QUIC
Internet-Draft
Intended status: Standards Track
Expires: 4 October 2026

K. Oku
Fastly
L. Pardue
Cloudflare
J. Iyengar
Netflix
E. Kinnear
Apple

2 April 2026

# QMux
## draft-ietf-quic-qmux-01

## Abstract

This document specifies QMux version 1. QMux version 1 provides, over bi-directional streams such as TLS, the same set of stream and datagram operations that applications rely upon in QUIC version 1.

## 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/quicwg/qmux.

## 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 4 October 2026.

Table of Contents

1. Introduction . . . . . 3
2. Conventions and Definitions . . . . . 4
3. The Protocol . . . . . 4
3.1. Properties of Underlying Transport . . . . . 4
3.2. QMux Records . . . . . 5
4. QUIC Frames . . . . . 6
4.1. Ordering of STREAM Frames . . . . . 7
4.2. QX_TRANSPORT_PARAMETERS Frames . . . . . 8
4.3. QX_PING Frames . . . . . 9
5. Transport Parameters . . . . . 10
5.1. Permitted and Forbidden Transport Parameters . . . . . 10
5.2. max_record_size Transport Parameter . . . . . 11
6. Forward Progress and Flow Control . . . . . 11
7. Closing the Connection . . . . . 11
8. Using TLS . . . . . 12
8.1. Protocol Negotiation . . . . . 12
8.2. Using 0-RTT . . . . . 13
9. Extensions . . . . . 13
9.1. Unreliable Datagram Extension . . . . . 13
10. Version Agility . . . . . 13
10.1. Negotiation Using ALPN . . . . . 14
10.2. In-band Upgrade . . . . . 14
11. Implementation Considerations . . . . . 15
12. Security Considerations . . . . . 15
13. IANA Considerations . . . . . 15
13.1. QUIC Frame Types . . . . . 15
13.2. QUIC Transport Parameters . . . . . 16
14. References . . . . . 16
14.1. Normative References . . . . . 16
14.2. Informative References . . . . . 17
Acknowledgments . . . . . 18
Authors' Addresses . . . . . 18

## 1. Introduction

QUIC version 1 [QUIC] is a bi-directional, authenticated transport-layer protocol built on top of UDP [UDP]. The protocol provides multiplexed flow-controlled streams without head-of-line blocking as a core service. It also offers low-latency connection establishment and efficient loss recovery.

However, there are downsides to QUIC.

One downside is that QUIC, being based on UDP, is not as universally accessible as TCP [TCP], due to occasionally being blocked by middleboxes.

Another downside is that QUIC is computationally more expensive compared to TLS [TLS13] over TCP. This increased cost is partly because QUIC encrypts each packet, which is smaller than the encryption unit of TLS, leading to more overhead, and partly because UDP is less optimized within computing infrastructures.

Due to these limitations, applications are often served using both QUIC and TCP. QUIC is employed to provide the optimal user experience, while TCP acts as a fallback for ensuring network reachability and computational efficiency as needed.

One such example is HTTP, which has different bindings for QUIC (HTTP/3 [HTTP3]) and TCP (HTTP/2 [HTTP2]). Recently, security concerns have prompted proposals to revise HTTP/2 ([h2-stream-limits]), which has sparked discussions about the costs of maintaining multiple HTTP versions.

Another example is WebTransport, a superset of HTTP. Because HTTP has different bindings for QUIC and TCP, WebTransport defines its own extensions for the two HTTP variants ([webtrans-h3], [webtrans-h2]).

To reduce or eliminate the costs associated with duplicated efforts in providing services on top of both transport protocols, this document specifies a polyfill that allows application protocols built on QUIC to run on transport protocols that provide single bi-directional, byte-oriented stream such as TCP or TLS.

The specified polyfill provides a compatibility layer for the set of operations (i.e., API) required by QUIC, as specified in Section 2.4 and Section 5.3 of [QUIC].

## 2. 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.

## 3. The Protocol

QMux can be used on any transport that provides a bi-directional, byte-oriented stream that is ordered and reliable; for details, see Section 3.1.

QMux transfers one or more QUIC frames within a QMux record; see Section 3.2. Neither QUIC frames nor QMux records are encrypted in this protocol; it is the task of the transport (e.g., TLS) to provide confidentiality and integrity.

QUIC packet headers are not used.

For exchanging the transport parameters, a new frame called QX_TRANSPORT_PARAMETERS frame is defined.

