TCP Maintenance Working Group
Internet-Draft
Intended status: Standards Track
Expires: 18 April 2026

K. Yang
N. Cardwell
Y. Cheng
E. Dumazet
Google, Inc

15 October 2025

TCP ETS: Extensible Timestamp Options
draft-yang-tcpm-ets-01

Abstract

This document presents ETS: an Extensible TimeStamps option for TCP. It allows hosts to use microseconds as the unit for timestamps to improve the precision of timestamps, and extends the information provided in the [RFC7323] TCP Timestamps Option by including the receiver delay in the TSecr echoing, so that the receiver of the ACK is able to more accurately estimate the portion of the RTT that resulted from time traveling through the network. The ETS option format is extensible, so that future extensions can add further information without the overhead of extra TCP option kind and length fields.

Table of Contents

1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here. In this document, these words will appear with that interpretation only when in UPPER CASE. Lower case uses of these words are not to be interpreted as carrying [RFC2119] significance.

2. Introduction

Accurate round-trip time (RTT) estimation is necessary for TCP to adapt to diverse and dynamic traffic conditions.

The TCP timestamp option specified in [RFC7323] is designed largely for RTT samples intended for computing TCP's retransmission (RTO) timer [RFC6298].

Some congestion control algorithms may wish to use a form of RTT measurement as one of several congestion signals, since elevated RTT measurements can reflect increases in network queueing delays. For example, the Swift congestion control algorithm [KDJWWM20], successfully deployed in data-center environments, requires precise and accurate measurements of both network and host delays. However, the existing TCP RTT sampling mechanisms that measure the delay between data transmission and ACK receipt [RFC6298] do not separate network and host delays, and cannot measure the RTT of retransmitted data. Even the TCP timestamp option specified in [RFC7323] is not well-suited to use as a congestion signal, for a number of reasons.

With the TCP Timestamps Option [RFC7323], data senders can measure an RTT sample by computing the difference between the data sender's current timestamp clock value and the received TSecr value. However, there are some drawbacks in this [RFC7323] measurement method:

1. The [RFC7323] RTT measurement can be greatly inflated by delayed ACKs: the host receiving a data segment can delay an ACK segment in hopes of piggybacking an ACK on the next outgoing data segment. This delay can be as large as 500 milliseconds (as in [RFC1122] Section 4.2.3.2).

2. There is delay included in the RTT measurement that is irrelevant to network queuing, e.g. host-side transmit and receive delays, including delays for waking CPUs from power management "C-states".

3. [RFC7323] specifies a "timestamp clock frequency in the range 1 ms to 1 sec per tick" ([RFC7323] Section 5.4). But today's datacenter networks are capable of delivering network packets within tens of microseconds [BMPR17]. In such networks the lower bound of 1 ms limits the usefulness of [RFC7323] timestamps. First, [RFC7323] is incapable of accurate RTT measurements in such networks. Second, [RFC7323] timestamps are not suitable for reverting spurious loss recovery and congestion control responses [RFC4015] due to packet reordering in such networks.

In many of the cases above, an RTT sample computed using [RFC7323] can be inflated for reasons other than network queuing. It is difficult for the ACK receiver to infer how long the non-network delay was, which makes it hard to use an [RFC7323] RTT measurement as a clean signal for congestion control.

Delayed ACKs, as mentioned above, are particularly problematic. TCP receivers typically implement a delayed ACK algorithm. To avoid spurious timeouts due to these delayed ACKs, TCP senders can adapt to this delayed ACK behavior by guessing the maximum delayed ACK value of the remote receiver. Historically, many implementations tended to delay ACKs by up to roughly 200ms [WS95], so some implementations have correspondingly used a minimum RTO of 200ms. However, this imposes a latency penalty that is very large compared to RTTs in some of today's datacenter networks.

