Quic Timestamps For Measuring One-Way Delays
draft-huitema-quic-ts-07

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 7 September 2022.

Copyright Notice

Copyright (c) 2022 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 (https://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 Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.
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.
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.

Choosing 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_lwd, 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.

\[
\text{latest}_\text{lwd} = \text{timestamp} - \text{send}_\text{time}_\text{of}_\text{largest}_\text{acked} - \text{phase}_\text{shift}
\]

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

\[
\text{phase}_\text{shift} = \text{timestamp} - \text{send}_\text{time}_\text{of}_\text{largest}_\text{acked} - \text{latest}_\text{rtt}/2
\]

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:

\[
\begin{align*}
  u[x] &= p[x] + f - s[x] \\
  d[x] &= a[x] - p[x] - f 
\end{align*}
\]

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:

\[
\begin{align*}
  u[x] &= p[x] + f - s[x] \\
  u[y] &= p[y] + f - s[y] \\
\end{align*}
\]

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.

3.2. Application to QUIC Multipath

Time Stamps are very useful in multipath environments, as mentioned in Section 5 of [MULTIPATH-QUIC].

4. 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.

5. 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

6. Acknowledgements

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

7. References

7.1. Normative References

[QUIC-RECOVERY]

[QUIC-TRANSPORT]


7.2. Informative References

[MULTIPATH-QUIC]


Author’s Address

Christian Huitema
Private Octopus Inc.
Email: huitema@huitema.net
QUIC Acknowledgement Frequency
draft-ietf-quic-ack-frequency-02

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. Source code and issues list for this draft can be found at https://github.com/quicwg/ack-frequency.

Working Group information can be found at https://github.com/quicwg.

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 12 January 2023.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.
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.
As discussed in Section 9 however, there can be undesirable consequences to congestion control and loss recovery if a receiver unilaterally reduces the acknowledgment frequency. A sender’s constraints on the acknowledgment frequency need to be taken into account to maximize congestion controller and loss recovery performance.

[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 that is received out of order (Section 13.2.1 of [QUIC-TRANSPORT]). This simple mechanism does not allow a sender to signal its constraints. 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 (0xff03de1a): A variable-length integer representing the minimum amount of time in microseconds by which the endpoint can delay an acknowledgement. This limit could be based on the receiver’s clock or timer granularity.

An endpoint’s min_ack_delay MUST NOT be greater than 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 TRANSPORT_PARAMETER_ERROR. 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.

The min_ack_delay transport parameter is a unilateral indication of support for receiving ACK_FREQUENCY frames. If an endpoint sends the transport parameter, the peer is allowed to send ACK_FREQUENCY frames independent of whether it also sends the min_ack_delay transport parameter or not.

Receiving a min_ack_delay transport parameter indicates that the peer might send ACK_FREQUENCY frames in the future. Until an ACK_FREQUENCY frame is received, receiving this transport parameter does not cause the endpoint to change its acknowledgement behavior.

Endpoints MUST NOT remember the value of the min_ack_delay transport parameter they received. Consequently, ACK_FREQUENCY frames cannot be sent in 0-RTT packets, as per Section 7.4.1 of [QUIC-TRANSPORT].
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 the frequency it wants to receive ACK frames to its peer using an ACK_FREQUENCY frame, shown below:

ACK_FREQUENCY Frame {
  Type (i) = 0xaf,
  Sequence Number (i),
  Ack-Eliciting Threshold (i),
  Request Max Ack Delay (i),
  Reserved (6),
  Ignore CE (1),
  Ignore Order (1)
}

Following the common frame format described in Section 12.4 of [QUIC-TRANSPORT], ACK_FREQUENCY frames have a type of 0xaf, 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.

Ack-Eliciting Threshold: A variable-length integer representing the maximum number of ack-eliciting packets the recipient of this frame can receive without sending an acknowledgment. In other words, an acknowledgement is sent when more than this number of ack-eliciting packets have been received. Since this is a maximum value, a receiver can send an acknowledgement earlier. A value of 0 results in a receiver immediately acknowledging every ack-eliciting packet.

Request Max Ack Delay: A variable-length integer representing the value to which the endpoint requests the peer update its max_ack_delay (Section 18.2 of [QUIC-TRANSPORT]). The value of this field is in microseconds, unlike the ‘max_ack_delay’ transport parameter, which is in milliseconds. Sending a value smaller than the min_ack_delay advertised by the peer is invalid. Receipt of an invalid value MUST be treated as a connection error of type PROTOCOL_VIOLATION.
Reserved: This field has no meaning in this version of ACK_FREQUENCY. The value of this field MUST be 0x00. Receipt of any other value MUST be treated as a connection error of type FRAME_ENCODING_ERROR.

Ignore CE: A 1-bit field representing a boolean truth value. This field is set to true by an endpoint that does not wish to receive an immediate acknowledgement when the peer receives CE-marked packets (Section 7.1). 0 represents ‘false’ and 1 represents ‘true’.

Ignore Order: A 1-bit field representing a boolean truth value. This field is set to true by an endpoint that does not wish to receive an immediate acknowledgement when the peer receives a packet out of order (Section 7.1). 0 represents ‘false’ and 1 represents ‘true’.

ACK_FREQUENCY frames are ack-eliciting. However, their loss does not require retransmission if an ACK_FREQUENCY frame with a larger Sequence Number value has been sent.

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 Request 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 in all fields. 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 Ack-Eliciting Threshold and Request Max Ack Delay values MUST be immediately used for delaying acknowledgements; see Section 7.
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 7. The endpoint MUST also replace the recorded sequence number.

6. IMMEDIATE_ACK Frame

A sender can use an ACK_FREQUENCY frame to reduce the number of acknowledgements sent by a receiver, but doing so increases the chances that time-sensitive feedback is delayed as well. For example, as described in Section 9.3, delaying acknowledgements can increase the time it takes for a sender to detect packet loss. The IMMEDIATE_ACK frame helps mitigate this problem.

An IMMEDIATE_ACK frame can be useful in other situations as well. For example, it can be used with a PING frame (Section 19.2 of [QUIC-TRANSPORT]) if a sender wants an immediate RTT measurement or if a sender wants to establish receiver liveness as quickly as possible.

An endpoint SHOULD send a packet containing an ACK frame immediately upon receiving an IMMEDIATE_ACK frame. An endpoint MAY delay sending an ACK frame despite receiving an IMMEDIATE_ACK frame. For example, an endpoint might do this if a large number of received packets contain an IMMEDIATE_ACK or if the endpoint is under heavy load.

IMMEDIATE_ACK Frame {
    Type (i) = 0xac,
}

7. 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 Ack-Eliciting Threshold 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
Ack-Eliciting Threshold.

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

Section 7.1, Section 7.2, and Section 7.3 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.

The receiver of an ACK_FREQUENCY frame can continue to process
multiple available packets before determining whether to send an ACK
frame in response, as stated in Section 13.2.2 of [QUIC-TRANSPORT].

7.1. Response to Out-of-Order Packets

As specified in Section 13.2.1 of [QUIC-TRANSPORT], endpoints are
expected to send an acknowledgement immediately on receiving a
reordered ack-eliciting packet. This extension modifies this
behavior.

If the endpoint has not yet received an ACK_FREQUENCY frame, or if
the most recent frame received from the peer has an Ignore Order
value of false (0x00), the endpoint MUST immediately acknowledge any
subsequent packets that are received out of order.

If the most recent ACK_FREQUENCY frame received from the peer has an
Ignore Order value of true (0x01), the endpoint does not make this
exception. That is, the endpoint MUST NOT send an immediate
acknowledgement in response to packets received out of order, and
instead continues to use the peer’s Ack-Eliciting Threshold and
max_ack_delay thresholds for sending acknowledgements.

7.2. Expediting Congestion Signals

An endpoint SHOULD send an immediate acknowledgement when a packet
marked with the ECN Congestion Experienced (CE) codepoint in the IP
header is received and the previously received packet was not marked
CE.
Doing this maintains the peer’s response time to congestion events, while also reducing the ACK rate compared to Section 13.2.1 of [QUIC-TRANSPORT] during extreme congestion or when peers are using DCTCP [RFC8257] or other congestion controllers that mark more frequently than classic ECN [RFC3168].

If the most recent ACK_FREQUENCY frame an endpoint has received from the peer has an Ignore CE value of true (0x01), receipt of a CE marked packet SHOULD NOT cause an endpoint to send an immediate acknowledgement. The endpoint still sends an immediate acknowledgement if it would have for a non CE marked packet. If an immediate acknowledgement is not sent, the CE marks are reported in the next acknowledgement.

The Ignore-CE bit SHOULD NOT be set if the sender sets ECT(1) in its outgoing packets, such as with L4S, because it delays the congestion controller’s ability to quickly respond to congestion.

7.3. 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.

8. 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. When using max_ack_delay for PTO computations, endpoints MUST use the maximum of the current value and all those in flight.
When the number of in-flight ack-eliciting packets is larger than the ACK-Eliciting Threshold, an endpoint can expect that the peer will not need to wait for its max_ack_delay period before sending an acknowledgement. In such cases, the endpoint MAY therefore exclude the peer’s 'max_ack_delay' from its PTO calculation. When Ignore Order is enabled and loss causes the peer to not receive enough packets to trigger an immediate acknowledgement, the receiver will wait 'max_ack_delay', increasing the chances of a premature PTO. Therefore, if Ignore Order is enabled, the PTO MUST be larger than the peer’s 'max_ack_delay'.

9. Determining Acknowledgement Frequency

This section provides some guidance on a sender’s choice of acknowledgment frequency and discusses some additional considerations. Implementers can select an appropriate strategy to meet the needs of their applications and congestion controllers.

9.1. Congestion Control

A sender needs to be responsive to notifications of congestion, such as a packet loss or an ECN CE marking. Also, window-based congestion controllers that strictly adhere to packet conservation, such as the one defined in [QUIC-RECOVERY], rely on receipt of acknowledgments to send additional data into the network, and will suffer degraded performance if acknowledgments are delayed excessively.

To enable a sender to respond to potential network congestion, a sender SHOULD cause a receiver to send an acknowledgement at least once per RTT if there are unacknowledged ack-eliciting packets in flight. A sender can accomplish this by sending an IMMEDIATE_ACK frame once per round-trip time (RTT), or it can set the Ack-Eliciting Threshold and Request Max Ack Delay values to be no more than a congestion window and an estimated RTT, respectively.

9.2. Burst Mitigation

Receiving an acknowledgement can allow a sender to release new packets into the network. If a sender is designed to rely on the timing of peer acknowledgments ("ACK clock"), delaying acknowledgments can cause undesirable bursts of data into the network. A sender MUST limit such bursts. In keeping with Section 7.7 of [QUIC-RECOVERY], a sender can either employ pacing or cause a receiver to send an acknowledgement for at least each initial congestion window of received data.
9.3. Loss Detection and Timers

Acknowledgements are fundamental to reliability in QUIC. Consequently, delaying or reducing the frequency of acknowledgments can cause loss detection at the sender to be delayed.

A QUIC sender detects loss using packet thresholds on receiving an acknowledgement (Section 6.1.1 of [QUIC-RECOVERY]); delaying the acknowledgement therefore delays this method of detecting losses.

Reducing acknowledgement frequency reduces the number of RTT samples that a sender receives (Section 5 of [QUIC-RECOVERY]), making a sender’s RTT estimate less responsive to changes in the path’s RTT. As a result, any mechanisms that rely on an accurate RTT estimate, such as time-threshold loss detection (Section 6.1.2 of [QUIC-RECOVERY]) or Probe Timeout (Section 6.2 of [QUIC-RECOVERY]), will be less responsive to changes in the path’s RTT, resulting in either delayed or unnecessary packet transmissions.

To limit these consequences of reduced acknowledgement frequency, a sender SHOULD cause a receiver to send an acknowledgement at least once per RTT if there are unacknowledged ack-eliciting packets in flight. A sender can accomplish this by sending an IMMEDIATE_ACK frame once per round-trip time (RTT), or it can set the Ack-Eliciting Threshold and Request Max Ack Delay values to be no more than a congestion window and an estimated RTT, respectively.

A sender might use timers to detect loss of PMTUD probe packets. A sender SHOULD bundle an IMMEDIATE_ACK frame with any PMTUD probes to avoid triggering such timers.

9.4. Connection Migration

To avoid additional delays to connection migration confirmation when using this extension, a client can bundle an IMMEDIATE_ACK frame with the first non-probing frame (Section 9.2 of [QUIC-TRANSPORT]) it sends or it can send only an IMMEDIATE_ACK frame, which is a non-probing frame.

An endpoint’s congestion controller and RTT estimator are reset upon confirmation of migration (Section 9.4 of [QUIC-TRANSPORT]), which can impact the number of acknowledgements received after migration. An endpoint that has sent an ACK_FREQUENCY frame earlier in the connection SHOULD update and send a new ACK_FREQUENCY frame immediately upon confirmation of connection migration.
10. Security Considerations

TBD.

11. IANA Considerations

TBD.

12. References

12.1. Normative References

[QUIC-TRANSPORT]

[QUIC-RECOVERY]


12.2. Informative References


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: ianswett@gmail.com
Abstract

This document specifies a multipath extension for the QUIC protocol to enable the simultaneous usage of multiple paths for a single connection.

Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the QUIC Working Group mailing list (quic@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/quic/.

Source for this draft and an issue tracker can be found at https://github.com/mirjak/draft-lmbdhk-quic-multipath.

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 12 January 2023.

Copyright Notice

Copyright (c) 2022 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 (https://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 Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction ............................................. 3
   1.1. Conventions and Definitions ........................ 5
2. High-level overview ...................................... 5
3. Handshake Negotiation and Transport Parameter .......... 6
4. Path Setup and Removal ................................... 7
   4.1. Path Initiation .................................... 8
   4.2. Path State Management .............................. 8
   4.3. Path Close ......................................... 9
      4.3.1. Use PATH_ABANDON Frame to Close a Path ...... 9
      4.3.2. Refusing a New Path ........................... 10
      4.3.3. Effect of RETIRE_CONNECTION_ID Frame .......... 11
      4.3.4. Idle Timeout .................................. 11
   4.4. Path States ........................................ 12
5. Congestion Control ....................................... 14
6. Computing Path RTT ....................................... 14
7. Packet Scheduling ....................................... 15
8. Recovery ................................................ 16
9. Packet Number Space and Use of Connection ID .......... 16
   9.1. Using Zero-Length connection ID .................... 16
      9.1.1. Sending Acknowledgements and Handling Ranges 17
      9.1.2. Loss and Congestion Handling With Zero-Length CID 18
      9.1.3. ACK Delay and Zero-Length CID Considerations 18
      9.1.4. ECN and Zero-Length CID Considerations 19
      9.1.5. Restricted Sending to Zero-Length CID Peer 19
   9.2. Using non-zero length CID and Multiple Packet Number
      Spaces ................................................. 20
      9.2.1. Packet Protection for QUIC Multipath .......... 20
      9.2.2. Key Update for QUIC Multipath ................ 21
10. Examples ............................................... 22
    10.1. Path Establishment ............................... 22
1. Introduction

This document specifies an extension to QUIC version 1 [QUIC-TRANSPORT] to enable the simultaneous usage of multiple paths for a single connection.

