webtrans   A. Frindell
Internet-Draft   Facebook Inc.
Intended status: Standards Track   E. Kinnear
Expires: 5 September 2024   T. Pauly
                                                         Apple Inc.
                                                        M. Thomson
                                                            Mozilla
                                                       V. Vasiliev
                                                            Google
                                                            G. Xie
                                                     Facebook Inc.
                                                       4 March 2024

                        WebTransport over HTTP/2
                       draft-ietf-webtrans-http2-08

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 (mailto: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 5 September 2024.

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 [HTTP2] 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_WEBTRANSPORT_MAX_SESSIONS setting with a
value greater than "0" to indicate that they both support
WebTransport over HTTP/2. For servers, the value of the setting is
the number of concurrent sessions the server is willing to receive.
Clients cannot receive incoming WebTransport sessions, so any value
greater than "0" sent by a client simply indicates support for
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 referred to as a
Session ID, is used to uniquely identify a given WebTransport
session within the connection. WebTransport using HTTP/2 uses
extended CONNECT with the same webtransport HTTP Upgrade Token as
[WEBTRANSPORT-H3]. This Upgrade Token uses the Capsule Protocol as
defined in [HTTP-DATAGRAM].

After the session is established, endpoints exchange WebTransport
messages using the Capsule Protocol on the bidirectional CONNECT
stream, the "data stream" as defined in Section 3.1 of
[HTTP-DATAGRAM].

Within this stream, _WebTransport streams_ and _WebTransport
datagrams_ are multiplexed. In HTTP/2, WebTransport capsules 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 capsules 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 the WT_STREAM capsule

Stream creation and data flow on streams uses flow control mechanisms
modeled on those in QUIC. Flow control is managed using the
WebTransport capsules: 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 capsule
and a receiver can request that a sender stop sending with a
WT_STOP_SENDING capsule.

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_WEBTRANSPORT_MAX_SESSIONS value
greater than "0" 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 needs to
send both SETTINGS_ENABLE_CONNECT_PROTOCOL and
SETTINGS_WEBTRANSPORT_MAX_SESSIONS for WebTransport to be enabled.

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 406 (Section 15.5.7 of [RFC9110]). If
it does, it MAY accept the session by replying with a 2xx series
status code, as defined in Section 15.3 of [SEMANTICS]. The
WebTransport server MUST verify the Origin header to ensure that the
specified origin is allowed to access the server in question.

A WebTransport session is established when the server sends a 2xx
response. A server generates that response from the request header,
not from the contents of the request. To enable clients to resend
data when attempting to re-establish a session that was rejected by a
server, a server MUST NOT process any capsules on the request stream
unless it accepts the WebTransport session.

A client MAY optimistically send any WebTransport capsules associated
with a CONNECT request, without waiting for a response, to the extent
allowed by flow control. This can reduce latency for data sent by a
client at the start of a WebTransport session. For example, a client
might choose to send datagrams or flow control updates before
receiving any response from the server.

3.4. Flow Control

Flow control governs the amount of resources that can be consumed or
data that can be sent. WebTransport over HTTP/2 allows a server to
limit the number of sessions that a client can create on a single
connection; see Section 3.4.1.

For data, there are five applicable levels of flow control for data
that is sent or received using WebTransport over HTTP/2:

1. TCP flow control.

2. HTTP/2 connection flow control, which governs the total amount of
    data in DATA frames for all HTTP/2 streams.

   3. HTTP/2 stream flow control, which limits the data on a single
       HTTP/2 stream. For a WebTransport session, this includes all
       capsules, including those that are exempt from WebTransport
       session-level flow control.

   4. WebTransport session-level flow control, which limits the total
       amount of stream data that can be sent or received on streams
       within the WebTransport session. Note that this does not limit
       other types of capsules within a WebTransport session, such as
       control messages or datagrams.

   5. WebTransport stream flow control, which limits data on individual
       streams within a session.

   TCP and HTTP/2 define the first three levels of flow control. This
   document defines the final two.

3.4.1. Limiting the Number of Simultaneous Sessions

   This document defines a SETTINGS_WEBTRANSPORT_MAX_SESSIONS parameter
   that allows the server to limit the maximum number of concurrent
   WebTransport sessions on a single HTTP/2 connection. The client MUST
   NOT open more sessions than indicated in the server SETTINGS
   parameters. The server MUST NOT close the connection if the client
   opens sessions exceeding this limit, as the client and the server do
   not have a consistent view of how many sessions are open due to the
   asynchronous nature of the protocol; instead, it MUST reset all of
   the CONNECT streams it is not willing to process with the
   REFUSED_STREAM error code (Section 8.7 of [HTTP2]).

