

TCP Maintenance and Minor Extensions
(tcpm)
Internet-Draft
Updates: [1323](#) (if approved)
Intended status: Experimental
Expires: April 25, 2013

R. Scheffenegger
NetApp, Inc.
M. Kuehlewind
University of Stuttgart
B. Trammell
ETH Zurich
October 22, 2012

**Additional negotiation in the TCP Timestamp Option field
during the TCP handshake
draft-scheffenegger-tcpm-timestamp-negotiation-05**

Abstract

A number of TCP enhancements in diverse fields as congestion control, loss recovery or side-band signaling could be improved by allowing both ends of a TCP session to interpret the value carried in the Timestamp option. Further enhancements are enabled by changing the receiver side processing of timestamps in the presence of Selective Acknowledgements.

This document updates [RFC1323](#) and specifies a backward-compatible method for negotiating for additional capabilities for the Timestamp option, 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 April 25, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	3
2.	Terminology	6
3.	Overview of the TCP Timestamp Option	7
4.	Extended Timestamp Capabilities	8
4.1.	Description	8
4.2.	Timestamp echo update for Selective Acknowledgments . . .	9
5.	Timestamp capability signaling and negotiation	10
5.1.	Capability Flags	10
5.2.	Timestamp clock interval encoding	12
5.3.	Negotiation error detection and recovery	12
5.4.	Interaction with Selective Acknowledgment	14
5.4.1.	Interaction with the Retransmission Timer	15
5.4.2.	Interaction with the PAWS test	16
5.5.	Discussion	16
6.	Acknowledgements	17
7.	Updates to Existing RFCs	17
8.	IANA Considerations	18
9.	Security Considerations	19
10.	References	19
10.1.	Normative References	19
10.2.	Informative References	19
Appendix A.	Possible use cases	21
A.1.	Timestamp clock rate exposure	21
A.2.	Early spurious retransmit detection	22
A.3.	Early lost retransmission detection	23
A.4.	Integrity of the Timestamp value	24
A.5.	Disambiguation with slow Timestamp clock	25
A.6.	Masked timestamps as segment digest	26
Appendix B.	Open Issues	27
Appendix C.	Revision history	27
	Authors' Addresses	29

1. Introduction

The Timestamp option originally introduced in [\[RFC1323\]](#) was designed to support only two very specific mechanisms, round trip time measurement (RTTM), and protection against wrapped sequence numbers (PAWS), assuming a particular TCP algorithm (Reno). The current semantics inhibit the use of the Timestamp option for other uses. Taking advantage of developments since TCP Reno, in particular Selective Acknowledgements (SACK) [\[RFC2018\]](#) allow different semantics, which in turn enable new uses for the Timestamp option, either for timing purposes (e.g. one-way delay variation measurement in the context of congestion control), or as unique token (e.g. for improved loss recovery).

This specification defines a protocol for the two ends of a TCP session to negotiate alternative semantics of the Timestamp option fields they will exchange during the rest of the session. It updates [RFC1323](#) but it is backwards compatible with implementations of [RFC1323](#) Timestamp options, and allows gradual deployment.

The [RFC1323](#) timestamp protocol presents the following problems when trying to extend it for alternative uses:

- a. Unclear meaning of 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 variation, 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 variation.

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. A version independent ability to mask a specified number of the lower significant bits of the timestamp values is present. These masked bits are not considered for timestamp calculations, or in an algorithm to protect against wrapped sequence numbers. Future extensions can thereby change the timestamp signaling without changing the modified treatment on the receiver side.
3. 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 time interval representation defined in [\[I-D.trammell-tcpm-timestamp-interval\]](#).
 - * 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.

With this new wire protocol, a number of new use-cases for the TCP timestamp option become possible. [Appendix A](#) gives some examples. Further extensions might be required in future. Two possible ways to extend the negotiation capabilities are mentioned, one maintaining some of the semantics specified herein, and a incompatible extension

to allow for other semantics.

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 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 of the TCP Timestamp Option

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

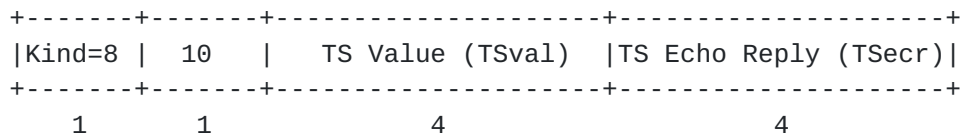


Figure 1: RFC1323 TSopt

"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 timestamp 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 received 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). With [RFC1323] the receiver is not supposed to interpret the TSval field for timing purposes, e.g. one-way delay variation measurements, but only to echo the content in the TSecr field. [RFC1323] specifies various cases when more than one timestamp is available to echo. The only property exposed to a receiver is a strict monotonic increase in value, for use with the protection against wrapped sequence numbers (PAWS) test. 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 on the same session.