This proposal is based on several basic design points:

* Re-use as much as possible mechanisms of QUIC version 1. In particular, this proposal uses path validation as specified for QUIC version 1 and aims to re-use as much as possible of QUIC’s connection migration.

* Use the same packet header formats as QUIC version 1 to avoid the risk of packets being dropped by middleboxes (which may only support QUIC version 1)

* Congestion Control must be per-path (following [QUIC-TRANSPORT]) which usually also requires per-path RTT measurements

* PMTU discovery should be performed per-path

* A path is determined by the 4-tuple of source and destination IP address as well as source and destination port. Therefore, there can be at most one active paths/connection ID per 4-tuple.

The path management specified in Section 9 of [QUIC-TRANSPORT] fulfills multiple goals: it directs a peer to switch sending through a new preferred path, and it allows the peer to release resources associated with the old path. Multipath requires several changes to that mechanism:
* Allow simultaneous transmission of non-probing frames on multiple paths.

* Continue using an existing path even if non-probing frames have been received on another path.

* Manage the removal of paths that have been abandoned.

As such, this extension specifies a departure from the specification of path management in Section 9 of [QUIC-TRANSPORT] and therefore requires negotiation between the two endpoints using a new transport parameter, as specified in Section 3.

This extension uses multiple packet number spaces. When multipath is negotiated, each destination connection ID is linked to a separate packet number space. Using multiple packet number spaces enables direct use of the loss recovery and congestion control mechanisms defined in [QUIC-RECOVERY].

Some deployments of QUIC use zero-length connection IDs. When a node selects to use zero-length connection IDs, it is not possible to use different connection IDs for distinguishing packets sent to that node over different paths. This extension also specifies a way to use zero-length CID by using the same packet number space on all paths. However, when using the same packet number space on multiple paths, out of order delivery is likely. This causes inflation of the number of acknowledgement ranges and therefore of the size of ACK frames. Senders that accept to use a single number space on multiple paths when sending to a node using zero-length CID need to take special care to minimize the impact of multipath delivery on loss detection, congestion control, and ECN handling. This proposal specifies algorithms for controlling the size of acknowledgement packets and ECN handling in Section Section 9.1 and Section 9.1.4.

This proposal does not cover address discovery and management. Addresses and the actual decision process to setup or tear down paths are assumed to be handled by the application that is using the QUIC multipath extension. Further, this proposal only specifies a simple basic packet scheduling algorithm, in order to provide some basic implementation guidance. However, more advanced algorithms as well as potential extensions to enhance signaling of the current path state are expected as future work.
1.1. Conventions and Definitions

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.

We assume that the reader is familiar with the terminology used in [QUIC-TRANSPORT]. In addition, we define the following terms:

* Path: refers to the 4-tuple {source IP address, source port number, destination IP address, destination port number}. A path refers to "network path" used in [QUIC-TRANSPORT].

* Path Identifier (Path ID): An identifier that is used to identify a path in a QUIC connection at an endpoint. Path Identifier is used in multipath control frames (etc. PATH_ABANDON frame) to identify a path. By default, it is defined as the sequence number of the destination Connection ID used for sending packets on that particular path, but alternative definitions can be used if the length of that connection ID is zero.

* Packet Number Space Identifier (PN Space ID): An identifier that is used to distinguish packet number spaces for different paths. It is used in 1-RTT packets and ACK_MP frames. Each node maintains a list of "Received Packets" for each of the CID that it provided to the peer, which is used for acknowledging packets received with that CID.

The difference between Path Identifier and Packet Number Space Identifier, is that the Path Identifier is used in multipath control frames to identify a path, and the Packet Number Space Identifier is used in 1-RTT packets and ACK_MP frames to distinguish packet number spaces for different paths. Both identifiers have the same value, which is the sequence number of the connection ID, if a non-zero connection ID is used. If the connection ID is zero length, the Packet Number Space Identifier is 0, while the Path Identifier is selected on path establishment.

2. High-level overview

The multipath extensions to QUIC proposed in this document enable the simultaneous utilization of different paths to exchange non-probing QUIC frames for a single connection. This contrasts with the base QUIC protocol [QUIC-TRANSPORT] that includes a connection migration mechanism that selects only one path to exchange such frames.
A multipath QUIC connection starts with a QUIC handshake as a regular QUIC connection. See further Section 3. The peers use the enable_multipath transport parameter during the handshake to negotiate the utilization of the multipath capabilities. The active_connection_id_limit transport parameter limits the maximum number of active paths that can be used during a connection. A multipath QUIC connection is thus an established QUIC connection where the enable_multipath transport parameter has been successfully negotiated.

To add a new path to an existing multipath QUIC connection, a client starts a path validation on the chosen path, as further described in Section 4. In this version of the document, a QUIC server does not initiate the creation of a path, but it can validate a new path created by a client. A new path can only be used once it has been validated. Each endpoint associates a Path identifier to each path. This identifier is notably used when a peer sends a PATH_ABANDON frame to indicate that it has closed the path whose identifier is contained in the PATH_ABANDON frame.

In addition to these core features, an application using Multipath QUIC will typically need additional algorithms to handle the number of active paths and how they are used to send packets. As these differ depending on the application’s requirements, their specification is out of scope of this document.

3. Handshake Negotiation and Transport Parameter

This extension defines a new transport parameter, used to negotiate the use of the multipath extension during the connection handshake, as specified in [QUIC-TRANSPORT]. The new transport parameter is defined as follows:

* name: enable_multipath (TBD - experiments use 0xbabf)
* value: 0 (default) for disabled.

Endpoints use 2-bits in the value field for negotiating one or more PN spaces, available option values are listed in Table 1:
### Table 1: Available value for enable_multipath

<table>
<thead>
<tr>
<th>Option</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>don’t support multipath</td>
</tr>
<tr>
<td>0x1</td>
<td>supports multipath as defined in this document</td>
</tr>
</tbody>
</table>

If for any one of the endpoints, the parameter is absent or set to 0, the endpoints MUST fallback to [QUIC-TRANSPORT] with single active path and MUST NOT use any frame or mechanism defined in this document.

If endpoint receives an unexpected value for the transport parameter "enable_multipath", it MUST treat this as a connection error of type MP_CONNECTION_ERROR and close the connection.

This extension does not change the definition of any transport parameter defined in Section 18.2. of [QUIC-TRANSPORT].

Inline with the definition in [QUIC-TRANSPORT] disable_active_migration also disables multipath support, except "after a client has acted on a preferred_address transport parameter" (Section 18.2. of [QUIC-TRANSPORT]).

The transport parameter "active_connection_id_limit" [QUIC-TRANSPORT] limits the number of usable Connection IDs, and also limits the number of concurrent paths. For the QUIC multipath extension this limit even applies when no connection ID is exposed in the QUIC header.

### 4. Path Setup and Removal

After completing the handshake, endpoints have agreed to enable multipath feature and can start using multiple paths. This document does not specify how an endpoint that is reachable via several addresses announces these addresses to the other endpoint. In particular, if the server uses the preferred_address transport parameter, clients SHOULD NOT assume that the initial server address and the addresses contained in this parameter can be simultaneously used for multipath. Furthermore, this document does not discuss when a client decides to initiate a new path. We delegate such discussion in separate documents.

This proposal adds one multipath control frame for path management:
* PATH_ABANDON frame for the receiver side to abandon the path (see Section 12.1)

All the new frames are sent in 1-RTT packets [QUIC-TRANSPORT].

4.1. Path Initiation

When the multipath option is negotiated, clients that want to use an additional path MUST first initiate the Address Validation procedure with PATH_CHALLENGE and PATH_RESPONSE frames described in Section 8.2 of [QUIC-TRANSPORT]. After receiving packets from the client on a new path, if the server decides to use the new path, the server MUST perform path validation (Section 8.2 of [QUIC-TRANSPORT]) unless it has previously validated that address.

If validation succeed, the client can send non-probing, 1-RTT packets on the new paths. In contrast with the specification in Section 9 of [QUIC-TRANSPORT], the server MUST NOT assume that receiving non-probing packets on a new path indicates an attempt to migrate to that path. Instead, servers SHOULD consider new paths over which non-probing packets have been received as available for transmission.

4.2. Path State Management

An endpoint uses PATH_STATUS frames to inform that the peer should send packets in the preference expressed by these frames. Notice that the endpoint might not follow the peer’s advertisements, but the PATH_STATUS frame is still a clear signal of suggestion for the preference of path usage by the peer.

PATH_STATUS frame describes 2 kinds of path states:

* Mark a path as "available", i.e., allow the peer to use its own logic to split traffic among available paths.

* Mark a path as "standby", i.e., suggest that no traffic should be sent on that path if another path is available.

Endpoints use Path Identifier field in PATH_STATUS frame to identify which path’s state is going to be changed. Notice that PATH_STATUS frame can be sent via a different path. An Endpoint MAY ignore the PATH_STATUS frame if it would make all the paths unavailable in a single connection.
4.3. Path Close

Each endpoint manages the set of paths that are available for transmission. At any time in the connection, each endpoint can decide to abandon one of these paths, following for example changes in local connectivity or changes in local preferences. After an endpoint abandons a path, the peer will not receive any more non-probing packets on that path.

An endpoint that wants to close a path SHOULD use explicit request to terminate the path by sending the PATH_ABANDON frame (see Section 4.3.1). Note that while abandoning a path will cause Connection ID retirement, only retiring the associated Connection ID does not necessarily advertise path abandon (see Section 4.3.3). However, implicit signals such as idle time or packet losses might be the only way for an endhost to detect path closure (see Section 4.3.4).

Note that other explicit closing mechanisms of [QUIC-TRANSPORT] still apply on the whole connection. In particular, the reception of either a CONNECTION_CLOSE (Section 10.2 of [QUIC-TRANSPORT]) or a Stateless Reset (Section 10.3 of [QUIC-TRANSPORT]) closes the connection.

4.3.1. Use PATH_ABANDON Frame to Close a Path

Both endpoints, namely the client and the server, can close a path, by sending PATH_ABANDON frame (see Section 12.1) which abandons the path with a corresponding Path Identifier. Once a path is marked as "abandoned", it means that the resources related to the path, such as the used connection IDs, can be released. However, information related to data delivered over that path SHOULD not be released immediately as acknowledgments can still be received or other frames that also may trigger retransmission of data on another path.

The endpoint sending the PATH_ABANDON frame SHOULD consider a path as abandoned when the packet that contained the PATH_ABANDON frame is acknowledged. When releasing resources of a path, the endpoint SHOULD send a RETIRE_CONNECTION_ID frame for the connection IDs used on the path, if any.

The receiver of a PATH_ABANDON frame SHOULD NOT release its resources immediately, but SHOULD wait for the reception of the RETIRE_CONNECTION_ID frame for the used connection IDs or 3 RTOs.

Usually, it is expected that the PATH_ABANDON frame is used by the client to indicate to the server that path conditions have changed such that the path is or will be not usable anymore, e.g. in case of
a mobility event. The PATH_ABANDON frame therefore indicates to the receiving peer that the sender does not intend to send any packets on that path anymore but also recommends to the receiver that no packets should be sent in either direction. The receiver of an PATH_ABANDON frame MAY also send an PATH_ABANDON frame to signal its own willingness not to send any packet on this path anymore.

If connection IDs are used, PATH_ABANDON frames can be sent on any path, not only the path that is intended to be closed. Thus, a path can be abandoned even if connectivity on that path is already broken. If no connection IDs are used and the PATH_ABANDON frame has to send on the path that is intended to be closed, it is possible that the packet containing the PATH_ABANDON frame or the packet containing the ACK for the PATH_ABANDON frame cannot be received anymore and the endpoint might need to rely on an idle time out to close the path, as described in Section 4.3.4.

Retransmittable frames, that have previously been sent on the abandoned path and are considered lost, SHOULD be retransmitted on a different path.

If a PATH_ABANDON frame is received for the only active path of a QUIC connection, the receiving peer SHOULD send a CONNECTION_CLOSE frame and enters the closing state. If the client received a PATH_ABANDON frame for the last open path, it MAY instead try to open a new path, if available, and only initiate connection closure if path validation fails or a CONNECTION_CLOSE frame is received from the server. Similarly the server MAY wait for a short, limited time such as one RTO if a path probing packet is received on a new path before sending the CONNECTION_CLOSE frame.

4.3.2. Refusing a New Path

An endpoint may deny the establishment of a new path initiated by its peer during the address validation procedure. According to [QUIC-TRANSPORT], the standard way to deny the establishment of a path is to not send a PATH_RESPONSE in response to the peer’s PATH_CHALLENGE. An endpoint that has negotiated the usage of the multipath extension MAY use an explicit method by sending on another active path a PATH_ABANDON frame containing the Path Identifier of the refused path, but only if the PATH_CHALLENGE arrives in a packet using a non-zero length Connection ID.
4.3.3. Effect of RETIRE_CONNECTION_ID Frame

Receiving a RETIRE_CONNECTION_ID frame causes the endpoint to discard the resources associated with that connection ID. If the connection ID was used by the peer to identify a path from the peer to this endpoint, the resources include the list of received packets used to send acknowledgements. The peer MAY decide to keep sending data using the same IP addresses and UDP ports previously associated with the connection ID, but MUST use a different connection ID when doing so.

Note that if the sender retires a Connection ID that is still used by in-flight packets, it may receive ACK_MP frames referencing the retired Connection ID. If the sender stops tracking sent packets with retired Connection ID, these would be spuriously marked as lost. To avoid such performance issue without keeping retired Connection ID state, an endpoint should first stop sending packets with the to-be-retired Connection ID, then wait for all in-flight packets to be either acknowledged or marked as lost, and finally retire the Connection ID.

4.3.4. Idle Timeout

[QUIC-TRANSPORT] allows for closing of connections if they stay idle for too long. The connection idle timeout in multipath QUIC is defined as "no packet received on any path for the duration of the idle timeout". When only one path is available, servers MUST follow the specifications in [QUIC-TRANSPORT].

