QUIC
Internet-Draft
Intended status: Standards Track
Expires: 25 July 2020
22 January 2020

J. Iyengar
Fastly
I. Swett
Google

Sender Control of Acknowledgement Delays in QUIC
draft-iyengar-quic-delayed-ack-00

Abstract

This document describes a QUIC extension for an endpoint to control its peer's delaying of acknowledgements.

Note to Readers

Discussion of this draft takes place on the QUIC working group mailing list (quic@ietf.org), which is archived at <https://mailarchive.ietf.org/arch/search/?email_list=quic>.

Working Group information can be found at <https://github.com/quicwg>; source code and issues list for this draft can be found at <https://github.com/quicwg/base-drafts/labels/-transport>.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document describes a QUIC extension for an endpoint to control its peer's delaying of acknowledgements.

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.

In the rest of this document, "sender" refers to a QUIC data sender (and acknowledgement receiver). Similarly, "receiver" refers to a QUIC data receiver (and acknowledgement sender).

An "acknowledgement packet" refers to a QUIC packet that contains only an ACK frame.

This document uses terms, definitions, and notational conventions described in Section 1.2 and Section 1.3 of [QUIC-TRANSPORT].

2. Motivation

A receiver acknowledges received packets, but it can delay sending these acknowledgements. The delaying of acknowledgements can impact connection throughput, loss detection and congestion controller performance at a data sender, and CPU utilization at both a data sender and a data receiver.

Reducing the frequency of acknowledgement packets can improve connection and endpoint performance in the following ways:

* Sending UDP packets can be noticeably CPU intensive on some platforms. Reducing the number of packets that only contain acknowledgements can therefore reduce the amount of CPU consumed at a data receiver. Experience shows that this cost reduction can be significant for high bandwidth connections.

* Similarly, receiving and processing UDP packets can also be CPU intensive, and reducing acknowledgement frequency reduces this cost at a data sender.

* Severely asymmetric link technologies, such as DOCSIS, LTE, and satellite links, connection throughput in the data direction becomes constrained when the reverse bandwidth is filled by acknowledgment packets. When traversing such links, reducing the number of acknowledgments allows connection throughput to scale much further.

Unfortunately, there are undesirable consequences to simply reducing the acknowledgement frequency, especially to an arbitrary fixed value, as follows:

* A sender relies on receipt of acknowledgements to determine the amount of data in flight and to detect losses, see [QUIC-RECOVERY]. Consequently, how often a receiver sends acknowledgments dictates how long it takes for losses to be detected at the sender.

* Starting a connection up quickly without inducing excess queue is important for latency reduction, for both short and long flows. The sender often needs frequent acknowledgments during this phase; see slow start and hystart.

* Congestion controllers that are purely window based and strictly adherent to packet conservation, such as the one defined in

[QUIC-RECOVERY], rely on receipt of acknowledgments to move the congestion window forward and release additional data. Such controllers suffer performance penalties when acknowledgements are not sent frequently enough. On the other hand, for long-running flows, congestion controllers that are not window-based, such as BBR, can perform well with very few acknowledgements per RTT.

* New sender startup mechanisms, such as paced chirping, will need a way for the sender to increase the frequency of acknowledgements when fine-grained feedback is required.

[QUIC-TRANSPORT] currently specifies a simple delayed acknowledgement mechanism that a receiver can use: send an acknowledgement for every other packet, and for every packet when reordering is observed. This simple mechanism does not allow a sender to signal its constraints, which in turn limits what a receiver can do to delay acknowledgements and reduce acknowledgement frequency. This extension provides a mechanism to solve this problem.

3. Negotiating Extension Use

Endpoints advertise their support of the extension described in this document by sending the following transport parameter (Section 7.2 of [QUIC-TRANSPORT]):

min_ack_delay (0xXXXX): A variable-length integer representing the minimum amount of time in microseconds by which the endpoint can delay an acknowledgement. Values of 0 and 2^24 or greater are invalid, and receipt of these values MUST be treated as a connection error of type PROTOCOL_VIOLATION.