4. Extended Timestamp Capabilities

4.1. Description

Timestamp values are carried in each segment if negotiated for. However, the content of these values is to be treated as an unmutable and largely uninterpreted entity by the receiver. The timestamp negotiation should allow for following criteria:

- o Allow to state timing information explicitly during the initial handshake, avoiding the proliferation of ad-hoc heuristics to determine this information via some other means. Heuristics that simply assume a specific timestamp clock intervals, or try to learn the clock interval used by the partner during a training phase extending beyond the initial handshake can thereby avoided. This is discussed further in [\[I-D.trammell-tcpm-timestamp-interval\]](#).
- o Indicate the (approximate) timestamp clock interval used by the sender in a wide range. The longest interval should be around 10 seconds, while the shorted interval should allow unique timestamps per segment, even at extremely high link speeds. A negotiation-method-independent representation for timestamp intervals is given in [\[I-D.trammell-tcpm-timestamp-interval\]](#).
- o 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).
- o Provide means to prevent or at least detect tampering with the echoed timestamp value, allowing for basic integrity and consistency checks.
- o Allow for future extensions that may use some of the timestamp value bits for other signaling purposes during the remainder of the session.
- o Signaling must be backwards compatible with existing TCP stacks implementing basic [\[RFC1323\]](#) timestamps. Current methods for timestamp value generation must be supported.
- o Allow for a means to disambiguate between retransmitted and delayed <SYN> segments.
- o Cater for broken implementations of [\[RFC1323\]](#), that either send a non-zero TSecr value in the initial <SYN>, or a zero TSecr value in <SYN,ACK>.

- o Provide flexibility to extend the negotiation protocol. Backwards-compatible and incompatible extensions of using timestamps should be available.

4.2. Timestamp echo update for Selective Acknowledgments

In [RFC1323], timing information is only considered in relation to calculating a (conservative) estimate of the round trip time, in order to arrive at a reasonable retransmission timeout (RTO). A retransmission timeout is a very expensive event in TCP, in terms of lost throughput and other metrics. For this reason, a receiver had to follow special rules in what timestamp to echo. This was to never underestimate the actual RTT, even during periods of loss or reordering on either the forward or return path. No other explicit signal could convey the presence of such events back to the sender at the time [RFC1323] was defined. Therefore a receiver had to make sure than at best, the timestamp of the last in-sequence segment would be echoed to the sender.

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.

When selective acknowledgment (SACK) is enabled on a session, the presence of a SACK option will explicitly signal reordering or loss to the sender. This information can be used to suspend the calculation of updated RTT estimates. As the SACK option will be present in multiple ACKs, this also prevents increasing RTT artificially when some of the ACKs, indicating loss, are dropped on the return path.

A receiver supporting the timestamp negotiation mechanism defined in this document MUST immediately reflect the value of TSval in the segment triggering an ACK, when the same session also supports SACK.

The rules to update the state variable TS.recent remain the identical to [RFC1323], and TS.recent must be evaluated when performing the PAWS test on the receiver side.

By this change of semantics when using the timestamps and selective acknowledgments [RFC2018] in the same session, enhancements in loss recovery are possible by removing any remaining retransmission and acknowledgment ambiguity. See [Appendix A](#) for a more detailed discussion. Through the modification to the handling of which timestamp to echo in the receiver, timestamps fulfill the properties of the "token", as described in [[I-D.sabatini-tcp-sack](#)].

5. Timestamp capability signaling and negotiation

In order to signal the supported capabilities, both the sender and the receiver will independently generate a timestamp capability negotiation field, as indicated below. The TSecr value field of the [RFC1323] TSopt is overloaded with the following flags and fields during the initial <SYN> and <SYN,ACK> segments. The connection initiator will send the timestamp capabilities in plain, as with [RFC1323] the TSecr is not used in the initial <SYN>. The receiver will XOR the local timestamp capabilities with the TSval received from the sender and send the result in the TSecr field. 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.

