Workgroup: webtrans
Internet-Draft: draft-ietf-webtrans-http2-03
Published: 7 March 2022
Intended Status: Standards Track
Expires: 8 September 2022
Authors: A. Frindell (Facebook Inc.)
E. Kinnear (Apple Inc.)
T. Pauly (Apple Inc.)
M. Thomson (Mozilla)
V. Vasiliev (Google)
G. Xie (Facebook Inc.)

WebTransport using HTTP/2

Abstract

WebTransport defines a set of low-level communications features designed for client-server interactions that are initiated by Web clients. This document describes a protocol that can provide many of the capabilities of WebTransport over HTTP/2. This protocol enables the use of WebTransport when a UDP-based protocol is not available.

Note to Readers

Discussion of this draft takes place on the WebTransport mailing list (webtransport@ietf.org), which is archived at https://mailarchive.ietf.org/arch/search/?email_list=webtransport.

The repository tracking the issues for this draft can be found at https://github.com/ietf-wg-webtrans/draft-webtransport-http2. The web API draft corresponding to this document can be found at https://w3c.github.io/webtransport/.

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 8 September 2022.

Copyright Notice

Table of Contents

1. Introduction

WebTransport [OVERVIEW] is designed to provide generic communication capabilities to Web clients that use HTTP/3 [HTTP3]. The HTTP/3 WebTransport protocol [WEBTRANSPORT-H3] allows Web clients to use QUIC [QUIC] features such as streams or datagrams [DATAGRAM]. However, there are some environments where QUIC cannot be deployed.

This document defines a protocol that provides all of the core functions of WebTransport using HTTP semantics. This includes unidirectional streams, bidirectional streams, and datagrams.

By relying only on generic HTTP semantics, this protocol might allow deployment using any HTTP version. However, this document only defines negotiation for HTTP/2 [H2] as the current most common TCP-based fallback to HTTP/3.

1.1. Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document follows terminology defined in Section 1.2 of [OVERVIEW]. Note that this document distinguishes between a WebTransport server and an HTTP/2 server. An HTTP/2 server is the server that terminates HTTP/2 connections; a WebTransport server is an application that accepts WebTransport sessions, which can be accessed using HTTP/2 and this protocol.

2. Protocol Overview

WebTransport servers are identified by an HTTPS URI as defined in Section 4.2.2 of [HTTP].

When an HTTP/2 connection is established, both the client and server have to send a SETTINGS_ENABLE_WEBTRANSPORT setting in order to indicate that they both support WebTransport over HTTP/2.

A client initiates a WebTransport session by sending an extended CONNECT request [RFC8441]. If the server accepts the request, a WebTransport session is established. The stream that carries the CONNECT request is used to exchange bidirectional data for the session. This stream will be referred to as a *CONNECT stream*. The

stream ID of a CONNECT stream, which will be referrred to as a *Session ID*, is used to uniquely identify a given WebTransport session within the connection.

After the session is established, endpoints exchange *WebTransport frames* using the bidirectional CONNECT stream. Within this stream, *WebTransport streams* and *WebTransport datagrams* are multiplexed. In HTTP/2, WebTransport frames are carried in HTTP/2 DATA frames. Multiple independent WebTransport sessions can share a connection if the HTTP version supports that, as HTTP/2 does.

WebTransport frames closely mirror a subset of QUIC frames and provide the essential WebTransport features. Within a WebTransport session, endpoints can

* create and use bidirectional or unidirectional streams with no additional round trips using WT_STREAM frames

Stream creation and data flow on streams uses flow control mechanisms modeled on those in QUIC. Flow control is managed using the WebTransport frames: WT_MAX_DATA, WT_MAX_STREAM_DATA, WT_MAX_STREAMS, WT_DATA_BLOCKED, WT_STREAM_DATA_BLOCKED, and WT_STREAMS_BLOCKED. Flow control for the CONNECT stream as a whole, as provided by the HTTP version in use, applies in addition to any WebTransport-session-level flow control.

WebTransport streams can be aborted using a WT_RESET_STREAM frame and a receiver can request that a sender stop sending with a WT_STOP_SENDING frame.

A WebTransport session is terminated when the CONNECT stream that created it is closed. This implicitly closes all WebTransport streams that were multiplexed over that CONNECT stream.