### 3.1. Properties of Underlying Transport

QMux is designed to work on top of transport layer protocols that provide the following capabilities:

In-order delivery of bytes in both direction: Underlying transport provides a byte-oriented and bi-directional stream that deliver the bytes in order; i.e., bytes that were sent in one order become available to the receiving side in the same order.

Guaranteed delivery: If the transport runs on top of a lossy network, that transport recovers the bytes lost; e.g., by retransmitting them. This requires buffering and reassembly, in order to achieve the first bullet point (in-order delivery).

Congestion control: When used on a shared network, the transport is congestion controlled. Implementations of QMux simply write outgoing frames to the transport when that transport permits.

Authentication, confidentiality, and integrity protection: Unless used upon endpoints between which tampering or monitoring is a non-concern, the transport provides peer authentication, confidentially, and integrity protection.

Application Protocol Negotiation: To avoid cross-protocol confusion, the underlying transport provides a mechanism for endpoints to agree on the protocols in use. Without such a mechanism, QMux and application protocols built on top of QMux can only be used between endpoints that have out-of-band agreement on those protocols.

TLS over TCP, combined with the Application-Layer Protocol Negotiation extension (ALPN) [ALPN], provides all these capabilities.

UNIX sockets are an example that provide in-order and guaranteed delivery only. Congestion control is not employed, as UNIX sockets do not face a shared bottleneck. Confidentiality and integrity protection are deemed unnecessary in environments where the operating system is trusted. Agreement on the application protocol can be achieved by using different listening sockets.

### 3.2. QMux Records

QMux Records are formatted as shown in Figure 1.

```
QMux Record {
  Size (i),
  Frames (..),
}
```

Figure 1: QMux Record Format

QMux Records contain the following fields:

Size: A variable-length integer specifying the length in bytes of the Frames field. Note that this length does not include the Size field itself.

Frames: A byte sequence that contains one or more QUIC frames.

Each QMux Record is self-delimiting. On a byte stream transport, endpoints parse QMux Records by first parsing Size and then reading exactly that many bytes as Frames. The bytes in Frames are interpreted as a contiguous series of QUIC frames encoded using QUIC version 1 framing.

As with QUIC packet payloads (Section 12.4 of [QUIC]), frames are complete units: a frame MUST fit entirely within a single QMux Record and MUST NOT span multiple QMux Records.

Senders can choose record boundaries freely, subject to the max_record_size transport parameter (Section 5.2). Receivers process frames within each record, using the record boundary as the payload boundary for frames that omit an explicit length.

If the end of the Frames field is not aligned to a frame boundary (that is, if the final frame in the record is truncated), the receiver MUST close the connection with an error of type FRAME_ENCODING_ERROR.

# 4. QUIC Frames

In QMux, the following QUIC frames can be used, as if they were sent or received in the application packet number space:

* PADDING
* RESET_STREAM
* STOP_SENDING
* STREAM
* MAX_DATA
* MAX_STREAM_DATA
* MAX_STREAMS
* DATA_BLOCKED
* STREAM_DATA_BLOCKED
* STREAMS_BLOCKED
* CONNECTION_CLOSE

The frame formats are identical to those in QUIC version 1. Likewise, the meaning and requirements for the use of these frames are consistent with QUIC version 1, with the exception to the specific changes made to the STREAM frames, as detailed in Section 4.1.

Use of other frames defined in QUIC version 1 is prohibited for various reasons.

Frames related to the cryptographic handshake are not used because an underlying security layer can provide equivalent features. Use of frames that communicate Connection IDs and those related to path migration is forbidden.

QMux relies on the underlying transport for reliable delivery and therefore does not use ACK frames. QMux stacks do not track delivery or retransmit lost data or frames. For the stream state machinery defined in Section 3 of [QUIC], references to acknowledgment are interpreted as though acknowledgments occurs as soon as data is passed to the underlying transport. As in QUIC version 1, applications cannot assume that the peer application has consumed data based solely on transport events.

The full list of prohibited frames is:

* PING
* ACK
* CRYPTO
* NEW_TOKEN
* NEW_CONNECTION_ID
* RETIRE_CONNECTION_ID
* PATH_CHALLENGE
* PATH_REPONSE
* HANDSHAKE_DONE

Endpoints MUST NOT send prohibited frames. If an endpoint receives one it MUST close the connection with an error of type FRAME_ENCODING_ERROR.