   Just like other HTTP requests, WebTransport sessions, and data sent
   on those sessions, are counted against flow control limits. Servers
   that wish to limit the rate of incoming requests on any particular
   session have multiple mechanisms:

   * The REFUSED_STREAM error code defined in Section 8.7 of [HTTP2]
      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.

An endpoint that wishes to reduce the value of
SETTINGS_WEBTRANSPORT_MAX_SESSIONS to a value that is below the
current number of open sessions can either close sessions that exceed
the new value or allow those sessions to complete. Endpoints MUST
NOT reduce the value of SETTINGS_WEBTRANSPORT_MAX_SESSIONS to "0"
after previously advertising a non-zero value.

3.4.2. Limiting the Number of Streams Within a Session

This document defines a WT_MAX_STREAMS capsule (Section 5.7) that
allows each endpoint to limit the number of streams its peer is
permitted to open as part of a WebTransport session. There is a
separate limit for bidirectional streams and for unidirectional
streams. Note that, unlike WebTransport over HTTP/3
[WEBTRANSPORT-H3], because the entire WebTransport session is
contained within HTTP/2 DATA frames on a single HTTP/2 stream, this
limit is the only mechanism for an endpoint to limit the number of
WebTransport streams that its peer can open on a session.

3.4.3. Initial Flow Control Limits

To allow stream data to be exchanged in the same flight as the
extended CONNECT request that establishes a WebTransport session,
initial flow control limits can be exchanged via HTTP/2 SETTINGS
(Section 3.4.3.1). Initial values for the flow control limits can
also be exchanged via the WebTransport-Init header field on the
extended CONNECT request (Section 3.4.3.2).

The limits communicated via HTTP/2 SETTINGS apply to all WebTransport
sessions opened on that HTTP/2 connection. Limits communicated via
the WebTransport-Init header field apply only to the WebTransport
session established by the extended CONNECT request carrying that
field.

If both the SETTINGS and the header field are present when a
WebTransport session is established, the endpoint MUST use the
greater of the two values for each corresponding initial flow control
value. Endpoints sending the SETTINGS and also including the header
field SHOULD ensure that the header field values are greater than or
equal to the values provided in the SETTINGS.

3.4.3.1. Flow Control SETTINGS

Initial flow control limits can be exchanged via HTTP/2 SETTINGS
(Section 9.1) by providing non-zero values for

*   WT_MAX_DATA via SETTINGS_WEBTRANSPORT_INITIAL_MAX_DATA

   *   WT_MAX_STREAM_DATA via
       SETTINGS_WEBTRANSPORT_INITIAL_MAX_STREAM_DATA_UNI and
       SETTINGS_WEBTRANSPORT_INITIAL_MAX_STREAM_DATA_BIDI

   *   WT_MAX_STREAMS via SETTINGS_WEBTRANSPORT_INITIAL_MAX_STREAMS_UNI
       and SETTINGS_WEBTRANSPORT_INITIAL_MAX_STREAMS_BIDI

3.4.3.2. Flow Control Header Field

   The WebTransport-Init HTTP header field can be used to communicate
   the initial values of the flow control windows, similar to how QUIC
   uses transport parameters. The WebTransport-Init is a Dictionary
   Structured Field (Section 3.2 of [RFC8941]). If any of the fields
   cannot be parsed correctly or do not have the correct type, the peer
   MUST reset the CONNECT stream. The following keys are defined for
   the WebTransport-Init header field:

   u: The initial flow control limit for unidirectional streams opened
       by the recipient of this header field. MUST be an Integer.

   bl: The initial flow control limit for the bidirectional streams
       opened by the sender of this header field. MUST be an Integer.

   br: The initial flow control limit for the bidirectional streams
       opened by the recipient of this header field. MUST be an Integer.

3.4.4. Flow Control and Intermediaries

   WebTransport over HTTP/2 uses several capsules for flow control, and
   all of these capsules define special intermediary handling as
   described in Section 3.2 of [HTTP-DATAGRAM]. These capsules,
   referred to as the "flow control capsules" are WT_MAX_DATA,
   WT_MAX_STREAM_DATA, WT_MAX_STREAMS, WT_DATA_BLOCKED,
   WT_STREAM_DATA_BLOCKED, and WT_STREAMS_BLOCKED.