When more than one path is available, hosts shall monitor the arrival of non-probing packets and the acknowledgements for the packets sent over each path. Hosts SHOULD stop sending traffic on a path if for at least the period of the idle timeout as specified in Section 10.1. of [QUIC-TRANSPORT] (a) no non-probing packet was received or (b) no non-probing packet sent over this path was acknowledged, but MAY ignore that rule if it would disqualify all available paths. To avoid idle timeout of a path, endpoints can send ack-eliciting packets such as packets containing PING frames (Section 19.2 of [QUIC-TRANSPORT]) on that path to keep it alive. Sending periodic PING frames also helps prevent middlebox timeout, as discussed in Section 10.1.2 of [QUIC-TRANSPORT].
Server MAY release the resource associated with paths for which no non-probing packet was received for a sufficiently long path-idle delay, but SHOULD only release resource for the last available path if no traffic is received for the duration of the idle timeout, as specified in Section 10.1 of [QUIC-TRANSPORT]. This means if all paths remain idle for the idle timeout, the connection is implicitly closed.

Server implementations need to select the sub-path idle timeout as a trade-off between keeping resources, such as connection IDs, in use for an excessive time or having to promptly reestablish a path after a spurious estimate of path abandonment by the client.

4.4. Path States

Figure 1 shows the states that an endpoint’s path can have.

```
+------------+                  o PATH_CHALLENGE sent/received on new path
                   v
+-----------------+      v Path validation abandoned
| Validating       +-----------------+ PATH_RESPONSE received
|                  +-----------------+ v Path blackhole detected
|                  +-----------------+          Active
|                  +-----------------+ PATH_ABANDONED sent/received
|                  +-----------------+ v Closing
|                  +-----------------+          Path’s draining timeout
|                  +-----------------+ (at least 3 PTO)
|                  +-----------------+ v Closed
```

Figure 1: States of a path

In non-final states, hosts have to track the following information.
* Associated 4-tuple: The tuple (source IP, source port, destination IP, destination port) used by the endhost to send packets over the path.

* Associated Destination Connection ID: The Connection ID used to send packets over the path.

If multiple packet number spaces are used over the connection, hosts MUST also track the following information.

* Path Packet Number Space: The endpoint maintains a separate packet number for sending and receiving packets over this path. Packet number considerations described in [QUIC-TRANSPORT] apply within the given path.

In the "Active" state, hosts MUST also track the following information.

* Associated Source Connection ID: The Connection ID used to receive packets over the path.

A path in the "Validating" state performs path validation as described in Section 8.2 of [QUIC-TRANSPORT]. An endhost should not send non-probing frames on a path in "Validating" state, as it has no guarantee that packets will actually reach the peer.

The endhost can use all the paths in the "Active" state, provided that the congestion control and flow control currently allow sending of new data on a path. Note that if a path became idle due to a timeout, endpoints SHOULD send PATH_ABANDONED frame before closing the path.

In the "Closing" state, the endhost SHOULD NOT send packets on this path anymore, as there is no guarantee that the peer can still map the packets to the connection. The endhost SHOULD wait for the acknowledgment of the PATH_ABANDONED frame before moving the path to the "Closed" state to ensure a graceful termination of the path.

When a path reaches the "Closed" state, the endhost releases all the path’s associated resources, including the associated Connection IDs. Endpoints SHOULD send RETIRE_CONNECTION_ID frames for releasing the associated Connection IDs following [QUIC-TRANSPORT]. Considering endpoints are not expected to send packets on the current path in the "Closed" state, endpoints can send RETIRE_CONNECTION_ID frames on other available paths. Consequently, the endhost is not able to send nor receive packets on this path anymore.
5. Congestion Control

Senders MUST manage per-path congestion status, and MUST NOT send more data on a given path than congestion control on that path allows. This is already a requirement of [QUIC-TRANSPORT].

When a Multipath QUIC connection uses two or more paths, there is no guarantee that these paths are fully disjoint. When two (or more paths) share the same bottleneck, using a standard congestion control scheme could result in an unfair distribution of the bandwidth with the multipath connection getting more bandwidth than competing single paths connections. Multipath TCP uses the LIA congestion control scheme specified in [RFC6356] to solve this problem. This scheme can immediately be adapted to Multipath QUIC. Other coupled congestion control schemes have been proposed for Multipath TCP such as [OLIA].

6. Computing Path RTT

Acknowledgement delays are the sum of two one-way delays, the delay on the packet sending path and the delay on the return path chosen for the acknowledgements. When different paths have different characteristics, this can cause acknowledgement delays to vary widely. Consider for example a multipath transmission using both a terrestrial path, with a latency of 50ms in each direction, and a geostationary satellite path, with a latency of 300ms in both directions. The acknowledgement delay will depend on the combination of paths used for the packet transmission and the ACK transmission, as shown in Table 2.

<table>
<thead>
<tr>
<th>ACK Path \ Data path</th>
<th>Terrestrial</th>
<th>Satellite</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terrestrial</td>
<td>100ms</td>
<td>350ms</td>
</tr>
<tr>
<td>Satellite</td>
<td>350ms</td>
<td>600ms</td>
</tr>
</tbody>
</table>

Table 2: Example of ACK delays using multiple paths
Using the default algorithm specified in [QUIC-RECOVERY] would result in suboptimal performance, computing average RTT and standard deviation from series of different delay measurements of different combined paths. At the same time, early tests showed that it is desirable to send ACKs through the shortest path because a shorter ACK delay results in a tighter control loop and better performances. The tests also showed that it is desirable to send copies of the ACKs on multiple paths, for robustness if a path experiences sudden losses.

An early implementation mitigated the delay variation issue by using time stamps, as specified in [QUIC-Timestamp]. When the timestamps are present, the implementation can estimate the transmission delay on each one-way path, and can then use these one way delays for more efficient implementations of recovery and congestion control algorithms.

If timestamps are not available, implementations could estimate one way delays using statistical techniques. For example, in the example shown in Table 1, implementations can use "same path" measurements to estimate the one way delay of the terrestrial path to about 50ms in each direction, and that of the satellite path to about 300ms. Further measurements can then be used to maintain estimates of one way delay variations, using logical similar to Kalman filters. But statistical processing is error-prone, and using time stamps provides more robust measurements.

7. Packet Scheduling

The transmission of QUIC packets on a regular QUIC connection is regulated by the arrival of data from the application and the congestion control scheme. QUIC packets can only be sent when the congestion window of at least one path is open.

Multipath QUIC implementations also need to include a packet scheduler that decides, among the paths whose congestion window is open, the path over which the next QUIC packet will be sent. Many factors can influence the definition of these algorithms and their precise definition is outside the scope of this document. Various packet schedulers have been proposed and implemented, notably for Multipath TCP. A companion draft [I-D.bonaventure-iccrg-schedulers] provides several general-purpose packet schedulers depending on the application goals.

Note that the receiver could use a different scheduling strategy to send ACK(_MP) frames. The recommended default behaviour consists in sending ACK(_MP) frames on the path they acknowledge packets. Other scheduling strategies, such as sending ACK(_MP) frames on the lowest
latency path, might be considered, but they could impact the sender
with side effects on, e.g., the RTT estimation or the congestion
control scheme. When adopting such asymetrical acknowledgment
scheduling, the receiver should at least ensure that the sender
negotiated one-way delay calculation mechanism (e.g.,
[QUIC-Timestamp]).

8. Recovery

Simultaneous use of multiple paths enables different retransmission
strategies to cope with losses such as: a) retransmitting lost frames
over the same path, b) retransmitting lost frames on a different or
dedicated path, and c) duplicate lost frames on several paths (not
recommended for general purpose use due to the network overhead).
While this document does not preclude a specific strategy, more
detailed specification is out of scope.

9. Packet Number Space and Use of Connection ID

If the connection ID is present (non-zero length) in the packet
header, the connection ID is used to identify the path. If no
connection ID is present, the 4 tuple identifies the path. The
initial path that is used during the handshake (and multipath
negotiation) has the path ID 0 and therefore all 0-RTT packets are
also tracked and processed with the path ID 0. For 1-RTT packets,
the path ID is the sequence number of the Destination Connection ID
present in the packet header, as defined in Section 5.1.1 of
[QUIC-TRANSPORT], or also 0 if the Connection ID is zero-length.

If non-zero-length Connection IDs are used, an endpoint MUST use
different Connection IDs on different paths. Still, the receiver may
observe the same Connection ID used on different 4-tuples due to,
e.g., NAT rebinding. In such case, the receiver reacts as specified
in Section 9.3 of [QUIC-TRANSPORT].

Acknowledgements of Initial and Handshake packets MUST be carried
using ACK frames, as specified in [QUIC-TRANSPORT]. The ACK frames,
as defined in [QUIC-TRANSPORT], do not carry path identifiers. If
for any reason ACK frames are received in 1-RTT packets while the
state of multipath negotiation is ambiguous, they MUST be interpreted
as acknowledging packets sent on path 0.

9.1. Using Zero-Length connection ID

If a zero-length connection ID is used, one packet number space for
all paths. That means the packet sequence numbers are allocated from
the common number space, so that, for example, packet number N could
be sent on one path and packet number N+1 on another.
In this case, ACK frames report the numbers of packets that have been received so far, regardless of the path on which they have been received. That means the sender needs to maintain an association between sent packet numbers and the path over which these packets were sent. This is necessary to implement per path congestion control, as explained in Section 9.1.2.

Further, the receiver of packets with zero-length connection IDs should implement handling of acknowledgements as defined in Section 9.1.1.

ECN handing is specified in Section 9.1.4, and mitigation of the RTT measurement is further explained in Section 9.1.3.

If a node does not want to implement this logic, it MAY instead limit its use of multiple paths as explained in Section 9.1.5.

### 9.1.1. Sending Acknowledgements and Handling Ranges

If zero-length CID and therefore also a single packet number space is used by the sender, the receiver MAY send ACK frames instead of ACK_MP frames to reduce overhead as the additional path ID field will anyway always carry the same value.

If senders decide to send packets on paths with different transmission delays, some packets will very likely be received out of order. This will cause the ACK frames to carry multiple ranges of received packets. The large number of range increases the size of ACK frames, causing transmission and processing overhead.

The size and overhead of the ACK frames can be controlled by the combination of one or several of the following:

* Not transmitting again ACK ranges that were present in an ACK frame acknowledged by the peer.

* Delay acknowledgements to allow for arrival of "hole filling" packets.

* Limit the total number of ranges sent in an ACK frame.

* Limiting the number of transmissions of a specific ACK range, on the assumption that a sufficient number of transmissions almost certainly ensures reception by the peer.
9.1.2. Loss and Congestion Handling With Zero-Length CID

When sending to a zero-length CID receiver, senders may receive acknowledgements that combine packet numbers received over multiple paths. However, even if one packet number space is used on multiple path the sender MUST maintain separate congestion control state for each path. Therefore, senders MUST be able to infer the sending path from the acknowledged packet numbers, for example by remembering which packet was sent on what path. The senders MUST use that information to perform congestion control on the relevant paths, and to correctly estimate the transmission delays on each path. (See Section 9.1.3 for specific considerations about using the ACK Delay field of ACK frames, and Section 9.1.4 for issues on using ECN marks.)

Loss detection as specified in [QUIC-RECOVERY] uses algorithms based on timers and on sequence numbers. When packets are sent over multiple paths, loss detection must be adapted to allow for different RTTs on different paths. When sending to zero-length CID receivers, packets sent on different paths may be received out of order. Therefore, senders cannot directly use the packet sequence numbers to compute the Packet Thresholds defined in Section 6.1.1 of [QUIC-RECOVERY]. Relying only on Time Thresholds produces correct results, but is somewhat suboptimal. Some implementations have been getting good results by not just remembering the path over which a packet was sent, but also maintaining an order list of packets sent on each path. That ordered list can then be used to compute acknowledgement gaps per path in Packet Threshold tests.

9.1.3. ACK Delay and Zero-Length CID Considerations

The ACK Delay field of the ACK frame is relative to the largest acknowledged packet number (see Section 13.2.5 of [QUIC-TRANSPORT]). When using paths with different transmission delays, the reported host delay will most of the time relate to the path with the shortest latency. To collect ACK delays on all the paths, hosts should rely on time stamps as described in [QUIC-Timestamp].
9.1.4. ECN and Zero-Length CID Considerations

ECN feedback in QUIC is provided based on counters in the ACK frame (see Section 19.3.2. of [QUIC-TRANSPORT]). That means if an ACK frame acknowledges multiple packets, the ECN feedback cannot be accounted to a specific packet.

There are separate counters for each packet number space. However, sending to zero-length CID receivers, the same number space is used for multiple paths. Respectively, if an ACK frames acknowledges multiple packets from different paths, the ECN feedback cannot unambiguously be assigned to a path.

If the sender marks its packets with the ECN capable flags, the network will expect standard reactions to ECN marks, such as slowing down transmission on the sending path. If zero-length CID is used, the sending path is however ambiguous. Therefore, the sender MUST treat a CE marking as a congestion signal on all sending paths that have been by a packet that was acknowledged in the ACK frame signaling the CE counter increase.

A host that is sending over multiple paths to a zero-length CID receiver MAY disable ECN marking and send all subsequent packets as Not-ECN capable.

9.1.5. Restricted Sending to Zero-Length CID Peer

Hosts that are designed to support multipath using multiple number spaces MAY adopt a conservative posture after negotiating multipath support with a peer using zero-length CID. The simplest posture is to only send data on one path at a time, while accepting packets on all acceptable paths. In that case:

* the attribution of packets to path discussed in Section 9.1.2 are easy to solve because packets are sent on a single path,

* the ACK Delays are correct,

* the vast majority of ECN marks relate to the current sending path.

Of course, the hosts will only take limited advantage from the multipath capability in these scenarios. Support for "make before break" migrations will improve, but load sharing between multiple paths will not work.
9.2. Using non-zero length CID and Multiple Packet Number Spaces

If packets contain a non-zero CID, each path has its own packet number space for transmitting 1-RTT packets and a new ACK frame format is used as specified in Section 12.3. Compared to the QUIC version 1 ACK frame, the ACK_MP frames additionally contain a Packet Number Space Identifier (PN Space ID). The PN Space ID used to distinguish packet number spaces for different paths and is simply derived from the sequence number of Destination Connection ID. Therefore, the packet number space for 1-RTT packets can be identified based on the Destination Connection ID in each packet.

As soon as the negotiation of multipath support with value 2 is completed, endpoints SHOULD use ACK_MP frames instead of ACK frames for acknowledgements of 1-RTT packets on path 0, as well as for 0-RTT packets that are acknowledged after the handshake concluded.

Following [QUIC-TRANSPORT], each endpoint uses NEW_CONNECTION_ID frames to issue usable connections IDs to reach it. Before an endpoint adds a new path by initiating path validation, it MUST check whether at least one unused Connection ID is available for each side.

