WEBTRANS
Internet-Draft
Intended status: Standards Track
Expires: 23 April 2026

A. Frindell
Facebook Inc.
E. Kinnear
T. Pauly
Apple Inc.
M. Thomson
Mozilla
V. Vasiliev
Google
G. Xie
Facebook Inc.

20 October 2025

WebTransport over HTTP/2
draft-ietf-webtrans-http2-13

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.

About This Document

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

The latest revision of this draft can be found at https://ietf-wg-webtrans.github.io/draft-ietf-webtrans-http2/#go.draft-ietf-webtrans-http2.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-webtrans-http2/.

Discussion of this document takes place on the WebTransport Working Group mailing list (mailto:webtransport@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/webtransport/. Subscribe at https://www.ietf.org/mailman/listinfo/webtransport/.

Source for this draft and an issue tracker can be found at https://github.com/ietf-wg-webtrans/draft-ietf-webtrans-http2.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

This Internet-Draft will expire on 23 April 2026.

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].

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 and Termination

A WebTransport session is a communication context between a client and server [OVERVIEW]. This section describes how sessions begin and end.

3.1. Establishing a WebTransport-Capable HTTP/2 Connection

In order to indicate potential support for WebTransport, the server MUST send the SETTINGS_ENABLE_CONNECT_PROTOCOL setting with a value of "1" in its SETTINGS frame. The client MUST NOT send a WebTransport request until it has received the setting indicating extended CONNECT support from the server.

3.2. 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.

3.3. Application Protocol Negotiation

WebTransport over HTTP/2 offers a subprotocol negotiation mechanism, similar to TLS Application-Layer Protocol Negotiation Extension (ALPN) [RFC7301]; the intent is to simplify porting pre-existing protocols that rely on this type of functionality.

The user agent MAY include a WT-Available-Protocols header field in the CONNECT request. The WT-Available-Protocols enumerates the possible protocols in preference order. If the server receives such a header, it MAY include a WT-Protocol field in a successful (2xx) response. If it does, the server MUST include a single choice from the client's list in that field. Servers MAY reject the request if the client did not include a suitable protocol.

Both WT-Available-Protocols and WT-Protocol are defined in Section 3.4 of [WEBTRANSPORT-H3].

3.4. Session Termination and Error Handling

An WebTransport session over HTTP/2 is terminated when either endpoint closes the stream associated with the CONNECT request that initiated the session.

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

Session errors result in the termination of a session. Errors can be reported using the WT_CLOSE_SESSION capsule, which includes an error code and an optional explanatory message.

An endpoint can terminate a session without sending a WT_CLOSE_SESSION capsule by closing the HTTP/2 stream.

An HTTP/2 stream that is reset terminates the session without providing an application-level signal, though there will be an HTTP/2 error code.

This document reserves the following HTTP/2 error codes for use with reporting WebTransport errors:

WEBTRANSPORT_ERROR (0xTBD): This generic error can be used for errors that do not have more specific error codes.

WEBTRANSPORT_STREAM_STATE_ERROR (0xTBD): A stream-related capsule identified a stream that was in an invalid state.

Prior to terminating a stream with an error, a WT_CLOSE_SESSION capsule with an application-specified error code MAY be sent.

Session errors do not necessarily result in any change of HTTP/2 connection state, except that an endpoint might choose to terminate a connection in response to stream errors; see Section 5.4 of [HTTP2].

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 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.

4.1. Limiting the Number of Simultaneous Sessions

HTTP/2 defines a SETTINGS_MAX_CONCURRENT_STREAMS parameter Section 6.5.2 of [HTTP2] that allows the server to limit the maximum number of concurrent streams on a single HTTP/2 session, which also limits the number of WebTransport sessions on that connection. Servers that wish to limit the rate of incoming WebTransport sessions on any particular HTTP/2 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.

4.2. Limiting the Number of Streams Within a Session

The WT_MAX_STREAMS capsule (Section 5.6.1 of [WEBTRANSPORT-H3]) 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.

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 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 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.