5.1. Capability Flags

Kind: 8

Length: 10 bytes

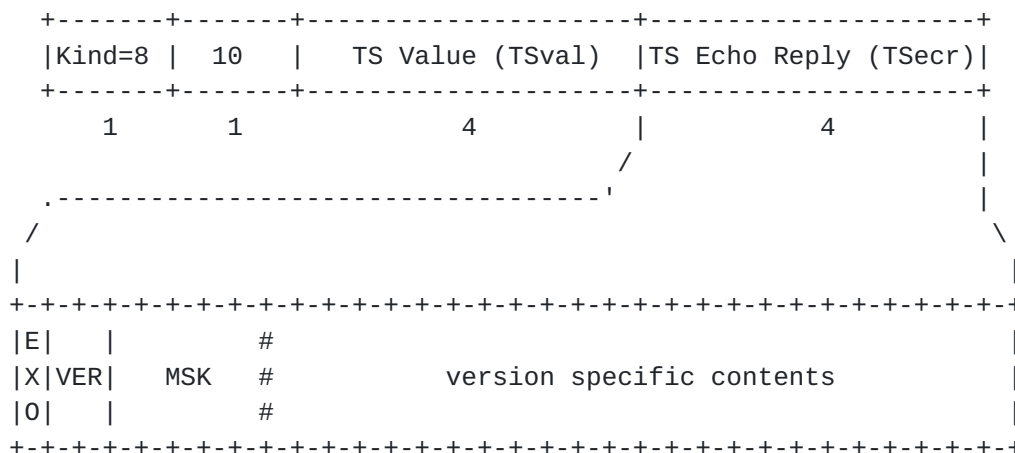


Figure 2: Timestamp Capability flags

Common fields to all versions:

EX0 - 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 (see [Section 4.2](#), and [Section 5.4](#)). 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 (00b). 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 not interpret the received timestamps and not use a timestamp value as input into advanced heuristics, if the version received is not supported. This is an identical requirement as with current [\[RFC1323\]](#) compliant implementations.

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.

MSK - 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, or 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 monotone increasing between segments, and strict monotone increasing for each TCP 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 to be ignored by any legacy heuristics, e.g. disabling PAWS. For PAWS to be effective, at least two not masked bits are required to discriminate between an increase (and roll-over) versus outdated segments.

5.2. Timestamp clock interval encoding

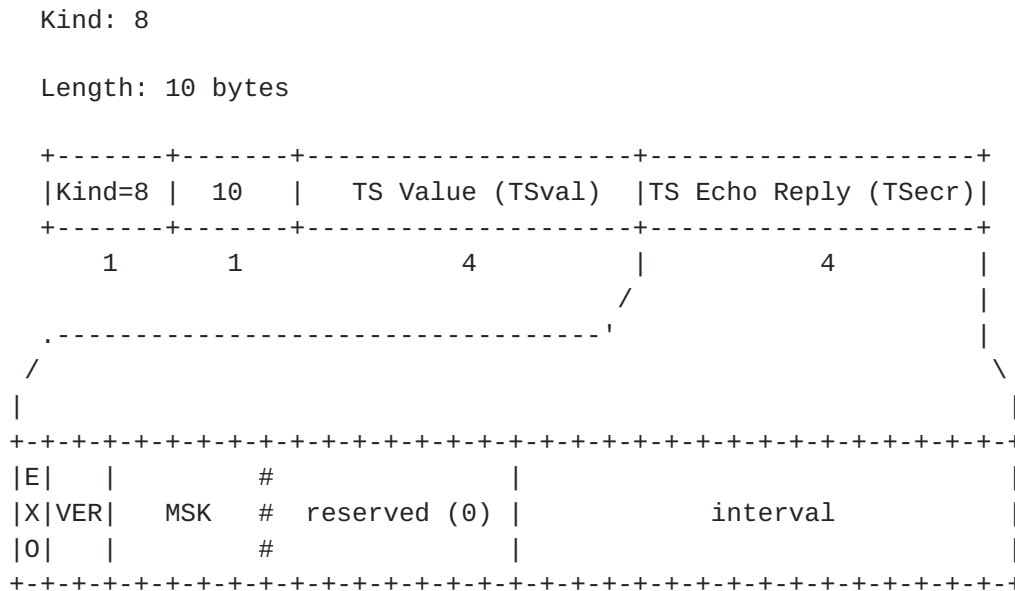


Figure 3: Timestamp Capability flags - version 0

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).

interval (16 bits)

The interval of the timestamp clock, as defined in [\[I-D.trammell-tcpm-timestamp-interval\]](#).

5.3. Negotiation error detection and recovery

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.