   Because flow control in WebTransport is hop-by-hop and does not
   provide an end-to-end signal, intermediaries MUST consume flow
   control signals and express their own flow control limits to the next
   hop. The intermediary can send these signals via HTTP/3 flow control
   messages, HTTP/2 flow control messages, or as WebTransport flow
   control capsules, where appropriate. Intermediaries are responsible
   for storing any data for which they advertise flow control credit if
   that data cannot be immediately forwarded to the next hop.

   In practice, an intermediary that translates flow control signals
   between similar WebTransport protocols, such as between two HTTP/2
   connections, can often simply reexpress the same limits received on
   one connection directly on the other connection.

An intermediary that does not want to be responsible for storing data
that cannot be immediately sent on its translated connection would
ensure that it does not advertise a higher flow control limit on one
connection than the corresponding limit on the translated connection.

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 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.

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. For example, endpoints 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
WT_STREAM capsules is contiguous and in order.

Below are details about support in WebTransport over HTTP/2 for the
properties defined by the WebTransport framework.

Stream Independence: WebTransport over HTTP/2 does not support
   stream independence, as HTTP/2 inherently has head-of-line
   blocking.

Partial Reliability: WebTransport over HTTP/2 does not support
   partial reliability, as HTTP/2 retransmits any lost data. This
   means that any datagrams sent via WebTransport over HTTP/2 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: WebTransport over HTTP/2 supports pooling, as

multiple transports using WebTransport over HTTP/2 may share the
same underlying HTTP/2 connection and therefore share a congestion
controller and other transport context. Note that WebTransport
streams over HTTP/2 are contained within a single HTTP/2 stream
and do not compete with other pooled WebTransport sessions for
per-stream resources.

Connection Mobility: WebTransport over HTTP/2 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, WebTransport
over HTTP/2 connections cannot survive network transitions.

4.2. WebTransport Streams

WebTransport streams have identifiers and states that mirror the
identifiers ((Section 2.1 of [RFC9000])) and states (Section 3 of
[RFC9000]) of QUIC streams as closely as possible to aid in ease of
implementation.

WebTransport streams are identified by a numeric value, or stream ID.
Stream IDs are only meaningful within the WebTransport session in
which they were created. They share the same semantics as QUIC
stream IDs, with client initiated streams having even-numbered stream
IDs and server-initiated streams having odd-numbered stream IDs.
Similarly, they can be bidirectional or unidirectional, indicated by
the second least significant bit of the stream ID.

Because WebTransport does not provide an acknowledgement mechanism
for WebTransport capsules, 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 Capsules

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

WebTransport capsules use the Capsule Protocol defined in Section 3.2
of [HTTP-DATAGRAM].

5.1. PADDING Capsule

   A PADDING capsule is an HTTP capsule [HTTP-DATAGRAM] of
   type=0x190B4D38 and has no semantic value. PADDING capsules can be
   used to introduce additional data between other HTTP datagrams and
   can also be used to provide protection against traffic analysis or
   for other reasons.

   Note that, when used with WebTransport over HTTP/2, the PADDING
   capsule exists alongside the ability to pad HTTP/2 frames
   (Section 10.7 of [RFC9113]). HTTP/2 padding is hop-by-hop and can be
   modified by intermediaries, while the PADDING capsule traverses
   intermedaries. The PADDING capsule is also constrained to be no
   smaller than the capsule overhead itself.

   PADDING Capsule {
     Type (i) = 0x190B4D38,
     Length (i),
     Padding (..),
   }

                    Figure 1: PADDING Capsule 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 Capsule

   A WT_RESET_STREAM capsule is an HTTP capsule [HTTP-DATAGRAM] of
   type=0x190B4D39 and allows either endpoint to abruptly terminate the
   sending part of a WebTransport stream.

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

   WT_RESET_STREAM Capsule {
     Type (i) = 0x190B4D39,
     Length (i),
     Stream ID (i),
     Application Protocol Error Code (i),
   }

                 Figure 2: WT_RESET_STREAM Capsule Format

   The WT_RESET_STREAM capsule 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 capsule does not include a
   Final Size field. In-order delivery of WT_STREAM capsules ensures
   that the amount of session-level flow control consumed by a stream is
   always known by both endpoints.

5.3. WT_STOP_SENDING Capsule

   An HTTP capsule [HTTP-DATAGRAM] called WT_STOP_SENDING
   (type=0x190B4D3A) 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 WebTransport stream.

   WT_STOP_SENDING Capsule {
     Type (i) = 0x190B4D3A,
     Length (i),
     Stream ID (i),
     Application Protocol Error Code (i),
   }

                  Figure 3: WT_STOP_SENDING Capsule Format

   The WT_STOP_SENDING capsule 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 Capsule

   WT_STREAM capsules implicitly create a WebTransport stream and carry
   stream data.

