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

A. Frindell
Facebook
E. Kinnear
Apple Inc.
V. Vasiliev
Google

20 October 2025

WebTransport over HTTP/3
draft-ietf-webtrans-http3-14

Abstract

WebTransport [OVERVIEW] is a protocol framework that enables application clients constrained by the Web security model to communicate with a remote application server using a secure multiplexed transport. This document describes a WebTransport protocol that is based on HTTP/3 [HTTP3] and provides support for unidirectional streams, bidirectional streams, and datagrams, all multiplexed within the same HTTP/3 connection.

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-http3/#go.draft-ietf-webtrans-http3.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-webtrans-http3/.

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-http3.

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/.

Copyright Notice

Table of Contents

1. Introduction

HTTP/3 [HTTP3] is a protocol defined on top of QUIC [RFC9000] that can multiplex HTTP requests over a QUIC connection. This document defines a mechanism for multiplexing non-HTTP data with HTTP/3 in a manner that conforms with the WebTransport protocol requirements and semantics [OVERVIEW]. Using the mechanism described here, multiple WebTransport instances, or sessions, can be multiplexed simultaneously with regular HTTP traffic on the same HTTP/3 connection.

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/3 server. An HTTP/3 server is the server that terminates HTTP/3 connections; a WebTransport server is an application that accepts WebTransport sessions, which can be accessed via an HTTP/3 server. An application client is user or developer-provided code, often untrusted, that utilizes the interface offered by the WebTransport client to communicate with an application server. The application server uses the interface offered by the WebTransport server to accept incoming WebTransport sessions.

2. Overview

2.1. QUIC, WebTransport, and HTTP/3

QUIC version 1 [RFC9000] is a secure transport protocol with flow control and congestion control. QUIC supports application data exchange via streams; reliable and ordered byte streams that can be multiplexed. Stream independence can mitigate head-of-line blocking. While QUIC provides streams as a transport service, it is unopinionated about their usage. The applicability of streams is described by section 4 of [RFC9308].

HTTP is an application-layer protocol, defined by "HTTP Semantics" [RFC9110]. HTTP/3 is the application mapping for QUIC, defined in [RFC9114]. It describes how QUIC streams are used to carry control data or HTTP request and response message sequences in the form of frames and describes details of stream and connection lifecycle management. HTTP/3 offers two features in addition to HTTP Semantics: QPACK header compression [RFC9208] and Server Push Section 4.6 of [RFC9114].

WebTransport session establishment involves interacting at the HTTP layer with a resource. For Web user agents and other WebTransport clients, this interaction is important for security reasons, especially to ensure that the resource is willing to use WebTransport.

Although WebTransport requires HTTP for its handshake, when HTTP/3 is in use, HTTP is not used for anything else related to an established session. Instead, QUIC streams begin with a sequence of header bytes that links them to the established session. The remainder of the stream is the body, which carries the payload supplied by the application using WebTransport. This process is similar to WebSockets over HTTP/1.1 [ORIGIN], where access to the underlying byte stream is enabled after both sides have completed an initial handshake.

The layering of QUIC, HTTP/3, and WebTransport is shown in Figure 1. Once a WebTransport session is established, applications have nearly direct access to QUIC.

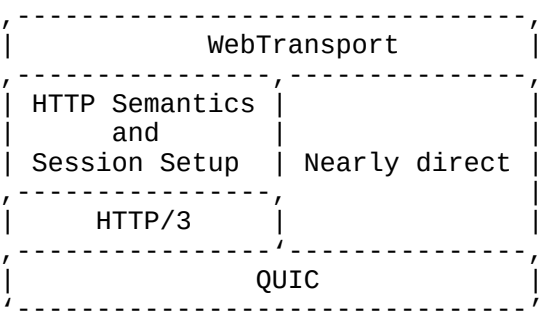

Figure 1: WebTransport Layering

2.1.1. Minimizing Implementation Complexity

WebTransport has minimal interaction with HTTP and HTTP/3. Clients and servers can constrain their use of features to only those required to complete a WebTransport handshake:

* Generating/parsing the request method, host, path, protocol, optional Origin header field, and perhaps some extra header fields.

* Generating/parsing the response status code and possibly some extra header fields.

A WebTransport endpoint, whether a client or a server, can likely perform several of its HTTP-level requirements using bytestring comparisons.

While HTTP/3 encodes HTTP messages using QPACK, this complexity can be minimized. When receiving, a WebTransport endpoint can disable dynamic decompression entirely but must always support static decompression and Huffman decoding. When sending, endpoints can opt to never use dynamic compression, static compression, or Huffman encoding.

2.2. Protocol Overview

WebTransport servers in general are identified by a pair of authority value and path value (defined in [RFC3986] Sections 3.2 and 3.3 correspondingly).

When an HTTP/3 connection is established, both the client and the server send a SETTINGS_WT_MAX_SESSIONS setting to indicate support for WebTransport over HTTP/3. This process also negotiates the use of additional HTTP/3 extensions to enable both endpoints to open WebTransport streams.

WebTransport sessions are initiated inside a given HTTP/3 connection by the client, who sends an extended CONNECT request [RFC9220]. If the server accepts the request, a WebTransport session is established. The resulting stream will be further referred to as a _CONNECT stream_, and its stream ID is used to uniquely identify a given WebTransport session within the connection. The ID of the CONNECT stream that established a given WebTransport session will be further referred to as a _Session ID_.

After the session is established, the endpoints can exchange data using the following mechanisms:

* A client can create a bidirectional stream and transfer its ownership to WebTransport by providing a special signal in the first bytes.

