Workgroup: QUIC Working Group
Internet-Draft:
draft-jholland-quic-multicast-02
Published: 11 July 2022
Intended Status: Experimental
Expires: 12 January 2023
Authors: J. Holland
Akamai Technologies, Inc.
L. Pardue
M. Franke
TU Berlin

Multicast Extension for QUIC

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

Table of Contents

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.

Term	Definition
SSM	Source-specific multicast, as described in [RFC4607]
ASM	Any-source multicast, as distinguished from SSM in [RFC4607]
(S,G)	A tuple of IP addresses (Source IP, Group IP) identifying a source-specific multicast channel as described in [RFC4607]

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

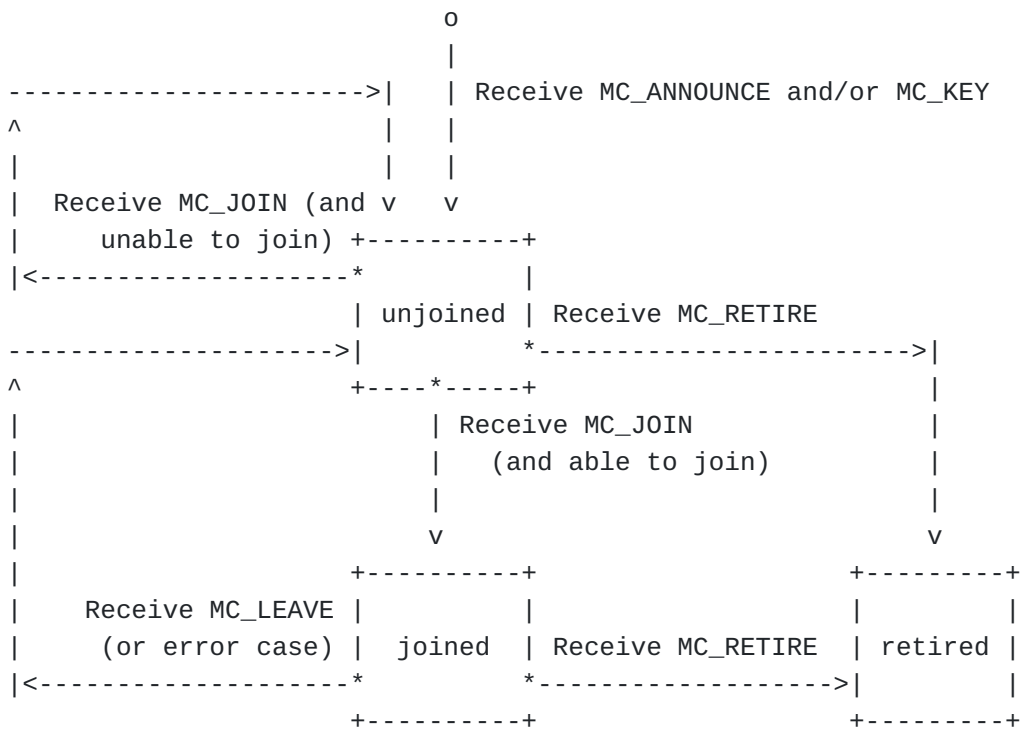
The client tells its server about some restrictions on resources that it is capable of processing with the initial values in the multicast_client_params transport parameter (Section 3) and later can update these limits with MC_LIMITS Section 10.7 frames. Servers ensure the set of channels the client 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 clients 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.