3. Session Establishment

3.1. Establishing a Transport-Capable HTTP/2 Connection

In order to indicate support for WebTransport, both the client and the server MUST send a SETTINGS_ENABLE_WEBTRANSPORT value set to "1" in their SETTINGS frame. Endpoints MUST NOT use any WebTransport-related functionality unless the parameter has been negotiated.

3.2. Extended CONNECT in HTTP/2

[RFC8441] defines an extended CONNECT method in Section 4, enabled by the SETTINGS_ENABLE_CONNECT_PROTOCOL parameter. An endpoint does not need to send both SETTINGS_ENABLE_CONNECT_PROTOCOL and SETTINGS_ENABLE_WEBTRANSPORT; the SETTINGS_ENABLE_WEBTRANSPORT setting implies that an endpoint supports extended CONNECT.

3.3. Creating a New Session

As WebTransport sessions are established over HTTP, they are identified using the https URI scheme [RFC7230].

In order to create a new WebTransport session, a client can send an HTTP CONNECT request. The :protocol pseudo-header field ([RFC8441]) MUST be set to webtransport (Section 7.1 of [WEBTRANSPORT-H3]). The :scheme field MUST be https. Both the :authority and the :path value MUST be set; those fields indicate the desired WebTransport server. In a Web context, the request MUST include an Origin header field [ORIGIN] that includes the origin of the site that requested the creation of the session.

Upon receiving an extended CONNECT request with a :protocol field set to webtransport, the HTTP server checks if the identified resource supports WebTransport sessions. If the resource does not, the server SHOULD reply with status code 404 (Section 6.5.4 of [RFC7231]). To accept a WebTransport session the server replies with 2xx status code. Before accepting a session, a server MUST ensure that it authorizes use of the session by the site identified in the Origin header.

From the client's perspective, a WebTransport session is established when the client receives a 200 response. From the server's perspective, a session is established once it sends a 200 response. Both endpoints MUST NOT send any WebTransport frames on a given session before that session is established.

3.4. Limiting the Number of Simultaneous Sessions

From a flow control perspective, WebTransport sessions count against HTTP/2 session flow control limits just like regular HTTP requests, since they are established via an HTTP CONNECT request. This document does not make any effort to introduce a separate flow control mechanism for WebTransport sessions. If the server needs to limit the rate of incoming requests, it has alternative mechanisms at its disposal:

* HTTP_STREAM_REFUSED error code defined in [RFC7540] indicates to the receiving HTTP/2 stack that the request was not processed in any way.

* HTTP status code 429 indicates that the request was rejected due to rate limiting [RFC6585]. Unlike the previous method, this signal is directly propagated to the application.

4. WebTransport Features

WebTransport over TCP-based HTTP semantics provides the following features described in [OVERVIEW]: unidirectional streams, bidirectional streams, and datagrams, initiated by either endpoint.

WebTransport streams and datagrams that belong to different WebTransport sessions are identified by the CONNECT stream on which they are transmitted, with one WebTransport session consuming one CONNECT stream.

4.1. Transport Considerations

Because WebTransport over TCP-based HTTP semantics relies on the underlying protocols to provide in order and reliable delivery, there are some notable differences from the way in which QUIC handles application data.

Endpoints MUST send stream data in order. As there is no ordering mechanism available for the receiver to reassemble incoming data, receivers assume that all data arriving in STREAM frames is contiguous and in order.

DATAGRAM frames are delivered to the remote WebTransport endpoint reliably, however this does not require that the receiving implementation deliver that data to the application in a reliable manner.

4.2. WebTransport Stream States

WebTransport streams have states that mirror the states of QUIC streams (Section 3 of [RFC9000]) as closely as possible to aid in ease of implementation.

Because WebTransport does not provide an acknowledgement mechanism for WebTransport frames, it relies on the underlying transport's in order delivery to inform stream state transitions. Wherever QUIC relies on receiving an ack for a packet to transition between stream states, WebTransport performs that transition immediately.

5. WebTransport Frames

WebTransport frames mirror their QUIC counterparts as closely as possible to enable maximal reuse of any applicable QUIC infrastructure by implementors.

A WebTransport frame begins with a Frame Type and Frame Length which are followed by zero or more fields that are type-dependent.

```
Frame {
  Frame Type (i),
  Frame Length (i),
  Type-Dependent Fields (..),
}
```

Figure 1: WebTransport Frame Format

The Frame Type field indicates the type of the frame, defining what type-dependent fields will be present.

