

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 28 April 2022

A. Frindell
Facebook
E. Kinnear
Apple Inc.
V. Vasiliev
Google
25 October 2021

WebTransport over HTTP/3
draft-ietf-webtrans-http3-02

Abstract

WebTransport [OVERVIEW] is a protocol framework that enables clients constrained by the Web security model to communicate with a remote 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.

Note to Readers

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

The repository tracking the issues for this draft can be found at <https://github.com/ietf-wg-webtrans/draft-ietf-webtrans-http3/issues>. The web API draft corresponding to this document can be found at <https://w3c.github.io/webtransport/>.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 28 April 2022.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	3
2. Protocol Overview	3
3. Session Establishment	4
3.1. Establishing a Transport-Capable HTTP/3 Connection . . .	4
3.2. Extended CONNECT in HTTP/3	4
3.3. Creating a New Session	4
3.4. Limiting the Number of Simultaneous Sessions	5
4. WebTransport Features	5
4.1. Unidirectional streams	6
4.2. Bidirectional Streams	6
4.3. Resetting Data Streams	7
4.4. Datagrams	8
4.5. Buffering Incoming Streams and Datagrams	8
5. Session Termination	9
6. Negotiating the Draft Version	10
7. Security Considerations	10
8. IANA Considerations	11
8.1. Upgrade Token Registration	11
8.2. HTTP/3 SETTINGS Parameter Registration	11
8.3. Frame Type Registration	12
8.4. Stream Type Registration	12
8.5. HTTP/3 Error Code Registration	12
8.6. Datagram Format Type	13
9. References	13
9.1. Normative References	13
9.2. Informative References	15
Authors' Addresses	15

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 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.

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 server have to send a `SETTINGS_ENABLE_WEBTRANSPORT` setting in order to indicate that they both support WebTransport over HTTP/3.

WebTransport sessions are initiated inside a given HTTP/3 connection by the client, who sends an extended `CONNECT` request [RFC8441]. If the server accepts the request, an 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 peers can exchange data using the following mechanisms:

- * A client can create a bidirectional stream using a special indefinite-length HTTP/3 frame that transfers ownership of the stream to WebTransport.
- * A server can create a bidirectional stream, which is possible since HTTP/3 does not define any semantics for server-initiated bidirectional streams.
- * Both client and server can create a unidirectional stream using a special stream type.
- * A datagram can be sent using HTTP Datagrams [HTTP-DATAGRAM].

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

3. Session Establishment

3.1. Establishing a Transport-Capable HTTP/3 Connection

In order to indicate support for WebTransport, both the client and the server MUST send a `SETTINGS_ENABLE_WEBTRANSPORT` value set to "1" in their `SETTINGS` frame. The `SETTINGS_ENABLE_WEBTRANSPORT` parameter value SHALL be either "0" or "1", with "0" being the default; an endpoint that receives a value other than "0" or "1" MUST close the connection with the `H3_SETTINGS_ERROR` error code.

The client MUST NOT send a WebTransport request until it has received the setting indicating WebTransport support from the server. Similarly, the server MUST NOT process any incoming WebTransport requests until the client settings have been received, as the client may be using a version of WebTransport extension that is different from the one used by the server.

3.2. Extended CONNECT in HTTP/3

[RFC8441] defines an extended `CONNECT` method in Section 4, enabled by the `SETTINGS_ENABLE_CONNECT_PROTOCOL` parameter. That parameter is only defined for HTTP/2. This document does not create a new multi-purpose parameter to indicate support for extended `CONNECT` in HTTP/3; instead, the `SETTINGS_ENABLE_WEBTRANSPORT` setting implies that an endpoint supports extended `CONNECT`.

3.3. Creating a New Session

As WebTransport sessions are established over HTTP/3, 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`. The `:scheme` field MUST be `https`. Both the `:authority` and the `:path` value MUST be set; those fields indicate the desired WebTransport server. An Origin header [RFC6454] MUST be provided within the request.

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 6.5.4, [RFC7231]). 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.

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. WebTransport over HTTP/3 does not support 0-RTT.

The `webtransport` HTTP Upgrade Token uses the Capsule Protocol as defined in [HTTP-DATAGRAM].

3.4. Limiting the Number of Simultaneous Sessions

From the flow control perspective, WebTransport sessions count against the stream flow control just like regular HTTP requests, since they are established via an HTTP CONNECT request. This document does not make any effort to introduce a separate flow control mechanism for sessions, nor to separate HTTP requests from WebTransport data streams. If the server needs to limit the rate of incoming requests, it has alternative mechanisms at its disposal:

- * HTTP_REQUEST_REJECTED error code defined in [HTTP3] indicates to the receiving HTTP/3 stack that the request was not processed in any way.
- * HTTP status code 429 indicates that the request was rejected due to rate limiting [RFC6585]. Unlike the previous method, this signal is directly propagated to the application.