### 4.1. Ordering of STREAM Frames

In QUIC version 1, the order in which STREAM frames are sent or received is not guaranteed, because packets can be lost and frames can be retransmitted in different packets. In contrast, QMux, which runs over an ordered and reliable byte stream transport, requires STREAM frames for each QUIC stream to be sent in order: i.e., for each QUIC stream being sent, senders MUST send the stream payload in order.

This change eliminates the need for implementations to buffer and reassemble the stream payload. As a result, the payload being received can be directly passed to the application as it is read from the transport. This efficiency is due to the underlying transport's guarantee of in-order delivery.

When receiving a STREAM frame that carries a payload not immediately following the payload of the previous STREAM frame for the same Stream ID, receivers MUST close the connection with an error of type PROTOCOL_VIOLATION_ERROR.

These changes do not impact the senders' capability to interleave STREAM frames from multiple streams.

### 4.2. QX_TRANSPORT_PARAMETERS Frames

In QMux, transport parameters are exchanged as frames.

QX_TRANSPORT_PARAMETERS frames are formatted as shown in Figure 2.

```
QX_TRANSPORT_PARAMETERS Frame {
  Type (i) = 0x3f5153300d0a0d0a,
  Length (i),
  Transport Parameters (..),
}
```

Figure 2: QX_TRANSPORT_PARAMETERS Frame Format

QX_TRANSPORT_PARAMETERS frames contain the following fields:

Length: A variable-length integer specifying the length of the transport parameters field in this QX_TRANSPORT_PARAMETERS frame.

Transport Parameters: The transport parameters. The encoding of the payload is as defined in Section 18 of [QUIC].

The QX_TRANSPORT_PARAMETERS frame is the first frame sent by endpoints. To allow peers to open streams and start sending data as early as possible, endpoints MUST send the QX_TRANSPORT_PARAMETERS frame as soon as the underlying transport becomes available for sending. Neither endpoint needs to wait for the peer's transport parameters before sending its own, as transport parameters are a unilateral declaration of an endpoint's capabilities (Section 7.4 of [QUIC]).

Except when sending 0-RTT data using remembered transport parameters as described in Section 7.4.1 of [QUIC], endpoints MUST NOT send frames whose use depends on peer transport parameters until the

peer's QX_TRANSPORT_PARAMETERS frame has been received and processed. This can delay use of peer-advertised flow control credit and can therefore block sending stream data before peer transport parameters arrive. When QMux runs over TLS 1.3, this does not necessarily add a full round trip for clients on a full handshake. Servers can send the QX_TRANSPORT_PARAMETERS frame immediately after the server's Finished message, and clients can receive and process the transport parameters as soon as they obtain the keys needed to process application data.

The QX_TRANSPORT_PARAMETERS frame MUST only be sent as the first frame. If the first frame being received by an endpoint is not a QX_TRANSPORT_PARAMETERS frame, or if a QX_TRANSPORT_PARAMETERS frame is received other than as the first frame, the endpoint MUST close the connection with an error of type TRANSPORT_PARAMETER_ERROR.

The frame type (0x3f5153300d0a0d0a; "\xffQMX\r\n\r\n" on wire) has been chosen so that it can be used to disambiguate QMux from HTTP/1.1 [HTTP1] and HTTP/2.

### 4.3. QX_PING Frames

In QMux, QX_PING frames allow endpoints to test peer reachability above the underlying transport.

QX_PING frames are formatted as shown in Figure 3.

```
QX_PING Frame {
  Type (i) = 0x348c67529ef8c7bd..0x348c67529ef8c7be,
  Sequence Number (i),
}
```

Figure 3: QX_PING Frame Format

Type 0x348c67529ef8c7bd is used for sending a ping (i.e., request the peer to respond). Type 0x348c67529ef8c7be is used in response.

QX_PING frames contain the following fields:

Sequence Number: A variable-length integer used to identify the ping.

When sending QX_PING frames of type 0x348c67529ef8c7bd, endpoints MUST send monotonically increasing values in the Sequence Number field, since that allows the endpoints to identify to which ping the peer has responded.

When sending QX_PING frames of type 0x348c67529ef8c7be in response, endpoints MUST echo the Sequence Number that they received.

When receiving multiple QX_PING frames of type 0x348c67529ef8c7bd before having the chance to respond, an endpoint MAY only respond with one QX_PING frame of type 0x348c67529ef8c7be carrying the largest Sequence Number that the endpoint has received.

# 5. Transport Parameters

QMux uses a subset of transport parameters defined in QUIC version 1. Also, one new transport parameter specific to QMux is defined.