If the transport parameter "active_connection_id_limit" is negotiated as N, the server provided N Connection IDs, and the client is already actively using N paths, the limit is reached. If the client wants to start a new path, it has to retire one of the established paths.

ACK_MP frame (defined in Section 12.3) can be returned via either a different path, or the same path identified by the Path Identifier, based on different strategies of sending ACK_MP frames.

Using multiple packet number spaces requires changes in the way AEAD is applied for packet protection, as explained in Section 9.2.1, and tighter constraints for key updates, as explained in Section 9.2.2.

9.2.1. Packet Protection for QUIC Multipath

Packet protection for QUIC version 1 is specified in Section 5 of [QUIC-TLS]. The general principles of packet protection are not changed for QUIC Multipath. No changes are needed for setting packet protection keys, initial secrets, header protection, use of 0-RTT keys, receiving out-of-order protected packets, receiving protected packets, or retry packet integrity. However, the use of multiple number spaces for 1-RTT packets requires changes in AEAD usage.
Section 5.3 of [QUIC-TLS] specifies AEAD usage, and in particular the use of a nonce, N, formed by combining the packet protection IV with the packet number. If multiple packet number spaces are used, the packet number alone would not guarantee the uniqueness of the nonce.

In order to guarantee the uniqueness of the nonce, the nonce N is calculated by combining the packet protection IV with the packet number and with the path identifier.

The path ID for 1-RTT packets is the sequence number of [QUIC-TRANSPORT], or zero if the Connection ID is zero-length. Section 19 of [QUIC-TRANSPORT] encodes the Connection ID Sequence Number as a variable-length integer, allowing values up to $2^{62}-1$; in this specification, a range of less than $2^{32}-1$ values MUST be used before updating the packet protection key.

To calculate the nonce, a 96 bit path-and-packet-number is composed of the 32 bit Connection ID Sequence Number in byte order, two zero bits, and the 62 bits of the reconstructed QUIC packet number in network byte order. If the IV is larger than 96 bits, the path-and-packet-number is left-padded with zeros to the size of the IV. The exclusive OR of the padded packet number and the IV forms the AEAD nonce.

For example, assuming the IV value is 6b26114b9cba2b63a9e8dd4f, the connection ID sequence number is 3, and the packet number is aead, the nonce will be set to 6b2611489cba2b63a9e873e2.

9.2.2. Key Update for QUIC Multipath

The Key Phase bit update process for QUIC version 1 is specified in Section 6 of [QUIC-TLS]. The general principles of key update are not changed in this specification. Following QUIC version 1, the Key Phase bit is used to indicate which packet protection keys are used to protect the packet. The Key Phase bit is toggled to signal each subsequent key update. Because of network delays, packets protected with the older key might arrive later than the packets protected with the new key. Therefore, the endpoint needs to retain old packet keys to allow these delayed packets to be processed and it must distinguish between the new key and the old key. In QUIC version 1, this is done using packet numbers so that the rule is made simple: Use the older key if packet number is lower than any packet number frame the current key phase.

When using multiple packet number spaces on different paths, some care is needed when initiating the Key Update process, as different paths use different packet number spaces but share a single key. When a key update is initiated on one path, packets sent to another
path needs to know when the transition is complete. Otherwise, it is possible that the other paths send packets with the old keys, but skip sending any packets in the current key phase and directly jump to sending packet in the next key phase. When that happens, as the endpoint can only retain two sets of packet protection keys with the 1-bit Key Phase bit, the other paths cannot distinguish which key should be used to decode received packets, which results in a key rotation synchronization problem.

To address such a synchronization issue, if key update is initialized on one path, the sender SHOULD send at least one packet with the new key on all active paths. Further, an endpoint MUST NOT initiate a subsequent key update until a packet with the current key has been acknowledged on each path.

Following Section 5.4 of [QUIC-TLS], the Key Phase bit is protected, so sending multiple packets with Key Phase bit flipping at the same time should not cause linkability issue.

10. Examples

10.1. Path Establishment

Figure 2 illustrates an example of new path establishment using multiple packet number spaces.

Client                                                  Server

(Exchanges start on default path)
1-RTT[]: NEW_CONNECTION_ID[C1, Seq=1] -->
    <- 1-RTT[]: NEW_CONNECTION_ID[S1, Seq=1]
    <- 1-RTT[]: NEW_CONNECTION_ID[S2, Seq=2]
...
(starts new path)
1-RTT[0]: DCID=S2, PATH_CHALLENGE[X] -->
    Checks AEAD using nonce(CID sequence 2, PN 0)
    <- 1-RTT[0]: DCID=C1, PATH_RESPONSE[X], PATH_CHALLENGE[Y],
                 ACK_MP[Seq=2,PN=0]
    Checks AEAD using nonce(CID sequence 1, PN 0)
1-RTT[1]: DCID=S2, PATH_RESPONSE[Y],
        ACK_MP[Seq=1, PN=0], ... -->

Figure 2: Example of new path establishment
In Figure 2, the endpoints first exchange new available Connection IDs with the NEW_CONNECTION_ID frame. In this example, the client provides one Connection ID (C1 with sequence number 1), and server provides two Connection IDs (S1 with sequence number 1, and S2 with sequence number 2).

Before the client opens a new path by sending a packet on that path with a PATH_CHALLENGE frame, it has to check whether there is an unused Connection IDs available for each side. In this example, the client chooses the Connection ID S2 as the Destination Connection ID in the new path.

If the client has used all the allocated CID, it is supposed to retire those that are not used anymore, and the server is supposed to provide replacements, as specified in [QUIC-TRANSPORT]. Usually, it is desired to provide one more connection ID as currently in use, to allow for new paths or migration.

10.2. Path Closure

In this example, the client detects the network environment change (client’s 4G/Wi-Fi is turned off, Wi-Fi signal is fading to a threshold, or the quality of RTT or loss rate is becoming worse) and wants to close the initial path.

Figure 3 illustrates an example of path closing when both the client and the server use non-zero-length CIDs. For the first path, the server’s 1-RTT packets use DCID C1, which has a sequence number of 1; the client’s 1-RTT packets use DCID S2, which has a sequence number of 2. For the second path, the server’s 1-RTT packets use DCID C2, which has a sequence number of 2; the client’s 1-RTT packets use DCID S3, which has a sequence number of 3. Note that the paths use different packet number spaces. In this case, the client is going to close the first path. It identifies the path by the sequence number of the received packet’s DCID over that path (path identifier type 0x00), hence using the path_id 1. Optionally, the server confirms the path closure by sending a PATH_ABANDON frame using the sequence number of the received packet’s DCID over that path (path identifier type 0x00) as path identifier, which corresponds to the path_id 2. Both the client and the server can close the path after receiving the RETIRE_CONNECTION_ID frame for that path.
Client                                                      Server

(client tells server to abandon a path)
1-RTT[X]: DCID=S2 PATH_ABANDON[path_id_type=0, path_id=1]->
(server tells client to abandon a path)
<-1-RTT[Y]: DCID=C1 PATH_ABANDON[path_id_type=0, path_id=2],
ACK_MP[Seq=2, PN=X]

(client retires the corresponding CID)
1-RTT[U]: DCID=S3 RETIRE_CONNECTION_ID[2], ACK_MP[Seq=1, PN=Y] ->
(server retires the corresponding CID)
<- 1-RTT[V]: DCID=C2 RETIRE_CONNECTION_ID[1], ACK_MP[Seq=3, PN=U]

Figure 3: Example of closing a path when both the client and the server choose to receive non-zero-length CIDs.

Figure 4 illustrates an example of path closing when the client chooses to receive zero-length CIDs while the server chooses to receive non-zero-length CIDs. Because there is a zero-length CID in one direction, single packet number spaces are used. For the first path, the client’s 1-RTT packets use DCID S2, which has a sequence number of 2. For the second path, the client’s 1-RTT packets use DCID S3, which has a sequence number of 3. Again, in this case, the client is going to close the first path. Because the client now receives zero-length CID packets, it needs to use path identifier type 0x01, which identifies a path by the DCID sequence number of the packets it sends over that path, and hence, it uses a path_id 2 in its PATH_ABANDON frame. The server SHOULD stop sending new data on the path indicated by the PATH_ABANDON frame after receiving it. However, the client may want to repeat the PATH_ABANDON frame if it sees the server continuing to send data. When the client’s PATH_ABANDON frame is acknowledged, it sends out a RETIRE_CONNECTION_ID frame for the CID used on the first path. The server can readily close the first path when it receives the RETIRE_CONNECTION_ID frame from the client. However, since the client will not receive a RETIRE_CONNECTION_ID frame, after sending out the RETIRE_CONNECTION_ID frame, the client waits for 3 RTO before closing the path.

Client                                                      Server

(client tells server to abandon a path)
1-RTT[X]: DCID=S2 PATH_ABANDON[path_id_type=1, path_id=2]->
(server stops sending on that path after receiving PATH_ABANDON)

(client retires the corresponding CID after PATH_ABANDON is acknowledged)
1-RTT[X+1]: DCID=S3 RETIRE_CONNECTION_ID[2]->
11. Implementation Considerations

11.1. Handling different PMTU sizes

An implementation should take care to handle different PMTU sizes across multiple paths. One simple option if the PMTUs are relatively similar is to apply the minimum PMTU of all paths to each path. The benefit of such an approach is to simplify retransmission processing as the content of lost packets initially sent on one path can be sent on another path without further frame scheduling adaptations.

12. New Frames

All the new frames MUST only be sent in 1-RTT packet, and MUST NOT use other encryption levels.

If an endpoint receives multipath-specific frames from packets of other encryption levels, it MUST return MP_PROTOCOL_VIOLATION as a connection error and close the connection.

12.1. PATH_ABANDON Frame

The PATH_ABANDON frame informs the peer to abandon a path. More complex path management can be made possible with additional extensions (e.g., PATH_STATUS frame in [I-D.liu-multipath-quic]).

PATH_ABANDON frames are formatted as shown in Figure 5.

```plaintext
PATH_ABANDON Frame {
    Type (i) = TBD-02 (experiments use 0xbaba05),
    Path Identifier (..),
    Error Code (i),
    Reason Phrase Length (i),
    Reason Phrase (..),
}
```

Figure 5: PATH_ABANDON Frame Format

PATH_ABANDON frames contain the following fields:

Path Identifier: An identifier of the path, which is formatted as shown in Figure 6.
* Identifier Type: Identifier Type field is set to indicate the type
  of path identifier.

  - Type 0: Refer to the connection identifier issued by the sender
    of the control frame. Note that this is the connection
    identifier used by the peer when sending packets on the to-be-
    closed path. This method SHOULD be used if this connection
    identifier is non-zero length. This method MUST NOT be used if
    this connection identifier is zero-length.

  - Type 1: Refer to the connection identifier issued by the
    receiver of the control frame. Note that this is the
    connection identifier used by the sender when sending packets
    on the to-be-closed path. This method MUST NOT be used if this
    connection identifier is zero-length.

  - Type 2: Refer to the path over which the control frame is sent
    or received.

* Path Identifier Content: A variable-length integer specifying the
  path identifier. If Identifier Type is 2, the Path Identifier
  Content MUST be empty.

Path Identifier {
  Identifier Type (i) = 0x00..0x02,
  [Path Identifier Content (i)],
}

Figure 6: Path Identifier Format

Note: If the receiver of the PATH_ABANDON frame is using non-zero
length Connection ID on that path, endpoint SHOULD use type 0x00 for
path identifier in the control frame. If the receiver of the
PATH_ABANDON frame is using zero-length Connection ID, but the peer
is using non-zero length Connection ID on that path, endpoints SHOULD
use type 0x01 for path identifier. If both endpoints are using
0-length Connection IDs on that path, endpoints SHOULD only use type
0x02 for path identifier.

Error Code: A variable-length integer that indicates the reason for
abandoning this path.

Reason Phrase Length: A variable-length integer specifying the
length of the reason phrase in bytes. Because an PATH_ABANDON
frame cannot be split between packets, any limits on packet size
will also limit the space available for a reason phrase.

Reason Phrase: Additional diagnostic information for the closure.
This can be zero length if the sender chooses not to give details beyond the Error Code value. This SHOULD be a UTF-8 encoded string [RFC3629], though the frame does not carry information, such as language tags, that would aid comprehension by any entity other than the one that created the text.

PATH_ABANDON frames SHOULD be acknowledged. If a packet containing a PATH_ABANDON frame is considered lost, the peer SHOULD repeat it.

If the Identifier Type is 0x00 or 0x01, PATH_ABANDON frames MAY be sent on any path, not only the path identified by the Path Identifier Content field. If the Identifier Type if 0x02, the PATH_ABANDON frame MUST only be sent on the path that is intended to be abandoned.

12.2. PATH_STATUS frame

PATH_STATUS Frame are used by endpoints to inform the peer of the current status of one path, and the peer should send packets according to the preference expressed in these frames. PATH_STATUS frames are formatted as shown in Figure 7.

PATH_STATUS Frame {
    Type (i) = TBD-03 (experiments use 0xbaba06),
    Path Identifier (..),
    Path Status sequence number (i),
    Path Status (i),
}

Figure 7: PATH_STATUS Frame Format

PATH_STATUS Frames contain the following fields:

Path Identifier: An identifier of the path, which is formatted as shown in Figure 6. Exactly the same as the definition of Path Identifier in [#path-abandon-frame].

Path Status sequence number: A variable-length integer specifying the sequence number assigned for this PATH_STATUS frame. The sequence number MUST be monotonically increasing generated by the sender of the Path Status frame in the same connection. The receiver of the Path Status frame needs to use and compare the sequence numbers separately for each Path Identifier.

Available values of Path Status field are:

* 1: Standby
* 2: Available
Endpoints use PATH_STATUS frame to inform the peer whether it prefer to use this path or not. If an endpoint receives a PATH_STATUS frame containing 1-Standby status, it SHOULD stop sending non-probing packets on the corresponding path, until it receive a new PATH_STATUS frame containing 2-Available status with a higher sequence number referring to the same path.

Frames may be received out of order. A peer MUST ignore an incoming PATH_STATUS frame if it previously received another PATH_STATUS frame for the same Path Identifier with a sequence number equal to or higher than the sequence number of the incoming frame.

PATH_STATUS frames SHOULD be acknowledged. If a packet containing a PATH_STATUS frame is considered lost, the peer should only repeat it if it was the last status sent for that path -- as indicated by the sequence number.