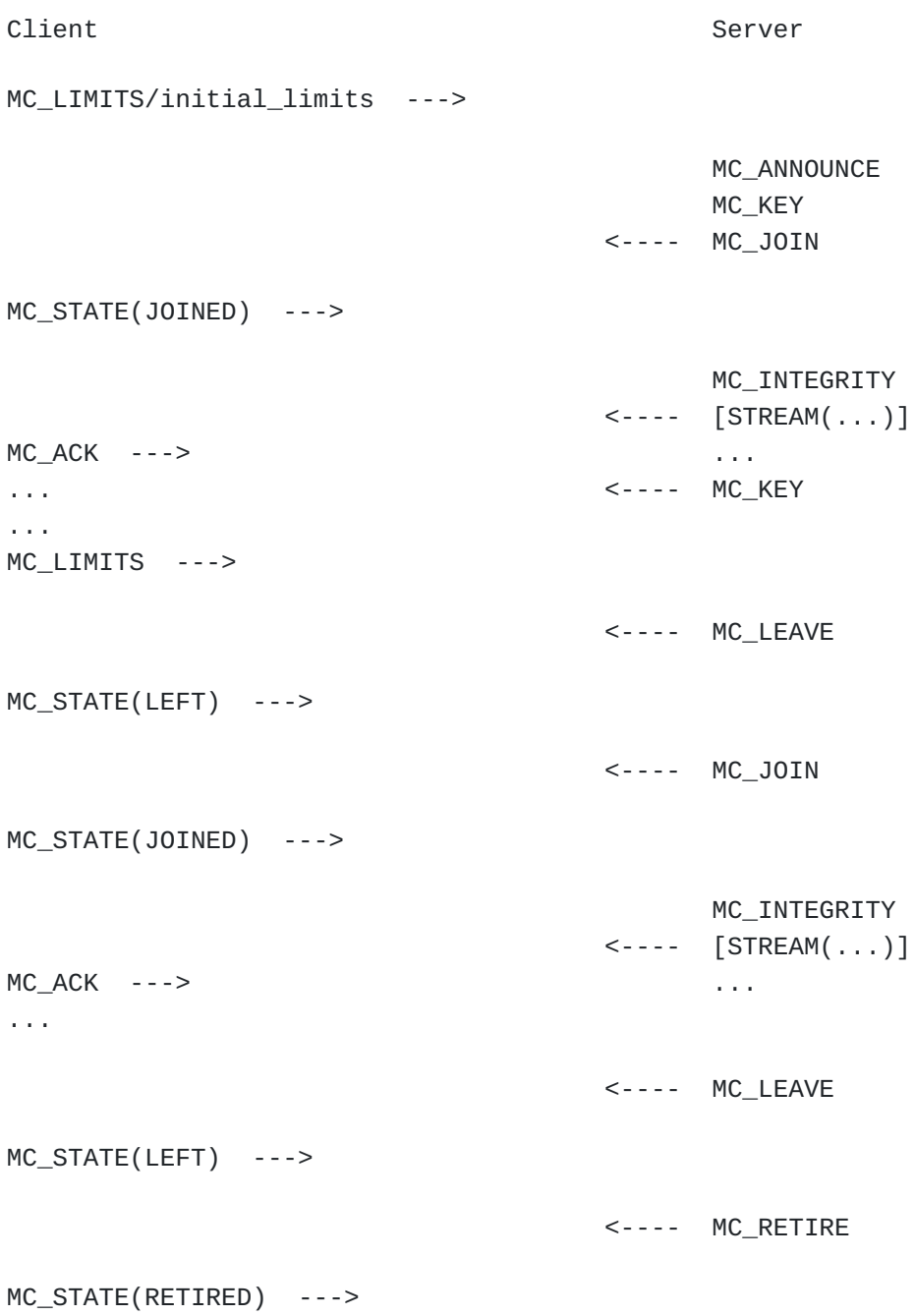


Figure 3: Example flow of frames for a channel. Frames in square brackets are sent over multicast.

TODO: incorporate server-side state diagram and explanation, latest proposed sketch at https://github.com/GrumpyOldTroll/draft-jholland-quic-multicast/issues/62

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), 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), 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:

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

o https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-18

o(text-only): 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 can not 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 cases 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?

e.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, <https://datatracker.ietf.org/doc/html/draft-ietf-mboned-ambi-03>.

[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, <https://datatracker.ietf.org/doc/html/draft-ietf-mboned-cbacc-04>.

[I-D.draft-ietf-quic-multipath] Liu, Y., Ma, Y., Coninck, Q. D., Bonaventure, O., Huitema, C., and M. Kuehlewind, "Multipath Extension for QUIC", Work in Progress, Internet-Draft, draft-ietf-quic-multipath-02, 11 July 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-multipath-02>.

[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, <https://datatracker.ietf.org/doc/html/draft-krose-multicast-security-02>.

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

[RFC8085] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017, <https://www.rfc-editor.org/rfc/rfc8085>.

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

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/rfc/rfc8446>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI

10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>.

[RFC9001] Thomson, M., Ed. and S. Turner, Ed., "Using TLS to Secure QUIC", RFC 9001, DOI 10.17487/RFC9001, May 2021, <https://www.rfc-editor.org/rfc/rfc9001>.

[RFC9002] Iyengar, J., Ed. and I. Swett, Ed., "QUIC Loss Detection and Congestion Control", RFC 9002, DOI 10.17487/RFC9002, May 2021, <https://www.rfc-editor.org/rfc/rfc9002>.

[RFC9221] Pauly, T., Kinnear, E., and D. Schinazi, "An Unreliable Datagram Extension to QUIC", RFC 9221, DOI 10.17487/RFC9221, March 2022, <https://www.rfc-editor.org/rfc/rfc9221>.

15.2. Informative References

[I-D.draft-ietf-masque-h3-datagram] Schinazi, D. and L. Pardue, "HTTP Datagrams and the Capsule Protocol", Work in Progress, Internet-Draft, draft-ietf-masque-h3-datagram-11, 17 June 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-masque-h3-datagram-11>.

[I-D.draft-ietf-webtrans-http3] Frindell, A., Kinnear, E., and V. Vasiliev, "WebTransport over HTTP/3", Work in Progress, Internet-Draft, draft-ietf-webtrans-http3-03, 6 July 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-webtrans-http3-03>.

[MERKLE] Merkle, R., "Secrecy, Authentication, and Public Key Systems", Computer Science Series, UMI Research Press, ISBN: 9780835713849 , 1983.

[RFC4607] Holbrook, H. and B. Cain, "Source-Specific Multicast for IP", RFC 4607, DOI 10.17487/RFC4607, August 2006, <https://www.rfc-editor.org/rfc/rfc4607>.

[RFC6363] Watson, M., Begen, A., and V. Roca, "Forward Error Correction (FEC) Framework", RFC 6363, DOI 10.17487/RFC6363, October 2011, <https://www.rfc-editor.org/rfc/rfc6363>.

[RFC6726] Paila, T., Walsh, R., Luby, M., Roca, V., and R. Lehtonen, "FLUTE - File Delivery over Unidirectional Transport", RFC 6726, DOI 10.17487/RFC6726, November 2012, <https://www.rfc-editor.org/rfc/rfc6726>.

[RFC6968] Roca, V. and B. Adamson, "FCAST: Object Delivery for the Asynchronous Layered Coding (ALC) and NACK-Oriented

Reliable Multicast (NORM) Protocols", RFC 6968, DOI 10.17487/RFC6968, July 2013, <https://www.rfc-editor.org/rfc/rfc6968>.

[RFC9114] Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <https://www.rfc-editor.org/rfc/rfc9114>.

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