

TCP Maintenance and Minor Extensions
(tcpm)
Internet-Draft
Intended status: Experimental
Expires: April 17, 2013

R. Scheffenegger
NetApp, Inc.
M. Kuehlewind
University of Stuttgart
B. Trammell
ETH Zurich
October 14, 2012

Exposure of Time Intervals for the TCP Timestamp Option
draft-trammell-tcpm-timestamp-interval-00.txt

Abstract

The TCP Timestamp option would be useful for additional measurements if it could be assumed that the interval between ticks of the timestamp clock are regular, and if that interval were known. In practice, many implementations do use a timestamp clock source that has a regular interval. This draft specifies a mechanism for exposing the timestamp interval to a receiver, and discusses applications therefor.

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 17, 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	3
3.	Timestamp interval exposure	3
3.1.	Interval encoding requirements	4
3.2.	Interval encoding specification	4
3.3.	Timestamp Interval experimental TCP option	6
3.4.	Interval export during TS negotiation	7
4.	Timestamp interval negotiation	7
5.	IANA Considerations	7
6.	Security Considerations	7
7.	References	7
7.1.	Normative References	7
7.2.	Informative References	8
Appendix A.	Detailed use cases for timestamp interval export	8
A.1.	Methodology for one-way delay variation measurement using known timestamp intervals	8
Authors' Addresses	10

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).

While [\[RFC1323\]](#) specifies only that timestamps "must be at least approximately proportional to real time" to support RTTM, many implementations generate timestamp values from a regular timing source. Determining the real-time interval represented by a single tick makes additional measurements possible. In addition to easing passive measurements using the timestamp option, it also makes possible the measurement of inter-departure time; the comparison of inter-departure time to inter-arrival time can be used to one-way delay variation measurement, useful for congestion control algorithms as well in QoS applications [FIXME: others?]

This document specifies a compact encoding for timestamp intervals which can be exported via multiple mechanisms, including an experimental TCP option, or the mechanism described in [\[I-D.scheffenegger-tcpm-timestamp-negotiation\]](#).

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 [\[RFC2119\]](#).

Terms defined in [\[RFC1323\]](#) are used in this document as defined there.

This document defines the following additional term:

Timestamp interval

The interval between two ticks of the timestamp clock source running at a constant frequency. Note that the timestamp clock is not required to be identical with the TCP clock, even though most implementations use the same clock for practical purposes.

3. Timestamp interval exposure

This section describes the requirements for interval encoding, then specifies an interval to meet these requirements based on a 16-bit reduced-precision encoding of a 42-bit fixed-point unsigned integer.

3.1. Interval encoding requirements

The choice of a timestamp interval is generally implementation-specific, and there are a small number of commonly chosen intervals. However, a general solution must support not only common cases, but uncommon ones, and provide future flexibility to allow an implementation to dynamically choose new timestamp intervals for new sockets, based on network conditions and specific requirements for timestamp measurements.

There are some sensible bounds on the range of timestamp intervals that must be reasonably supported. The minimum inter-packet interval for 64-byte packets (i.e., back-to-back ACK segments) on a future 400 Gigabit Ethernet would be about 1ns; smaller intervals need not be supported with current technology, even for applications for which a unique timestamp for every packet would be useful. On the other side of the scale, low-bandwidth, high-latency links may operate with timestamp intervals on the order of seconds.

The precision required by timestamp interval export, on the other hand, is determined by the applications for which the information will be used and the precision of the underlying clock source. As many clock sources may provide less than maximum precision (due to e.g. interrupt jitter), there should be some way to represent variable precision. [FIXME: justify why 11 bits is enough here.]

As a timestamp interval will need to be bound to a connection in-band at runtime, a space-efficient encoding is necessary.

These requirements indicate a reduced-precision encoding of a fixed-point interval, expressed in seconds, as described in the next subsection.