   The Type field in the WT_STREAM capsule is either 0x190B4D3B or
   0x190B4D3C. The least significant bit in the capsule type is the FIN
   bit (0x01), indicating when set that the capsule marks the end of the
   stream in one direction. Stream data consists of any number of
   0x190B4D3B capsules followed by a terminal 0x190B4D3C capsule.

```
WT_STREAM Capsule {
  Type (i) = 0x190B4D3B..0x190B4D3C,
  Length (i),
  Stream ID (i),
  Stream Data (..),
}
```

                   Figure 4: WT_STREAM Capsule Format

WT_STREAM capsules 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 capsules MUST NOT be used unless they open or close a
   stream; an endpoint MAY treat an empty WT_STREAM capsule that
   neither starts nor ends a stream as a session error.

5.5. WT_MAX_DATA Capsule

An HTTP capsule [HTTP-DATAGRAM] called WT_MAX_DATA (type=0x190B4D3D)
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 Capsule {
  Type (i) = 0x190B4D3D,
  Length (i),
  Maximum Data (i),
}
```

                  Figure 5: WT_MAX_DATA Capsule Format

WT_MAX_DATA capsules 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 capsules counts toward this limit. The
sum of the lengths of Stream Data fields in WT_STREAM capsules MUST
NOT exceed the value advertised by a receiver.

The WT_MAX_DATA capsule defines special intermediary handling, as
described in Section 3.2 of [HTTP-DATAGRAM]. Intermediaries MUST
consume WT_MAX_DATA capsules for flow control purposes and MUST
generate and send appropriate flow control signals for their limits;
see Section 3.4.4.

   The initial value for this limit MAY be communicated by sending a
   non-zero value for SETTINGS_WEBTRANSPORT_INITIAL_MAX_DATA.

5.6. WT_MAX_STREAM_DATA Capsule

   An HTTP capsule [HTTP-DATAGRAM] called WT_MAX_STREAM_DATA
   (type=0x190B4D3E) is introduced to inform a peer of the maximum
   amount of data that can be sent on a WebTransport stream.

   WT_MAX_STREAM_DATA Capsule {
     Type (i) = 0x190B4D3E,
     Length (i),
     Stream ID (i),
     Maximum Stream Data (i),
   }

                  Figure 6: WT_MAX_STREAM_DATA Capsule Format

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

   The WT_MAX_STREAM_DATA capsule defines special intermediary handling,
   as described in Section 3.2 of [HTTP-DATAGRAM]. Intermediaries MUST
   consume WT_MAX_STREAM_DATA capsules for flow control purposes and
   MUST generate and send appropriate flow control signals for their
   limits; see Section 3.4.4.

   Initial values for this limit for unidirectional and bidirectional
   streams MAY be communicated by sending non-zero values for
   SETTINGS_WEBTRANSPORT_INITIAL_MAX_STREAM_DATA_UNI and
   SETTINGS_WEBTRANSPORT_INITIAL_MAX_STREAM_DATA_BIDI respectively.

5.7. WT_MAX_STREAMS Capsule

   An HTTP capsule [HTTP-DATAGRAM] 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 capsule with a type
   of 0x190B4D3F applies to bidirectional streams, and a WT_MAX_STREAMS
   capsule with a type of 0x190B4D40 applies to unidirectional streams.

   Note that, because Maximum Streams is a cumulative value representing
   the total allowed number of streams, including previously closed
   streams, endpoints repeatedly send new WT_MAX_STREAMS capsules with
   increasing Maximum Streams values as streams are opened.

