Network Working Group
Internet-Draft
Intended status: Experimental
Expires: 1 March 2023

C. Huitema
Private Octopus Inc.

28 August 2022

Quic Timestamps For Measuring One-Way Delays
draft-huitema-quic-ts-08

Abstract

The TIMESTAMP frame can be added to Quic packets when one way delay measurements are useful. The timestamp is set to the number of microseconds from the beginning of the node's epoch to the time at which the packet is sent. The draft defines the "enable_timestamp" transport parameter for negotiating the use of this extension frame, and the TIMESTAMP frame.

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 https://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 1 March 2023.

Copyright Notice

Table of Contents

1. Introduction

The QUIC Transport Protocol [QUIC-TRANSPORT] provides a secure, multiplexed connection for transmitting reliable streams of application data. The algorithms for QUIC Loss Detection and Congestion Control [QUIC-RECOVERY] use measurement of Round Trip Time (RTT) to determine when packets should be retransmitted. RTT measurements are useful, but there are however many cases in which more precise One-Way Delay (1WD) measurements enable more efficient Loss Detection and Congestion Control.

An example would be the Low Extra Delay Background Transport (LEDBAT) [RFC6817] which uses variations in transmission delay to detect competition for transmission resource. Experience shows that while LEDBAT may be implemented using RTT measurements, it is somewhat inefficient because it will cause unnecessary slowdowns in case of queues or delayed ACKs on the return path. Using 1WD solves these issues. Similar argument can be made for most delay-based

algorithms.

We propose to enable one way delay measurements in QUIC by defining a TIMESTAMP frame carrying the time at which a packet is sent. The use of this extension frame is negotiated with a transport parameter,

"enable_timestamp". When the extension is negotiated by both parties, this frame can be used in conjunction with other such as ACK to measure one way delays.

1.1. Terms and Definitions

The keywords "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.

2. Specification

The enable_timestamp transport parameter used for negotiating the use of the extension frame is defined in Section 2.1. The timestamp frame format is defined in Section 2.3.

2.1. Negotiation

The use of the timestamp frame extension is negotiated using a transport parameter:

* enable_timestamp (TBD)

The enable timestamp transport parameter is included if the endpoint wants to receive or accepts to send timestamp frames for this connection. This parameter is encoded as a variable integer as specified in section 16 of [QUIC-TRANSPORT]. It can take one of the following three values:

1. I would like to receive TIMESTAMP frames

2. I am able to generate TIMESTAMP frames

3. I am able to generate TIMESTAMP frames and I would like to

receive them

Peers receiving another value SHOULD terminate the connection with a TRANSPORT PARAMETER error.

A peer that advertises its capability of sending TIMESTAMP frames using option values 2 or 3 MUST NOT send these frames if the other peer does not announce advertise its desire to receive them by sending the enable_timestamp TP with option 1 or 3. This condition is described as "successful sending negotiation" in Section 2.2.

Peers that receive TIMESTAMP frames when they have not advertised their desire to receive them MAY terminate the connection with a PROTOCOL VIOLATION error.

2.1.1. Zero RTT and Timestamp Option

Implementations MUST NOT remember the value of the enable_timestamp parameter and try to use it when attempting 0-RTT on subsequent connections. This rules is in line with the suggestions in section 7.4.2 of [QUIC-TRANSPORT] to adopt conservative defaults and avoid compatibility issues. It is also consistent with the specification to only use TIMESTAMP frames in 1RTT packets, see Section 2.2.

2.2. Sending TIMESTAMP frames

Following successful sending negotiation, a peer SHOULD add a timestamp frame to 1RTT packets carrying an ACK frame. This specification does not impose a placement of TIMESTAMP frames in the packet. They MAY be sent either before or after the ACK frame.

Implementations SHOULD NOT send more than one TIMESTAMP frame per packet, but they MAY send more than one in rare circumstances. When multiple TIMESTAMP frames are present in a packet, the receiver retains the frame indicating the largest timestamp.