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 that is known to the sender. Therefore, the receiver also uses TSecr to indicate its capabilities, but calculates the XOR sum with the received TSval. This allows the receiver to remain stateless and functionality like SYN Cache (see [[RFC4987](#)]) can be maintained with no change.

If a sender has to retransmit the <SYN>, this encoding also allows to detect which segment was received and responded to. This is possible by changing the timestamp clock offset between retransmissions in such a way, that the decoding on the sender side would result in an invalid timestamp capability negotiation field (e.g. some RES bits are set). If the sender does not require the capability to differentiate which <SYN> was received, the timestamp clock offset for each new <SYN> can be set in such a way, that the TSopt of the <SYN> is identical for each retransmission.

As a receiver MAY report back a zero value at any time, in particular during the <SYN,ACK>, the sender is slightly constrained in its selection of an initial Timestamp value. The Timestamp value sent in the <SYN> should be selected in such a way, that it does not resemble a valid Timestamp capabilities field. One approach to ensure this property is that the sender makes sure that at least one bit of the RES field is set. This prevents a compliant sender to erroneously detect a compliant receiver, if the returned TSecr value is zero.

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.

To mitigate the effect from misbehaving TCP senders appearing to negotiate for timestamp capabilities, a receiver MUST verify that one specific bit (EX0) 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 in the presence of a misbehaving sender to around 0.05%. The prevalence of misbehaving senders, and distribution of observed TSecr values, limits this to less than 1 in 6 million. The modifications described in [[I-D.ietf-tcpm-1323bis](#)] and implemented in a receiver would further decrease the false negotiation to less than 10^{-7} .

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 (from an approximate population of 0.00036% of sessions being started without a cleared TSecr). This may lead to an increased number of spurious retransmission timeouts, putting such a session from a misbehaving TCP sender to a disadvantage.

Once timestamp capabilities are successfully negotiated, the receiver MUST ignore an indicated number of masked, low-order bits, before applying the heuristics defined in [\[RFC1323\]](#). The monotonic increase of the timestamp value for each new segment could be violated if the full 32 bit field, including the masked bits, are used. This conflicts 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)

[5.4.](#) Interaction with Selective Acknowledgment

If both Timestamp capabilities and Selective Acknowledgement options [\[RFC2018\]](#) are negotiated (both hosts send these options in their respective handshake 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 timestamp is 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). An updated algorithm to calculate the retransmission timeout timer (RTO) is beyond the scope of this document.

The immediate echoing of the last received timestamp value allowed by the simultaneous 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 [Appendix A](#)) 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.

For legacy applications of the timestamp option such as RTTM and PAWS, the presence of the SACK option gives a clear indication of loss or reordering. Under these circumstances, RTTM should not be invoked even under [\[RFC1323\]](#), but often is, due to separate handling of timestamp and SACK options).

The use of RTT and OWD measurements during loss episodes is an open research topic. A sender has to accommodate for the changed timestamp semantics in order to maintain a conservative estimate of the Retransmission Timer as defined in [\[RFC6298\]](#), when a TCP sender has negotiated for an immediate reflection of the timestamp triggering an ACK (i.e. both timestamp capability negotiation and Selective Acknowledgements are enabled for the session). As the presence of a SACK option in an ACK signals an ongoing reordering or loss episode, timestamps conveyed in such segments MUST NOT be used to update the retransmission timeout. Also note that the presence of a SACK option alleviates the need of the receiver to reflect the last in-order timestamp, as lost ACKs can no longer cause erroneous updates of the retransmission timeout.

[5.4.1.](#) Interaction with the Retransmission Timer

The above stated rule, to ignore timestamps as soon as a SACK option is present, is fully consistent with the guidance given in [\[RFC1323\]](#), even though most implementations skip over such an additional verification step in the presence of the SACK option.

To address the additional delay imposed by delayed ACKs, a compliant sender SHOULD modify the update procedure when receiving normal, in-sequence ACKs that acknowledge more than SMSS bytes, so that the input (denoted R in [\[RFC6298\]](#)) is calculated as

$$R = RTT * (1 + 1/(cwnd/smss))$$

If RTT (as measured in units of the timestamp clock) is smaller than the congestion window measured in full sized segments, the above heuristic MAY be bypassed before updating the retransmission timeout value.

5.4.2. Interaction with the PAWS test

The PAWS test as defined in [RFC1323] requires constant monotonic increasing values at the receiver side. As TS.Recent is no longer used to track which timestamp to echo, this variable can be reused. Instead of tracking the timestamp sent in the most recent ACK, a more strict update rule could be used:

"For example, we might save the timestamp from the segment that last advanced the left edge of the receive window, i.e., the most recent in-sequence segment."