* A server can create a bidirectional stream and transfer its ownership to WebTransport by providing a special signal in the first bytes.

* Both client and server can create a unidirectional stream using a special stream type.

* Both client and server can send datagrams using HTTP Datagrams [HTTP-DATAGRAM].

A WebTransport session is terminated when the CONNECT stream that created it is closed.

3. Session Establishment

3.1. Establishing a WebTransport-Capable HTTP/3 Connection

A WebTransport-Capable HTTP/3 connection requires the client and server to both signal support for WebTransport over HTTP/3 using a setting.

This document defines a SETTINGS_WT_MAX_SESSIONS setting for indicating the number of WebTransport sessions a connection supports. The default value for the SETTINGS_WT_MAX_SESSIONS setting is "0", meaning that the endpoint is not willing to receive any WebTransport sessions. Both clients and servers supporting WebTransport over

HTTP/3 MUST send the SETTINGS_WT_MAX_SESSIONS setting with a value greater than "0". Clients MUST NOT attempt to establish WebTransport sessions until they have received the setting indicating WebTransport support from the server.

WebTransport over HTTP/3 uses extended CONNECT in HTTP/3 as described in [RFC9220], which defines the SETTINGS_ENABLE_CONNECT_PROTOCOL setting. Clients also signal support for WebTransport by using the "webtransport" upgrade token in extended CONNECT requests when establishing sessions (see Section 9.1).

WebTransport over HTTP/3 requires support for HTTP/3 datagrams and the Capsule Protocol, and both the client and the server indicate support for HTTP/3 datagrams by sending a SETTINGS_H3_DATAGRAM setting value set to 1 in their SETTINGS frame (see Section 2.1.1 of [HTTP-DATAGRAM]).

WebTransport over HTTP/3 also requires support for QUIC datagrams. To indicate support, both the client and the server send a max_datagram_frame_size transport parameter with a value greater than 0 (see Section 3 of [QUIC-DATAGRAM]).

WebTransport over HTTP/3 relies on the RESET_STREAM_AT frame defined in [RESET-STREAM-AT]. To indicate support, both the client and the server enable the extension by sending an empty reset_stream_at transport parameter as described in Section 3 of [RESET-STREAM-AT].

In summary, servers supporting WebTransport over HTTP/3 send:

* A SETTINGS_WT_MAX_SESSIONS setting with a value greater than "0"
* A SETTINGS_ENABLE_CONNECT_PROTOCOL setting with a value of "1"
* A SETTINGS_H3_DATAGRAM setting with a value of 1
* A max_datagram_frame_size transport parameter with a value greater than 0
* An empty reset_stream_at transport parameter

Clients supporting WebTransport over HTTP/3 send:

* A SETTINGS_WT_MAX_SESSIONS setting with a value greater than "0"
* A SETTINGS_H3_DATAGRAM setting with a value of 1
* A max_datagram_frame_size transport parameter with a value greater than 0

* An empty reset_stream_at transport parameter

Servers should note that CONNECT requests to establish new WebTransport sessions, in addition to other messages, can arrive before the client's SETTINGS are received (see Section 4.6). If the server receives SETTINGS that do not have correct values for every required setting, or transport parameters that do not have correct values for every required transport parameter, the server MUST treat all established and newly incoming WebTransport sessions as malformed, as described in Section 4.1.2 of [HTTP3].

A client MUST NOT establish WebTransport sessions if the server's SETTINGS do not have correct values for every required setting or if the server's transport parameters do not have correct values for every required transport parameter.

[[RFC editor: please remove the following paragraph before publication.]]

For draft versions of WebTransport only, the server MUST NOT process any incoming WebTransport requests until the client settings have been received, as the client might be using a version of the WebTransport extension that is different from the one used by the server.

3.2. Creating a New Session

As WebTransport sessions are established over HTTP/3, they are identified using the https URI scheme (Section 4.2.2 of [HTTP]).

In order to create a new WebTransport session, a WebTransport client sends an HTTP extended CONNECT request. In this request:

* The :protocol pseudo-header field([RFC8441]) MUST be set to webtransport.

* The :scheme field MUST be https.

* Both the :authority and the :path value MUST be set; these fields identify the desired WebTransport server resource.

* If the WebTransport session is coming from a browser client, an Origin header [RFC6454] MUST be provided within the request. Otherwise, the header is OPTIONAL.

Upon receiving an extended CONNECT request with a :protocol field set to webtransport, the HTTP/3 server can check if it has a WebTransport server associated with the specified :authority and :path values. If

it does not, it SHOULD reply with status code 404 (Section 15.5.5 of [HTTP]). When the request contains the Origin header, the WebTransport server MUST verify the Origin header to ensure that the specified origin is allowed to access the server in question. If the verification fails, the WebTransport server SHOULD reply with status code 403 (Section 15.5.4 of [HTTP]). If all checks pass, the WebTransport server MAY accept the session by replying with a 2xx series status code, as defined in Section 15.3 of [HTTP].

From the client's perspective, a WebTransport session is established when the client receives a 2xx response. From the server's perspective, a session is established once it sends a 2xx response.

The server may reply with a 3xx response, indicating a redirection (Section 15.4 of [HTTP]). The WebTransport client MUST NOT automatically follow such redirects, as it potentially could have already sent data for the WebTransport session in question; it MAY notify the application client about the redirect.