3.2. Interval encoding specification

A 42-bit fixed-point unsigned integer with 4 bits before the decimal point and 38 bits after, expressed in seconds, is sufficient to encode an interval range from just under 16 seconds (0x3ff ffff ffff) down to 2^{-38} s or 3.64 ps (0x000 0000 0001), meeting the range requirement. Sufficient precision for the applications envisioned by this document is provided by exporting just the 11 most significant bits of the interval value (here, the "value"), coupled with a 5-bit "scale" which locates the least significant bit of the value within the larger field: a scale of 31 places the value field between bits 41 and 31 inclusive of the fixed-point integer for the largest intervals, while a scale of 0 places the value field between bits 10 and 0 inclusive. By using a scale such that the most significant bit of the value is not 1, less than 11 bits of precision can be

signaled, as well; implementations SHOULD NOT represent more precision in an exported timestamp interval Full precision export is available down to 2^{-27} s (or 7.45 ns) with diminishing precision down to 3.64 ps. This arrangement therefore allows the representation of timestamp intervals over 13 orders of magnitude and 11 bits of precision with only two octets. The details of this encoding are illustrated in Figure 1.

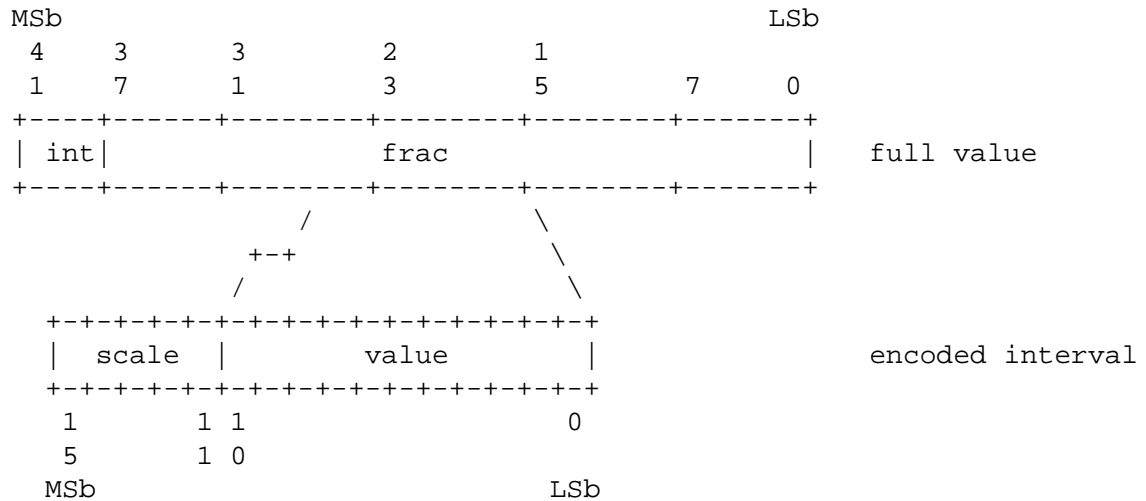


Figure 1: Timestamp interval encoding using scaled fixed-point integer

This encoded 16-bit interval is then exported for a given connection as a standalone TCP option or as part of the extended timestamp negotiation described in the following subsections.

A sender explicitly signals that it uses an irregular timestamp clock by sending 0 for both scale and value.

For implementations that support only a single timestamp interval for all flows in all situations, the encoded interval can be implemented as a constant. Encodings for common timestamp intervals are given in Table 1.

interval	frequency	scale	value	combined
16 s	0.06 Hz	0x1f	0x7ff	0xffff
1 s	1 Hz	0x1c	0x400	0xe400
0.5 s	2 Hz	0x1b	0x400	0xdc00
100 ms	10 Hz	0x18	0x666	0xc666
10 ms	100 Hz	0x15	0x51f	0xad1f
4 ms	250 Hz	0x14	0x419	0xa419
1 ms	1 kHz	0x12	0x418	0x9418
200 us	5 kHz	0x0f	0x68e	0x7e8e
50 us	20 kHz	0x0d	0x68e	0x6e8e
1 us	1 MHz	0x08	0x432	0x4432
60 ns	16.7 MHz	0x04	0x407	0x2407
none	-----	0x00	0x000	0x0000

Table 1: Encodings for common timestamp intervals at maximum precision

3.3. Timestamp Interval experimental TCP option

This section specifies an experimental TCP option, using arbitrarily chosen magic numbers as described in [I-D.ietf-tcpm-experimental-options], for exporting timestamp intervals. This option MAY appear in any TCP segment after the SYN segment to advertise the sender's timestamp interval, encoded as in Section 3.2 above. If the receiver uses timestamp interval information, it stores the interval for the duration of the connection, or until a subsequent Timestamp Interval option is received.