```
WT_MAX_STREAMS Capsule {
  Type (i) = 0x190B4D3F..0x190B4D40,
  Length (i),
  Maximum Streams (i),
}
```

                     Figure 7: WT_MAX_STREAMS Capsule Format

   WT_MAX_STREAMS capsules 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.

   The WT_MAX_STREAMS capsule defines special intermediary handling, as
   described in Section 3.2 of [HTTP-DATAGRAM]. Intermediaries MUST
   consume WT_MAX_STREAMS capsules for flow control purposes and MUST
   generate and send appropriate flow control signals for their limits.

   Initial values for these limits MAY be communicated by sending non-
   zero values for SETTINGS_WEBTRANSPORT_INITIAL_MAX_STREAMS_UNI and
   SETTINGS_WEBTRANSPORT_INITIAL_MAX_STREAMS_BIDI.

5.8. WT_DATA_BLOCKED Capsule

   A sender SHOULD send a WT_DATA_BLOCKED capsule (type=0x190B4D41) when
   it wishes to send data but is unable to do so due to WebTransport
   session-level flow control. WT_DATA_BLOCKED capsules can be used as
   input to tuning of flow control algorithms.

   WT_DATA_BLOCKED Capsule {
     Type (i) = 0x190B4D41,
     Length (i),
     Maximum Data (i),
   }

                   Figure 8: WT_DATA_BLOCKED Capsule Format

   WT_DATA_BLOCKED capsules contain the following field:

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

   The WT_DATA_BLOCKED capsule defines special intermediary handling, as
   described in Section 3.2 of [HTTP-DATAGRAM]. Intermedaries MUST
   consume WT_DATA_BLOCKED capsules for flow control purposes and MUST
   generate and send appropriate flow control signals for their limits;
   see Section 3.4.4.

5.9. WT_STREAM_DATA_BLOCKED Capsule

   A sender SHOULD send a WT_STREAM_DATA_BLOCKED capsule
   (type=0x190B4D42) when it wishes to send data but is unable to do so
   due to stream-level flow control. This capsule is analogous to
   WT_DATA_BLOCKED.

   WT_STREAM_DATA_BLOCKED Capsule {
     Type (i) = 0x190B4D42,
     Length (i),
     Stream ID (i),
     Maximum Stream Data (i),
   }

               Figure 9: WT_STREAM_DATA_BLOCKED Capsule Format

   WT_STREAM_DATA_BLOCKED capsules 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.

The WT_STREAM_DATA_BLOCKED capsule defines special intermediary
handling, as described in Section 3.2 of [HTTP-DATAGRAM].
Intermedaries MUST consume WT_STREAM_DATA_BLOCKED capsules for flow
control purposes and MUST generate and send appropriate flow control
signals for their limits; see Section 3.4.4.

5.10. WT_STREAMS_BLOCKED Capsule

A sender SHOULD send a WT_STREAMS_BLOCKED capsule (type=0x190B4D43 or
0x190B4D44) 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 capsule of type 0x190B4D43 is used to indicate
reaching the bidirectional stream limit, and a STREAMS_BLOCKED
capsule of type 0x190B4D44 is used to indicate reaching the
unidirectional stream limit.

A WT_STREAMS_BLOCKED capsule 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 Capsule {
  Type (i) = 0x190B4D43..0x190B4D44,
  Length (i),
  Maximum Streams (i),
}
```

Figure 10: WT_STREAMS_BLOCKED Capsule Format

WT_STREAMS_BLOCKED capsules contain the following field:

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

The WT_STREAMS_BLOCKED capsule defines special intermediary handling,
as described in Section 3.2 of [HTTP-DATAGRAM]. Intermedaries MUST
consume WT_STREAMS_BLOCKED capsules for flow control purposes and
MUST generate and send appropriate flow control signals for their
limits.

5.11. DATAGRAM Capsule

WebTransport over HTTP/2 uses the DATAGRAM capsule defined in
Section 3.5 of [HTTP-DATAGRAM] to carry datagram traffic.

```
DATAGRAM Capsule {
  Type (i) = 0x00,
  Length (i),
  HTTP Datagram Payload (..),
}
```

Figure 11: DATAGRAM Capsule Format

When used in WebTransport over HTTP/2, DATAGRAM capsules contain the following fields:

HTTP Datagram Payload: The content of the datagram to be delivered.

The data in DATAGRAM capsules 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 DATAGRAM capsule 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.

Section 3.5 of [HTTP-DATAGRAM] indicates that intermediaries that forward DATAGRAM capsules where QUIC datagrams [DATAGRAM] are available forward the contents of the capsule as native QUIC datagrams, rather than as HTTP datagrams in a DATAGRAM capsule. Similarly, when forwarding DATAGRAM capsules used as part of a WebTransport over HTTP/2 session on a WebTransport session that natively supports QUIC datagrams, such as WebTransport over HTTP/3 [WEBTRANSPORT-H3], intermediaries follow the requirements in [WEBTRANSPORT-H3] to use native QUIC datagrams.

5.12. CLOSE_WEBTRANSPORT_SESSION Capsule

WebTransport over HTTP/2 uses the CLOSE_WEBTRANSPORT_SESSION capsule defined in Section 5 of [WEBTRANSPORT-H3] to terminate a WebTransport session with an application error code and message.