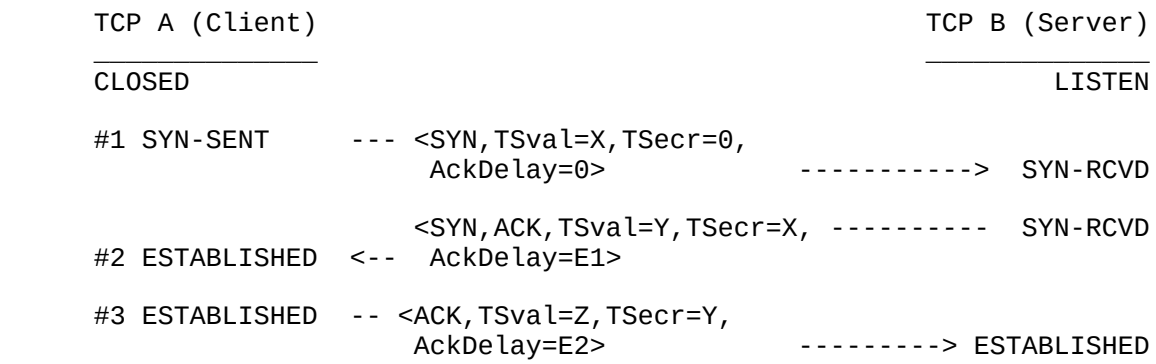
This document presents ETS: an Extensible TimeStamps option for TCP. ETS extends the information provided in the [RFC7323] TCP Timestamps Option, adding several features. First, ETS allows connections to use microseconds as the unit for timestamps, to improve the precision of timestamps. Second, ETS allows connections to include information about the delay between data receipt and ACK generation, so that the receiver of the ACK is able to more accurately estimate the portion of the RTT that resulted from time that data and ACK segments spent traveling through the network. Third, the ETS option format is extensible, so that future extensions can add further information without the overhead of extra TCP option kind and length fields.

2.1. High-level Design

The ETS protocol has two phases: an exchange of ETS options (ETSopt) in the negotiation handshake in <SYN> and <SYN,ACK> segments, and then ETS options included in all following segments.

All segments include AckDelay, the delay in the TSecr echoing process that was inserted by the data receiver, helping the receiver of an ACK to estimate the portion of the RTT delay caused by the network.

An example of a handshake exchange is illustrated below:



Active connect: An actively connecting host that wishes to negotiate ETSopt MUST include the ETSopt in the <SYN>. For backward compatibility, the endpoint performing the active connect MAY also include a [RFC7323] TSopt in the <SYN> segment, so that if the passive side or middleboxes do not support and respond to the ETSopt, the active and passive sides can proceed with the [RFC7323] TSopt negotiation for the connection.

Passive connect: For a passively connecting host that is willing to proceed with ETSopt negotiation, if the <SYN> includes an ETSopt, the host MUST include a TCP ETS option in the initial <SYN,ACK> segment. A retransmission of the <SYN,ACK> segment may omit the ETSopt, to increase robustness in the presence of middleboxes that block segments containing ETSopt.

Processing of <SYN,ACK> for active connect: If the ETSopt is absent from the <SYN,ACK> segment received by the actively connecting endpoint, suggesting that the passive endpoint does not support ETSopt, or some middlebox has stripped the option from the <SYN,ACK> segment, then the actively connecting endpoint MUST disable ETSopt for this connection. In such cases the actively connecting endpoint MAY fall back to using [RFC7323] timestamps if both the <SYN> and <SYN,ACK> segments include valid [RFC7323] timestamps.

3. Detailed Protocol

3.1. Definitions

The reader is expected to be familiar with the TCP Timestamps Option (TSopt), including TSval, TSecr, and TS.Recent [RFC7323].

Variables introduced by this document are described below:

TSval: Same as the TSval field described in [RFC7323] except the unit is in microseconds.