Implementations MUST NOT send the TIMESTAMP frame in Initial, 0-RTT or Handshake packets, because there is a risk that the peer will receive such packets before the negotiation completes. This restriction may appear excessive because some Handshake packets are

typically sent after the negotiation completes, but restricting TIMESTAMP frames to 1RTT packets is simpler and less error prone than allowing the TIMESTAMP frame in just a fraction of Handshake packets.

2.3. TIMESTAMP frame format

TIMESTAMP frames are identified by the frame type:

* TIMESTAMP (TBD)

TIMESTAMP frames carry a single parameter, the timestamp.

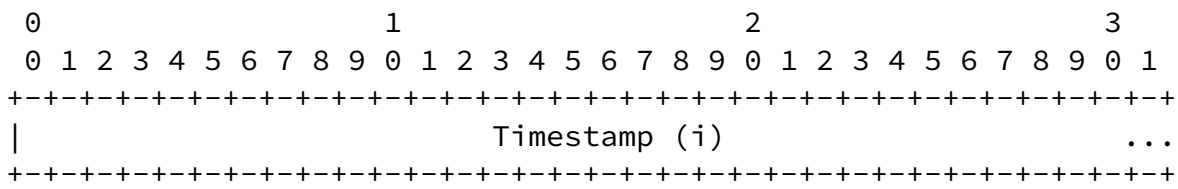

Figure 1: TIMESTAMP Frame Format with Timestamp

The timestamp encodes the number of microseconds since the beginning of the epoch, as measured by the peer at the time at which the packet is sent. It is encoded using the exponent selected by the peer in the ack_delay_exponent. The exponent reduced timestamp is encoded as a variable length integer.

TIMESTAMP frames are not ack-eliciting. Their loss does not require retransmission.

For congestion control, TIMESTAMP frames are treated like ACK frames. Section 7 of [QUIC-RECOVERY] specifies that "packets containing only ACK frames do not count towards bytes in flight and are not congestion controlled". The same applies to packets containing only TIMESTAMP frames, or a combination of ACK frames and TIMESTAMP frames.

2.4. RTT Measurements

RTT measurements are performed as specified in Section 4 of [QUIC-RECOVERY], without reference to the Timestamp parameter of the Timestamped ACK frames.

2.5. Choice of Epoch

Each peer can chose its epoch as it sees fit, but it MUST remain constant for the duration of the connection, and the resulting timestamps MUST be positive integers. Plausible values for the epoch could be:

* the beginning of the connection, i.e., the time at which the first packet for that connection was sent or received.

* the time at which the first timestamp is sent.

Chosing values close to the beginning of the connection ensures that the timestamps value will be at most equal to the duration of the connection, which limits the amount of bytes required to encode the timestamps.

2.6. One-Way Delay Measurements

An endpoint generates a One Way Delay Sample on receiving a packet containing both a TIMESTAMP frame and an ACK frame that meets the following two conditions:

* the largest acknowledged packet number is newly acknowledged, and

* at least one of the newly acknowledged packets was ack-eliciting.

The One Way Delay sample, latest_1wd, is generated as the time elapsed since the largest acknowledged packet was sent, corrected for the difference between local time at the sending peer and connection time at the receiving peer, phase_shift.

latest_1wd = timestamp - send_time_of_largest_acked - phase_shift

By convention, the phase_shift is estimated upon reception of the first RTT sample, first_rtt. It is set to:

phase_shift = timestamp - send_time_of_largest_acked - latest_rtt/2

In that formula, we assume that the local time are measured in microseconds since the beginning of the connection. The formula does not depend on the choice of epoch by each peer, but simply of the

hypothesis that delays on the data path and the return path are about equal.

We understand that clocks may drift over time, and that simply estimating a phase shift at the beginning of a connection may be too simplistic for long duration connections. Implementations MAY adopt different strategies to reestimate the phase shift at appropriate intervals. Specifying these strategies is beyond the scope of this document.