The Frame Length field indicates the length of the WebTransport frame, including all type-dependent fields and other information. It does not include the size of the Frame Type or Frame Length fields themselves.

Both of these fields use a variable-length integer encoding (see Section 16 of [RFC9000]), with one exception. To ensure simple and efficient implementations of frame parsing, the frame type and length MUST use the shortest possible encoding. For example, for the frame types defined in this document, this means a single-byte encoding, even though it is possible to encode these values as a two-, four-, or eight-byte variable-length integer.

5.1. WT_PADDING Frames

A WT_PADDING frame (type=0x00) has no semantic value. PADDING frames can be used to introduce additional data between other WebTransport frames and can also be used to provide protection against traffic analysis or for other reasons.

```
WT_PADDING Frame {
  Type (i) = 0x00,
  Length (i),
  Padding (..),
}
```

Figure 2: WT_PADDING Frame Format

The Padding field MUST be set to an all-zero sequence of bytes of any length as specified by the Length field.

5.2. WT_RESET_STREAM Frames

A WebTransport frame called WT_RESET_STREAM is introduced for either endpoint to abruptly terminate the sending part of a WebTransport stream.

An endpoint uses a WT_RESET_STREAM frame (type=0x04) to abruptly terminate the sending part of a stream.

After sending a WT_RESET_STREAM, an endpoint ceases transmission and retransmission of WT_STREAM frames on the identified stream. A receiver of WT_RESET_STREAM can discard any data that it already received on that stream.

```
WT_RESET_STREAM Frame {
  Type (i) = 0x04,
  Length (i),
  Stream ID (i),
  Application Protocol Error Code (i),
}
```

Figure 3: WT_RESET_STREAM Frame Format

The WT_RESET_STREAM frame defines the following fields:

Stream ID: A variable-length integer encoding of the WebTransport stream ID of the stream being terminated.

Application Protocol Error Code: A variable-length integer containing the application protocol error code that indicates why the stream is being closed.

Unlike the equivalent QUIC frame, this frame does not include a Final Size field. In-order delivery of WT_STREAM frames ensures that the amount of session-level flow control consumed by a stream is always known by both endpoints.

5.3. WT_STOP_SENDING Frames

A WebTransport frame called WT_STOP_SENDING is introduced to communicate that incoming data is being discarded on receipt per application request. WT_STOP_SENDING requests that a peer cease transmission on a stream.

```
WT_STOP_SENDING Frame {
  Type (i) = 0x05,
  Length (i),
  Stream ID (i),
  Application Protocol Error Code (i),
}
```

Figure 4: WT_STOP_SENDING Frame Format

The WT_STOP_SENDING frame defines the following fields:

Stream ID:
A variable-length integer carrying the WebTransport stream ID of the stream being ignored.

Application Protocol Error Code: A variable-length integer containing the application-specified reason the sender is ignoring the stream.

5.4. WT_STREAM Frames

WT_STREAM frames implicitly create a stream and carry stream data.

The Type field in the WT_STREAM frame is either 0x0a or 0x0b. This uses the same frame types as a QUIC STREAM frame with the OFF bit clear and the LEN bit set. The FIN bit (0x01) in the frame type indicates that the frame marks the end of the stream in one direction. Stream data consists of any number of 0x0a frames followed by a terminal 0x0b frame.

```
WT_STREAM Frame {
  Type (i) = 0x0a..0x0b,
  Length (i),
  Stream ID (i),
  Stream Data (..),
}
```

Figure 5: WT_STREAM Frame Format

WT_STREAM frames contain the following fields:

Stream ID: The stream ID for the stream.

Stream Data: Zero or more bytes of data for the stream. Empty WT_STREAM frames MUST NOT be used unless they open or close a stream; an endpoint MAY treat an empty WT_STREAM frame that neither starts nor ends a stream as a session error.

5.5. WT_MAX_DATA Frames

A WebTransport frame called WT_MAX_DATA is introduced to inform the peer of the maximum amount of data that can be sent on the WebTransport session as a whole.

```
WT_MAX_DATA Frame {
  Type (i) = 0x10,
  Length (i),
  Maximum Data (i),
}
```

Figure 6: WT_MAX_DATA Frame Format

WT_MAX_DATA frames contain the following field:

Maximum Data: A variable-length integer indicating the maximum amount of data that can be sent on the entire connection, in units of bytes.