TSecr: The echo of TS.recent

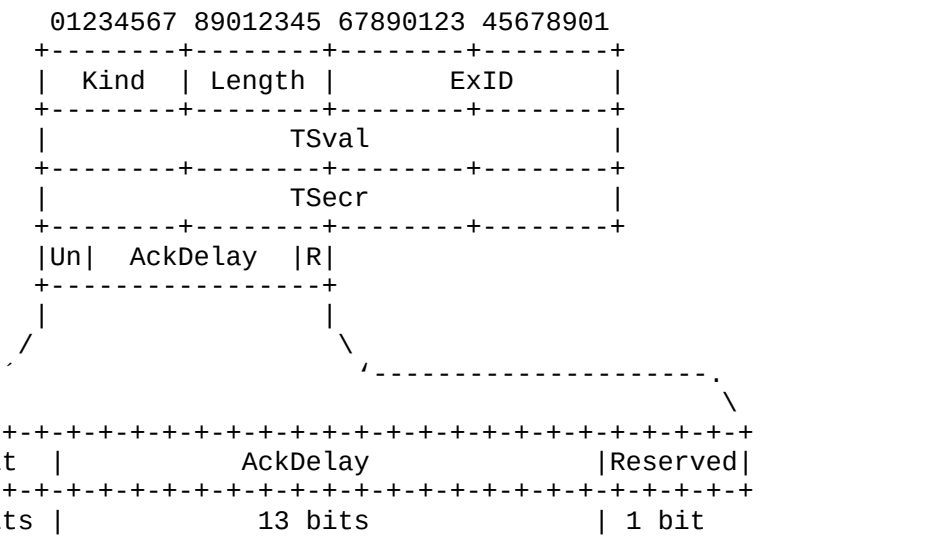
TS.Recent: The recently received TSval sent by the remote TCP endpoint in the TSval field of an ETSopt, updated using the TS.Recent rules specified in [RFC7323].

AckDelay: The field quantifying the delay between data receipt and ACK generation in the TSecr echoing process, so that the receiver of the ACK is able to more accurately estimate the NetworkRTT.

NetworkRTT: The time from when the data segment leaves the sender until when it arrives at the receiver, plus the time from when the corresponding ACK leaves the (data) receiver until the ACK arrives at the data sender (here sender and receiver refer to the TCP layer only).

3.2. TCP ETS option header format

The header format for TCP ETS options (ETSopt) is as follows:



Kind: 1 byte, has value 254,
TCP experimental option codepoint [RFC6994]
Length: 1 byte option length, value is 14
(value MAY be higher in later ETSOpt protocol versions).
ExID: 2 byte [RFC6994] experiment ID; MUST be 0x4554.
TSval and TSecr: 32 bits each, have the same definition as
[RFC7323] except that both are in microseconds.
AckDelay.Unit: 2 bits, has value:
0: indicates AckDelay is in microsecond units
1: indicates AckDelay is in millisecond units
2: indicates AckDelay is invalid
3: reserved
AckDelay: 13 bits, the value of AckDelay.
Reserved: 1 bit, in this protocol version; sender MUST set to 0;
receiver MUST ignore field and tolerate 0 or 1

The semantics of the option fields are as follows:

TSval: Same as the TSval field described in [RFC7323] except the unit is in microseconds, contains the value of the sender host's timestamp clock, in, at the time the sender schedules transmission of the segment.

TSecr: The TSecr field contains the current value of TS.Recent, the recently received TSval sent by the remote TCP that is recorded using the TS.recent update algorithm described in [RFC7323], section 4.3. TSecr is only valid when the ACK bit is set. When the ACK bit is not set, senders MUST set this field to 0, and receivers MUST ignore the value in this field (and MUST tolerate any value in this field).

AckDelay: Field AckDelay contains the delay inserted by the receiver in the TSecr echoing process. Field AckDelay is only valid when the ACK bit is set, otherwise MUST be set to 0 by the sender and ignored by the receiver. When the ACK bit is set, the sender computes the AckDelay using the following algorithm:

(1) When TS.Recent is updated by a received segment SEG (as in [RFC7323]):
TS.RecentClock = SEG.ArrivalTime
(2) When an ETSopt is sent in a segment ACK:
TSecr = TS.Recent (as in [RFC7323])
AckDelay = ACK.SendTime - TS.RecentClock

In (1), it is RECOMMENDED that the Network Interface Controller (NIC) receiving timestamp of the segment SEG be used as the ArrivalTime of SEG. This practice aids endpoints in more accurately estimating NetworkRTT by excluding delays unrelated to network queuing, such as host-side receive delays, including those incurred by CPU wake-up from power management "C-states".