12.3. ACK_MP Frame

The ACK_MP frame (types TBD-00 and TBD-01; experiments use 0xbaba00..0xbaba01) is an extension of the ACK frame defined by [QUIC-TRANSPORT]. It is used to acknowledge packets that were sent on different paths when using multiple packet number spaces. If the frame type is TBD-01, ACK_MP frames also contain the sum of QUIC packets with associated ECN marks received on the connection up to this point.

ACK_MP frame is formatted as shown in Figure 8.

ACK_MP Frame {
  Type (i) = TBD-00..TBD-01 (experiments use 0xbaba00..0xbaba01),
  Packet Number Space Identifier (i),
  Largest Acknowledged (i),
  ACK Delay (i),
  ACK Range Count (i),
  First ACK Range (i),
  ACK Range (..) ...,
  [ECN Counts (..)],
}

Figure 8: ACK_MP Frame Format

Compared to the ACK frame specified in [QUIC-TRANSPORT], the following field is added.

Packet Number Space Identifier: An identifier of the path packet number space, which is the sequence number of Destination Connection ID of the 1-RTT packets which are acknowledged by the ACK_MP frame.
If the endpoint receives 1-RTT packets with zero-length Connection ID, it SHOULD use Packet Number Space Identifier 0 in ACK_MP frames. If an endpoint receives an ACK_MP frame with a packet number space ID which was never issued by endpoints (i.e., with a sequence number larger than the largest one advertised), it MUST treat this as a connection error of type MP_PROTOCOL_VIOLATION and close the connection. If an endpoint receives an ACK_MP frame with a packet number space ID which is no more active (e.g., retired by a RETIRE_CONNECTION_ID frame or belonging to closed paths), it MUST ignore the ACK_MP frame without causing a connection error.

When using a single packet number space, endhosts MUST NOT send ACK_MP frames. If an endhost receives an ACK_MP frame while a single packet number space was negotiated, it MUST treat this as a connection error of type MP_PROTOCOL_VIOLATION and close the connection.

13. Error Codes

Multipath QUIC transport error codes are 62-bit unsigned integers following [QUIC-TRANSPORT].

This section lists the defined multipath QUIC transport error codes that can be used in a CONNECTION_CLOSE frame with a type of 0x1c. These errors apply to the entire connection.

MP_PROTOCOL_VIOLATION (experiments use 0xba01): An endpoint detected an error with protocol compliance that was not covered by more specific error codes.

14. IANA Considerations

This document defines a new transport parameter for the negotiation of enable multiple paths for QUIC, and two new frame types. The draft defines provisional values for experiments, but we expect IANA to allocate short values if the draft is approved.

The following entry in Table 3 should be added to the "QUIC Transport Parameters" registry under the "QUIC Protocol" heading.

<table>
<thead>
<tr>
<th>Value</th>
<th>Parameter Name.</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBD (experiments use 0xbabf)</td>
<td>enable_multipath</td>
<td>Section 3</td>
</tr>
</tbody>
</table>

Table 3: Addition to QUIC Transport Parameters Entries
The following frame types defined in Table 4 should be added to the "QUIC Frame Types" registry under the "QUIC Protocol" heading.

<table>
<thead>
<tr>
<th>Value</th>
<th>Frame Name</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBD-00 - TBD-01 (experiments use 0xbaba00-0xbaba01)</td>
<td>ACK_MP</td>
<td>Section 12.3</td>
</tr>
<tr>
<td>TBD-02 (experiments use 0xbaba05)</td>
<td>PATH_ABANDON</td>
<td>Section 12.1</td>
</tr>
</tbody>
</table>

Table 4: Addition to QUIC Frame Types Entries

The following transport error code defined in Table 5 should be added to the "QUIC Transport Error Codes" registry under the "QUIC Protocol" heading.

<table>
<thead>
<tr>
<th>Value</th>
<th>Code</th>
<th>Description</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBD (experiments use 0xba01)</td>
<td>MP_PROTOCOL_VIOLATION</td>
<td>Multipath protocol violation</td>
<td>Section 13</td>
</tr>
</tbody>
</table>

Table 5: Error Code for Multipath QUIC

15. Security Considerations

TBD

16. Contributors

This document is a collaboration of authors that combines work from three proposals. Further contributors that were also involved one of the original proposals are:

* Qing An
* Zhenyu Li

17. Acknowledgments

TBD

18. References
18.1. Normative References


18.2. Informative References


[QUIC-Invariants]

[QUIC-RECOVERY]

[QUIC-Timestamp]


Authors’ Addresses
Yanmei Liu (editor)
Alibaba Inc.
Email: miaoji.lym@alibaba-inc.com

Yunfei Ma
Alibaba Inc.
Email: yunfei.ma@alibaba-inc.com

Quentin De Coninck (editor)
UCLouvain
Email: quentin.deconinck@uclouvain.be

Olivier Bonaventure
UCLouvain
Email: olivier.bonaventure@uclouvain.be

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

Multicast Extension for QUIC
draft-jholland-quic-multicast-02

Abstract

This document defines a multicast extension to QUIC to enable the efficient use of multicast-capable networks to send identical data streams to many clients at once, coordinated through individual unicast QUIC connections.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://GrumpyOldTroll.github.io/draft-jholland-quic-multicast/draft-jholland-quic-multicast.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-jholland-quic-multicast/.

Discussion of this document takes place on the QUIC Individual Draft mailing list (mailto:quic@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/quic/.

Source for this draft and an issue tracker can be found at https://github.com/GrumpyOldTroll/draft-jholland-quic-multicast.

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 12 January 2023.

Copyright Notice

Copyright (c) 2022 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 (https://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 Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction .............................................................. 3
1.1. Conventions and Definitions ........................................ 4
2. Multicast Channel ....................................................... 4
3. Transport Parameters ................................................... 6
4. Extension Overview ....................................................... 7
  4.1. Channel Management ................................................ 8
  4.2. Client Response .................................................... 11
  4.3. Data Carried in Channels .......................................... 11
  4.4. Stream Processing ................................................ 12
5. Flow Control .............................................................. 12
6. Congestion Control ...................................................... 13
7. Data Integrity ............................................................ 14
  7.1. Packet Hashes ....................................................... 14
8. Recovery ................................................................. 15
9. Connection Termination .................................................. 15
  9.1. Stateless Reset ..................................................... 16
10. New Frames ............................................................... 16
  10.1. MC_ANNOUNCE ........................................................ 16
  10.2. MC_KEY ............................................................... 19
  10.3. MC_JOIN .............................................................. 21
  10.4. MC_LEAVE ............................................................. 22
  10.5. MC_INTEGRITY ....................................................... 23
  10.6. MC_ACK ............................................................... 23
  10.7. MC_LIMITS ........................................................... 24
  10.8. MC_RETIRE ........................................................... 25
1. Introduction

This document specifies an extension to QUIC version 1 [RFC9000] to enable the use of multicast IP transport of identical packets for use in many individual QUIC connections.

The multicast data can only be consumed in conjunction with a unicast QUIC connection. When the client has support for multicast as described in Section 3, the server can tell the client about multicast channels and ask the client to join and leave them as described in Section 4.1.

The client reports its joins and leaves to the server and acknowledges the packets received via multicast after verifying their integrity.

The purpose of this multicast extension is to realize the large scalability benefits for popular traffic over multicast-capable networks without compromising on security, network safety, or implementation reliability. Thus, this specification has several design goals:

* Re-use as much as possible the mechanisms and packet formats of QUIC version 1

* Provide flow control and congestion control mechanisms that work with multicast traffic
Maintain the confidentiality, integrity, and authentication guarantees of QUIC as appropriate for multicast traffic, fully meeting the security goals described in [I-D.draft-krose-multicast-security]

Leverage the scalability of multicast IP for data that is transmitted identically to many clients

This document does not define any multicast transport except server to client and only includes semantics for source-specific multicast.

1.1. Conventions and Definitions

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.

Commonly used terms in this document are described below.

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSM</td>
<td>Source-specific multicast, as described in [RFC4607]</td>
</tr>
<tr>
<td>ASM</td>
<td>Any-source multicast, as distinguished from SSM in [RFC4607]</td>
</tr>
<tr>
<td>(S,G)</td>
<td>A tuple of IP addresses (Source IP, Group IP) identifying a source-specific multicast channel as described in [RFC4607]</td>
</tr>
</tbody>
</table>

Table 1

2. Multicast Channel

A QUIC multicast channel (or just channel) is a one-way network path that a server can use as an alternate path to send QUIC connection data to a client.

Multicast channels are designed to leverage multicast IP and to be shared by many different connections simultaneously for unidirectional server-initiated data.
One or more servers can use the same QUIC multicast channel to send the same data to many clients, as a supplement to the individual QUIC connections between those servers and clients. (Note that QUIC connections are defined in Section 5 of [RFC9000] and are not changed in this document; each connection is a shared state between a client and a server.)

Each QUIC multicast channel has exactly one associated (S,G) that is used for the delivery of the multicast packets on the IP layer. Channels do not support any-source multicast (ASM) semantics.

Channels carry only 1-RTT packets. Packets associated with a channel contain a Channel ID in place of a Destination Connection ID. (A Channel ID cannot be zero length.) This adds a layer of indirection to the process described in Section 5.2 of [RFC9000] for matching packets to connections upon receipt. Incoming packets received on the network path associated with a channel use the Channel ID to associate the packet with a joined channel.

A client with a matching joined channel always has at least one connection associated with the channel. If a client has no matching joined channel, the packet is discarded.

Each channel has an independent packet number space. Since the network path for a channel is unidirectional and uses a different packet number space than the unicast part of the connection, packets associated with a channel are acknowledged with MC_ACK frames Section 10.6 instead of ACK frames.

The use of any particular channel is OPTIONAL for both the server and the client. It is recommended that applications designed to leverage the multicast capabilities of this extension also provide graceful degradation for endpoints that do not or cannot make use of the multicast functionality (see Section 12.4).

The server has access to all data transmitted on any multicast channel it uses, and could optionally send this data with unicast instead.

No special handling of the data is required in a client application that has enabled multicast. A datagram or any particular bytes from a server-initiated unidirectional stream can be delivered over the unicast connection or a multicast channel transparently to a client application consuming the stream or datagram.
Client applications should have a mechanism that disables the use of multicast on connections with enhanced privacy requirements for the privacy-related reasons covered in [I-D.draft-krose-multicast-security].

3. Transport Parameters

Support for multicast extensions in a client is advertised by means of QUIC transport parameters:

* name: multicast_server_support (TBD - experiments use 0xff3e808)
* name: multicast_client_params (TBD - experiments use 0xff3e800)

If a multicast_server_support transport parameter is not included, clients MUST NOT send any frames defined in this document.

If a multicast_client_params transport parameter is not included, servers MUST NOT send any frames defined in this document.

The multicast_server_support parameter is a 0-length value. Presence indicates that multicast-capable clients MAY send frames defined in this document, and SHOULD send MC_LIMITS (Section 10.7) frames as appropriate when their capabilities or client-side limitations change.

The multicast_client_params parameter has the structure shown below in Figure 1.

multicast_client_params {
    Reserved (6),
    IPv6 Channels Allowed (1),
    IPv4 Channels Allowed (1),
    Max Aggregate Rate (i),
    Max Channel IDs (i),
    Hash Algorithms Supported (i),
    AEAD Algorithms Supported (i),
    Hash Algorithms List (16 * Hash Algorithms Supported),
    AEAD Algorithms List (16 * AEAD Algorithms Supported)
}

Figure 1: multicast_client_params Format

The Reserved, IPv6 Channels Allowed, IPv4 Channels Allowed, Max Aggregate Rate, and Max Channel ID fields are identical to their analogous fields in the MC_LIMITS frame (Section 10.7) and hold the initial values.
A server MUST NOT send MC_ANNOUNCE (Section 10.1) frames with addresses using an IP Family that is not allowed according to the IPv4 and IPv6 Channels Allowed fields in the multicast_client_params, unless and until a later MC_LIMITS (Section 10.7) frame adds permission for a different address family.

The AEAD Algorithms List field is in order of preference (most preferred occurring first) using values from the TLS Cipher Suite registry (https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-4). It lists the algorithms the client is willing to use to decrypt data in multicast channels, and the server MUST NOT send an MC_ANNOUNCE to this client for any channels using unsupported algorithms. If the server does send an MC_ANNOUNCE with an unsupported cipher suite, the client SHOULD treat it as a connection error of type MC_EXTENSION_ERROR.

The Hash Algorithms List field is in order of preference (most preferred occurring first) using values from the registry below. It lists the algorithms the client is willing to use to check integrity of data in multicast channels, and the server MUST NOT send an MC_ANNOUNCE to this client for any channels using unsupported algorithms, or the client SHOULD treat it as a connection error of type MC_EXTENSION_ERROR:

* https://www.iana.org/assignments/named-information/named-information.xhtml#hash-alg

4. Extension Overview

A client has the option of refusal and the power to impose upper bound maxima on several resources (see Section 5), but otherwise its join status for all multicast channels is entirely managed by the server.

* A client MUST NOT join a channel without receiving instructions from a server to do so.

* A client MUST leave joined channels when instructed by the server to do so.

* A client MAY leave channels or refuse to join channels, regardless of instructions from the server.
4.1. Channel Management

The client tells its server about some restrictions on resources that it is capable of processing with the initial values in the multicast_client_params transport parameter (Section 3) and later can update these limits with MC_LIMITS Section 10.7 frames. Servers ensure the set of channels the client is currently requested to join remains within these advertised client limits as covered in Section 5.

The server asks the client to join channels with MC_JOIN (Section 10.3) frames and to leave channels with MC_LEAVE (Section 10.4) frames.

The server uses the MC_ANNOUNCE (Section 10.1) frame before any join or leave frames for the channel to describe the channel properties to the client, including values the client can use to ensure the server’s requests remain within the limits it has sent to the server, as well as the secrets necessary to decode the headers of packets in the channel. MC_KEY frames provide the secrets necessary to decode the payload of packets in the channel. Figure 2 shows the states a channel has from the clients point of view.

Joining a channel after receiving an MC_JOIN frame is OPTIONAL for clients. If a client decides not to join after being asked to do so, it can indicate this decision by sending an MC_STATE (Section 10.9) frame with state DECLINED_JOIN and an appropriate reason.

The server ensures that in aggregate, all channels that the client has currently been asked to join and that the client has not left or declined to join fit within the limits indicated by the initial values in the transport parameter or last MC_LIMITS (Section 10.7) frame the server received.
*: Each transition except the initial receiving of MC_ANNOUNCE and MC_KEY frames causes the client to send an MC_STATE frame describing the state transition (for LEFT or DECLINED_JOIN, this includes a reason for the transition).

"able to join" means:
- Both MC_KEY and MC_ANNOUNCE have been received
- Result will be within latest advertised client limits
- Nothing preventing a join is active (e.g. a hold-down timer, administrative blocking, etc.)