3. Discussion

This document replaces an earlier proposal to modify the format of the ACK frame by including a timestamp inside the modified frame. The revised proposal encodes the timestamp independently of the ACK frame, which requires slightly more overhead to encode the type of the TIMESTAMP frame.

Defining an independent frame allows for more flexibility. This draft defines the combination of TIMESTAMP with ACK frames, but they could be combined with other frames as well. For example, adding a TIMESTAMP to packets carrying a Path Response could allow measuring one way delays before deciding to migrate to a new path.

3.1. Management of Time

There are two known issues with deducing one way delays from RTT measurements: clock drift and undefined phase difference.

The phase difference problem is easy to understand. We start from a list of measurements associating the send time of packet number x (s[x]), the receive time of the acknowledgement of packet (a[x]), and the timestamp indicating when packet x was received by the peer (p[x]). The peer's timestamp are expressed in the peer's clock.

Suppose that we model the peer's clock as local time plus phase difference f, and that we model the rtt as the sum of two one way

delays, up (u[x]) and down (d[x]). We get:

u[x] = p[x] + f - s[x]

d[x] = a[x] - p[x] - f

Just looking at the equation shows that the value of f cannot be determined from the a series of measurement (s[x], a[x], p[x]). You can just add constraints that all u[x] and d[x] are positive numbers, which gives a range of plausible values for f: max(s[x] - p[x]) < f < min(a[x]-p[x]). In case you wonder, you get similar formulations in a multipath scenario. The plausible range may narrow to the min rtt of the shortest path, but no further.

The phase difference uncertainty is not a big issue in practice, because control algorithms are much more interested in the variations of the delays than by their absolute values. Suppose we want to compare one way delays at measurement (x) and (y). We get:

u[x] = p[x] + f - s[x]

u[y] = p[y] + f - s[y]

u[x] - u[y] = p[x] - p[y] - s[x] + s[y]

The phase difference does not affect the measurement of variations in the one way delay.

The clock drift is another matter. All the equations above assume that the local clock and the remote clock have the same frequency. This is an approximation. Clocks drift over time. Instead of just considering a stable phase difference, one should consider the sum of a phase difference and a time-varying drift component. Estimating drift is a complex problem. This was studied in detail in the development of the Network Time Protocol (NTP) [RFC5905]. In theory, implementations of Quic could copy the algorithms of NTP to build a model of the clocks used by the local node and the peer. That would be very complex.

Fortunately, implementations of Quic no not need to implement

something as complex as NTP. Most time based algorithms are only interested in variations of delays over a short horizon. Clock drift happens at a slow pace, maybe 1 millisecond per minute. Time base congestion control algorithms already have to cope with the potential drift of the minimum RTT due to changing network conditions. They do that by periodically restarting the measurement of the minimum RTT after some delay, typically less than a minute. A simple implementation of one way delay measurements could follow the same approach, for example resetting the phase difference every 30 seconds or so.

4. Use cases

Time stamps have been found useful in multiple environments, including avoid spurious exit from slow start with the Hybrid Slow Start algorithm [HyStart], and monitoring delays for individual paths for QUIC multipath as mentioned in Section 5 of [MULTIPATH-QUIC].

4.1. Application to Hystart

Implementations of the Cubic congestion control [RFC8312] benefit from the Hybrid Slow Start algorithm [HyStart]. Hystart works by monitoring the transmission delay during the initial Slow Start phase, exiting slow start and moving to congestion avoidance when a delay increase is noticed, before usage of a too large congestion window causes many losses. Hystart generally improves performance of Cubic, but can cause performance degradation if spurious delay measurement cause an early exit.