We discuss how the ACK receiver can estimate the NetworkRTT using AckDelay in the next section.

3.3. Estimating the Network RTT

When an endpoint receives an ACK with TSecr=X, we define NetworkRTT as the time from when the data segment, which has TSval=X, is sent until when it arrives at the receiver, plus the time from when the corresponding ACK is sent by the (data) receiver until the ACK arrives at the data sender. It should be noted that despite an ACK may acknowledge multiple segments, only those packets involved in the TSval echoing process are related to NetworkRTT calculation.

With AckDelay in ETSopt, when a data sender (TCP A) receives an ACK segment with ETSopt from the remote endpoint (TCP B), NetworkRTT can be estimated by:

```
  NetworkRTT = ACK.ArrivalTime - ACK.TSecr - ACK.AckDelay
```

For better accuracy, it is also RECOMMENDED that the NIC receiving timestamp of the segment (ACK) be used as the ArrivalTime.

The following example shows how NetworkRTT is computed:

```
   TCP A                                                    TCP B
   _____________                                            _____________
                  -- <TSval=1> -->                          arrives at t=2
                                                          TS.Recent=TSval=1
                                                           TS.RecentClock=2
                  -- <TSval=2> --> lost
                  -- <TSval=3> -->                          arrives at t=10
 arrive at t=11   <-- <ACK, TSecr=1, AckDelay=8> -- Send ACK at t=10
```

In this example, AckDelay in the last ACK segment is ACK.SendTime - TS.RecentClock = 10 - 2 = 8, and the NetworkRTT from the last ACK segment is computed as 11 - 1 - 8 = 2, which is the network RTT of the segment sent with TSval = 1.

In order to make use of this NetworkRTT estimate as a clean signal that more precisely reflects network queuing, it is RECOMMENDED that timestamps ACK.ArrivalTime and TS.RecentClock use the time at which the segment arrives at the host, e.g. the time the NIC receives the segment, if the NIC supports hardware receive timestamping. Within this context, NetworkRTT is then defined as the time from when the data segment leaves the sender's TCP until when it is received at the receiver's NIC, plus the time from when the corresponding ACK leaves the (data) receiver's TCP until the ACK is received at the data sender's NIC.

4. Interaction with other TCP Mechanisms

4.1. Interaction with PAWS

Protection Against Wrapped Sequences (PAWS), introduced by [RFC7323] Section 5, is a mechanism to reject old duplicate segments that might corrupt an open TCP connection. In the PAWS mechanism, a segment can be discarded as an old duplicate if it is received with a timestamp SEG.TSval that is "before" some timestamps recently received on this connection.

As in [RFC7323], ETS receivers need to exercise care to avoid spurious PAWS discards due to wrapping 32-bit timestamp values during periods in which the connection is idle. When microsecond units are used, as in ETS, the 32-bit timestamp could trigger wrapping issues and spurious PAWS discards after 2^31 ticks of idleness, which is

around 2147 seconds (or around 35.7 minutes). To prevent a false positive PAWS rejection of a valid segment, an ETS receiver MUST skip the PAWS check for the first arriving segment after the timestamp used by PAWS, e.g. TS.Recent, has not been updated for 2147 seconds or more.

4.2. Eifel Considerations

The Eifel Detection Algorithm [RFC3522] detects a spurious recovery by comparing a received TSecr to RetransmitTS, the value of the TSval in the retransmit sent when loss recovery is initiated. ETS allows Eifel to work as-is because the fields TSval and TSecr in the ETSopt have the same semantics as in TSopt [RFC7323]. Further in sub-millisecond environments, ETS microsecond precision is more effective at detecting spurious retransmission compared to TSopt's more coarse unit.