4.3.1. Flow Control SETTINGS

Initial flow control limits can be exchanged via HTTP/2 SETTINGS (Section 11.1) by providing non-zero values for

* WT_MAX_DATA via SETTINGS_WT_INITIAL_MAX_DATA

* WT_MAX_STREAM_DATA via SETTINGS_WT_INITIAL_MAX_STREAM_DATA_UNI and SETTINGS_WT_INITIAL_MAX_STREAM_DATA_BIDI

* WT_MAX_STREAMS via SETTINGS_WT_INITIAL_MAX_STREAMS_UNI and SETTINGS_WT_INITIAL_MAX_STREAMS_BIDI

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 the WebTransport-Init field cannot be parsed correctly or does not have the correct type, the endpoint MUST reject the CONNECT request with a 4xx status code. 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.

If any of these keys are present but contain invalid values, the endpoint MUST reject the CONNECT request with a 4xx status code. Unknown keys and parameters in the dictionary MUST be ignored.

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.

5. 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.

5.1. 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.

Unreliable Delivery: WebTransport over HTTP/2 does not support unreliable delivery, 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: WebTransport over HTTP/2 provides support for pooling. Every WebTransport session is an independent HTTP/2 stream and does not compete with other pooled WebTransport sessions for per-stream resources.

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

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.

5.2. 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.

5.3. Use of Keying Material Exporters

WebTransport over HTTP/2 supports the use of TLS keying material exporters Section 7.5 of [TLS]. Since the underlying HTTP/2 connection could be shared by multiple WebTransport sessions, WebTransport defines a mechanism for deriving a TLS exporter that separates keying material for different sessions. If the application requests an exporter for a given WebTransport session with a specified label and context, the resulting exporter SHALL be a TLS exporter as defined in Section 7.5 of [TLS] with the label set to "EXPORTER-WebTransport" and the context set to the serialization of the "WebTransport Exporter Context" struct as defined below.

```
WebTransport Exporter Context {
  WebTransport Session ID (64),
  WebTransport Application-Supplied Exporter Label Length (8),
  WebTransport Application-Supplied Exporter Label (8..),
  WebTransport Application-Supplied Exporter Context Length (8),
  WebTransport Application-Supplied Exporter Context (..)
}
```

Figure 1: WebTransport Exporter Context struct

A TLS exporter API might permit the context field to be omitted. In this case, as with TLS 1.3, the WebTransport Application-Supplied Exporter Context becomes zero-length if omitted.

6. 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].

6.1. 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 intermediaries. The PADDING capsule cannot be smaller than its own header, which means that the minimum size of the capsule is two bytes: one byte for the Type and one byte to encode "0" for the Length.

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

Figure 2: 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. A receiver is not obligated to verify padding but MAY treat non-zero padding as a stream error (Section 3.4).

6.2. 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 in excess of the Reliable Size indicated, even if that data was already received.

The WT_RESET_STREAM capsule follows the design of the QUIC RESET_STREAM_AT frame [PARTIAL-RESET]. Consequently, it includes a Reliable Size field. A WT_RESET_STREAM capsule MUST be sent after

WT_STREAM capsules that include an amount of data equal to or in excess of the value in the Reliable Size field. A receiver MUST treat the receipt of a WT_RESET_STREAM with a Reliable Size smaller than the number of bytes it has received on the stream as a session error.

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

Figure 3: 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.

Reliable Size: A variable-length integer indicating the amount of data that needs to be delivered to the application even though the stream is reset.

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.

A WT_RESET_STREAM capsule MUST NOT be sent after a stream is closed or reset. While QUIC permits redundant RESET_STREAM frames, the ordering guarantee in HTTP/2 makes this unnecessary. A stream error (Section 3.4) of type WEBTRANSPORT_STREAM_STATE_ERROR MUST be sent if a WT_RESET_STREAM capsule is received for a stream that is not in a valid state.

6.3. 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 4: 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.

As defined in Section 3.5 of [RFC9000], the recipient of a WT_STOP_SENDING capsule sends a WT_RESET_STREAM capsule in response, including the same error code, if the stream is the "Ready" or "Send" state.