Figure 2: States a channel from the client's point of view.

When the server has asked the client to join a channel and has not received any MC_STATE frames Section 10.9 with state DECLINED_JOIN or LEFT, it also sends MC_INTEGRITY frames (Section 10.5) to enable the client to verify packet integrity before processing the packet. A client MUST NOT decode packets for a channel for which it has not received an applicable MC_ANNOUNCE (Section 10.1), or for which it has not received a matching packet hash in an MC_INTEGRITY (Section 10.5) frame, or for which it has not received an applicable MC_KEY frame Section 10.2.

Figure 3 shows the frames that are being exchanged about and over a channel during the lifetime of an example channel.
Client

MC_LIMITS/initial_limits  --->

Server

MC_ANNOUNCE
MC_KEY
<----  MC_JOIN

MC_STATE(JOINED)  --->

MC_INTEGRITY
<----  [STREAM(...)]

MC_ACK  --->

...  <----  MC_KEY

...  <----  MC_LIMITS  --->

<----  MC_LEAVE

MC_STATE(LEFT)  --->

<----  MC_JOIN

MC_STATE(JOINED)  --->

MC_INTEGRITY
<----  [STREAM(...)]

MC_ACK  --->

...  <----  MC_LEAVE

MC_STATE(LEFT)  --->

<----  MC_LEAVE

MC_STATE(RETIRED)  --->

<----  MC_RETIRE

Figure 3: Example flow of frames for a channel. Frames in square brackets are sent over multicast.

4.2. Client Response

The client sends back information about how it has responded to the server’s requests to join and leave channels in MC_STATE (Section 10.9) frames. MC_STATE frames are only sent for channels after the server has requested the client to join the channel, and are thereafter sent any time the state changes.

Clients that receive and decode data on a multicast channel send acknowledgements for the data on the unicast connection using MC_ACK (Section 10.6) frames.

A server can determine if a client receives packets for a multicast channel if it receives MC_ACK frames associated with that channel. As such, it is in general up to the server to decide on the time after which it deems a client to be unable to receive packets on a given channel and take appropriate steps, e.g., sending an MC_LEAVE frame to the client. Note that clients willing to join a channel SHOULD remain joined to the channel even if they receive no channel data for an extended period, to enable multicast-capable networks to perform popularity-based admission control for multicast channels.

4.3. Data Carried in Channels

Data transmitted in a multicast channel is encrypted with symmetric keys so that on-path observers without access to these keys cannot decode the data. However, since potentially many receivers receive identical packets and identical keys for the multicast channel and some receivers might be malicious, the packets are also protected by MC_INTEGRITY (Section 10.5) frames transmitted over a separate integrity-protected path.

A client MUST NOT decode packets on a multicast channel for which it has not received a matching hash in an MC_INTEGRITY frame over a different integrity-protected communication path. The different path can be either the unicast connection or another multicast channel with packets that were verified with an earlier MC_INTEGRITY frame.

Note that MC_INTEGRITY frames MAY be carried in packets on multicast channels, however such packets will not be accepted unless another accepted MC_INTEGRITY frame contains its packet hash. Hashes of packets containing hashes of other packets can thus form a Merkle tree [MERKLE] with a root that is carried in the unicast connection.

See Section 7 for a more complete overview of the security issues involved here.
4.4. Stream Processing

Stream IDs in channels are restricted to unidirectional server initiated streams, or those with the least significant 2 bits of the stream ID equal to 3 (see Section 2.1 of [RFC9000]).

When a channel contains streams with IDs above the client’s unidirectional MAX_STREAMS, the server MUST NOT instruct the client to join that channel and SHOULD send a STREAMS_BLOCKED frame, as described in Sections 4.6 and 19.14 of [RFC9000].

If the client is already joined to a channel that carries streams that exceed or will soon exceed the client’s unidirectional MAX_STREAMS, the server SHOULD send an MC_LEAVE frame.

If a client receives a STREAM frame with an ID above its MAX_STREAMS on a channel, the client MAY increase its unidirectional MAX_STREAMS to a value greater than the new ID and send an update to the server, otherwise it MUST drop the packet and leave the channel with reason "MAX_STREAMS_EXCEEDED".

Since clients can join later than a channel began, it is RECOMMENDED that clients supporting the multicast extensions to QUIC be prepared to handle stream IDs that do not begin at early values, since by the time a client joins a channel in progress the stream ID count might have been increasing for a long time. Clients should therefore begin with a high initial_max_streams_uni or send an early MAX_STREAMS type 0x13 value (see Section 19.11 of [RFC9000]) with a high limit. Clients MAY use the maximum 2^60 for this high initial limit, but the specific choice is implementation-dependent.

The same stream ID may be used in both one or more multicast channels and the unicast connection. As described in Section 2.2 of [RFC9000], stream data received multiple times for the same offset MUST be identical, even across different network paths; if it’s not identical it MAY be treated as a connection error of type MC_EXTENSION_ERROR.

5. Flow Control

The values used for unicast flow control cannot be used to limit the transmission rate of a multicast channel because a single client with a low MAX_STREAM_DATA or MAX_DATA value that did not acknowledge receipt could block many other receivers if the servers had to ensure that channels responded to each client’s limits.
Instead, clients advertise resource limits via MC_LIMITS (Section 10.7) frames and their initial values from the transport parameter (Section 3). The server is responsible for keeping the client within its advertised limits, by ensuring via MC_JOIN and MC_LEAVE frames that the set of channels the client is asked to be joined to will not, in aggregate, exceed the client’s advertised limits. The server also advertises the expected maxima of the values that can contribute toward client resource limits within a channel in an MC_ANNOUNCE (Section 10.1) frame, and the client also ensures that the set of channels it’s joined to does not exceed its limits, according to the advertised values. The client also monitors the packets received to ensure that channels don’t exceed their advertised values, and leaves channels that do.

If the server asks the client to join a channel that would exceed the client’s limits with an up-to-date Client Limit Sequence Number, the client should send back an MC_STATE frame (Section 10.9) with "DECLINED_JOIN" and reason "PROPERTY_VIOLATION". If the server asks the client to join a channel that would exceed the client’s limits with an out-of-date Client Limit Sequence Number or a Channel Key Sequence Number that the client has not yet seen, the client should instead send back a "DECLINED_JOIN" with "Desynchronized Limit Violation". If the actual contents sent in the channel exceed the advertised limits from the MC_ANNOUNCE, clients SHOULD leave the stream and send an MC_STATE(LEFT) frame, using the Limit Violated reason.

6. Congestion Control

Both the server and the client perform congestion control operations, so that according to the guidelines in Section 4.1 of [RFC8085], mechanisms for both feedback-based and receiver-driven styles of congestion control are present and operational.

The server maintains a full view of the traffic received by the client via the MC_ACK (Section 10.6) frames and ACK frames it receives, and can detect loss experienced by the client. Under sustained persistent loss that exceeds server-configured thresholds, the server SHOULD instruct the client to leave channels as appropriate to avoid having the client continue to see sustained persistent loss.
Under sustained persistent loss that exceeds client-configured thresholds, the client SHOULD reduce its Max Rate and tell the server via MC_LIMITS frames, which also will result in the server instructing the client to leave channels until the clients aggregate rate is below its advertised Max Rate. Under a higher threshold of sustained persistent loss, the client also SHOULD leave channels, using an MC_STATE(LEFT) frame with the "HIGH_LOSS" reason, as well as reducing the Max Rate in MC_LIMITS.

The unicast connection’s congestion control is unaffected. However a few potential interactions with the unicast connection are worth highlighting:

* if the client notices high loss on the unicast connection while multicast channel packets are arriving, the client MAY leave channels with reason "HIGH_LOSS".

* if the client notices congestion from unicast this MAY also drive reductions in the client’s Max Rate, and a lack of unicast congestion under unicast load MAY also drive increases to the client’s Max Rate (along with an updated MC_LIMITS frame).

Hybrid multicast-unicast congestion control is still an experimental research topic. Implementations SHOULD follow the guidelines given in Section 4.1.1 of [RFC8085] under the assumption that applications using QUIC multicast will operate as Bulk-Transfer applications.

7. Data Integrity

TODO: import the [I-D.draft-krose-multicast-security] explanation for why extra integrity protection is necessary (many client have the shared key, so AEAD doesn’t provide authentication against other valid clients on its own, since the same key is given to multiple clients and as the client count grows so does the chance that at least one client is controlled by an attacker.)

7.1. Packet Hashes

TODO: explanation and example for how to calculate the packet hash. Note that the hash is on the encrypted packet to avoid leaking data about the encrypted contents to those who can see a hash but not the key. (This approach also may help make better use of [I-D.draft-ietf-mboned-ambi] by making it possible to generate the same hashes for use in both AMBI and QUIC MC_INTEGRITY frames.)
8. Recovery

TODO: Articulate key differences with [RFC9002]. The main known difference is that servers might not be running on the same devices that are sending the channel packets, therefore the RTT for channel packets might use an estimated send time that can vary according to the clock synchronization among servers and the deployment and implementation details of how the servers find out the sending timestamps of channel packets. Experience-based guidance on the recovery timing estimates is one anticipated outcome of experimenting with deployments of this experimental extension.

All the new frames defined in this document except MC_ACK are ack-eliciting and are retransmitted until acknowledged to provide reliable, though possibly out of order, delivery.

Note that recovery MAY be achieved either by retransmitting frame data that was lost and needs reliable transport either by sending the frame data on the unicast connection or by coordinating to cause an aggregated retransmission of widely dropped data on a multicast channel, at the server’s discretion. However, the server in each connection is responsible for ensuring that any necessary server-to-client frame data lost by a multicast channel packet loss ultimately arrives at the client.

9. Connection Termination

Termination of the unicast connection behaves as described in Section 10 of [RFC9000], with the following notable differences:

* On the client side, termination of the unicast connection means that it MUST leave all multicast channels and discard any state associated with them. Servers MAY stop sending to multicast channels if there are no unicast connections left that are associated with them.

* For determining the liveness of a connection, the client MUST only consider packets received on the unicast connection. Any packets received on a multicast channel MUST NOT be used to reset a timer checking if a potentially specified max_idle_timeout has been reached. If the unicast connection becomes idle, as described in Section 10.1 of [RFC9000], the client MUST terminate the connection as described above.
9.1. Stateless Reset

As clients can unilaterally stop the delivery of multicast packets by leaving the relevant (S,G), channels do not need stateless reset tokens. Clients therefore do not share the stateless reset tokens of channels with the server. Instead, if an endpoint receives packets addressed to an (S,G) that it can not associate with any existing channel, it MAY take the necessary steps to prevent the reception of further such packets, without the need to signal to the server that it should stop sending.

If a server or client detect a stateless reset for a channel, they MUST ignore it.

10. New Frames

10.1. MC_ANNOUNCE

Once a server learns that a client supports multicast through its transport parameters, it can send one or multiple MC_ANNOUNCE frames (type=TBD-11..TBD-12) to share information about available channels with the client. The MC_ANNOUNCE frame contains the properties of a channel that do not change during its lifetime.

MC_ANNOUNCE frames are formatted as shown in Figure 4.

MC_ANNOUNCE Frame {
    Type (i) = TBD-11..TBD-12 (experiments use 0xff3e811/0xff3e812),
    ID Length (8),
    Channel ID (8..160),
    Source IP (32..128),
    Group IP (32..128),
    UDP Port (16),
    Header AEAD Algorithm (16),
    Header Secret Length (i),
    Header Secret (..),
    AEAD Algorithm (16),
    Integrity Hash Algorithm (16),
    Max Rate (i),
    Max ACK Delay (i)
}

Figure 4: MC_ANNOUNCE Frame Format

Frames of type TBD-11 are used for IPv4 and both Source and Group address are 32 bits long. Frames of type TBD-12 are used for IPv6 and both Source and Group address are 128 bits long.
MC_ANNOUNCE frames contain the following fields:

* ID Length: The length in bytes of the Channel ID field.

* Channel ID: The channel ID of the channel that is getting announced.

* Source IP: The IP Address of the source of the (S,G) for the channel. Either a 32-bit IPv4 address or a 128-bit IPv6 address, as indicated by the frame type (TBD-11 indicates IPv4, TBD-12 indicates IPv6).

* Group IP: The IP Address of the group of the (S,G) for the channel. Either a 32-bit IPv4 address or a 128-bit IPv6 address, as indicated by the frame type (TBD-11 indicates IPv4, TBD-12 indicates IPv6).

* UDP Port: The 16-bit UDP Port of traffic for the channel.

* Header AEAD Algorithm: A value from the TLS Cipher Suite registry (https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-4 (https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-4)), used to protect the header fields in the channel packets. The value MUST match a value provided in the "AEAD Algorithms List" of the transport parameter (see Section 3).

* Header Secret Length: Provides the length of the Secret field.

* Header Secret: A secret for use with the Header AEAD Algorithm for protecting the header fields of 1-RTT packets in the channel as described in [RFC9001]. The Key and Initial Vector for the application data carried in the 1-RTT packet header fields are derived from this secret as described in Section 7.3 of [RFC8446].

* AEAD Algorithm: A value from the TLS Cipher Suite registry (https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-4 (https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-4)), used to protect the payloads in the channel packets. The value MUST match a value provided in the "AEAD Algorithms List" of the transport parameter (see Section 3).

* Integrity Hash Algorithm: The hash algorithm used in integrity frames.
*Author’s Note:* Several candidate IANA registries, not sure which one to use? Some have only text for some possibly useful values. For now we use the first of these:

- [https://www.iana.org/assignments/named-information/named-information.xhtml#hash-alg](https://www.iana.org/assignments/named-information/named-information.xhtml#hash-alg)

- [https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-18](https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-18)

- (text-only): [https://www.iana.org/assignments/hash-function-text-names/hash-function-text-names.xhtml](https://www.iana.org/assignments/hash-function-text-names/hash-function-text-names.xhtml)

*Max Rate:* The maximum rate in Kibps of the payload data for this channel. Channel data MUST NOT exceed this rate over any 5s window, if it does clients SHOULD leave the channel with reason "MAX_RATE_EXCEEDED".

*Max ACK Delay:* A value used similarly to max_ack_delay (Section 18.2 of [RFC9000]) that applies to traffic in this channel. Clients SHOULD NOT intentionally add delay to MC_ACK frames for traffic in this channel beyond this value, in milliseconds, and SHOULD NOT add any delay to the first MC_ACK of data packets for a channel. As long as they stay inside these limits, clients can improve efficiency and network load for the uplink by aggregating MC_ACK frames whenever possible.

