

[BSD10] Hayes, D., "Timing enhancements to the FreeBSD kernel to support delay and rate based TCP mechanisms", Feb 2010, <http://caia.swin.edu.au/reports/100219A/CAIA-TR-100219A.pdf>.

[CUBIC] Rhee, I., Ha, S., and L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant", Feb 2005, <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.153.3152&rep=rep1&type=pdf>.

[Chirp] Kuehlewind, M. and B. Briscoe, "Chirping for Congestion Control - Implementation Feasibility", Nov 2010, <http://bobbriscoe.net/projects/netsvc_i-f/chirp_pfldnet10.pdf>.

[Cho08] Cho, I., Han, J., and J. Lee, "Enhanced Response Algorithm for Spurious TCP Timeout (ER-SRTO)", Jan 2008, <http://ubinet.yonsei.ac.kr/v2/publication/hpmn_papaers/ic/2008_Enhanced%20Response%20Algorithm%20for%20Spurious%

20TCP.pdf>.

[I-D.blanton-tcp-reordering]
Blanton, E., Dimond, R., and M. Allman, "Practices for TCP Senders in the Face of Segment Reordering", draft-blanton-tcp-reordering-00 (work in progress), February 2003.

[I-D.ietf-ledbat-congestion]
Shalunov, S., Hazel, G., Iyengar, J., and M. Kuehlewind, "Low Extra Delay Background Transport (LEDBAT)", draft-ietf-ledbat-congestion-05 (work in progress), May 2011.

[I-D.ietf-tcpm-anumita-tcp-stronger-checksum]
Biswas, A., "Support for Stronger Error Detection Codes in TCP for Jumbo Frames", draft-ietf-tcpm-anumita-tcp-stronger-checksum-00 (work in progress), May 2010.

[I-D.ietf-tcpm-tcp-security]
Gont, F., "Security Assessment of the Transmission Control Protocol (TCP)", draft-ietf-tcpm-tcp-security-02 (work in progress), January 2011.

[Linux] Sarolahti, P., "Linux TCP", Apr 2007, <http://www.cs.clemson.edu/~westall/853/linuxtcp.pdf>.

[PH04] Eckstroem, H. and R. Ludwig, "The Peak-Hopper: A New End-to-End Retransmission Timer for Reliable Unicast Transport", Apr 2004, <citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.76.2748&rep=rep1&type=pdf>.

[Path99] Allman, M. and V. Paxson, "On Estimating End-to-End Network Path Properties", Sep 1999, <http://www.icir.org/mallman/papers/estimation.ps>.

[RFC1122] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.

[RFC2883] Floyd, S., Mahdavi, J., Mathis, M., and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP", RFC 2883, July 2000.

[RFC2988] Paxson, V. and M. Allman, "Computing TCP's Retransmission Timer", RFC 2988, November 2000.

[RFC3522] Ludwig, R. and M. Meyer, "The Eifel Detection Algorithm

for TCP", RFC 3522, April 2003.

[RFC4015] Ludwig, R. and A. Gurtov, "The Eifel Response Algorithm for TCP", RFC 4015, February 2005.

[RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, August 2007.

[RFC6013] Simpson, W., "TCP Cookie Transactions (TCPCT)", RFC 6013, January 2011.

[RFC6247] Eggert, L., "Moving the Undeployed TCP Extensions RFC 1072, RFC 1106, RFC 1110, RFC 1145, RFC 1146, RFC 1379, RFC 1644, and RFC 1693 to Historic Status", RFC 6247, May 2011.

Appendix A. Possible Extension

This section is not intended as normative description of an extension, but merely as an example of a possible extension. Future extensions MUST set the common fields in such a way that a receiver capable of version 0 only can react appropriately.

Certain hosts may want to negotiate a common optimal timestamp clock rate between each other for various purposes. For example, the balance between PAWS ([RFC1323]) and the timestamp clock resolution should be more towards one or the other. Also, if a hosts wants to have identical timestamp clock rates both at the sender and receiver to simplify one-way delay variation calculation, negotiating the clock rate could be useful. With identical timestamp clock rates, instead of multiplications and divisions, only additions and subtractions are required for OWD variation calculation.

Without a full three way handshake, full negotiation of the timestamp clock rate is not possible. For this reason, a special semantic is required during negotiation. This allows both ends know the exact timestamp clock rate with only two exchanged segments, while at the same time remaining compatible with version 0.

For this purpose, the following extension (version 1) of this proposal is one suggestion. Depending on the exact requirements, a different signaling may be more appropriate. For example, only the two different EXP fields could be required, while a single, but higher precision FRAC field for both low and high boundaries could suffice, and some additional signaling bits could be made available.