4.3. RTO Calculation Considerations

The RTT measurement used in the calculation of RTO (retransmission timeout) [RFC6298] stays the same as described in [RFC7323]. It is NOT RECOMMENDED to use only NetworkRTT measurements for RTO calculation because RTO needs to include the host side delays to avoid spurious RTO events due to host delays.

4.4. SACK Considerations

The ETSopt has a length of 14 bytes. This leaves a remaining space of 26 bytes for other TCP options. A SACK option [RFC2018] with at most 3 SACK blocks is able to coexist with ETSopt in a single TCP segment as with TSopt.

4.5. Interaction with Middleboxes

[HNRGHT11] shows that middleboxes could drop an unrecognized TCP option or even drop the whole segment.

In order to fall back on [RFC7323] TSopt, the sender MAY include a [RFC7323] TSopt in the <SYN> and <SYN,ACK> segments, so that the [RFC7323] TSopt can be adopted when the ETSopt is stripped by a middlebox.

Once an expected ETSopt is missing from an incoming segment, the sender MUST NOT include an ETSopt for all future segments of this TCP connection. An implementation could negatively cache such incidents to avoid using ETS on these hosts or routes on future connections.

Another consideration is the interaction with hardware offloads like Receive Segment Coalescing (RSC). RSC should remain compatible with ETSopt, provided that RSC coalesces segments with bitwise identical TCP headers. Some NICs require to parse the TCP options and impose additional conditions on them. In such cases, new TCP options, like ETSopt, may be unrecognizable by the NIC, and thereby disables RSC.

5. IANA Considerations

This document specifies a new TCP option that uses the shared experimental options format [RFC6994], with ExID in network-standard byte order.

The authors plan to request the allocation of ExID value 0x4554 for the TCP option specified in this document.

6. Security Considerations

A malicious receiver can manipulate the sender's network RTT estimated by forging an AckDelay value. However this does not introduce a new vulnerability relative to the Timestamp option [RFC7323], because a malicious receiver could already forge the TSecr field to manipulate the RTT measured by the other side.

7. References

7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997, <http://www.rfc-editor.org/rfc/rfc2119.txt>.

[RFC6994] Touch, J., "Shared Use of Experimental TCP Options", August 2013.

7.2. Informative References

[WS95] Wright, G. and W. Stevens, "TCP/IP Illustrated, Volume 2: The Implementation", 1995.

[BMPR17] Barroso, L., Marty, M., Patterson, D., and P. Ranganathan, "Attack of the Killer Microseconds", Communications of the ACM , April 2017.

[KDJWWM20] Kumar, G., Dukkipati, N., Jang, K., Wassel, HM., Wu, X., Montazeri, B., Wang, Y., Springborn, K., Alfeld, C., Ryan, M., and D. Wetherall, "Swift: Delay is Simple and Effective for Congestion Control in the Datacenter",

InProceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication 2020 Jul 30 (pp514-528) , 2020.

[HNRGHT11] Honda, M., Nishida, Y., Raiciu, C., Greenhalgh, A., Handley, M., and H. Tokuda, "Is it still possible to extend TCP?", InProceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference 2011 Nov 2 (pp.181-194) , 2011.

[RFC1122] Braden, R., "Requirements for Internet hosts-communication layers", October 1989.

[RFC7323] Borman, D., Braden, B., Jacobson, V., and R. Scheffenegger, "TCP Extensions for High Performance", September 2014.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP’s Retransmission Timer", June 2011.

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

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

Authors’ Addresses

Kevin (Yudong) Yang
Google, Inc
Email: yyd@google.com

Neal Cardwell
Google, Inc
Email: ncardwell@google.com

Yuchung Cheng
Google, Inc
Email: ycheng@google.com

Eric Dumazet
Google, Inc
Email: edumazet@google.com