Clients cannot initiate WebTransport in 0-RTT packets, as the CONNECT method is not considered safe (see Section 10.9 of [HTTP3]). However, WebTransport-related SETTINGS parameters may be retained from the previous session as described in Section 7.2.4.2 of [HTTP3]. If the server accepts 0-RTT, the server MUST NOT reduce the limit of maximum open WebTransport sessions, or other initial flow control values, from the values negotiated during the previous session; such change would be deemed incompatible, and MUST result in a H3_SETTINGS_ERROR connection error.

The webtransport HTTP Upgrade Token uses the Capsule Protocol as defined in [HTTP-DATAGRAM]. The Capsule Protocol is negotiated when the server sends a 2xx response. The capsule-protocol header field Section 3.4 of [HTTP-DATAGRAM] is not required by WebTransport and can safely be ignored by WebTransport endpoints.

3.3. Application Protocol Negotiation

WebTransport over HTTP/3 offers a protocol negotiation mechanism, similar to TLS Application-Layer Protocol Negotiation Extension (ALPN) [RFC7301]; the intent is to simplify porting existing protocols that use QUIC and rely on this functionality.

The client MAY include a WT-Available-Protocols header field in the CONNECT request. The WT-Available-Protocols field enumerates the possible protocols in preference order, with the most preferred protocol listed first. 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 Structured Fields [FIELDS]. WT-Available-Protocols is a List. WT-Protocol is defined as an Item. In both cases, the only valid value type is a String. Any value type other than String MUST be treated as an error that causes the entire field to be ignored. No semantics are defined for parameters on either field; parameters MUST be ignored.

The value in the WT-Protocol response header field MUST be one of the values listed in WT-Available-Protocols of the request. Otherwise, the WT-Protocol field MUST be ignored.

The semantics of individual values used in WT-Available-Protocols and WT-Protocol are determined by the WebTransport resource in question and are not required to be registered in IANA's "ALPN Protocol IDs" registry.

3.4. Prioritization

WebTransport sessions are initiated using extended CONNECT. While Section 11 of [RFC9218] describes how extensible priorities can be applied to data sent on a CONNECT stream, WebTransport extends the types of data that are exchanged in relation to the request and response, which requires additional considerations.

WebTransport CONNECT requests and responses MAY contain the Priority header field (Section 5 of [RFC9218]); clients MAY reprioritize by sending PRIORITY_UPDATE frames (Section 7 of [RFC9218]). In extension to [RFC9218], it is RECOMMENDED that clients and servers apply the scheduling guidance in both Section 9 of [RFC9218] and Section 10 of [RFC9218] for all data that they send in the enclosing WebTransport session, including Capsules, WebTransport streams and datagrams. WebTransport does not provide any priority signaling mechanism for streams and datagrams within a WebTransport session; such mechanisms can be defined by application protocols using WebTransport. It is RECOMMENDED that such mechanisms only affect scheduling within a session and not scheduling of other data on the same HTTP/3 connection.

The client/server priority merging guidance in Section 8 of [RFC9218] also applies to WebTransport sessions. For example, a client that receives a response Priority header field could alter its view of a WebTransport session priority and alter the scheduling of outgoing data as a result.

Endpoints that prioritize WebTransport sessions need to consider how they interact with other sessions or requests on the same HTTP/3 connection.

4. WebTransport Features

WebTransport over HTTP/3 provides the following features described in [OVERVIEW]: unidirectional streams, bidirectional streams, and datagrams, all of which can be initiated by either endpoint. Protocols designed for use with WebTransport over HTTP/3 are constrained to these features. The Capsule Protocol is an implementation detail of WebTransport over HTTP/3 and is not a WebTransport feature.

Session IDs are used to demultiplex streams and datagrams belonging to different WebTransport sessions. On the wire, session IDs are encoded using the QUIC variable length integer scheme described in [RFC9000].

The client MAY optimistically open unidirectional and bidirectional streams, as well as send datagrams, on a session for which it has sent the CONNECT request, even if it has not yet received the server’s response to the request. On the server side, opening streams and sending datagrams is possible as soon as the CONNECT request has been received.

If at any point a session ID is received that cannot be a valid ID for a client-initiated bidirectional stream, the recipient MUST close the connection with an H3_ID_ERROR error code.

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.

Below are details about support in WebTransport over HTTP/3 for the properties defined by the WebTransport framework.

Unreliable Delivery: WebTransport over HTTP/3 supports unreliable

delivery. Resetting a stream results in lost stream data no longer being retransmitted. WebTransport over HTTP/3 also supports datagrams, which are not retransmitted.

Pooling: WebTransport over HTTP/3 provides optional support for pooling. Endpoints can use the SETTINGS_WT_MAX_SESSIONS setting to indicate if pooling is supported on a particular HTTP/3 connection (Section 3.1).

4.2. Unidirectional streams

WebTransport endpoints can initiate unidirectional streams. The HTTP/3 unidirectional stream type SHALL be 0x54. The body of the stream SHALL be the stream type, followed by the session ID, encoded as a variable-length integer, followed by the user-specified stream data (Figure 2).

```
Unidirectional Stream {
    Stream Type (i) = 0x54,
    Session ID (i),
    User-Specified Stream Data (..)
}
```

Figure 2: Unidirectional WebTransport stream format

4.3. Bidirectional Streams

All client-initiated bidirectional streams are reserved by HTTP/3 as request streams, which are a sequence of HTTP/3 frames with a variety of rules (see Sections 4.1 and 6.1 of [HTTP3]).

WebTransport extends HTTP/3 to allow clients to declare and to use alternative request stream rules. Once a client receives settings indicating WebTransport support (Section 3.1), it MUST send a special signal value, encoded as a variable-length integer, as the first bytes of each bidirectional WebTransport stream it initiates to indicate how the remaining bytes on the stream are used.