An endpoint's min_ack_delay MUST NOT be greater than the its max_ack_delay. Endpoints that support this extension MUST treat receipt of a min_ack_delay that is greater than the received max_ack_delay as a connection error of type PROTOCOL_VIOLATION. Note that while the endpoint's max_ack_delay transport parameter is in milliseconds (Section 18.2 of [QUIC-TRANSPORT]), min_ack_delay is specified in microseconds.

This Transport Parameter is encoded as per Section 18 of [QUIC-TRANSPORT].

4. ACK-FREQUENCY Frame

Delaying acknowledgements as much as possible reduces both work done by the endpoints and network load. An endpoint's loss detection and congestion control mechanisms however need to be tolerant of this delay at the peer. An endpoint signals its tolerance to its peer using an ACK-FREQUENCY frame, shown below:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            0xXX (i)                         ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Sequence Number (i)                    ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Packet Tolerance (i)                   ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Update Max Ack Delay (i)                 ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Following the common frame format described in Section 12.4 of [QUIC-TRANSPORT], ACK-FREQUENCY frames have a type of 0xXX, and contain the following fields:

Sequence Number: A variable-length integer representing the sequence number assigned to the ACK-FREQUENCY frame by the sender to allow receivers to ignore obsolete frames, see Section 5.

Packet Tolerance: A variable-length integer representing the maximum number of ack-eliciting packets after which the receiver sends an acknowledgement. A value of 1 will result in an acknowledgement being sent for every ack-eliciting packet received. A value of 0 is invalid.

Update Max Ack Delay: A variable-length integer representing an update to the peer's "max_ack_delay" transport parameter (Section 18.2 of [QUIC-TRANSPORT]). The value of this field is in microseconds. Any value smaller than the "min_ack_delay" advertised by this endpoint is invalid.

Receipt of invalid values in an ACK-FREQUENCY frame MUST be treated as a connection error of type FRAME_ENCODING_ERROR.

ACK-FREQUENCY frames are ack-eliciting. However, their loss does not require retransmission.

An endpoint MAY send ACK-FREQUENCY frames multiple times during a connection and with different values.

An endpoint will have committed a "max_ack_delay" value to the peer, which specifies the maximum amount of time by which the endpoint will delay sending acknowledgments. When the endpoint receives an ACK-FREQUENCY frame, it MUST update this maximum time to the value proposed by the peer in the Update Max Ack Delay field.

5. Multiple ACK-FREQUENCY Frames

An endpoint can send multiple ACK-FREQUENCY frames, and each one of them can have different values. An endpoint MUST use a sequence number of 0 for the first ACK-FREQUENCY frame it constructs and sends, and a strictly increasing value thereafter.

An endpoint MUST allow reordered ACK-FREQUENCY frames to be received and processed, see Section 13.3 of [QUIC-TRANSPORT].

On the first received ACK-FREQUENCY frame in a connection, an endpoint MUST immediately record all values from the frame. The sequence number of the frame is recorded as the largest seen sequence number. The new Packet Tolerance and Update Max Ack Delay values MUST be immediately used for delaying acknowledgements; see Section 6.

On a subsequently received ACK-FREQUENCY frame, the endpoint MUST check if this frame is more recent than any previous ones, as follows:

* If the frame's sequence number is not greater than the largest one seen so far, the endpoint MUST ignore this frame.

* If the frame's sequence number is greater than the largest one seen so far, the endpoint MUST immediately replace old recorded state with values received in this frame. The endpoint MUST start using the new values immediately for delaying acknowledgements; see Section 6. The endpoint MUST also replace the recorded sequence number.

6. Sending Acknowledgments

Prior to receiving an ACK-FREQUENCY frame, endpoints send acknowledgements as specified in Section 13.2.1 of [QUIC-TRANSPORT].