4. WebTransport Features

WebTransport over HTTP/3 provides the following features described in [OVERVIEW]: unidirectional streams, bidirectional streams and datagrams, initiated by either endpoint.

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].

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

4.1. Unidirectional streams

Once established, both endpoints can open 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 1).

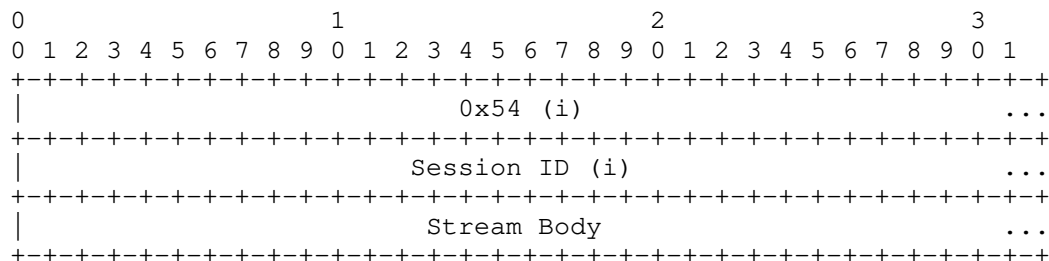


Figure 1: Unidirectional WebTransport stream format

4.2. Bidirectional Streams

WebTransport endpoints can initiate bidirectional streams by opening an HTTP/3 bidirectional stream and sending an HTTP/3 frame with type WEBTRANSPORT_STREAM (type=0x41). The format of the frame SHALL be the frame type, followed by the session ID, encoded as a variable-length integer, followed by the user-specified stream data (Figure 2). The frame SHALL last until the end of the stream.

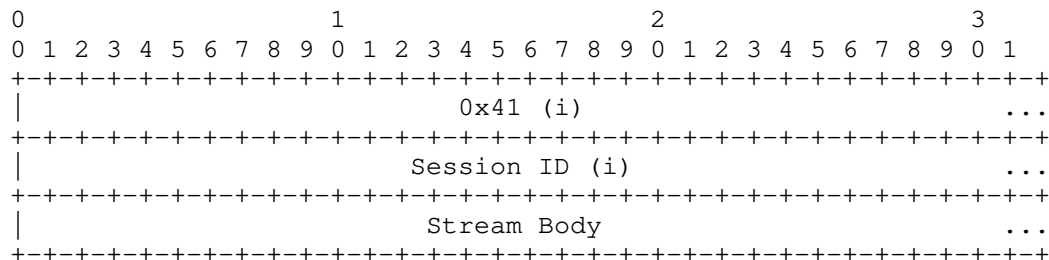


Figure 2: WEBTRANSPORT_STREAM frame format

HTTP/3 does not by itself define any semantics for server-initiated bidirectional streams. If WebTransport setting is negotiated by both endpoints, the syntax of the server-initiated bidirectional streams SHALL be the same as the syntax of client-initiated bidirectional streams, that is, a sequence of HTTP/3 frames. The only frame defined by this document for use within server-initiated bidirectional streams is WEBTRANSPORT_STREAM.

TODO: move the paragraph above into a separate draft; define what happens with already existing HTTP/3 frames on server-initiated bidirectional streams.

4.3. 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 SHALL 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 8-bit integer, assuming values between 0x00 and 0xff. WebTransport implementations SHALL remap those error codes into an error range where 0x00 corresponds to 0x52e4a40fa8db, and 0xff corresponds to 0x52e4a40fa9e2. Note that there are code points inside that range of form "0x1f * N + 0x21" that are reserved by Section 8.1 of [HTTP3]; those have to be accounted for when mapping the error codes by skipping them (i.e. the two HTTP/3 error codepoints adjacent to a GREASE codepoint would map to two adjacent WebTransport application error codepoints). An example pseudocode can be seen in Figure 3.

```
first = 0x52e4a40fa8db
last = 0x52e4a40fa9e2

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 - shifted // 0x1f
```

Figure 3: Pseudocode for converting between WebTransport application errors and HTTP/3 error codes; here, `//` is integer division

WebTransport data streams are associated with sessions through a header at the beginning of the stream; resetting a stream may result in that data being discarded. Because of that, WebTransport application error codes are best effort, as the WebTransport stack is not always capable of associating the reset code with a session. The only exception is the situation where there is only one session on a given HTTP/3 connection, and no intermediaries between the client and the server.