A client MUST NOT use the channel ID included in an MC_ANNOUNCE frame as a connection ID for the unicast connection. If it is already in use, the client should retire it as soon as possible. As the server knows which connection IDs are in use by the client, it MUST wait with the sending of an MC_JOIN frame until the channel ID associated with it has been retired by the client.

As all the properties in MC_ANNOUNCE frames are immutable during the lifetime of a channel, a server SHOULD NOT send an MC_ANNOUNCE frame for the same channel more than once to each client except as needed for recovery.
A server SHOULD send an MC_ANNOUNCE frame for a channel before sending an MC_KEY and SHOULD send an MC_KEY frame for a channel before sending an MC_JOIN frame for it. Each of these recommended orderings MAY occur within the same packet.

10.2. MC_KEY

An MC_KEY frame (type=TBD-01) is sent from server to client, either with the unicast connection or in an existing joined multicast channel. The MC_KEY frame contains an updated secret that is used to generate the keying material for the payload of 1-RTT packets received on the multicast channel.

A server can send a new MC_KEY frame with a sequence number increased by one. A server MUST generate continuous sequence numbers, and MAY start at a value higher than 0. Note that while not joined, a client will not receive updates to channel secrets, and thus may see jumps in the Key Sequence Number values between MC_KEY frames. However, while joined the Key Sequence Numbers in the MC_KEY frames MUST increment by 1 for each new secret.

Secrets with even-valued Key Sequence Numbers have a Key Phase of 0 in the 1-RTT packet, and secrets with odd-valued Key Sequence Numbers have a Key Phase of 1 in the 1-RTT packet. Secrets with a Key Phase indicating an unknown key SHOULD be discarded without attempting to decrypt them. (An unknown key might happen after loss of the latest MC_KEY frame, so that packets on a channel have an updated Key Phase starting at a particular packet number, but the client does not yet know about the key change.)

It is RECOMMENDED that servers send regular secret updates.

MC_KEY frames are formatted as shown in Figure 5.

MC_KEY Frame {
    Type (i) = TBD-01 (experiments use 0xff3e801),
    ID Length (8),
    Channel ID (8..160),
    Key Sequence Number (i),
    From Packet Number (i),
    Secret Length (i),
    Secret (..)
}

Figure 5: MC_KEY Frame Format

MC_KEY frames contain the following fields:
* ID Length: The length in bytes of the Channel ID field.

* Channel ID: The channel ID for the channel associated with this frame.

* Key Sequence Number: Increases by 1 each time the secret for the channel is changed by the server. If there is a gap in sequence numbers due to reordering or retransmission of packets, on receipt of the older MC_KEY frame, the client MUST apply the secret contained and the packet numbers on which it applies as if they arrived in order.

* From Packet Number: The values in this MC_KEY frame apply only to packets starting at From Packet Number and continuing until they are overwritten by a new MC_KEY frame with a higher From Packet Number. The Packet Number MUST never decrease with an increased Key Sequence Number.

* Secret Length: Provides the length of the secret field.

* Secret: Used to protect the packet contents of 1-RTT packets for the channel as described in [RFC9001]. The Key and Initial Vector for the application data carried in the 1-RTT packet payloads are derived from the secret as described in Section 7.3 of [RFC8446]. To maintain forward secrecy and prevent malicious clients from decrypting packets long after they have left or were removed from the unicast connection, servers SHOULD periodically send key updates using only unicast.

Clients MUST delete old secrets and the keys derived from them after receiving new MC_KEY frames. Deleting old keys prevents later compromise of a client from discovering an otherwise uncompromised key, thus improving the chances of achieving forward secrecy for data sent before a key rotation.

Client implementations MAY institute a delay before deleting secrets to allow for decoding of packets for the channel that arrive shortly after a new MC_KEY frame. For this experimental specification, it is RECOMMENDED that clients delete old keys 10 seconds after receiving a new key or after 3 seconds that elapse without receiving any new data to decode with the old key, whichever is shorter. Clients MUST NOT delay more than 60 seconds before deleting the old keys.

The delay values for this specification are somewhat arbitrary and allow for implementation-dependent experimentation. One of the target discoveries for experimental evaluation is to determine good default delay values to use, and to understand whether there are use cases that would benefit from a negotiation between server and client.
to determine the delays to use dynamically. (A poor delay choice results in either overhead from dropping packets instead of decoding them with old keys for too short a delay or in extra forward secrecy exposure time for too long a delay, and the purpose of the delays are to bound the forward secrecy exposure without inducing unreasonable overhead.)

The From Packet Number is used to indicate the starting packet number (Section 17.1 of [RFC9000]) of the 1-RTT packets for which the secret contained in an MC_KEY frame is applicable. This secret is applicable to all future packets until it is updated by a new MC_KEY frame.

A server SHOULD NOT send MC_KEY frames for channels except those the client has joined or will be imminently asked to join.

10.3. MC_JOIN

An MC_JOIN frame (type TBD-02) is sent from server to client and requests that a client join the given transport addresses and ports and process packets with the given Channel ID according to the corresponding MC_ANNOUNCE frame and the latest MC_KEY frame for the channel.

A client cannot join a multicast channel without first receiving an MC_ANNOUNCE frame and an MC_KEY frame, which together set all the values necessary to process the channel.

If a client receives an MC_JOIN for a channel for which it has not received both an MC_ANNOUNCE frame and an MC_KEY frame, it MUST respond with an MC_STATE with State "DECLINED_JOIN" and reason "Missing Properties". The server MAY send another MC_JOIN after receiving an acknowledgement indicating receipt of the MC_ANNOUNCE frame and the MC_KEY frame.

MC_JOIN frames are formatted as shown in Figure 6.

```
MC_JOIN Frame {
  Type (i) = TBD-02 (experiments use 0xff3e802),
  MC_LIMITS Sequence Number (i),
  MC_STATE Sequence Number (i),
  MC_KEY Sequence Number (i),
  ID Length (8),
  Channel ID (8..160)
}
```

Figure 6: MC_JOIN Frame Format
The sequence numbers are the most recently processed sequence number by the server from the respective frame type. They are present to allow the client to distinguish between a broken server that has performed an illegal action and an instruction that’s based on updates that are out of sync (either one or more missing updates to MC_KEY not yet received by the client or one or more missing updates to MC_LIMITS or MC_STATE not yet received by the server).

A client MAY perform the join if it has the sequence number of the corresponding channel properties and the client’s limits will not be exceeded, even if the client sequence numbers are not up-to-date.

If the client does not join, it MUST send an MC_STATE frame with "DECLINED_JOIN" and a reason.

If the client does join, it MUST send an MC_STATE frame with "JOINED".

10.4. MC_LEAVE

An MC_LEAVE frame (type=TBD-03) is sent from server to client, and requests that a client leave the given channel.

If the client has already left or declined to join the channel, the MC_LEAVE is ignored.

If an MC_JOIN or an MC_LEAVE with the same Channel ID and a higher MC_STATE Sequence number has previously been received, the MC_LEAVE is ignored.

Otherwise, the client MUST leave the channel and send a new MC_STATE frame with reason LEFT as requested by server.

MC_LEAVE frames are formatted as shown in Figure 7.

MC_LEAVE Frame {
  Type (i) = TBD-03 (experiments use 0xff3e803),
  ID Length (8),
  Channel ID (8..160),
  MC_STATE Sequence Number (i),
  After Packet Number (i)
}

Figure 7: MC_LEAVE Frame Format

If After Packet Number is nonzero, wait until receiving that packet or a higher valued number before leaving.
10.5. MC_INTEGRITY

MC_INTEGRITY frames are sent from server to client and are used to convey packet hashes for validating the integrity of packets received over the multicast channel as described in Section 7.1.

MC_INTEGRITY frames are formatted as shown in Figure 8.

MC_INTEGRITY Frame {
  Type (i) = TBD-04..TBD-05 (experiments use 0xff3e804/0xff3e805),
  ID Length (8),
  Channel ID (8..160),
  Packet Number Start (i),
  [Length (i)],
  Packet Hashes (..)
}

Figure 8: MC_INTEGRITY Frame Format

For type TBD-05, Length is present and is a count of packet hashes. For TBD-04, Length is not present and the packet hashes extend to the end of the packet.

The first hash in the Packet Hashes list is a hash of a 1-RTT packet with the Channel ID equal to the Channel ID in the MC_INTEGRITY frame and packet number equal to the Packet Number Start field. Subsequent hashes refer to the packets for the channel with packet numbers increasing by 1.

Packet hashes MUST have length with an integer multiple of the length indicated by the Hash Algorithm from the MC_ANNOUNCE frame.

See Section 7.1 for a description of the packet hash calculation.

10.6. MC_ACK

The MC_ACK frame (types TBD-06 and TBD-07; experiments use 0xff3e806..0xff3e807) is an extension of the ACK frame defined by [RFC9000]. It is used to acknowledge packets that were sent on multicast channels. If the frame type is TBD-07, MC_ACK frames also contain the sum of QUIC packets with associated ECN marks received on the connection up to this point.

(TODO: Would there be value in reusing the multiple packet number space version of ACK_MP from Section 12.2 of [I-D.draft-ietf-quic-multipath], defining channel ID as the packet number space? at 2022-05 they’re identical except the Channel ID and types.)
MC_ACK frames are formatted as shown in Figure 9.

MC_ACK Frame {
    Type (i) = TBD-06..TBD-07 (experiments use 0xff3e806, 0xff3e807),
    ID Length (8),
    Channel ID (8..160),
    Largest Acknowledged (i),
    ACK Delay (i),
    ACK Range Count (i),
    First ACK Range (i),
    ACK Range (..) ...,
    [ECN Counts (..)],
}

Figure 9: MC_ACK Frame Format

10.7. MC_LIMITS

MC_LIMITS frames are formatted as shown in Figure 10.

MC_LIMITS Frame {
    Type (i) = TBD-09 (experiments use 0xff3e809),
    Client Limits Sequence Number (i),
    Reserved (6),
    IPv6 Channels Allowed (1),
    IPv4 Channels Allowed (1),
    Max Aggregate Rate (i),
    Max Channel IDs (i),
    Max Joined Count (i),
}

Figure 10: MC_LIMITS Frame Format

The sequence number is implicitly 0 before the first MC_LIMITS frame from the client, and increases by 1 each new frame that’s sent. Newer frames override older ones.

The 6 Reserved bits MUST be set to 0 by the client and MUST be ignored by the server. These are reserved to advertise future capabilities.

IPv6 Channels Allowed is a 1-bit field set to 1 if IPv6 channels can be joined and 0 if IPv6 channels cannot be joined.

IPv4 Channels Allowed is a 1-bit field set to 1 if IPv4 channels can be joined and 0 if IPv4 channels cannot be joined.

Max Aggregate Rate allowed across all joined channels is in Kibps.
Max Channel IDs is the count of channel IDs that can be announced to this client and have keys. Retired Channel IDs don’t count against this value.

Max Joined Count is the count of channels that are allowed to be joined concurrently.

10.8. MC_RETIRE

MC_RETIRE frames are formatted as shown in Figure 11.

```
MC_RETIRE Frame {
    Type (i) = TBD-0a (experiments use 0xff3e80a),
    ID Length (8),
    Channel ID (8..160),
    After Packet Number (i)
}
```

Figure 11: MC_RETIRE Frame Format

Retires a channel by ID, discarding any state associated with it. (Author comment: We can’t use RETIRE_CONNECTION_ID because we don’t have a coherent sequence number.) If After Packet Number is nonzero and the channel is joined and has received any data, the channel will be retired after receiving that packet or a higher valued number, otherwise it will be retired immediately.

After retiring a channel, the client MUST send a new MC_STATE frame with reason RETIRED to the server.

If the client is still joined in the channel that is being retired, it MUST also leave it. If a channel is left this way, it does not need to send an additional MC_STATE frame with state LEFT, as state RETIRED also implies the channel was left.

10.9. MC_STATE

MC_STATE frames (type=TBD-0b or TBD-0c) are sent from client to server to report changes in the client’s channel state. Each time the channel state changes, the Client Channel State Sequence number is increased by one. It is a state change to the channel if the server requests that a client join a channel and the client declines the join, even though no join occurs on the network.

Frames of type TBD-0b are used for cases in which the reason for the state change occur in the QUIC multicast layer while frames of type TBD-0c are used for reasons that are application specific.
MC_STATE frames are formatted as shown in Figure 12.

MC_STATE Frame {
  Type (i) = TBD-0b..TBD-0c (experiments use 0xff3e80b and 0xff3e80c),
  Client Channel State Sequence Number (i),
  ID Length (8),
  Channel ID (8..160),
  State (8),
  Reason Code (i),
  Reason Phrase Length (i),
  Reason Phrase (..)
}

Figure 12: MC_STATE Frame Format

State has these defined values:
* 0x1: LEFT
* 0x2: DECLINED_JOIN
* 0x3: JOINED
* 0x4: RETIRED

If a server receives an undefined value, it SHOULD close the connection with reason MC_EXTENSION_ERROR.

If State is JOINED or RETIRED, the Reason Code MUST be REQUESTED_BY_SERVER (0x1).

If State is LEFT or DECLINED_JOIN, for frames of type TBD-0b the Reason Code field is set to one of:
* 0x0: UNSPECIFIED_OTHER
* 0x1: REQUESTED_BY_SERVER
* 0x2: ADMINISTRATIVE_BLOCK
* 0x3: PROTOCOL_ERROR
* 0x4: PROPERTY_VIOLATION
* 0x5: UNSYNCHRONIZED_PROPERTIES
* 0x6: ID_COLLISION
* 0x10: HELD_DOWN
* 0x12: MAX_RATE_EXCEEDED
* 0x13: HIGH_LOSS
* 0x14: EXCESSIVE_SPURIOUS_TRAFFIC
* 0x15: MAX_STREAMS_EXCEEDED

(Author’s note TODO: consider whether that these reasons should be added to the QUIC Transport Error Codes registry (Section 22.5 of [RFC9000]) instead of defining a new registry specific to multicast.)

For frames of type TBD-0c, the Reason Code is left to the application, as described in Section 20.2 of [RFC9000]

The Reason Phrase field, in combination with the Reason Phrase Length field, can optionally be used to give further details for the state change.

A client might receive multicast packets that it cannot associate with any channel ID, or that cannot be verified as matching hashes from MC_INTEGRITY frames, or cannot be decrypted. This traffic is presumed either to have been corrupted in transit or to have been sent by someone other than the legitimate sender of traffic for the channel, possibly by an attacker or a misconfigured sender. If these packets are addressed to an (S,G) that is used for reception in one or more known channels, the client MAY leave these channels with reason "Excessive Spurious traffic".