A WT_STOP_SENDING capsule MUST NOT be sent multiple times for the same stream. While QUIC permits redundant STOP_SENDING frames, the ordering guarantee in HTTP/2 makes this unnecessary. A stream error (Section 3.4) of type WEBTRANSPORT_STREAM_STATE_ERROR MUST be sent if a second WT_STOP_SENDING capsule is received.

6.4. 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 5: WT_STREAM Capsule Format

WT_STREAM capsules contain the following fields:

Stream ID: The stream ID for the stream. The second least significant bit of the Stream ID indicates whether the stream is bidirectional or unidirectional, as described in Section 5.2.

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.

A WT_STREAM capsule MUST NOT be sent after a stream is closed or reset. While QUIC permits redundant STREAM frames, the ordering guarantee in HTTP/2 makes this unnecessary. A stream error (Section 3.4) of type WEBTRANSPORT_STREAM_STATE_ERROR MUST be sent if a WT_STREAM capsule is received for a stream that is not in a valid state.

6.5. WT_MAX_DATA Capsule

An HTTP capsule [HTTP-DATAGRAM] called WT_MAX_DATA (type=0x190B4D3D) (Section 5.4.3 of [WEBTRANSPORT-H3]) is used 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 6: 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 4.4.

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

6.6. 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 7: 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 4.4.

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

A WT_MAX_STREAM_DATA capsule MUST NOT be sent after a sender requests that a stream be closed with WT_STOP_SENDING. While QUIC permits redundant MAX_STREAM_DATA frames, the ordering guarantee in HTTP/2 makes this unnecessary. A stream error (Section 3.4) of type WEBTRANSPORT_STREAM_STATE_ERROR MUST be sent if a WT_MAX_STREAM_DATA capsule is received after a WT_STOP_SENDING capsule for the same stream.

6.7. WT_MAX_STREAMS Capsule

An HTTP capsule [HTTP-DATAGRAM] called WT_MAX_STREAMS is defined by Section 5.6.1 of [WEBTRANSPORT-H3] 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 8: 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_WT_INITIAL_MAX_STREAMS_UNI and SETTINGS_WT_INITIAL_MAX_STREAMS_BIDI.

6.8. WT_DATA_BLOCKED Capsule

A sender SHOULD send a WT_DATA_BLOCKED capsule (type=0x190B4D41) (Section 5.6.4 of [WEBTRANSPORT-H3]) 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 9: 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]. Intermediaries MUST consume WT_DATA_BLOCKED capsules for flow control purposes and MUST generate and send appropriate flow control signals for their limits; see Section 4.4.

6.9. 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 10: 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]. Intermediaries 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 4.4.

A WT_STREAM_DATA_BLOCKED capsule MUST NOT be sent after a stream is closed or reset. While QUIC permits redundant STREAM_DATA_BLOCKED frames, the ordering guarantee in HTTP/2 makes this unnecessary. A stream error (Section 3.4) of type WEBTRANSPORT_STREAM_STATE_ERROR MUST be sent if a WT_STREAM_DATA_BLOCKED capsule is received for a stream that is not in a valid state.

6.10. WT_STREAMS_BLOCKED Capsule

A sender SHOULD send a WT_STREAMS_BLOCKED capsule (type=0x190B4D43 or 0x190B4D44) (Section 5.4.2 of [WEBTRANSPORT-H3]) 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 11: 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]. Intermediaries MUST consume WT_STREAMS_BLOCKED capsules for flow control purposes and MUST generate and send appropriate flow control signals for their limits.

6.11. 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 12: 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.

6.12. WT_CLOSE_SESSION Capsule

WebTransport over HTTP/2 uses the WT_CLOSE_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 WT_CLOSE_SESSION capsule and then by closing the HTTP/2 stream associated with the session (see Section 3.4).

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

Figure 13: WT_CLOSE_SESSION Capsule Format

When used in WebTransport over HTTP/2, WT_CLOSE_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 WT_CLOSE_SESSION capsule MUST then half-close the stream by sending an HTTP/2 frame with the END_STREAM flag set (Section 5.1 of [HTTP2]). The recipient MUST close the stream upon receipt of the capsule by replying with an HTTP/2 frame with the END_STREAM flag set; note that it does not need to send a WT_CLOSE_SESSION capsule in response.