## 5.1. Permitted and Forbidden Transport Parameters

In QMux, use of the following transport parameters is allowed.

* max_idle_timeout
* initial_max_data
* initial_max_stream_data_bidi_local
* initial_max_stream_data_bidi_remote
* initial_max_stream_data_uni
* initial_max_streams_bidi
* initial_max_streams_uni

The definition of these transport parameters are unchanged.

Use of other transport parameters defined in QUIC version 1 is prohibited. When an endpoint receives one of the prohibited transport parameters, the endpoint MUST close the connection with an error of type TRANSPORT_PARAMETER_ERROR.

Endpoints MUST NOT send transport parameters that extend QUIC version 1, unless they are specified to be compatible with QMux.

When receiving transport parameters not defined in QUIC version 1, receivers MUST ignore them unless they are specified to be usable on QMux.

### 5.2. max_record_size Transport Parameter

The max_record_size transport parameter (0x0571c59429cd0845) is a variable-length integer specifying the maximum value of the Size field of a QMux Record that the peer can send, in the unit of bytes.

The initial value of the max_record_size transport parameter is 16382. This value allows a sender to construct a 16KB QMux Record by using a 2-byte Size field and a 16382-byte Frames field, aligning with the default capacity of a full-sized TLS record.

By sending the transport parameter, the maximum record size can only be increased. When receiving a value below the initial value, receivers MUST close the connection with an error of type TRANSPORT_PARAMETER_ERROR.

Endpoints MUST NOT send QMux Records with a Frames field that exceeds the maximum declared by the peer.

When receiving a QMux Record with a Frames field that exceeds the declared maximum, receivers MUST close the connection with an error of type FRAME_ENCODING_ERROR.

## 6. Forward Progress and Flow Control

To avoid deadlock due to flow control in the underlying transport, endpoints MUST continue reading from the underlying transport even when delivery of STREAM data to the application is temporarily blocked.

Endpoints MUST NOT couple reads from the underlying transport to application reads on any single QUIC stream, as doing so can prevent processing of frames required for connection progress.

Continuing to read does not imply unbounded buffering of STREAM data, as the amount of stream data a peer can send is limited by flow control (Section 4 of [QUIC]). For DATAGRAM frames, endpoints MAY drop received datagrams when they cannot be promptly delivered to the application.

## 7. Closing the Connection

As is with QUIC version 1, a connection can be closed either by a CONNECTION_CLOSE frame or by an idle timeout.

Unlike QUIC version 1, idle timeout handling does not rely on ACK frames. Endpoints reset the idle timer when sending or receiving QMux frames. When no other traffic is available, QX_PING frames can be used to elicit a peer response and keep the connection active.

Unlike QUIC version 1, there is no draining period; once an endpoint sends or receives the CONNECTION_CLOSE frame or reaches the idle timeout, all the resources allocated for the Service are freed and the underlying transport is closed immediately.

## 8. Using TLS

When QMux is used over TLS, TLS provides capabilities in addition to confidentiality and integrity protection.

### 8.1. Protocol Negotiation

As in QUIC Section 8.1 of [QUIC-TLS], when running an application protocol over QMux over TLS, endpoints MUST use ALPN [ALPN] to agree on an application protocol, unless another mechanism is used for agreeing on an application protocol.

ALPN protocol identifiers identify the application protocol in use. Application protocols that use QMux over TLS MUST designate their ALPN identifier and specify that they use QMux version 1. The identifier for a mapping to QMux MUST be different from the mapping of the same protocol to QUIC, to retain compatibility with Service Binding and Parameter Specification via the DNS [SVCB].

When using ALPN, endpoints MUST abort the TLS handshake with a no_application_protocol TLS alert (Section 3.2 of [ALPN]) if an application protocol is not negotiated. While ALPN only requires that servers use this alert, QMux clients MUST also abort the handshake when ALPN negotiation fails.

QMux is not itself an application protocol and does not have an ALPN identifier.

TO BE REMOVED BEFORE PUBLICATION: During the development of QMux, its wire format might change. Therefore, when testing interoperability of application protocols using a draft version of QMux, applications should specify, for each ALPN identifier they define, which draft version of QMux is used. As an example, an ALPN identifier "myapp-12qx" could identify version 12 of "myapp" over TCP and QMux, identifying the use of a specific QMux draft version in its specification. the use of QMux draft-05.