11. Frames Carried in Channel Packets

Multicast channels will contain normal QUIC 1-RTT data packets (see Section 17.3.1 of [RFC9000]) except using the Channel ID instead of a Connection ID. The packets are protected with the keys derived from the secrets in MC_KEY frames for the corresponding channel.

Data packet hashes will also be sent in MC_INTEGRITY frames, as keys cannot be trusted for integrity due to giving them to too many receivers, as described in [I-D.draft-krose-multicast-security].

The 1-RTT packets in multicast channels will have a restricted set of frames. Since the channel is strictly 1-way server to client, the general principle is that broadcastable shared server->client data frames can be sent, but frames that make sense only for individualized connections or that are sent client-to-server cannot.
Should a not permitted frame arrive on a multicast channel, the connection MUST be closed with a connection error of type MC_EXTENSION_ERROR.

Permitted:

* PADDING Frames (Section 19.1 of [RFC9000])
* PING Frames (Section 19.2 of [RFC9000])
* RESET_STREAM Frames (Section 19.4 of [RFC9000])
* STREAM Frames (Section 19.8 of [RFC9000])
* DATAGRAM Frames (both types) (Section 4 of [RFC9221])
* MC_KEY
* MC_LEAVE (however, join must come over unicast?)
* MC_INTEGRITY (not for this channel, only for another)
* MC_RETIRE

Not permitted:

* 19.3. ACK Frames
* 19.6. CRYPTO Frames (crypto handshake does not happen on mc channels)
* 19.7. NEW_TOKEN Frames

* Flow control is different:
  - 19.5. STOP_SENDING Frames
  - 19.9. MAX_DATA Frames (flow control for mc channels is by rate)
  - 19.10. MAX_STREAM_DATA Frames
  - 19.11. MAX_STREAMS Frames
  - 19.12. DATA_BLOCKED Frames
  - 19.13. STREAM_DATA_BLOCKED Frames
- 19.14. STREAMS_BLOCKED Frames

* Channel ID Migration can't use the "prior to" concept within a channel, not 0-starting

- 19.15. NEW_CONNECTION_ID Frames
- 19.16. RETIRE_CONNECTION_ID Frames

* Channels don’t have the same kind of path validation, as there’s a unicast anchor with acks for the multicast packets:

- 19.17. PATH_CHALLENGE Frames
- 19.18. PATH_RESPONSE Frames

* 19.19. CONNECTION_CLOSE Frames
* 19.20. HANDSHAKE_DONE Frames
* MC_ANNOUNCE
* MC_LIMITS
* MC_STATE
* MC_ACK

12. Implementation and Operational Considerations

12.1. Constraints on Stream Data

Note that when a newly connected client joins a channel, the client will only be able to receive application data carried in stream frames delivered on that channel when they have received the stream data starting from offset 0 of the stream.

This usually means that new streams must be started for application data carried in channel packets whenever there might be new clients that have joined since an earlier stream started.

With broadcast video, this usually means a new stream is necessary for every video segment or group of video frames since new clients will join throughout the broadcast, whereas for video conferencing, it could be possible to start a new stream whenever new clients join the conference without needing a new stream per object.
12.2. Application Use Cases

There are several known applications that could benefit from using multicast QUIC, either with their own custom application-layer transport or with one of the transports discussed in Section 12.3. A few examples include:

* Existing multicast-capable applications that are modified to use QUIC datagrams instead of UDP payloads can potentially get improved encryption and congestion feedback, while keeping existing error recovery techniques (e.g. techniques based on the forward error correction (FEC) framework in [RFC6363]).
  - An external tunnel could supply this kind of encapsulation without modification to the sender or receiver for some applications, while retaining the benefits of multicast scalability
  - Using QUIC datagrams in place of UDP packets could usefully support existing implementations of file-transfer protocols like FLUTE [RFC6726] or FCAST [RFC6968] to enable file downloads such as operating system updates or popular game downloads, but adding encryption, packet-level authentication, and congestion control as provided by QUIC.

* Conferencing systems, especially within an enterprise that can deploy multicast network support, often can save significantly on server costs by using multicast

* The traditional multicast use case of broadcasting of live sports with a set-top box would benefit from an interoperable system such as these QUIC extensions that can fall back to unicast transparently as needed, for example if there are a few customers who installed a non-multicast-capable home router.

* Smart TVs or other video playing in-home devices could interoperate with a standard sender using multicast QUIC, rather than requiring proprietary integrations with TV operators.

12.3. Data Transport Use Cases

This section outlines considerations for some known transport mechanisms that are worth highlighting as potentially useful with multicast QUIC.
12.3.1. HTTP/3 Server Push

HTTP/3 Server Push is defined in Section 4.6 of [RFC9114].

Server push is a good use case for multicast transport because the same data can be pushed to many different receivers on a multicast channel. Applications designed to work well with server push can leverage multicast QUIC very effectively with only a few extra considerations.

A QUIC connection using HTTP/3 can use multicast channels to deliver server-initiated streams that implement HTTP/3 Server Push.

Applications expecting to use server push with multicast SHOULD use a high MAX_PUSH_ID in order to work with channels that have been active for a long time already when the connection is first established. Servers SHOULD NOT allow clients to remain joined to channels if their MAX_PUSH_ID will be exceeded by push streams that are to be sent imminently.

If a client receives data from a push ID that exceeds its MAX_PUSH_ID causing an H3_ID_ERROR on a multicast channel, it SHOULD leave the channel with reason 0x1000108 (computed by adding the H3_ID_ERROR value 0x0108 to the Application-defined Reason start value 0x1000000). This SHOULD NOT cause a close of the whole connection but MAY cause a stream error and reset of the stream.

TODO: flesh out this principle for application-level error code assignment in general for known error code values, and specifically all HTTP/3 ones? (Or is there a better approach?)

12.3.2. HTTP/3 WebTransport Streams

WebTransport over HTTP/3 is defined in [I-D.draft-ietf-webtrans-http3].

Popular data that can be sent with server-initiated streams and carried over WebTransport is a good use case for multicast transport because the same server-to-client data can be pushed to many different receivers on a multicast channel.

A QUIC connection using HTTP/3 and WebTransport can use multicast channels to deliver WebTransport server-initiated streams.

However, because the WebTransport Session ID is a client-specific value, the bytes that carry the WebTransport Session ID value within the stream would need to be carried over unicast, since it’s not the same for different clients.
For this situation, note that the Session ID is a variable length integer, and that a variable length integer can be encoded in any size that’s big enough to hold it. In particular, it’s possible to use the largest size of any Session IDs of any of the WebTransport sessions of any clients (or 8 octets, the maximum size for a variable length integer), and that all clients receiving stream data on a channel will need to use the same size for the Session ID so that the rest of the stream data will be at the same offset for every client.

12.3.3. Datagrams

DATAGRAM frames ([RFC9221]) can be carried in multicast channels, and can be a good way to deliver popular content to receivers. Doing so can align well with existing multicast UDP-based applications, since a datagram API in a QUIC application offers similar functionality to a UDP API for sending and receiving packets.

However, at the time of this writing (version -05 of [I-D.draft-ietf-masque-h3-datagram]) multicast channels generally cannot deliver HTTP/3 datagrams, including WebTransport datagrams (version -02 of [I-D.draft-ietf-webtrans-http3]), since the demuxing of WebTransport datagrams uses a Session ID based on a client-specific value (the HTTP/3 Session ID comes from the Stream ID of the client-initiated stream that issued the initial extended CONNECT request).

It is therefore hoped that an extension or revision to WebTransport and HTTP/3 datagrams can be adopted in a future version of their specifications that make it possible to use a server-chosen Session ID value for demuxing WebTransport datagrams (and HTTP/3 datagrams in general).

Such a value could for instance be sent in an HTTP/3 response header, and as long as it is unique within the connection and avoids collision with any client-initiated stream ID values, it could still be used to multiplex data associated with different HTTP/3 traffic and different WebTransport sessions carried on the same connection. Then by choosing the same server-chosen session ID for all the connections, the server would be able to use the same channel to carry the identical complete datagrams, including the server-chosen Session ID, to multiple receivers that the server asks to join the same channel. Such a change could either replace the current client-chosen definition for Session ID in server-to-client datagrams, or could add new HTTP/3 frame types that allow a server-chosen Session ID when the client has advertised support for this extended functionality.
12.4. Graceful Degradation

Clients with multicast QUIC support can stop accepting multicast for a variety of reasons.

Applications like live broadcast-scale video that rely on multicast QUIC may benefit from anticipating that clients might stop using multicast and providing data feeds with similar content that can scale even if many clients stop using multicast, for example by ensuring that a lower-bitrate rendition can still be delivered over unicast to all or most of the clients simultaneously, and ensuring that the server has a way to make the client start using the low-bitrate version when it switches to unicast.

While some existing Adaptive Bitrate video players might have an easy way to provide this, other video players might need specialized logic to provide the server a way to control what bitrate individual clients consume. Although under ideal conditions it may often be possible using features like server push (Section 12.3.1) to use unmodified existing HTTP-based video players with multicast QUIC, in practice it may require extra development at the application level to make a player that robustly delivers a good user experience under variable network conditions, depending on the scalability gains that multicast transport is providing and the Adaptive Bitrate algorithms the player is using.

12.4.1. Circuit Breakers

Operators of multicast QUIC services should consider that some networks may implement circuit breakers such as the one described in [I-D.draft-ietf-mboned-cbacc], or similar network-level safety features that might cut off previously operational multicast transport under certain conditions.

The servers will notice the transport loss from the lack of MC_ACK frames from receivers in a network that cut off multicast transport, but it may be beneficial when possible in a transport cutoff event correlated across many clients to pace the recovery response according to aggregations of the affected clients so that a sudden unicast storm doesn’t overload the network further.

12.5. Server Scalability

Use of QUIC multicast channels can provide large scalability gains, but there still will be significant scaling requirements on server operators to support a large client footprint.
Servers, possibly many of them, still will be required to maintain unicast connections with all the clients and provide for handling MC_ACK frames from the clients, delivering MC_INTEGRITY frames, managing the clients' channel join states, and providing recovery for lost packets.

Further, the use of multicast channels likely requires increased coordination between the different servers, relative to services that operate completely independently.

For large deployments, server implementations will often need to operate on separate devices from the ones generating the multicast channel packets, and will need to be designed accordingly.

12.6. Address Collisions

Multicast channels at the network layer are addressed with a source IP, a destination group IP address, and a destination UDP port.

These offers a number of potential address collision considerations that are worth mentioning:

1. If properties change for the data being used in a channel (for example, new video encoding settings might result in a change to the expected max rate for a video feed), a server might reuse the same network addresses in a new QUIC multicast channel, and might send a join for the new channel and a leave for the old channel to clients that can support the new max rate. If they arrive together, this could be handled by the client without making a change to the IGMP or MLD membership state, as an optimization that can prevent the need for some recovery, or even by reusing the same UDP socket. Doing so does not change any requirements for the channel state management at the QUIC layer, and as long as the situation is transient, should not result in leaving due to Excessive Spurious Traffic even if some packets were reordered or may still be in flight.

2. As described in Section 6 of [RFC4607], link-layer addresses can be linked to the low-order bits of multicast addresses, and may be the same for different group destinations. Collisions in the link-layer addressing, even with traffic that comes from other sources, can cause congestion or receiver CPU load for colliding channels that might be different from that seen with other channels that were delivered with apparently the same network paths.
3. Even though multicast QUIC uses only source-specific multicast, older networks with devices that don’t have IGMPv3 or MLDv2 support can propagate the joins as any-source multicast. If there are active senders sending to that destination, this can cause network congestion and CPU load due to discarding packets from the wrong source, even though at the application layer the UDP socket won’t receive those packets from the wrong source.

4. If different channels use the same (S,G) but different UDP ports, they will share the same multicast forwarding tree in an IP network. This is often useful when the data in the channels are linked, for example if MC_INTEGRITY frames are carried on one channel for packets carried on another channel, because it provides some fate-sharing for the linked data. However, for data that is not so linked, it would generally be a disadvantage to share the (S,G) because the network link of any receiver joined to one of those channels but not the other would receive both packets and throw away the data for the un-joined port, causing extra congestion and CPU load for the receiving device.

13. Security Considerations

(Authors comment: Mostly incorporate [I-D.draft-krose-multicast-security]. Anything else?

ej.g. if a different legitimate quic connection says someone else’s quic multicast stream is theirs, that’s maybe a problem worth protecting against. Maybe we need a periodic asymmetric challenge?

I’m thinking send a public key on the multicast channel and in the unicast channels send an individualized MAC signed with the private key and verify it with the public key, so that in addition to validating that the unicast server knows the contents of the multicast packets via the hashes it supplies, the multicast stream provides a way for the client to validate that the unicast stream is authorized to use it for data transport via proof they know the private key corresponding to the public key that arrived on the multicast channel. Note this doesn’t prevent unauthorized receipt of multicast data packets, but does prevent a quic server from lying when claiming a multicast data channel belongs to it, preventing legit receivers from consuming it.

alternatively, can the multicast channel just periodically say what domain name is expected for the quic connection and get the same crypto guarantee of a proper sender via the domain’s cert, which was already checked on the unicast channel?)
14. IANA Considerations

TODO: MC_EXTENSION_ERROR error code

TODO: lots

15. References

15.1. Normative References

[I-D.draft-ietf-mboned-ambi]
Holland, J. and K. Rose, "Asymmetric Manifest Based Integrity", Work in Progress, Internet-Draft, draft-ietf-mboned-ambi-03, 7 March 2022,

[I-D.draft-ietf-mboned-cbacc]
Holland, J., "Circuit Breaker Assisted Congestion Control", Work in Progress, Internet-Draft, draft-ietf-mboned-cbacc-04, 7 March 2022,

[I-D.draft-ietf-quic-multipath]
Liu, Y., Ma, Y., Coninck, Q. D., Bonaventure, O., Huiitema, C., and M. Kuehlewind, "Multipath Extension for QUIC", Work in Progress, Internet-Draft, draft-ietf-quic-multipath-02, 11 July 2022,

[I-D.draft-krose-multicast-security]
Rose, K. and J. Holland, "Security and Privacy Considerations for Multicast Transports", Work in Progress, Internet-Draft, draft-krose-multicast-security-02, 31 January 2022,

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,

DOI 10.17487/RFC8085, March 2017,
15.2. Informative References

[I-D.draft-ietf-masque-h3-datagram]

[I-D.draft-ietf-webtrans-http3]

[MERKLE]


Acknowledgments

TODO acknowledge.

Authors’ Addresses

Jake Holland
Akamai Technologies, Inc.
Email: jakeholland.net@gmail.com

Lucas Pardue
Email: lucaspardue.24.7@gmail.com

Max Franke
TU Berlin
Email: mfranke@inet.tu-berlin.de