TS.Recent is only to be updated whenever the left window advances, but no longer has to consider delayed ACKs.

5.5. 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 change in signaling with immediate timestamp value echoes would however break some 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 signaling scheme with directly echoing the timestamp value of the segment triggering the ACK, when the SACK options is also negotiated for. 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 also D-SACK [RFC2883] enabled, such premature RTOs can be detected and mitigated by the sender (for example, by increasing minRTO for low bandwidth flows).

Allowing timestamps to play a more important role in TCP signaling also gives rise to concerns. When the timestamp is used for congestion control purposes, this gives an incentive for malicious receivers to reflect tampered timestamps. During the early phases of the introduction of Cubic, such modifications were shown to result in unfair advantages to malicious receivers, that selectively alter the reflected timestamp values (see [CUBIC]). For that very reason, this document introduces the explicit possibility to include a signal in the timestamp values that is excluded from any processing by the receiver. A sender can then decide how to make use of this capability, e.g. for use as additional security information, improvements of loss recovery or other, yet unknown, means.

6. Acknowledgements

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

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

We further want to thank Michael Welzl for his input and discussion.

7. 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](#)]). Also see [[I-D.ietf-tcpm-1323bis](#)], as this stipulates, that the TSval sent in a <RST> should be zeroed, further reducing the chance for a false positive. It is expected, that these changes are implemented in stacks making use of timestamp negotiation. 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.

In the unlikely case of an erroneous negotiation of timestamp capabilities between a compliant receiver, and a misbehaving sender, the proposed semantic changes will trigger a higher rate of spurious retransmissions, while time-based heuristics on the receiver side may further negatively impact congestion control decisions. Overall, misbehaving receivers will suffer from self-inflicted reductions in TCP performance.

8. 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 interval to the receiver.

9. Security Considerations

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

An implementation can 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 returned timestamp value by the receiver. A MASK field has been provided to explicitly 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 allowing the receiver to extract useful information nevertheless.

10. References

10.1. Normative References

- [I-D.trammell-tcpm-timestamp-interval]
Scheffenegger, R., Kuehlewind, M., and B. Trammell,
"Exposure of Time Intervals for the TCP Timestamp Option",
[draft-trammell-tcpm-timestamp-interval-00](#) (work in
progress), October 2012.
- [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.

10.2. 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](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](http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.153.3152&rep=rep1&type=pdf)>.
- [Cho08] Cho, I., Han, J., and J. Lee, "Enhanced Response Algorithm
for Spurious TCP Timeout (ER-SRT0)", 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-tcpm-1323bis]

Borman, D., Braden, R., Jacobson, V., and R.
Scheffenegger, "TCP Extensions for High Performance",
[draft-ietf-tcpm-1323bis-04](#) (work in progress),
August 2012.

[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., "Survey of Security Hardening Methods for
Transmission Control Protocol (TCP) Implementations",
[draft-ietf-tcpm-tcp-security-03](#) (work in progress),
March 2012.

[I-D.sabatini-tcp-sack]

Sabatini, A., "Highly Efficient Selective Acknowledgement
(SACK) for TCP", [draft-sabatini-tcp-sack-01](#) (work in
progress), August 2012.

[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](http://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.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", [RFC 6298](#), June 2011.

[Appendix A](#). Possible use cases

[A.1](#). Timestamp clock rate exposure

Today, each TCP host may use an arbitrary, locally defined clock source to derive the timestamp value from. Even though only a handful of typically used clock rates are implemented in common TCP stacks, this does not guarantee that any future stack will choose the same clock rate. This poses a problem for current state of the art heuristics, which try to determine the senders timestamp clock rate by pure passive observation of the TCP stream, and affects both advanced heuristics in the partner host of a TCP session, or arbitrarily located passive observation points to estimate TCP session parameters.

The proposed mechanism would reveal this information explicitly, even though other environmental factors, such as the operation of a TCP stack in a virtualized environment, may result in some deviations in the actually used clock rate.

High-speed and real-time stacks would be expected to operate with higher clock rates, while the observed variance in (known) timestamp clock vs. reference clock could help in determining between physical and virtual end hosts, for example.

A.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 shorter than the TCP timestamp clock interval). 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 losing 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.