All data sent in WT_STREAM frames counts toward this limit. The sum of the lengths of Stream Data fields in WT_STREAM frames MUST NOT exceed the value advertised by a receiver.

5.6. WT_MAX_STREAM_DATA Frames

A WebTransport frame called WT_MAX_STREAM_DATA is introduced to inform a peer of the maximum amount of data that can be sent on a stream.

```
WT_MAX_STREAM_DATA Frame {
  Type (i) = 0x11,
  Length (i),
  Stream ID (i),
  Maximum Stream Data (i),
}
```

Figure 7: WT_MAX_STREAM_DATA Frame Format

WT_MAX_STREAM_DATA frames contain the following fields:

Stream ID: The stream ID of the affected WebTransport stream, encoded as a variable-length integer.

Maximum Stream Data: A variable-length integer indicating the maximum amount of data that can be sent on the identified stream, in units of bytes.

All data sent in WT_STREAM frames for the identified stream counts toward this limit. The sum of the lengths of Stream Data fields in WT_STREAM frames on the identified stream MUST NOT exceed the value advertised by a receiver.

5.7. WT_MAX_STREAMS Frames

A WebTransport frame called WT_MAX_STREAMS is introduced to inform the peer of the cumulative number of streams of a given type it is permitted to open. A WT_MAX_STREAMS frame with a type of 0x12 applies to bidirectional streams, and a WT_MAX_STREAMS frame with a type of 0x13 applies to unidirectional streams.

```
WT_MAX_STREAMS Frame {
  Type (i) = 0x12..0x13,
  Length (i),
  Maximum Streams (i),
}
```

Figure 8: WT_MAX_STREAMS Frame Format

WT_MAX_STREAMS frames contain the following field:

Maximum Streams: A count of the cumulative number of streams of the corresponding type that can be opened over the lifetime of the connection. This value cannot exceed 2^{60}, as it is not possible to encode stream IDs larger than $2^{62}-1$.

An endpoint MUST NOT open more streams than permitted by the current stream limit set by its peer. For instance, a server that receives a unidirectional stream limit of 3 is permitted to open streams 3, 7, and 11, but not stream 15.

Note that this limit includes streams that have been closed as well as those that are open.

5.8. WT_DATA_BLOCKED Frames

A sender SHOULD send a WT_DATA_BLOCKED frame (type=0x14) when it wishes to send data but is unable to do so due to WebTransport session-level flow control. WT_DATA_BLOCKED frames can be used as input to tuning of flow control algorithms.

```
WT_DATA_BLOCKED Frame {
  Type (i) = 0x14,
  Length (i),
  Maximum Data (i),
}
```

Figure 9: WT_DATA_BLOCKED Frame Format

WT_DATA_BLOCKED frames contain the following field:

Maximum Data: A variable-length integer indicating the session-level limit at which blocking occurred.

5.9. WT_STREAM_DATA_BLOCKED Frames

A sender SHOULD send a WT_STREAM_DATA_BLOCKED frame (type=0x15) when it wishes to send data but is unable to do so due to stream-level flow control. This frame is analogous to WT_DATA_BLOCKED.

```
WT_STREAM_DATA_BLOCKED Frame {
  Type (i) = 0x15,
  Length (i),
  Stream ID (i),
  Maximum Stream Data (i),
}
```

Figure 10: WT_STREAM_DATA_BLOCKED Frame Format

WT_STREAM_DATA_BLOCKED frames contain the following fields:

Stream ID: A variable-length integer indicating the WebTransport stream that is blocked due to flow control.

Maximum Stream Data: A variable-length integer indicating the offset of the stream at which the blocking occurred.

5.10. WT_STREAMS_BLOCKED Frames

A sender SHOULD send a WT_STREAMS_BLOCKED frame (type=0x16 or 0x17) when it wishes to open a stream but is unable to do so due to the maximum stream limit set by its peer. A WT_STREAMS_BLOCKED frame of type 0x16 is used to indicate reaching the bidirectional stream limit, and a STREAMS_BLOCKED frame of type 0x17 is used to indicate reaching the unidirectional stream limit.