WebTransport extends HTTP/3 by defining rules for all server-initiated bidirectional streams. Once a server receives an incoming CONNECT request establishing a WebTransport session (Section 3.1), it can open a bidirectional stream for use with that session and MUST send a special signal value, encoded as a variable-length integer, as the first bytes of the stream in order to indicate how the remaining bytes on the stream are used.

Clients and servers use the signal value 0x41 to open a bidirectional WebTransport stream. Following this is the associated session ID, encoded as a variable-length integer; the rest of the stream is the application payload of the WebTransport stream (Figure 3).

```
Bidirectional Stream {
    Signal Value (i) = 0x41,
    Session ID (i),
    Stream Body (..)
}
```

Figure 3: Bidirectional WebTransport stream format

This document reserves the special signal value 0x41 as a WT_STREAM frame type. While it is registered as an HTTP/3 frame type to avoid collisions, WT_STREAM lacks length and is not a proper HTTP/3 frame; it is an extension of HTTP/3 frame syntax that MUST be supported by any peer negotiating WebTransport. Endpoints that implement this extension are also subject to additional frame handling requirements. Endpoints MUST NOT send WT_STREAM as a frame type on HTTP/3 streams other than the very first bytes of a request stream. Receiving this frame type in any other circumstances MUST be treated as a connection error of type H3_FRAME_ERROR.

4.4. Resetting Data Streams

A WebTransport endpoint may send a RESET_STREAM or a STOP_SENDING frame for a WebTransport data stream. Those signals are propagated by the WebTransport implementation to the application.

A WebTransport application MUST provide an error code for those operations. Since WebTransport shares the error code space with HTTP/3, WebTransport application errors for streams are limited to an unsigned 32-bit integer, assuming values between 0x00000000 and 0xffffffff. WebTransport implementations MUST remap those error codes into the error range reserved for WT_APPLICATION_ERROR, where 0x00000000 corresponds to 0x52e4a40fa8db, and 0xffffffff corresponds to 0x52e5ac983162. Note that there are codepoints inside that range of form "0x1f * N + 0x21" that are reserved by Section 8.1 of [HTTP3]; those have to be skipped when mapping the error codes (i.e., the two HTTP/3 error codepoints adjacent to a reserved codepoint would map to two adjacent WebTransport application error codepoints). An example pseudocode can be seen in Figure 4.

```
first = 0x52e4a40fa8db
last = 0x52e5ac983162

def webtransport_code_to_http_code(n):
    return first + n + floor(n / 0x1e)

def http_code_to_webtransport_code(h):
    assert(first <= h <= last)
    assert((h - 0x21) % 0x1f != 0)
    shifted = h - first
    return shifted - floor(shifted / 0x1f)
```

Figure 4: Pseudocode for converting between WebTransport application errors and HTTP/3 error codes

WebTransport data streams are associated with sessions through a header at the beginning of the stream; resetting a stream might result in that data being discarded when using a RESET_STREAM frame. To prevent this, WebTransport implementations MUST use the RESET_STREAM_AT frame [RESET-STREAM-AT] with a Reliable Size set to at least the size of the WebTransport header when resetting a WebTransport data stream. This ensures reliable delivery of the ID field associating the data stream with a WebTransport session.

WebTransport endpoints MUST forward the error code for a stream associated with a known session to the application that owns that session; similarly, intermediaries MUST reset such streams with a corresponding error code when receiving a reset from their peer.

4.5. Datagrams

Datagrams can be sent using HTTP Datagrams. The WebTransport datagram payload is sent unmodified in the "HTTP Datagram Payload" field of an HTTP Datagram (Section 2.1 of [HTTP-DATAGRAM]). Note that the payload field directly follows the Quarter Stream ID field, which is at the start of the QUIC DATAGRAM frame payload and refers to the CONNECT stream that established the WebTransport session.

4.6. Buffering Incoming Streams and Datagrams

In WebTransport over HTTP/3, the client MUST wait for receipt of the server's SETTINGS frame before establishing any WebTransport sessions by sending CONNECT requests using the WebTransport upgrade token (see Section 3.1). This ensures that the client will always know what versions of WebTransport can be used on a given HTTP/3 connection.

Clients can, however, send a SETTINGS frame, multiple WebTransport CONNECT requests, WebTransport data streams, and WebTransport datagrams all within a single flight. As those can arrive out of order, a WebTransport server could be put into a situation where it receives a stream or a datagram without a corresponding session. Similarly, a client may receive a server-initiated stream or a datagram before receiving the CONNECT response headers from the server.

To handle this case, WebTransport endpoints SHOULD buffer streams and datagrams until they can be associated with an established session. To avoid resource exhaustion, endpoints MUST limit the number of buffered streams and datagrams. When the number of buffered streams is exceeded, a stream SHALL be closed by sending a RESET_STREAM and/or STOP_SENDING with the WT_BUFFERED_STREAM_REJECTED error code. When the number of buffered datagrams is exceeded, a datagram SHALL be dropped. It is up to an implementation to choose what stream or datagram to discard.

4.7. Interaction with the HTTP/3 GOAWAY frame

HTTP/3 defines a graceful shutdown mechanism (Section 5.2 of [HTTP3]) that allows a peer to send a GOAWAY frame indicating that it will no longer accept any new incoming requests or pushes.