An example of early exit was noticed when implementing Cubic congestion control in QUIC [CH09]. Tests over a wireless connection showed significant jitter in the RTT measurements. This was fixed by exiting only after 5 measurements showed a delay increase. Further investigation showed that this jitter was mostly happening on the "upload" path. Timestamps enable measurement of the delay variations independently on each path, and thus improved exit decisions.

4.2. Application to QUIC Multipath

Time Stamps are very useful in multipath environments, as mentioned in [MULTIPATH-QUIC]. In the absence of time stamps, it is very hard to estimate the contribution of each path to the end to end delay, and the specification mandates that acknowledgements be sent on the same path over which packets were received. If time stamps are available, experiments show that performance are improved if the acknowlegments are sent on the shortest available path, because the implementation can detect packet losses or congestion events faster.

5. Security Considerations

The Timestamp value in the TIMESTAMP frame is asserted by the sender of the packet. Adversarial peers could chose values of the timestamp designed to exercise side effects in congestion control algorithms or other algorithms relying on the one-way delays. This can be mitigated by running plausibility checks on the received values. For example, each peer can maintain statistics not just on the One Way Delays, but also on the differences between One Way Delays and RTT, and detect outlier values. Peers can also compare the differences between timestamps in packets carrying acknowledgements and the differences between the sending times of corresponding packets, and detect anomalies if the delays between acknowledging packets appears shorter than the delays when sending them.

6. IANA Considerations

This document registers a new value in the QUIC Transport Parameter Registry:

Value: TBD (using value 0x7158 in early deployments)

Parameter Name: enable_timestamp

Specification: Indicates that the connection should use TimeStamped ACK frames

This document also registers a new value in the QUIC Frame Type registry:

Value: TBD (using value 757 in early deployments)

Frame Name: TIMESTAMP

Specification: Timestamp set at the time packet was sent

7. Acknowledgements

Thanks to Dmitri Tikhonov, Tal Misrahi, Watson Ladd, Martin Thomson and Ian Swett for their reviews and suggestions.

8. References

8.1. Normative References

[QUIC-RECOVERY]
Iyengar, J., Ed. and I. Swett, Ed., "QUIC Loss Detection and Congestion Control", RFC 9002, <https://www.rfc-editor.org/rfc/rfc9002>.

[QUIC-TRANSPORT]
Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, <https://www.rfc-editor.org/rfc/rfc9000>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8312] Rhee, I., Xu, L., Ha, S., Zimmermann, A., Eggert, L., and R. Scheffenegger, "CUBIC for Fast Long-Distance Networks", RFC 8312, DOI 10.17487/RFC8312, February 2018, <https://www.rfc-editor.org/info/rfc8312>.

8.2. Informative References

[CH09] Huitema, C., "Implementing Cubic congestion control in Quic", Blog entry, 2019, <https://huitema.wordpress.com/2019/11/11/implementing-cubic-congestion-control-in-quic/>.

[HyStart] Ha, S. and I. Rhee,, "Hybrid Slow Start for High-Bandwidth and Long-Distance Networks", Conference International Workshop on Protocols for Fast Long-Distance Networks, 2008, <https://pdfs.semanticscholar.org/25e9/ef3f03315782c7f1cbcd31b587857adae7d1.pdf>.

[MULTIPATH-QUIC]
Liu, Y., Ed., Ma, Y., De Coninck, Q., Ed., Bonaventure,

O., Huitema, C., and M. Kuehlewind, Ed., "Multipath Extension for QUIC", Work in Progress, Internet-Draft, draft-ietf-quic-multipath, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-multipath>.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <https://www.rfc-editor.org/info/rfc5905>.

[RFC6817] Shalunov, S., Hazel, G., Iyengar, J., and M. Kuehlewind, "Low Extra Delay Background Transport (LEDBAT)", RFC 6817, DOI 10.17487/RFC6817, December 2012, <https://www.rfc-editor.org/info/rfc6817>.

Author's Address

Christian Huitema
Private Octopus Inc.
Email: huitema@huitema.net