### 8.2. Using 0-RTT

TLS 1.3 introduced the concept of early data (also known as 0-RTT data).

When using QMux over TLS that supports early data, clients MAY use early data when resuming a connection, by reusing certain transport parameters as defined in Section 7.4.1 of [QUIC].

Similarly, when accepting early data, servers MUST send transport parameters that comply with the restrictions in Section 7.4.1 of [QUIC]. This preserves QUIC's 0-RTT compatibility model and avoids requiring an additional round trip to learn peer transport parameters on resumed connections.

## 9. Extensions

Not all the extensions of QUIC version 1 can be used. Each extension have to define its mapping for QMux, or explicitly allow the use; see Section 5.1.

As is the case with QUIC version 1, use of extension frames have to be negotiated before use; see Section 19.21 of [QUIC].

This specification defines the mapping of the Unreliable Datagram Extension.

### 9.1. Unreliable Datagram Extension

The use of the Unreliable Datagram Extension [QUIC_DATAGRAM] is permitted. The encoding and semantics of the Unreliable Datagram Extension remain unchanged, and the use of the extension is negotiated via transport parameters.

As discussed in Section 5 of [QUIC_DATAGRAM], senders can drop DATAGRAM frames if the transport is blocked by flow or congestion control.

## 10. Version Agility

As new versions of QUIC are specified, there may be a desire to define their reliable-byte-stream counterparts as different versions of QMux, and to provide ways of negotiating the version to be used.

QUIC establishes connections using packets carrying an explicit version number. Using that field, Version-Independent Properties of QUIC [QUIC-INVARIANTS] defines a version negotiation mechanism that involves a retry. Compatible Version Negotiation for QUIC [QUIC-CVN] defines another negotiation mechanism for switching between compatible versions during connection establishment without retrying.

By contrast, QMux does not establish connections by itself; the connections are set up by the underlying substrate, and QMux exchanges only the transport parameters after they are established.

Due to these differences, the negotiation mechanisms used by QUIC and QMux will differ.

This section explores some options that future versions of QMux might employ for version negotiation and upgrade.

### 10.1. Negotiation Using ALPN

When a new QUIC version that provides a different interface to applications is specified, application protocols developed for that version might be assigned a new identifier for the TLS Application-Layer Protocol Negotiation (ALPN) extension [ALPN].

Similarly, when TLS is the underlying transport, application protocols built on top of the QMux counterparts of such QUIC versions can rely on ALPN to negotiate both the application protocol and the underlying QMux version (Section 8.1).

When TLS is not the underlying transport, endpoints can use the first 8 bytes exchanged on the transport (i.e., the type field of the QX_TRANSPORT_PARAMETERS frame in the encoded form) to identify whether QMux is in use.

[TODO: discuss how endpoints should behave when the first 8 bytes received are not QX_TRANSPORT_PARAMETERS.]

### 10.2. In-band Upgrade

A new version of QMux might first start communication using QMux version 1 and then switch versions in-band during the session. The advantage of this approach is that, even when TLS is not in use, no additional round-trip is incurred for version negotiation.

While QMux version 1 does not specify a concrete method, new versions might use the version_information transport parameter (Section 3 of [QUIC-CVN]) to discover supported versions and coordinate the switch.

## 11. Implementation Considerations

Similar to HTTP/3 with Extensible Priorities [HTTP_PRIORITY], application protocols using QUIC may employ stream multiplexing along with a system to tune the delivery sequence of QUIC streams.

To alternate between QUIC streams of varying priorities in a timely manner, it is advisable for QMux implementations to avoid creating deep buffers holding QUIC frames. Instead, endpoints should wait for the transport layer to be ready for writing. Upon becoming writable, they should write QUIC frames according to the latest prioritization signals.

Additionally, implementations may consider monitoring or adjusting the flow and congestion control parameters of the underlying transport. This approach aims to minimize data buffering within the transport layer before transmission. However, improper adjustment of these parameters could potentially lead to lower throughput.

## 12. Security Considerations

Failure to follow the forward-progress requirements in Section 6 can lead to deadlock and can be exploited for resource-exhaustion attacks.

## 13. IANA Considerations

This document defines new frame types and a transport parameter for use with QUIC. IANA is requested to register the following values in the registries under https://www.iana.org/assignments/quic (https://www.iana.org/assignments/quic).