A client receiving GOAWAY cannot initiate CONNECT requests for new WebTransport sessions on that HTTP/3 connection; it must open a new HTTP/3 connection to initiate new WebTransport sessions with the same peer.

An HTTP/3 GOAWAY frame is also a signal to applications to initiate shutdown for all WebTransport sessions. To shut down a single WebTransport session, either endpoint can send a WT_DRAIN_SESSION (0x78ae) capsule.

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

Figure 5: WT_DRAIN_SESSION Capsule Format

After sending or receiving either a WT_DRAIN_SESSION capsule or a HTTP/3 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.

The WT_DRAIN_SESSION capsule is useful when an end-to-end WebTransport session passes through an intermediary. For example, when the backend shuts down, it sends a GOAWAY to the intermediary. The intermediary can convert this signal to a WT_DRAIN_SESSION capsule on the client-facing session, without impacting other requests or sessions carried on that connection.

4.8. Use of Keying Material Exporters

WebTransport over HTTP/3 supports the use of TLS keying material exporters Section 7.5 of [RFC8446]. Since the underlying QUIC connection may 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 [RFC8446] 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 6: 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.

5. Flow Control

Flow control governs the amount of resources that can be consumed or data that can be sent. When using WebTransport over HTTP/3, endpoints can limit the number of sessions that a peer can create on a single HTTP/3 connection and the number of streams that a peer can create within a session. Endpoints can also limit the amount of data that can be consumed by each session and by each stream within a session.

WebTransport over HTTP/3 provides a connection-level limit that governs the number of sessions that can be created on an HTTP/3 connection (see Section 5.2). It also provides session-level limits that govern the number of streams that can be created in a session and limit the amount of data that can be exchanged across all streams in each session (see Section 5.3).

The underlying QUIC connection provides connection and stream level flow control. The QUIC connection data limit defines the total amount of data that can be sent across all WebTransport sessions and other non-WebTransport streams. A QUIC stream's data limit controls the amount of data that can be sent on that stream, WebTransport or otherwise (see Section 4 of [RFC9000]).

5.1. Negotiating the Use of Flow Control

A WebTransport endpoint that allows a WebTransport session to share an underlying transport connection with other WebTransport sessions MUST enable flow control. This prevents an application from consuming excessive resources on a single session and starving traffic for other sessions (see Section 8).

Flow control is enabled when both endpoints declare their intent to use flow control, even if SETTINGS_WT_MAX_SESSIONS is sent with a value of "1". Endpoints declare their intent to use flow control by taking any of the following actions:

* Sending SETTINGS_WT_MAX_SESSIONS with a value greater than "1".
* Sending SETTINGS_WT_INITIAL_MAX_STREAMS_UNI with any value other than "0".
* Sending SETTINGS_WT_INITIAL_MAX_STREAMS_BIDI with any value other than "0".
* Sending SETTINGS_WT_INITIAL_MAX_DATA with any value other than "0".

If both endpoints take at least one of these actions, flow control is enabled, and the limits described in the entirety of Section 5 apply.

The inclusion of the flow control SETTINGS in these criteria allows endpoints to agree to explicitly enable flow control, even if only a single WebTransport session is supported.

If flow control is not enabled, clients MUST NOT attempt to establish more than one simultaneous WebTransport session. A server that receives more than one session on an underlying transport connection when flow control is not enabled MUST reset the excessive CONNECT streams with a H3_REQUEST_REJECTED status (see Section 5.2).

Also, if flow control is not enabled, an endpoint MUST ignore receipt of any flow control capsules (see Section 5.6), since the peer might not have received SETTINGS at the time they were sent or packets might have been reordered.

5.2. Limiting the Number of Simultaneous Sessions

This document defines a SETTINGS_WT_MAX_SESSIONS setting that allows the server to limit the maximum number of concurrent WebTransport sessions on a single HTTP/3 connection. The client MUST NOT open more simultaneous sessions than indicated in the server SETTINGS parameter. 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 H3_REQUEST_REJECTED status defined in Section 8.1 of [HTTP3].

5.3. Limiting the Number of Streams Within a Session

The WT_MAX_STREAMS capsule (Section 5.6.2) establishes a limit on the number of streams within a WebTransport session. Like the QUIC MAX_STREAMS frame (Section 19.11 of [RFC9000]), this capsule has two types that provide separate limits for unidirectional and bidirectional streams that a peer initiates.

Note that the CONNECT stream for the session is not included in either the bidirectional or the unidirectional stream limits; the number of CONNECT streams a client can open is limited by the SETTINGS_WT_MAX_SESSIONS setting and QUIC flow control’s stream limits.

The session-level stream limit applies in addition to the QUIC MAX_STREAMS frame, which provides a connection-level stream limit. New streams can only be created within the session if both the stream- and the connection-level limit permit, see Section 4.6 of [RFC9000] for details on how QUIC stream limits are applied.

Unlike the the QUIC MAX_STREAMS frame, there is no simple relationship between the value in this frame and stream IDs in QUIC STREAM frames. This especially applies if there are other users of streams on the connection.

The WT_STREAMS_BLOCKED capsule (Section 5.6.3) can be sent to indicate that an endpoint was unable to create a stream due to the session-level stream limit.

Note that enforcing this limit requires reliable resets for stream headers so that both endpoints can agree on the number of streams that are open.

5.4. Data Limits

The WT_MAX_DATA capsule (Section 5.6.4) establishes a limit on the amount of data that can be sent within a WebTransport session. This limit counts all data that is sent on streams of the corresponding type, excluding the stream header (see Section 4.2 and Section 4.3). The stream header is excluded from this limit so that this limit does not prevent the sending of information that is essential in linking new streams to a specific WebTransport session.