A.1. Capability Flags

Kind: 8

Length: 10 bytes

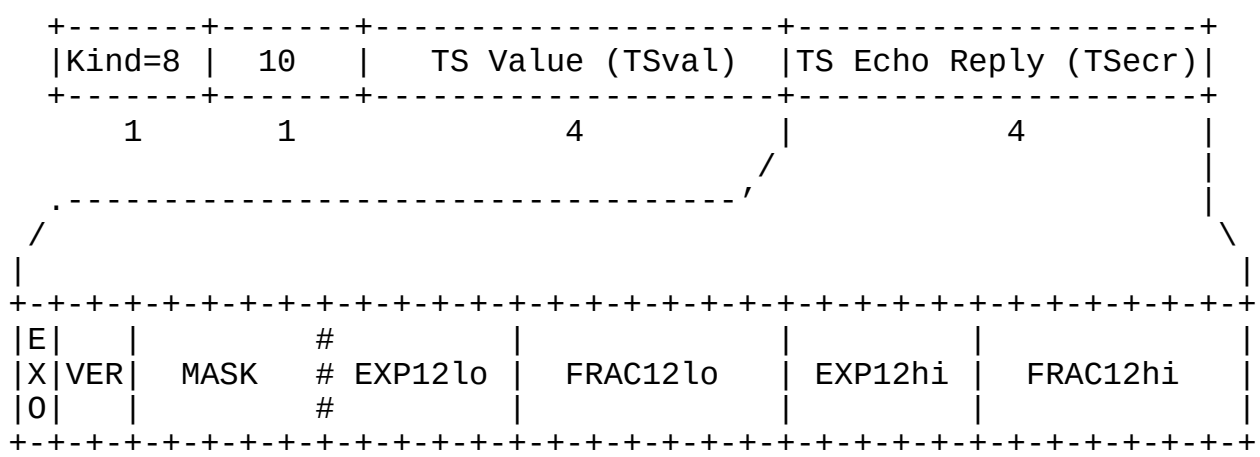

Figure 5: Timestamp Capability enhanced flags

The following additional fields are defined:

VER - version (2 bits)
: Version 1 could indicated that the sender is capable of adjusting the timestamp clock rate within the bounds of the two 12 bit fields (see Appendix A.2). A receiver that only implements version 0 SHOULD NOT ignore the timestamp capability negotiation entirely when encountering an unsupported version, any SHOULD respond with a version 0 response nevertheless (see below) - thereby enabling enhanced uses of the timestamp value and the modification of the receiver side timestamp processing.

EXP12lo and

EXP12hi - binary12 Exponent (5 bits each)
: The exponent component of a truncated, 12 bit floating point number indicating the possible timestamp clock ranges. The exponent bias is also 28, and no special numbers (infinity, NaN) are allowed. The exponent value 31 is treated like any other exponent value.

FRAC12lo and

FRAC12hi - binary12 Fraction (7 bits each)
The fraction component of a 12 bit floating point number. Subnormal numbers are allowed (Exponent value 0). This allows a range between 7.45 ns and 15.99 s with full resolution (lower bound is 0.06 ns using subnormal values). As a value of zero (both exponent and fraction set to zero) has a special meaning, it is not a valid number for range negotiation.

A.2. Range Negotiation

Only the host initiating a TCP session MAY offer a timestamp clock range, while the receiver SHOULD select a timestamp clock within these bounds. If the receiver can not adjust it’s timestamp clock to match the range, it MAY use a timestamp clock rate outside these bounds. If the receiver indicated a timestamp clock rate within the indicated bounds, the sender MUST set it’s timestamp clock rate to the negotiated rate. If the receiver uses a timestamp clock rate outside the indicated bounds, the sender MUST set the local timestamp clock rate to the value indicated by the closer boundary.

The following example sequence is provided to demonstrate how timestamp clock range negotiation works. Both sender and receiver finally know the clock rate of their respective partner.

SYN, TSopt=<X>, TSecr=EXO|VER=1|MASK|12bit-lo=1ms|12bit-hi=100ms

SYN,ACK, TSopt=<Y>, TSecr=<X>^EXO|VER=0|MASK|16bit=10ms

In this example, both hosts would run their respective timestamp clocks with a resolution of 10 ms.

SYN, TSopt=<X>, TSecr=EXO|VER=1|MASK|12bit-lo=1ms|12bit-hi=100ms

SYN,ACK, TSopt=<Y>, TSecr=<X>^EXO|VER=0|MASK|16bit=1000ms

In this example, the sender would set the timestamp clock rate to a resolution of 100 ms (closer to the receivers clock rate of 1 sec), while the receiver will have a timestamp clock rate running at 1 sec.

SYN, TSopt=<X>, TSecr=EXO|VER=1|MASK|12bit-lo=1ms|12bit-hi=100ms

SYN,ACK, TSopt=<Y>, TSecr=<X>^EXO|VER=0|MASK|16bit=100us

In this example, the sender would set the timestamp clock rate to a resolution of 10 ms (closest to the receiver’s clock rate of 100 us), while the receiver will have the timestamp clock running at 100 us.

Appendix B. Revision history

00 ... initial draft, early submission to meet deadline.

01 ... refined draft, focusing only on those capabilities that have an immediate use case. Also excluding flags that can be substituted by other means (MIR - synergistic with SACK option only, RNG moved to appendix A, BIA removed and the exponent bias set to a fixed value. Also extended other paragraphs.

02 ... updated document after IETF80 - referrals to "timestamp options" were seen to be ambiguous with "timestamp option", and therefore replaced by "timestamp capabilities". Also, the document was reworked to better align with RFC4101. Removed SGN and increased FRAC to allow higher precision.

Authors' Addresses

Richard Scheffenegger
NetApp, Inc.
Am Euro Platz 2
Vienna, 1120
Austria

Phone: +43 1 3676811 3146
Email: rs@netapp.com

Mirja Kuehlewind
University of Stuttgart
Pfaffenwaldring 47
Stuttgart 70569
Germany

Email: mirja.kuehlewind@ikr.uni-stuttgart.de