WebTransport sessions can be terminated by optionally sending a CLOSE_WEBTRANSPORT_SESSION capsule and then by closing the HTTP/2 stream associated with the session (see Section 7).

```
CLOSE_WEBTRANSPORT_SESSION Capsule {
  Type (i) = CLOSE_WEBTRANSPORT_SESSION,
  Length (i),
  Application Error Code (32),
  Application Error Message (..8192),
}
```

Figure 12: CLOSE_WEBTRANSPORT_SESSION Capsule Format

When used in WebTransport over HTTP/2, CLOSE_WEBTRANSPORT_SESSION capsules contain the following fields:

Application Error Code: A 32-bit error code provided by the application closing the connection.

Application Error Message: A UTF-8 encoded error message string provided by the application closing the connection. The message takes up the remainder of the capsule, and its length MUST NOT exceed 1024 bytes.

An endpoint that sends a CLOSE_WEBTRANSPORT_SESSION capsule MUST set the FIN bit on the frame carrying the capsule. The recipient MUST close the stream upon receipt of the capsule.

Cleanly terminating a WebTransport session without a CLOSE_WEBTRANSPORT_SESSION capsule is semantically equivalent to terminating it with a CLOSE_WEBTRANSPORT_SESSION capsule that has an error code of 0 and an empty error string.

5.13. DRAIN_WEBTRANSPORT_SESSION Capsule

HTTP/2 uses GOAWAY frames (Section 6.8 of [HTTP2]) to allow an endpoint to gracefully stop accepting new streams while still finishing processing of previously established streams.

WebTransport over HTTP/2 uses the DRAIN_WEBTRANSPORT_SESSION capsule defined in Section 4.6 of [WEBTRANSPORT-H3] to gracefully shut down a WebTransport session.

```
DRAIN_WEBTRANSPORT_SESSION Capsule {
  Type (i) = DRAIN_WEBTRANSPORT_SESSION,
  Length (i) = 0
}
```

Figure 13: DRAIN_WEBTRANSPORT_SESSION Capsule Format

   After sending or receiving either a DRAIN_WEBTRANSPORT_SESSION
   capsule or HTTP/2 GOAWAY frame, an endpoint MAY continue using the
   session and MAY open new WebTransport streams. The signal is
   intended for the application using WebTransport, which is expected to
   attempt to gracefully terminate the session as soon as possible.

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_CONNECT_PROTOCOL = 1
SETTINGS_WEBTRANSPORT_MAX_SESSIONS = 1

                                        SETTINGS
                                        SETTINGS_ENABLE_CONNECT_PROTOCOL = 1
                                        SETTINGS_WEBTRANSPORT_MAX_SESSIONS = 100

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 = 0
WebTransport Data

                                        WT_STREAM + FIN
                                        Stream ID = 0
                                        WebTransport Data

WT_STREAM + FIN
Stream ID = 0
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 capsule.

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

SETTINGS
SETTINGS_ENABLE_CONNECT_PROTOCOL = 1
SETTINGS_WEBTRANSPORT_MAX_SESSIONS = 1

                                         SETTINGS
                                         SETTINGS_ENABLE_CONNECT_PROTOCOL = 1
                                         SETTINGS_WEBTRANSPORT_MAX_SESSIONS = 100

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 = 1
                                         WebTransport Data

WT_STREAM + FIN
Stream ID = 1
WebTransport Data

                                         WT_STREAM + FIN
                                         Stream ID = 1
                                         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.

Prior to closing the stream associated with the CONNECT request,
either endpoint can send a CLOSE_WEBTRANSPORT_SESSION capsule with an
application error code and message to convey additional information
about the reasons for the closure of the session.

8. 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.

9. IANA Considerations

9.1. HTTP/2 SETTINGS Parameter Registration

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

The SETTINGS_WEBTRANSPORT_MAX_SESSIONS parameter indicates that the
specified HTTP/2 connection is WebTransport-capable and the number of
concurrent sessions an endpoint is willing to receive. The default
value for the SETTINGS_WEBTRANSPORT_MAX_SESSIONS parameter is "0",
meaning that the endpoint is not willing to receive any WebTransport
sessions.

Setting Name: WEBTRANSPORT_MAX_SESSIONS

Value: 0x2b60

   Default: 0

   Specification: This document

   The SETTINGS_WEBTRANSPORT_INITIAL_MAX_DATA parameter indicates the
   initial value for the session data limit, otherwise communicated by
   the WT_MAX_DATA capsule (see Section 5.5). The default value for the
   SETTINGS_WEBTRANSPORT_INITIAL_MAX_DATA parameter is "0", indicating
   that the endpoint needs to send a WT_MAX_DATA capsule within each
   session before its peer is allowed to send any stream data within
   that session.