For streams that were reset, implementing WT_MAX_DATA requires that the QUIC stack provide the WebTransport implementation with information about the final size of streams; see Section 4.5 of [RFC9000]. This guarantees that both endpoints agree on how much WebTransport session flow control credit was consumed by the sender on that stream.

The WT_DATA_BLOCKED capsule (Section 5.6.5) can be sent to indicate that an endpoint was unable to send data due to a limit set by the WT_MAX_DATA capsule.

Because WebTransport over HTTP/3 uses a native QUIC stream for each WebTransport stream, per-stream data limits are provided by QUIC natively (see Section 4.1 of [RFC9000]). The WT_MAX_STREAM_DATA and WT_STREAM_DATA_BLOCKED capsules (Part XX of [I-D.ietf-webtrans-http2]) are not used and so are prohibited. Endpoints MUST treat receipt of a WT_MAX_STREAM_DATA or a WT_STREAM_DATA_BLOCKED capsule as a session error.

5.5. Flow Control SETTINGS

Initial flow control limits can be exchanged via HTTP/3 SETTINGS (Section 9.2) by providing non-zero values for

* WT_MAX_STREAMS via SETTINGS_WT_INITIAL_MAX_STREAMS_UNI and SETTINGS_WT_INITIAL_MAX_STREAMS_BIDI

* WT_MAX_DATA via SETTINGS_WT_INITIAL_MAX_DATA

5.6. Flow Control Capsules

WebTransport over HTTP/3 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_STREAMS, WT_DATA_BLOCKED, and WT_STREAMS_BLOCKED.

5.6.1. Flow Control and Intermediaries

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/3 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 can ensure that it does not advertise a higher flow control limit on one connection than the corresponding limit on the translated connection.

5.6.2. 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 session. 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.

Unlike in QUIC, where MAX_STREAMS frames can be delivered in any order, WT_MAX_STREAMS capsules are sent on the WebTransport session's connect stream and are delivered in order. If an endpoint receives a WT_MAX_STREAMS capsule with a Maximum Streams value less than a previously received value, it MUST close the WebTransport session by resetting the connect stream with the WT_FLOW_CONTROL_ERROR error code.

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.

5.6.3. 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 8: 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.

5.6.4. 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.

This limit counts all data that is sent on streams of the corresponding type, excluding the stream header (see Section 4.2 and Section 4.3). For streams that were reset, implementing WT_MAX_DATA requires that the QUIC stack provide the WebTransport implementation with information about the final size of streams (see Section 4.5 of [RFC9000]).

```
WT_MAX_DATA Capsule {
  Type (i) = 0x190B4D3D,
  Length (i),
  Maximum Data (i),
}
```

Figure 9: 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 session, 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.

Unlike in QUIC, where MAX_DATA frames can be delivered in any order, WT_MAX_DATA capsules are sent on the WebTransport session's connect stream and are delivered in order. If an endpoint receives a WT_MAX_DATA capsule with a Maximum Data value less than a previously received value, it MUST close the WebTransport session by resetting the connect stream with the WT_FLOW_CONTROL_ERROR error code.

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 5.6.1).

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

5.6.5. 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 10: 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 5.6.1).

6. Session Termination

A WebTransport session over HTTP/3 is considered terminated when either of the following conditions is met:

* the CONNECT stream is closed, either cleanly or abruptly, on either side; or

* a WT_CLOSE_SESSION capsule is either sent or received.

Upon learning that the session has been terminated, the endpoint MUST reset the send side and abort reading on the receive side of all unidirectional and bidirectional streams associated with the session (see Section 2.4 of [RFC9000]) using the WT_SESSION_GONE error code; it MUST NOT send any new datagrams or open any new streams.

To terminate a session with a detailed error message, an application MAY provide such a message for the WebTransport endpoint to send in an HTTP capsule [HTTP-DATAGRAM] of type WT_CLOSE_SESSION (0x2843). The format of the capsule SHALL be as follows:

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

Figure 11: WT_CLOSE_SESSION Capsule Format

WT_CLOSE_SESSION has the following fields:

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

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

Note that the Application Error Code field does not mirror the Error Code field in QUIC's CONNECTION_CLOSE frame (Section 19.19 of [RFC9000]) because WebTransport application errors use a subset of the HTTP/3 Error Code space and need to fit within those bounds, see Section 4.4.

An endpoint that sends a WT_CLOSE_SESSION capsule MUST immediately send a FIN on the CONNECT Stream. The endpoint MAY also send a STOP_SENDING with error code WT_SESSION_GONE to indicate it is no longer reading from the CONNECT stream. The recipient MUST either close or reset the stream in response. If any additional stream data is received on the CONNECT stream after receiving a WT_CLOSE_SESSION capsule, the stream MUST be reset with code H3_MESSAGE_ERROR.

Cleanly terminating a CONNECT stream without a WT_CLOSE_SESSION capsule SHALL be semantically equivalent to terminating it with a WT_CLOSE_SESSION capsule that has an error code of 0 and an empty error string.

In some scenarios, an endpoint might want to send a WT_CLOSE_SESSION with detailed close information and then immediately close the underlying QUIC connection. If the endpoint were to do both of those simultaneously, the peer could potentially receive the CONNECTION_CLOSE before receiving the WT_CLOSE_SESSION, thus never receiving the application error data contained in the latter. To avoid this, the endpoint SHOULD wait until all CONNECT streams have been closed by the peer before sending the CONNECTION_CLOSE; this gives WT_CLOSE_SESSION properties similar to that of the QUIC CONNECTION_CLOSE mechanism as a best-effort mechanism of delivering application close metadata.