The codepoints listed below, unless otherwise stated, were selected using the deterministic method described at https://martinthomson.github.io/quic-pick/ (https://martinthomson.github.io/quic-pick/), with seeds of the form draft-ietf-quic-qmux-NN_<fieldtype>. Upon publication as an RFC, IANA is requested to replace provisional codepoints with permanent assignments from the standards-action range.

### 13.1. QUIC Frame Types

The following entries should be added to the "QUIC Frame Types" registry.

| Value | Frame Type Name | Status | Specification |
|---|---|---|---|
| 0x3f5153300d0a0d0a | QX_TRANSPORT_PARAMETERS | provisional | Figure 2 |
| 0x348c67529ef8c7bd - 0x348c67529ef8c7be | QX_PING | provisional | Figure 3 |

Table 1

The value 0x3f5153300d0a0d0a for QX_TRANSPORT_PARAMETERS was chosen deliberately to function as a protocol magic number see (Figure 2).

### 13.2. QUIC Transport Parameters

The following entry should be added to the "QUIC Transport Parameters" registry.

| Value | Parameter Name | Status | Specification |
|---|---|---|---|
| 0x0571c59429cd0845 | max_record_size | provisional | Section 5.2 |

Table 2

## 14. References

### 14.1. Normative References

[ALPN] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <https://www.rfc-editor.org/rfc/rfc7301>.

[QUIC] 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>.

[QUIC_DATAGRAM] 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>.

[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>.

[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>.

[TLS13] 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>.

### 14.2. Informative References

[h2-stream-limits] Thomson, M. and L. Pardue, "Using HTTP/3 Stream Limits in HTTP/2", Work in Progress, Internet-Draft, draft-thomson-httpbis-h2-stream-limits-00, 6 November 2023, <https://datatracker.ietf.org/doc/html/draft-thomson-httpbis-h2-stream-limits-00>.

[HTTP1] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1", STD 99, RFC 9112, DOI 10.17487/RFC9112, June 2022, <https://www.rfc-editor.org/rfc/rfc9112>.

[HTTP2] Thomson, M., Ed. and C. Benfield, Ed., "HTTP/2", RFC 9113, DOI 10.17487/RFC9113, June 2022, <https://www.rfc-editor.org/rfc/rfc9113>.

[HTTP3] Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <https://www.rfc-editor.org/rfc/rfc9114>.

[HTTP_PRIORITY] Oku, K. and L. Pardue, "Extensible Prioritization Scheme for HTTP", RFC 9218, DOI 10.17487/RFC9218, June 2022, <https://www.rfc-editor.org/rfc/rfc9218>.

[QUIC-CVN] Schinazi, D. and E. Rescorla, "Compatible Version Negotiation for QUIC", RFC 9368, DOI 10.17487/RFC9368, May 2023, <https://www.rfc-editor.org/rfc/rfc9368>.

[QUIC-INVARIANTS] Thomson, M., "Version-Independent Properties of QUIC", RFC 8999, DOI 10.17487/RFC8999, May 2021, <https://www.rfc-editor.org/rfc/rfc8999>.

[QUIC-TLS] 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>.

[SVCB] Schwartz, B., Bishop, M., and E. Nygren, "Service Binding and Parameter Specification via the DNS (SVCB and HTTPS Resource Records)", RFC 9460, DOI 10.17487/RFC9460, November 2023, <https://www.rfc-editor.org/rfc/rfc9460>.

[TCP] Eddy, W., Ed., "Transmission Control Protocol (TCP)", STD 7, RFC 9293, DOI 10.17487/RFC9293, August 2022, <https://www.rfc-editor.org/rfc/rfc9293>.

[UDP] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/rfc/rfc768>.

[webtrans-h2] Frindell, A., Kinnear, E., Pauly, T., Thomson, M., Vasiliev, V., and W. Xie, "WebTransport over HTTP/2", Work in Progress, Internet-Draft, draft-ietf-webtrans-http2-14, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-webtrans-http2-14>.

[webtrans-h3] Frindell, A., Kinnear, E., and V. Vasiliev, "WebTransport over HTTP/3", Work in Progress, Internet-Draft, draft-ietf-webtrans-http3-15, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-webtrans-http3-15>.

## Acknowledgments

TODO acknowledge.

## Authors' Addresses

Kazuho Oku
Fastly
Email: kazuhooku@gmail.com

Lucas Pardue
Cloudflare
Email: lucas@lucaspardue.com

Jana Iyengar
Netflix
Email: jri.ietf@gmail.com

Eric Kinnear
Apple
Email: ekinnear@apple.com