A WT_STREAMS_BLOCKED frame does not open the stream, but informs the peer that a new stream was needed and the stream limit prevented the creation of the stream.

```
WT_STREAMS_BLOCKED Frame {
  Type (i) = 0x16..0x17,
  Length (i),
  Maximum Streams (i),
}
```

Figure 11: WT_STREAMS_BLOCKED Frame Format

WT_STREAMS_BLOCKED frames contain the following field:

Maximum Streams: A variable-length integer indicating the maximum number of streams allowed at the time the frame was sent. This value cannot exceed 2^{60}, as it is not possible to encode stream IDs larger than $2^{62}-1$.

5.11. WT_DATAGRAM Frames

The WT_DATAGRAM frame type (0x31) is used to carry datagram traffic. Frame type 0x30 is also reserved to maintain parity with QUIC, but unused, as all WebTransport frames MUST contain a length field.

```
WT_DATAGRAM Frame {
  Type (i) = 0x31,
  Length (i),
  Datagram Data (..),
}
```

Figure 12: WT_DATAGRAM Frame Format

WT_DATAGRAM frames contain the following fields:

Datagram Data: The content of the datagram to be delivered.

The data in WT_DATAGRAM frames is not subject to flow control. The receiver MAY discard this data if it does not have sufficient space to buffer it.

An intermediary could forward the data in a WT_DATAGRAM frame over another protocol, such as WebTransport over HTTP/3. In QUIC, a datagram frame can span at most one packet. Because of that, the applications have to know the maximum size of the datagram they can send. However, when proxying the datagrams, the hop-by-hop MTUs can vary.

6. Examples

An example of negotiating a WebTransport Stream on an HTTP/2 connection follows. This example is intended to closely follow the example in Section 5.1 of [RFC8441] to help illustrate the differences defined in this document.

```
[[ From Client ]]                    [[ From Server ]]

SETTINGS
SETTINGS_ENABLE_WEBTRANSPORT = 1

                                     SETTINGS
                                     SETTINGS_ENABLE_WEBTRANSPORT = 1

HEADERS + END_HEADERS
Stream ID = 3
:method = CONNECT
:protocol = webtransport
:scheme = https
:path = /
:authority = server.example.com
origin: server.example.com

                                     HEADERS + END_HEADERS
                                     Stream ID = 3
                                     :status = 200

WT_STREAM
Stream ID = 5
WebTransport Data

                                     WT_STREAM + FIN
                                     Stream ID = 5
                                     WebTransport Data

WT_STREAM + FIN
Stream ID = 5
WebTransport Data
```

An example of the server initiating a WebTransport Stream follows. The only difference here is the endpoint that sends the first WT_STREAM frame.

```
[[ From Client ]]                    [[ From Server ]]

SETTINGS
SETTINGS_ENABLE_WEBTRANSPORT = 1

                                     SETTINGS
                                     SETTINGS_ENABLE_WEBTRANSPORT = 1

HEADERS + END_HEADERS
Stream ID = 3
:method = CONNECT
:protocol = webtransport
:scheme = https
:path = /
:authority = server.example.com
origin: server.example.com
                                     HEADERS + END_HEADERS
                                     Stream ID = 3
                                     :status = 200

                                     WT_STREAM
                                     Stream ID = 2
                                     WebTransport Data

WT_STREAM + FIN
Stream ID = 2
WebTransport Data

                                     WT_STREAM + FIN
                                     Stream ID = 2
                                     WebTransport Data
```

7. Session Termination

An WebTransport session over HTTP/2 is terminated when either endpoint closes the stream associated with the CONNECT request that initiated the session. Upon learning about the session being terminated, the endpoint MUST stop sending new datagrams and reset all of the streams associated with the session.

8. Transport Properties

The WebTransport framework [OVERVIEW] defines a set of optional transport properties that clients can use to determine the presence of features which might allow additional optimizations beyond the common set of properties available via all WebTransport protocols. Below are details about support in Http2Transport for those properties.

Stream Independence:

Http2Transport does not support stream independence, as HTTP/2 inherently has head of line blocking.

Partial Reliability: Http2Transport does not support partial reliability, as HTTP/2 retransmits any lost data. This means that any datagrams sent via Http2Transport will be retransmitted regardless of the preference of the application. The receiver is permitted to drop them, however, if it is unable to buffer them.

Pooling Support: Http2Transport supports pooling, as multiple transports using Http2Transport may share the same underlying HTTP/2 connection and therefore share a congestion controller and other transport context.

Connection Mobility: Http2Transport does not support connection mobility, unless an underlying transport protocol that supports multipath or migration, such as MPTCP [MPTCP], is used underneath HTTP/2 and TLS. Without such support, Http2Transport connections cannot survive network transitions.