7. 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 (see Section 9.1).

Servers that support future incompatible versions of WebTransport signal that support by changing the codepoint used for the SETTINGS_WT_MAX_SESSIONS setting (see Section 9.2). Clients can select the associated upgrade token, if applicable, to use when establishing a new session, ensuring that servers will always know the syntax in use for every incoming request.

Changes to future stream formats require changes to the Unidirectional Stream type (see Section 4.2) and Bidirectional Stream signal value (see Section 4.3) to allow recipients of incoming frames to determine the WebTransport version, and corresponding wire format, used for the session associated with that stream.

7.1. Negotiating the Draft Version

[[RFC editor: please remove this section before publication.]]

The wire format aspects of the protocol are negotiated by changing the codepoint used for the SETTINGS_WT_MAX_SESSIONS setting. Because of that, any WebTransport endpoint MUST wait for the peer's SETTINGS frame before sending or processing any WebTransport traffic. When multiple versions are supported by both of the peers, the most recent version supported by both is selected.

8. Security Considerations

WebTransport over HTTP/3 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 application is potentially untrusted.

WebTransport over HTTP/3 requires explicit opt-in through the use of an HTTP/3 setting; this avoids potential protocol confusion attacks by ensuring the HTTP/3 server explicitly supports it. It also requires the use of the Origin header for browser traffic, providing the server with the ability to deny access to Web-based applications that do not originate from a trusted origin.

Just like HTTP traffic going over HTTP/3, 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 WebTransport sessions share both congestion control and flow control context, a single application aggressively using up those resources can cause other sessions to stall. A WebTransport endpoint MUST implement flow control mechanisms if it allows a WebTransport session to share the transport connection with other WebTransport sessions. WebTransport endpoints SHOULD implement a fairness scheme that ensures that each session that shares a transport connection gets a reasonable share of controlled resources; this applies both to sending data and to opening new streams.

An application could attempt to exhaust resources by opening too many WebTransport sessions at once. In cases when the application is untrusted, the WebTransport client SHOULD limit the number of outgoing sessions it will open.

9. IANA Considerations

9.1. Upgrade Token Registration

The following entry is added to the "Hypertext Transfer Protocol (HTTP) Upgrade Token Registry" registry established by Section 16.7 of [HTTP].

The "webtransport" label identifies HTTP/3 used as a protocol for WebTransport:

Value: webtransport

Description: WebTransport over HTTP/3

Reference: This document and [I-D.ietf-webtrans-http2]

9.2. HTTP/3 SETTINGS Parameter Registration

The following entry is added to the "HTTP/3 Settings" registry established by [HTTP3]:

The SETTINGS_WT_MAX_SESSIONS setting indicates that the specified HTTP/3 endpoint is WebTransport-capable and the number of concurrent sessions it is willing to receive. The default value for the SETTINGS_WT_MAX_SESSIONS setting is "0", meaning that the endpoint is not willing to receive any WebTransport sessions.

Setting Name: WT_MAX_SESSIONS

Value: 0x14e9cd29

Default: 0

Specification: This document

The SETTINGS_WT_INITIAL_MAX_STREAMS_UNI setting indicates the initial value for the unidirectional max stream limit, otherwise communicated by the WT_MAX_STREAMS capsule (see Section 5.6.2). The default value for the SETTINGS_WT_INITIAL_MAX_STREAMS_UNI setting 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/3 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 setting indicates the initial value for the bidirectional max stream limit, otherwise communicated by the WT_MAX_STREAMS capsule (see Section 5.6.2). The default value for the SETTINGS_WT_INITIAL_MAX_STREAMS_BIDI setting 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/3 connection on which this SETTING is sent.

Setting Name: SETTINGS_WT_INITIAL_MAX_STREAMS_BIDI

Value: 0x2b65

Default: 0

Specification: This document

The SETTINGS_WT_INITIAL_MAX_DATA setting indicates the initial value for the session data limit, otherwise communicated by the WT_MAX_DATA capsule (see Section 5.6.4). The default value for the SETTINGS_WT_INITIAL_MAX_DATA setting 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/3 connection on which this SETTING is sent.

Setting Name: SETTINGS_WT_INITIAL_MAX_DATA

Value: 0x2b61

Default: 0

Specification: This document

9.3. Frame Type Registration

The following entry is added to the "HTTP/3 Frame Type" registry established by [HTTP3]:

The WT_STREAM frame is reserved for the purpose of avoiding collision with WebTransport HTTP/3 extensions:

Code: 0x41

Frame Type: WT_STREAM

Specification: This document

9.4. Stream Type Registration

The following entry is added to the "HTTP/3 Stream Type" registry established by [HTTP3]:

The "WebTransport stream" type allows unidirectional streams to be used by WebTransport:

Code: 0x54

Stream Type: WebTransport stream

Specification: This document

Sender: Both

9.5. HTTP/3 Error Code Registration

The following entries are added to the "HTTP/3 Error Code" registry established by [HTTP3]:

Name: WT_BUFFERED_STREAM_REJECTED

Value: 0x3994bd84

Description: WebTransport data stream rejected due to lack of associated session.

Specification: This document.

Name: WT_SESSION_GONE

Value: 0x170d7b68

Description: WebTransport data stream aborted because the associated WebTransport session has been closed. Also used to indicate that the endpoint is no longer reading from the CONNECT stream.

Specification: This document.