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

6.13. WT_DRAIN_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 WT_DRAIN_SESSION capsule defined in Section 4.6 of [WEBTRANSPORT-H3] to gracefully shut down a WebTransport session.

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

Figure 14: WT_DRAIN_SESSION Capsule Format

After sending or receiving either a WT_DRAIN_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.

6.14. Capsule Ordering and Reliability

The use of an ordered and reliable transport means that a receiver does not need to tolerate capsules that arrive out of order. This differs from QUIC in that a receiver is required to treat the arrival of out of order frames rather than being tolerant.

For an intermediary that forwards from an strongly-ordered transport (like [WEBTRANSPORT-H3]) to a reliable transport (like this protocol), it is necessary to maintain state for streams. A simple forwarding intermediary that directly translates one type of protocol unit into another without understanding the underlying state might cause a receiver to abort the session.

For instance, after a RESET_STREAM frame is forwarded, an intermediary cannot forward a RESET_STREAM frame as a WT_RESET_STREAM capsule or a STREAM frame as a WT_STREAM capsule without error.

7. Requirements on TLS Usage

Because TLS keying material exporters are only secure for authentication when they are uniquely bound to the TLS session [RFC7627], WebTransport requires either one of the following conditions:

* The TLS version in use is greater than or equal to 1.3 [TLS].

* The TLS version in use is 1.2, and the extended master secret extension [RFC7627] has been negotiated.

Clients MUST NOT send WebTransport over HTTP/2 requests on connections that do not meet one of the two conditions above. If a server receives a WebTransport over HTTP/2 request on a connection that meets neither, the server MUST treat the request as malformed, as specified in Section 8.1.1 of [HTTP2].

8. 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
                                     SETTINGS_ENABLE_CONNECT_PROTOCOL = 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 = 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
                                     SETTINGS_ENABLE_CONNECT_PROTOCOL = 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 = 1
                                     WebTransport Data

