TCP Maintenance and Minor Extensions (tcpm)
Internet-Draft
Intended status: Experimental
Expires: January 16, 2014

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

July 15, 2013

Encoding of Time Intervals for the TCP Timestamp Option
draft-trammell-tcpm-timestamp-interval-01.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 compact encoding 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 January 16, 2014.

Table of Contents

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.

This document specifies a compact encoding for timestamp intervals which can be exported via any number of mechanisms, either through a new TCP option, by piggybacking on the timestamp option as in [I-D.scheffenegger-tcpm-timestamp-negotiation], or through other in- or out-of-band means. This document specifies an experimental TCP

option for experiments with interval exposure separate from any other mechanism.

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.

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 than they actually support. 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.

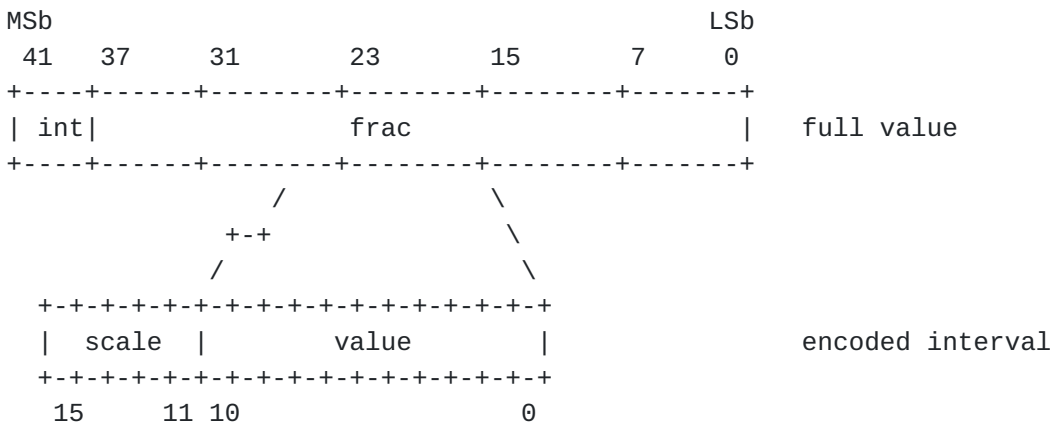

Figure 1: Timestamp interval encoding using scaled fixed-point integer

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 zero for both scale and value (i.e., 0x0000). Combinations of a value of zero and a non-zero scale are reserved for future use. These values MUST NOT be sent as a timestamp interval, and SHOULD presently be interpreted by the receiver as exposing an irregular timestamp clock.

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 with maximum precision are given in Table 1. Encodings for 9-bit precision, the maximum available from common software interrupt clock sources, are given in Table 2.

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 with maximum precision

interval	frequency	scale	value	combined
1.0 s	1 Hz	0x1e	0x100	0xf100
0.5 s	2 Hz	0x1d	0x100	0xe900
100 ms	10 Hz	0x1a	0x199	0xd199
10 ms	100 Hz	0x17	0x147	0xb947
4 ms	250 Hz	0x16	0x106	0xb106
1 ms	1 kHz	0x14	0x106	0xa106
200 us	5 kHz	0x11	0x1a3	0x89a3
50 us	20 kHz	0x0f	0x1a3	0x79a3
1 us	1 MHz	0x0a	0x10c	0x510c
60 ns	16.7 MHz	0x06	0x101	0x3101
none	--------	0x00	0x000	0x0000

Table 2: Encodings for common timestamp intervals with 9-bit precision

3.3. Timestamp Interval experimental TCP option

This section specifies an experimental TCP option, using an ExID and magic number 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. The receiver may assume the Timestamp Interval is applicable from the point of receipt of the option; i.e. that all subsequent received segments with the same or a subsequent sequence number as the segment containing the option export timestamps with the stated option.

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.

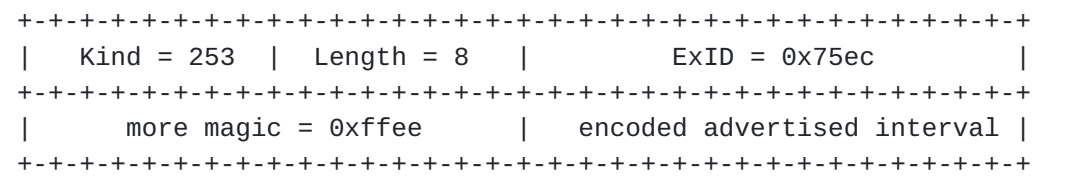

Figure 2: Structure of Timestamp Interval Experimental TCP option for interval export

Should timestamp interval exposure prove useful, and a separate TCP Option be chosen as the preferred method to send it in-band, this option would have a length of 4, and the use of an ExID and magic number would preserve word alignment in implementations transitioning from experimental to production TCP Option usage.

4. Guidelines for defined-interval timestamp export

As noted above, implementations SHOULD NOT indicate more precision than they support. As common software interrupt clock sources provide about 9 bits of precision, these should be indicated with 2 leading zero bits in the value field. Low variance software clocks (e.g. CPU cycle counters) should be indicated with a single leading zero bit, and hardware injecting the timestamp into the header with high precision should use the full precision. Similarly, if the clock source exhibits a very high variability (e.g. when running in a virtualized environment), 3 or more leading zeros should be used in the value field.

Timestamp intervals faster than about 1 ms SHOULD be implemented by inserting the timestamp "late" before transmitting a segment to avoid unnecessary timing jitter.

Intervals on the order of 1us or less are intended for use with for hardware-assisted implementations, e.g. direct use of a (shifted) CPU cycle counter as clock source.

5. IANA Considerations

This document uses the Experimental Option Experiment Identifier (ExID) 0x75ec ffee to identify the Timestamp Interval experimental option in Section 3.3; an application for this codepoint in the IANA TCP Experimental Option ExID registry has already been submitted.

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-06 (work in progress), June 2013.

[I-D.scheffenegger-tcpm-timestamp-negotiation]
Scheffenegger, R., Kuehlewind, M., and B. Trammell, "Additional negotiation in the TCP Timestamp Option field during the TCP handshake", draft-scheffenegger-tcpm-timestamp-negotiation-05 (work in progress), October 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. 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.

[EDITOR'S NOTE: the following refers to a mask field which doesn't exist anymore, needs a rewrite. Shouldn't we define C(t) = (TSecr(t) - TSval(t)) * (TSinterval(remote) / TSinterval(local))?]

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):

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

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

[EDITOR'S NOTE: the following refers to field definitions from the old TS nego draft; needs a rewrite.]

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 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.

Though specified for endpoint usage for congestion control, the difference betwen interarrival and interdeparture time used by this algorithm is applicable for passive measurement of jitter, as well.

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