On receiving an ACK-FREQUENCY frame and updating its recorded "max_ack_delay" and "Packet Tolerance" values (Section 5), the endpoint MUST send an acknowledgement when one of the following conditions are met:

* Since the last acknowledgement was sent, the number of received ack-eliciting packets is greater than or equal to the recorded "Packet Tolerance".

* Since the last acknowledgement was sent, "max_ack_delay" amount of time has passed.

Section 6.1, Section 6.2, and Section 6.3 describe exceptions to this strategy.

An endpoint is expected to bundle acknowledgements when possible. Every time an acknowledgement is sent, bundled or otherwise, all counters and timers related to delaying of acknowledgments are reset.

6.1. Response to Reordering

As specified in Section 13.3.1 of [QUIC-TRANSPORT], endpoints SHOULD send an acknowledgement immediately on receiving a reordered ack-eliciting packet, unless the peer has sent a "disable_ack_on_reordering" transport parameter, described below:

disable_ack_on_reordering (0xXXXX): This optional transport parameter is sent by an endpoint that is reordering tolerant or expects the connection to experience reordering. An endpoint that receives this transport parameter MUST NOT make the exception of sending an immediate acknowledgement when reordering is observed. This parameter is a zero-length value, and is encoded as per Section 18 of [QUIC-TRANSPORT].

6.2. Expediting Congestion Signals

As specified in Section 13.3.1 of [QUIC-TRANSPORT], an endpoint SHOULD immediately acknowledge packets marked with the ECN Congestion Experienced (CE) codepoint in the IP header. Doing so reduces the peer's response time to congestion events.

6.3. Batch Processing of Packets

For performance reasons, an endpoint can receive incoming packets from the underlying platform in a batch of multiple packets. This batch can contain enough packets to cause multiple acknowledgements to be sent.

To avoid sending multiple acknowledgements in rapid succession, an endpoint MAY process all packets in a batch before determining whether a threshold has been met and an acknowledgement is to be sent in response.

7. Computation of Probe Timeout Period

On sending an update to the peer's "max_ack_delay", an endpoint can use this new value in later computations of its Probe Timeout (PTO) period; see Section 5.2.1 of [QUIC-RECOVERY]. The endpoint MUST however wait until the ACK-FREQUENCY frame that carries this new value is acknowledged by the peer.

Until the frame is acknowledged, the endpoint MUST use the greater of the current "max_ack_delay" and the value that is in flight when computing the PTO period. Doing so avoids spurious PTOs that can be caused by an update that increases the peer's "max_ack_delay".

While it is expected that endpoints will have only one ACK-FREQUENCY frame in flight at any given time, this extension does not prohibit having more than one in flight. Generally, when using "max_ack_delay" for PTO computations, endpoints MUST use the maximum of the current value and all those in flight.

8. Security Considerations

TBD.

9. IANA Considerations

TBD.

10. Normative References

[QUIC-RECOVERY]
Iyengar, J., Ed. and I. Swett, Ed., "QUIC Loss Detection and Congestion Control", Work in Progress, Internet-Draft, draft-ietf-quic-recovery-latest, <https://tools.ietf.org/html/draft-ietf-quic-recovery-latest>.

[QUIC-TRANSPORT]
Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", Work in Progress, Internet-Draft, draft-ietf-quic-transport-latest, <https://tools.ietf.org/html/draft-ietf-quic-transport-latest>.

[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>.

Appendix A. Change Log

RFC Editor's Note: Please remove this section prior to publication of a final version of this document.

Acknowledgments

The following people directly contributed key ideas that shaped this draft: Bob Briscoe, Kazuho Oku, Marten Seemann.

Authors' Addresses

Jana Iyengar
Fastly

Email: jri.ietf@gmail.com

Ian Swett
Google

Email: ian.swett@google.com