If a sender has previously sent a timestamp interval to a receiver, and changes the timestamp interval on the connection, it MUST send a new Timestamp Interval option.

This option MUST NOT appear in a segment in which a TCP Timestamp option is also not present.

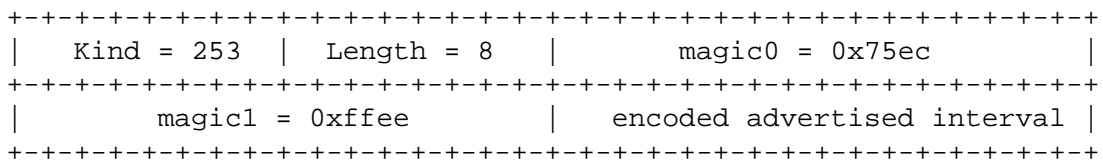


Figure 2: Structure of Timestamp Interval Experimental TCP option for interval export

[FIXME: specify how long after an advertisement of a new or changed interval the interval must be valid for the connection.]

3.4. Interval export during TS negotiation

[EDITOR'S NOTE: bind to new revision of the TS negotiation draft; requires TS negotiation that can flexibly add 16 bits of content to the negotiation handshake.]

4. Timestamp interval negotiation

[EDITOR'S NOTE: describe here how a receiver could ask a sender for a specific TS rate: an option with two encoded intervals could be handled as consisting of an advertised interval (first interval) and a requested interval (second interval). A sender that gets an interval request must then send a ts interval option which advertises the closest interval it is willing to support. This mechanism could also be used to implicitly request that timestamps be turned on, if it is decided that 1323 could be updated to support mid-connection initialization of TS.]

5. IANA Considerations

This document has no considerations for IANA.

6. Security Considerations

[EDITOR'S NOTE: discuss implications of misuse -- what can I break by sending a bad interval?]

7. References

7.1. Normative References

[I-D.ietf-tcpm-experimental-options]

Touch, J., "Shared Use of Experimental TCP Options",
[draft-ietf-tcpm-experimental-options-02](#) (work in progress), October 2012.

[I-D.scheffenegger-tcpm-timestamp-negotiation]

Scheffenegger, R. and M. Kuehlewind, "Additional negotiation in the TCP Timestamp Option field during the TCP handshake",
[draft-scheffenegger-tcpm-timestamp-negotiation-04](#) (work in progress), October 2012.

progress), July 2012.

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

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

7.2. Informative References

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

[I-D.ietf-ledbat-congestion] Shalunov, S., Hazel, G., Iyengar, J., and M. Kuehlewind, "Low Extra Delay Background Transport (LEDBAT)", [draft-ietf-ledbat-congestion-10](#) (work in progress), September 2012.

[Appendix A](#). Detailed use cases for timestamp interval export

[FIXME: frontmatter]

A.1. Methodology for one-way delay variation measurement using known timestamp intervals

New congestion control algorithms are currently proposed, that react on the measured one-way delay variation (see [\[I-D.ietf-ledbat-congestion\]](#), [\[Chirp\]](#)). 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 clocks at both ends are running at roughly the same rate. Without prior knowledge of the timestamp clock interval used by the partner, a sender can try to learn this interval 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 much shorter clock intervals 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 tick duration.

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 interval (typically, the local clock interval):

$$C = \frac{(TSecr \gg \text{local mask}) - (TSval \gg \text{remote mask})}{t} * \frac{\text{remote interval}}{\text{local interval}}$$

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

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

$$EXP.min = \min(EXP.loc, EXP.rem)$$

$$EXP.rem -= EXP.min$$

$$EXP.loc -= EXP.min$$

$$FRAC.rem = (0x800 | FRAC.rem) \ll EXP.rem$$

$$FRAC.loc = (0x800 | FRAC.loc) \ll EXP.loc$$

and assuming that the local clock 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 interval 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 interval are eliminated completely.

Authors' Addresses

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

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

Mirja Kuehlewind
University of Stuttgart
Pfaffenwaldring 47
70569 Stuttgart
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