WT_STREAM + FIN
Stream ID = 1
WebTransport Data

                                     WT_STREAM + FIN
                                     Stream ID = 1
                                     WebTransport Data
```

9. Considerations for Future Versions

Future versions of WebTransport that change the syntax of the CONNECT requests used to establish WebTransport sessions will need to modify the upgrade token used to identify WebTransport, allowing servers to offer multiple versions simultaneously (Section 9.1 of [WEBTRANSPORT-H3]).

10. 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.

11. IANA Considerations

This document registers new HTTP/2 settings (Section 11.1), HTTP/2 error codes (Section 11.2), new capsules (Section 11.3), and the WebTransport-Init header field (Section 11.4).

11.1. HTTP/2 SETTINGS Parameter Registration

The following entries are added to the "HTTP/2 Settings" registry established by [HTTP2]:

The SETTINGS_WT_INITIAL_MAX_DATA parameter indicates the initial value for the session data limit, otherwise communicated by the WT_MAX_DATA capsule (Section 6.5). The default value for the SETTINGS_WT_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_WT_INITIAL_MAX_DATA

Value: 0x2b61

Default: 0

Specification: This document

The SETTINGS_WT_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(Section 6.6). The default value for the SETTINGS_WT_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_WT_INITIAL_MAX_STREAM_DATA_UNI

Value: 0x2b62

Default: 0

Specification: This document

The SETTINGS_WT_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 (Section 6.6). The default value for the SETTINGS_WT_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_WT_INITIAL_MAX_STREAM_DATA_BIDI

Value: 0x2b63

Default: 0

Specification: This document

The SETTINGS_WT_INITIAL_MAX_STREAMS_UNI parameter indicates the initial value for the unidirectional max stream limit, otherwise communicated by the WT_MAX_STREAMS capsule (Section 6.7). The default value for the SETTINGS_WT_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_WT_INITIAL_MAX_STREAMS_UNI

Value: 0x2b64

Default: 0

Specification: This document

The SETTINGS_WT_INITIAL_MAX_STREAMS_BIDI parameter indicates the initial value for the bidirectional max stream limit, otherwise communicated by the WT_MAX_STREAMS capsule (Section 6.7). The default value for the SETTINGS_WT_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_WT_INITIAL_MAX_STREAMS_BIDI

Value: 0x2b65

Default: 0

Specification: This document

11.2. HTTP/2 Error Code Registration

The following entries are added to the "HTTP/2 Error Code" registry established by [HTTP2]:

For WEBTRANSPORT_ERROR:

Code: 0xTBD

Name: WEBTRANSPORT_ERROR

Description: General WebTransport error detected

Reference: Section 3.4

For WEBTRANSPORT_STREAM_STATE_ERROR:

Code: 0xTBD

Name: WEBTRANSPORT_STREAM_STATE_ERROR

Description: Unexpected WebTransport stream-related capsule received

Reference: Section 3.4

11.3. 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_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

11.4. 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

12. References

12.1. 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] Kinnear, E. and V. Vasiliev, "The WebTransport Protocol Framework", Work in Progress, Internet-Draft, draft-ietf-webtrans-overview-11, 20 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-webtrans-overview-11>.

[PARTIAL-RESET] Seemann, M. and K. Oku, "QUIC Stream Resets with Partial Delivery", Work in Progress, Internet-Draft, draft-ietf-quic-reliable-stream-reset-07, 14 June 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-reliable-stream-reset-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>.

[RFC7627] Bhargavan, K., Ed., Delignat-Lavaud, A., Pironti, A., Langley, A., and M. Ray, "Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension", RFC 7627, DOI 10.17487/RFC7627, September 2015, <https://www.rfc-editor.org/rfc/rfc7627>.

[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>.

[TLS] 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>.

[WEBTRANSPORT-H3]
Frindell, A., Kinnear, E., and V. Vasiliev, "WebTransport over HTTP/3", Work in Progress, Internet-Draft, draft-ietf-webtrans-http3-13, 7 July 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-webtrans-http3-13>.

12.2. 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>.

[RFC7301] 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>.

[RFC9729] Schinazi, D., Oliver, D., and J. Hoyland, "The Concealed HTTP Authentication Scheme", RFC 9729, DOI 10.17487/RFC9729, February 2025, <https://www.rfc-editor.org/rfc/rfc9729>.

Acknowledgments

Thanks to Anthony Chivetta, Eric Gorbaty, Ankshit Jain, Joshua Otto, and Valentin Pistol for their contributions in the design and implementation of this work. The requirements on TLS usage were inspired by [RFC9729].

Index

1; Section 6.2, Paragraph 2; Section 6.2, Paragraph 3;
Section 6.2, Paragraph 5; Section 6.2, Paragraph 8;
Section 6.3, Paragraph 5; Section 6.14, Paragraph 3;
Section 11.3, Paragraph 5.4.1

WT_STOP_SENDING Section 2, Paragraph 7; Section 6.3, Paragraph 1; Section 6.3, Paragraph 3; Section 6.3, Paragraph 5; Section 6.3, Paragraph 6; Section 6.6, Paragraph 8; Section 11.3, Paragraph 7.4.1

WT_STREAM Section 2, Paragraph 5; Section 5.1, Paragraph 2; Section 6.2, Paragraph 2; Section 6.2, Paragraph 3; Section 6.2, Paragraph 7; Section 6.4, Paragraph 1; Section 6.4, Paragraph 2; Section 6.4, Paragraph 4; Section 6.4, Paragraph 5.4.1; Section 6.4, Paragraph 6; Section 6.5, Paragraph 5; Section 6.6, Paragraph 5; Section 6.14, Paragraph 3; Section 8, Paragraph 3; Section 11.3, Paragraph 9.4.1

WT_STREAM_DATA_BLOCKED Section 2, Paragraph 6; Section 4.4, Paragraph 1; Section 6.9, Paragraph 1; Section 6.9, Paragraph 3; Section 6.9, Paragraph 5; Section 6.9, Paragraph 6

WT_STREAMS_BLOCKED Section 2, Paragraph 6; Section 4.4, Paragraph 1; Section 6.10, Paragraph 1; Section 6.10, Paragraph 2; Section 6.10, Paragraph 4; Section 6.10, Paragraph 6

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