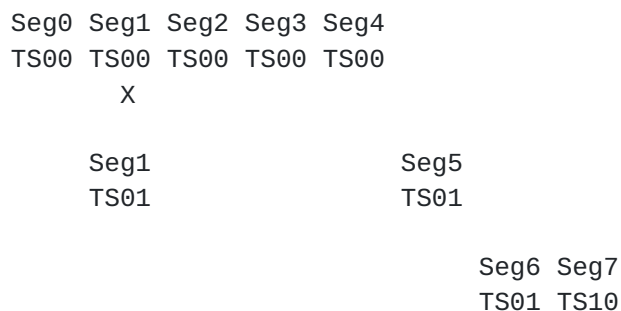


Figure 4: timestamp for spurious retransmit detection

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

A.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 in conjunction with each other, a sender can infer the exact order in which original and retransmitted segments are received. This allows faster recovery from lost retransmissions while maintaining the principle of packet conservations and avoiding costly retransmission timeouts.

The implementation can be done in combination with the masked bit approach described in the previous paragraph, or without. However, if the timestamp clock interval 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, it 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.

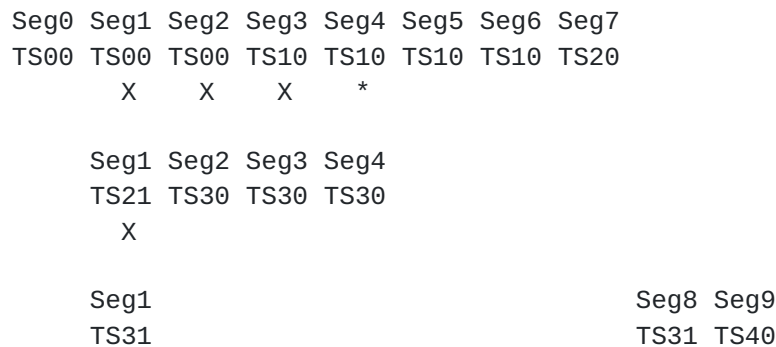


Figure 5: timestamp under loss

If Seg1,TS00 is lost twice, and Seg4,TS10 is also lost, the sender could resend Seg1 once more after observing dupthresh number of segments sent after the first retransmission of Seg1 being received (ie, when Seg4 is SACKed). However, there is an 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 simultaneous 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.

[A.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).

[A.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 interval that can guarantee unique timestamps for retransmitted segments. This may happen whenever the TCP timestamp clock interval is higher 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.

[A.6](#). Masked timestamps as segment digest

After making TCP alternate checksums historic (see [[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 a completely masked 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 [Appendix A.2](#), and [Appendix A.3](#)).

In effect, the masked portion of the timestamp value 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](#)]).

Appendix B. Open Issues

- o The split between this draft and [\[I-D.trammell-tcpm-timestamp-interval\]](#) is cursory; additional separation of timestamp interval export may be necessary.
- o [bht] suggest changing the "versioning" construct to a "capabilities" construct, especially since two bits of versioning might as well be none. The base specification would then define the alternate semantics WRT SACK and could use capabilities to define further semantics.
- o [bht] does it make sense to move masking out of the base spec and into the 8 "unused" bits in "version 0" (in order to get more capabilities bits / "magic bits" to reduce erroneous negotiation)?
- o [bht] does it make sense to define SACK-echo as version/capability independent?

Appendix C. Revision history

This appendix should be removed by the RFC Editor before publishing this document as a RFC.

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.

03 ... removed references to "opaque" and "transparent". substituted "timestamp clock interval" for all instances of rate. Changed signal encoding to resemble a scale/value approach like what is done with Window Scaling. As added benefit, clock quality can be implicitly signaled, since multiple representations can map to identical time intervals. Added discussion around resilience against broken [RFC1323](#) implementations (Win95, Linux 2.3.41+), which deviate from expected Timestamp signaling behavior.

04 ... removed previous [appendix A](#) (range negotiation); minor edit to improve wording; moved [Section 6](#) to the Appendix, and removed covert channels from the potential uses; added some text to discuss future versioning (compatible and incompatible variants); changed document structure; added guidance around PAWS; added pseudo-code examples (probably to be removed again)

05 ... added new Open Issues section, added reference to separate interval draft, removed content on timestamp interval exposure which now appears in the interval draft. Removed pseudocode examples until they can be reworked on finalization of the mechanism, as they refer to fields which have changed / moved to the interval draft.

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

Brian Trammell
Swiss Federal Institute of Technology Zurich
Gloriastrasse 35
8092 Zurich
Switzerland

Phone: +41 44 632 70 13
Email: trammell@tik.ee.ethz.ch