   Note that this limit applies to all WebTransport sessions that use
   the HTTP/2 connection on which this SETTING is sent.

   Setting Name: SETTINGS_WEBTRANSPORT_INITIAL_MAX_DATA

   Value: 0x2b61

   Default: 0

   Specification: This document

   The SETTINGS_WEBTRANSPORT_INITIAL_MAX_STREAM_DATA_UNI parameter
   indicates the initial value for the stream data limit for incoming
   unidirectional streams, otherwise communicated by the
   WT_MAX_STREAM_DATA capsule (see Section 5.6). The default value for
   the SETTINGS_WEBTRANSPORT_INITIAL_MAX_STREAM_DATA_UNI parameter is
   "0", indicating that the endpoint needs to send WT_MAX_STREAM_DATA
   capsules for each stream within each individual WebTransport session
   before its peer is allowed to send any stream data on those streams.

   Note that this limit applies to all WebTransport streams on all
   sessions that use the HTTP/2 connection on which this SETTING is
   sent.

   Setting Name: SETTINGS_WEBTRANSPORT_INITIAL_MAX_STREAM_DATA_UNI

   Value: 0x2b62

   Default: 0

   Specification: This document

   The SETTINGS_WEBTRANSPORT_INITIAL_MAX_STREAM_DATA_BIDI parameter
   indicates the initial value for the stream data limit for incoming
   data on bidirectional streams, otherwise communicated by the
   WT_MAX_STREAM_DATA capsule (see Section 5.6). The default value for

the SETTINGS_WEBTRANSPORT_INITIAL_MAX_STREAM_DATA_BIDI parameter is
"0", indicating that the endpoint needs to send WT_MAX_STREAM_DATA
capsules for each stream within each individual WebTransport session
before its peer is allowed to send any stream data on those streams.

Note that this limit applies to all WebTransport streams on all
sessions that use the HTTP/2 connection on which this SETTING is
sent.

Setting Name: SETTINGS_WEBTRANSPORT_INITIAL_MAX_STREAM_DATA_BIDI

Value: 0x2b63

Default: 0

Specification: This document

The SETTINGS_WEBTRANSPORT_INITIAL_MAX_STREAMS_UNI parameter indicates
the initial value for the unidirectional max stream limit, otherwise
communicated by the WT_MAX_STREAMS capsule (see Section 5.7). The
default value for the SETTINGS_WEBTRANSPORT_INITIAL_MAX_STREAMS_UNI
parameter is "0", indicating that the endpoint needs to send
WT_MAX_STREAMS capsules on each individual WebTransport session
before its peer is allowed to create any unidirectional streams
within that session.

Note that this limit applies to all WebTransport sessions that use
the HTTP/2 connection on which this SETTING is sent.

Setting Name: SETTINGS_WEBTRANSPORT_INITIAL_MAX_STREAMS_UNI

Value: 0x2b64

Default: 0

Specification: This document

The SETTINGS_WEBTRANSPORT_INITIAL_MAX_STREAMS_BIDI parameter
indicates the initial value for the bidirectional max stream limit,
otherwise communicated by the WT_MAX_STREAMS capsule (see
Section 5.7). The default value for the
SETTINGS_WEBTRANSPORT_INITIAL_MAX_STREAMS_BIDI parameter is "0",
indicating that the endpoint needs to send WT_MAX_STREAMS capsules on
each individual WebTransport session before its peer is allowed to
create any bidirectional streams within that session.

Note that this limit applies to all WebTransport sessions that use
the HTTP/2 connection on which this SETTING is sent.

    Setting Name: SETTINGS_WEBTRANSPORT_INITIAL_MAX_STREAMS_BIDI

    Value: 0x2b65

    Default: 0

    Specification: This document

9.2. Capsule Types

    The following entries are added to the "HTTP Capsule Types" registry
    established by [HTTP-DATAGRAM]:

    The PADDING capsule.

    Value: 0x190B4D38
    Capsule Type: PADDING
    Status: permanent
    Specification: This document
    Change Controller: IETF
    Contact: WebTransport Working Group webtransport@ietf.org
       (mailto:webtransport@ietf.org)
    Notes: None

    The WT_RESET_STREAM capsule.

    Value: 0x190B4D39
    Capsule Type: WT_RESET_STREAM
    Status: permanent
    Specification: This document
    Change Controller: IETF
    Contact: WebTransport Working Group webtransport@ietf.org
       (mailto:webtransport@ietf.org)
    Notes: None

    The WT_STOP_SENDING capsule.