Name: WT_FLOW_CONTROL_ERROR

Value: 0x045d4487

Description: WebTransport session aborted because a flow control error was encountered.

Specification: This document.

In addition, the following range of entries is registered:

Name: WT_APPLICATION_ERROR

Value: 0x52e4a40fa8db to 0x52e5ac983162 inclusive, with the exception of the codepoints of form 0x1f * N + 0x21.

Description: WebTransport application error codes.

Specification: This document.

9.6. Capsule Types

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

The WT_CLOSE_SESSION capsule.

Value: 0x2843
Capsule Type: WT_CLOSE_SESSION
Status: permanent
Specification: This document
Change Controller: IETF
Contact: WebTransport Working Group webtransport@ietf.org (mailto:webtransport@ietf.org)
Notes: None

The WT_DRAIN_SESSION capsule.

Value: 0x78ae
Capsule Type: WT_DRAIN_SESSION
Status: provisional (when this document is approved this will become 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_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

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_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

9.7. Protocol Negotiation HTTP Header Fields

The following HTTP header fields are used for negotiating a protocol (Section 3.3. These are added to the "HTTP Field Name" registry established in Section 18.4 of [HTTP]:

The WT-Available-Protocols field:

Field Name: WT-Available-Protocols
Status: permanent
Structured Type: List
Reference: Section 3.3
Comments: None

The WT-Protocol field:

Field Name: WT-Protocol
Status: permanent
Structured Type: Item
Reference: Section 3.3
Comments: None

10. References

10.1. Normative References

[FIELDS] Nottingham, M. and P. Kamp, "Structured Field Values for HTTP", RFC 9651, DOI 10.17487/RFC9651, September 2024, <https://www.rfc-editor.org/rfc/rfc9651>.

[HTTP] 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>.

[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>.

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

[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>.

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

[RESET-STREAM-AT]
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>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/rfc/rfc3986>.

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

[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>.

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

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>.

[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>.

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

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

[RFC9220] Hamilton, R., "Bootstrapping WebSockets with HTTP/3", RFC 9220, DOI 10.17487/RFC9220, June 2022, <https://www.rfc-editor.org/rfc/rfc9220>.

10.2. Informative References

[I-D.ietf-webtrans-http2] Frindell, A., Kinnear, E., Pauly, T., Thomson, M., Vasiliev, V., and G. Xie, "WebTransport over HTTP/2", Work in Progress, Internet-Draft, draft-ietf-webtrans-http2-12, 7 July 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-webtrans-http2-12>.

[ORIGIN] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <https://www.rfc-editor.org/rfc/rfc6455>.

[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>.

[RFC9208] Melnikov, A., "IMAP QUOTA Extension", RFC 9208, DOI 10.17487/RFC9208, March 2022, <https://www.rfc-editor.org/rfc/rfc9208>.

[RFC9308] K端hlewind, M. and B. Trammell, "Applicability of the QUIC Transport Protocol", RFC 9308, DOI 10.17487/RFC9308, September 2022, <https://www.rfc-editor.org/rfc/rfc9308>.

Appendix A. Changelog

A.1. Changes between draft versions 02 and 07

The following changes make the draft-02 and draft-07 versions of this protocol incompatible:

* draft-07 requires SETTINGS_WEBTRANSPORT_MAX_SESSIONS (#86) and uses it for version negotiation (#129)

* draft-07 explicitly requires SETTINGS_ENABLE_CONNECT_PROTOCOL to be enabled (#93)

* draft-07 explicitly requires SETTINGS_H3_DATAGRAM to be enabled (#106)

* draft-07 only allows WEBTRANSPORT_STREAM at the beginning of the stream

The following changes that are present in draft-07 can be also implemented by a draft-02 implementation safely:

* Expanding stream reset error code space from 8 to 32 bits (#115)

* WEBTRANSPORT_SESSION_GONE error code (#75)

* Handling for HTTP GOAWAY (#76)

* DRAIN_WEBTRANSPORT_SESSION capsule (#79)

* Disallowing following redirects automatically (#113)

Index

2; Section 5.4, Paragraph 3; Section 5.5, Paragraph 2.2.1;
Section 5.6, Paragraph 1; Section 5.6.4, Paragraph 1;
Section 5.6.4, Paragraph 2; Section 5.6.4, Paragraph 4;
Section 5.6.4, Paragraph 7; Section 5.6.4, Paragraph 8;
Section 9.2, Paragraph 10; Section 9.6, Paragraph 11.4.1

WT_MAX_STREAMS Section 5.3, Paragraph 1; Section 5.5,
Paragraph 2.1.1; Section 5.6, Paragraph 1; Section 5.6.2,
Paragraph 1; Section 5.6.2, Paragraph 2; Section 5.6.2,
Paragraph 4; Section 5.6.2, Paragraph 8; Section 5.6.2,
Paragraph 9; Section 9.2, Paragraph 4; Section 9.2,
Paragraph 7; Section 9.6, Paragraph 7.4.1

WT_STREAMS_BLOCKED Section 5.3, Paragraph 5; Section 5.6,
Paragraph 1; Section 5.6.3, Paragraph 1; Section 5.6.3,
Paragraph 2; Section 5.6.3, Paragraph 4; Section 5.6.3,
Paragraph 6; Section 9.6, Paragraph 9.4.1

Authors' Addresses

Alan Frindell
Facebook
Email: afrind@fb.com

Eric Kinnear
Apple Inc.
Email: ekinnear@apple.com

Victor Vasiliev
Google
Email: vasilvv@google.com