9. Security Considerations

WebTransport over HTTP/2 satisfies all of the security requirements imposed by [OVERVIEW] on WebTransport protocols, thus providing a secure framework for client-server communication in cases when the client is potentially untrusted.

WebTransport over HTTP/2 requires explicit opt-in through the use of HTTP SETTINGS; this avoids potential protocol confusion attacks by ensuring the HTTP/2 server explicitly supports it. It also requires the use of the Origin header, providing the server with the ability to deny access to Web-based clients that do not originate from a trusted origin.

Just like HTTP traffic going over HTTP/2, WebTransport pools traffic to different origins within a single connection. Different origins imply different trust domains, meaning that the implementations have to treat each transport as potentially hostile towards others on the same connection. One potential attack is a resource exhaustion attack: since all of the transports share both congestion control and flow control context, a single client aggressively using up those resources can cause other transports to stall. The user agent thus SHOULD implement a fairness scheme that ensures that each transport within connection gets a reasonable share of controlled resources; this applies both to sending data and to opening new streams.

10. IANA Considerations

10.1. HTTP/2 SETTINGS Parameter Registration

The following entry is added to the "HTTP/2 Settings" registry established by [RFC7540]:

The SETTINGS_ENABLE_WEBTRANSPORT parameter indicates that the specified HTTP/2 connection is WebTransport-capable.

Setting Name: ENABLE_WEBTRANSPORT

Value: 0x2b603742

Default: 0

Specification: This document

11. References

11.1. Normative References

[H2] Thomson, M. and C. Benfield, "HTTP/2", Work in Progress, Internet-Draft, draft-ietf-httpbis-http2bis-07, 24 January 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-http2bis-07>.

[HTTP] Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP Semantics", Work in Progress, Internet-Draft, draft-ietf-httpbis-semantics-19, 12 September 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-semantics-19>.

[ORIGIN] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <https://www.rfc-editor.org/rfc/rfc6454>.

[OVERVIEW] Vasiliev, V., "The WebTransport Protocol Framework", Work in Progress, Internet-Draft, draft-ietf-webtrans-overview-03, 7 March 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-webtrans-overview-03>.

[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>.

[RFC6585] Nottingham, M. and R. Fielding, "Additional HTTP Status Codes", RFC 6585, DOI 10.17487/RFC6585, April 2012, <https://www.rfc-editor.org/rfc/rfc6585>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/rfc/rfc7230>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/rfc/rfc7231>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/rfc/rfc7540>.

[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>.

[RFC8441] McManus, P., "Bootstrapping WebSockets with HTTP/2", RFC 8441, DOI 10.17487/RFC8441, September 2018, <https://www.rfc-editor.org/rfc/rfc8441>.

[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>.

[WEBTRANSPORT-H3] Frindell, A., Kinnear, E., and V. Vasiliev, "WebTransport over HTTP/3", Work in Progress, Internet-Draft, draft-ietf-webtrans-http3-02, 25 October 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-webtrans-http3-02>.

11.2. Informative References

[DATAGRAM] Pauly, T., Kinnear, E., and D. Schinazi, "An Unreliable Datagram Extension to QUIC", Work in Progress, Internet-Draft, draft-ietf-quic-datagram-10, 4 February 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-datagram-10>.

[HTTP3] Bishop, M., "Hypertext Transfer Protocol Version 3 (HTTP/3)", Work in Progress, Internet-Draft, draft-ietf-quic-http-34, 2 February 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-34>.

[MPTCP] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple

Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <https://www.rfc-editor.org/rfc/rfc6824>.

[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>.

Acknowledgments

Thanks to Anthony Chivetta, Joshua Otto, and Valentin Pistol for their contributions in the design and implementation of this work.

Index

Authors' Addresses

Alan Frindell
Facebook Inc.

Email: afrind@fb.com

Eric Kinnear
Apple Inc.
One Apple Park Way
Cupertino, California 95014,
United States of America

Email: ekinnear@apple.com

Tommy Pauly
Apple Inc.
One Apple Park Way
Cupertino, California 95014,
United States of America

Email: tpauly@apple.com

Martin Thomson
Mozilla

Email: mt@lowentropy.net

Victor Vasiliev
Google

Email: vasilvv@google.com

Guowu Xie
Facebook Inc.

Email: woo@fb.com