    Value: 0x190B4D3A
    Capsule Type: WT_STOP_SENDING
    Status: permanent
    Specification: This document
    Change Controller: IETF
    Contact: WebTransport Working Group webtransport@ietf.org
       (mailto:webtransport@ietf.org)
    Notes: None

    The WT_STREAM capsule.

      Value: 0x190B4D3B..0x190B4D3C
      Capsule Type: WT_STREAM
      Status: permanent
      Specification: This document
      Change Controller: IETF
      Contact: WebTransport Working Group webtransport@ietf.org
         (mailto:webtransport@ietf.org)
      Notes: None

      The WT_MAX_DATA capsule.

      Value: 0x190B4D3D
      Capsule Type: WT_MAX_DATA
      Status: permanent
      Specification: This document
      Change Controller: IETF
      Contact: WebTransport Working Group webtransport@ietf.org
         (mailto:webtransport@ietf.org)
      Notes: None

      The WT_MAX_STREAM_DATA capsule.

      Value: 0x190B4D3E
      Capsule Type: WT_MAX_STREAM_DATA
      Status: permanent
      Specification: This document
      Change Controller: IETF
      Contact: WebTransport Working Group webtransport@ietf.org
         (mailto:webtransport@ietf.org)
      Notes: None

      The WT_MAX_STREAMS capsule.

      Value: 0x190B4D3F..0x190B4D40
      Capsule Type: WT_MAX_STREAMS
      Status: permanent
      Specification: This document
      Change Controller: IETF
      Contact: WebTransport Working Group webtransport@ietf.org
         (mailto:webtransport@ietf.org)
      Notes: None

      The WT_DATA_BLOCKED capsule.

      Value: 0x190B4D41
      Capsule Type: WT_DATA_BLOCKED
      Status: permanent
      Specification: This document

      Change Controller: IETF
      Contact: WebTransport Working Group webtransport@ietf.org
         (mailto:webtransport@ietf.org)
      Notes: None

      The WT_STREAM_DATA_BLOCKED capsule.

      Value: 0x190B4D42
      Capsule Type: WT_STREAM_DATA_BLOCKED
      Status: permanent
      Specification: This document
      Change Controller: IETF
      Contact: WebTransport Working Group webtransport@ietf.org
         (mailto:webtransport@ietf.org)
      Notes: None

      The WT_STREAMS_BLOCKED capsule.

      Value: 0x190B4D43..0x190B4D44
      Capsule Type: WT_STREAMS_BLOCKED
      Status: permanent
      Specification: This document
      Change Controller: IETF
      Contact: WebTransport Working Group webtransport@ietf.org
         (mailto:webtransport@ietf.org)
      Notes: None

9.3. HTTP Header Field Name

   IANA will register the following entry in the "Hypertext Transfer
   Protocol (HTTP) Field Name Registry" maintained at
   https://www.iana.org/assignments/http-fields
   (https://www.iana.org/assignments/http-fields):

   Field Name: WebTransport-Init

   Template: None

   Status: permanent

   Reference: This document

   Comments: None

10. References

10.1. Normative References

   [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>.

   [HTTP-DATAGRAM]
                 Schinazi, D. and L. Pardue, "HTTP Datagrams and the
                 Capsule Protocol", RFC 9297, DOI 10.17487/RFC9297, August
                 2022, <https://www.rfc-editor.org/rfc/rfc9297>.

   [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>.

   [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-
                 07, 4 March 2024, <https://datatracker.ietf.org/doc/html/
                 draft-ietf-webtrans-overview-07>.

   [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>.

   [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>.

   [RFC8941] Nottingham, M. and P. Kamp, "Structured Field Values for
              HTTP", RFC 8941, DOI 10.17487/RFC8941, February 2021,
              <https://www.rfc-editor.org/rfc/rfc8941>.

   [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>.

   [RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke,
              Ed., "HTTP Semantics", STD 97, RFC 9110,
              DOI 10.17487/RFC9110, June 2022,
              <https://www.rfc-editor.org/rfc/rfc9110>.

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

   [SEMANTICS]
              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>.

   [WEBTRANSPORT-H3]
              Frindell, A., Kinnear, E., and V. Vasiliev, "WebTransport
              over HTTP/3", Work in Progress, Internet-Draft, draft-
              ietf-webtrans-http3-08, 23 October 2023,
              <https://datatracker.ietf.org/doc/html/draft-ietf-
              webtrans-http3-08>.

10.2. Informative References

   [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>.

   [HTTP3]   Bishop, M., "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

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