WebTransport implementations SHALL forward the error code for a stream associated with a known session to the application that owns that session; similarly, the intermediaries SHALL reset the streams with corresponding error code when receiving a reset from the peer. If a WebTransport implementation intentionally allows only one session over a given HTTP/3 connection, it SHALL forward the error codes within WebTransport application error code range to the application that owns the only session on that connection.

4.4. Datagrams

Datagrams can be sent using HTTP Datagrams, using the `WEB_TRANSPORT` HTTP Datagram Format Type (see value in Section 8.6). When using the `WEB_TRANSPORT` HTTP Datagram Format Type, the WebTransport datagram payload is sent unmodified in the "HTTP Datagram Payload" field of an HTTP Datagram. When sending a registration capsule using the "Datagram Format Type" set to `WEB_TRANSPORT`, the "Datagram Format Additional Data" field SHALL be empty.

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. TODO: Describe how the path MTU can be computed, specifically propagation across HTTP proxies.

4.5. Buffering Incoming Streams and Datagrams

In WebTransport over HTTP/3, the client MAY send its `SETTINGS` frame, as well as 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 those can be associated with an established session. To avoid resource exhaustion, the 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 H3_WEBTRANSPORT_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.

5. 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 CLOSE_WEBTRANSPORT_SESSION capsule is either sent or received.

Upon learning that the session has been terminated, the endpoint MUST reset all of the streams associated with the session; it MUST NOT send any new datagrams or open any new streams.

To terminate a session with a detailed error message, an application MAY send an HTTP capsule [HTTP-DATAGRAM] of type CLOSE_WEBTRANSPORT_SESSION (0x2843). The format of the capsule SHALL be as follows:

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

CLOSE_WEBTRANSPORT_SESSION has 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.

A `CLOSE_WEBTRANSPORT_SESSION` capsule MUST be followed by a FIN on the sender side. If any additional stream data is received on the `CONNECT` stream after `CLOSE_WEBTRANSPORT_SESSION`, the stream MUST be reset with code `H3_MESSAGE_ERROR`. The recipient MUST close the stream upon receiving a FIN. If the sender of `CLOSE_WEBTRANSPORT_SESSION` does not receive a FIN after some time, it SHOULD send `STOP_SENDING` on the `CONNECT` stream.

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

6. Negotiating the Draft Version

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

WebTransport over HTTP/3 uses two different mechanisms to negotiate versions for the different parts of the draft.

The hop-by-hop wire format aspects of the protocol are negotiated by changing the codepoint used for the `SETTINGS_ENABLE_WEBTRANSPORT` parameter. 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.

The data exchanged over the `CONNECT` stream is transmitted across intermediaries, and thus cannot be versioned using a `SETTINGS` parameter. To indicate support for different versions of the protocol defined in this draft, the clients SHALL send a header for each version of the draft supported. The header corresponding to the version described in this draft is `Sec-Webtransport-Http3-Draft02`; its value SHALL be 1. The server SHALL reply with a `Sec-Webtransport-Http3-Draft` header indicating the selected version; its value SHALL be `draft02` for the version described in this draft.

7. 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 client is potentially untrusted.

WebTransport over HTTP/3 requires explicit opt-in through the use of a QUIC transport parameter; this avoids potential protocol confusion attacks by ensuring the HTTP/3 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/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 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.

8. IANA Considerations

8.1. Upgrade Token Registration

The following entry is added to the "Hypertext Transfer Protocol (HTTP) Upgrade Token Registry" registry established by [RFC7230]:

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]

8.2. HTTP/3 SETTINGS Parameter Registration

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

The `SETTINGS_ENABLE_WEBTRANSPORT` parameter indicates that the specified HTTP/3 connection is WebTransport-capable.

Setting Name: `ENABLE_WEBTRANSPORT`

Value: `0x2b603742`

Default: `0`

Specification: This document

8.3. Frame Type Registration

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

The WEBTRANSPORT_STREAM frame allows HTTP/3 client-initiated bidirectional streams to be used by WebTransport:

Code: 0x41

Frame Type: WEBTRANSPORT_STREAM

Specification: This document

8.4. 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

8.5. HTTP/3 Error Code Registration

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

Name: H3_WEBTRANSPORT_BUFFERED_STREAM_REJECTED

Value: 0x3994bd84

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

Specification: This document.

In addition, the following range of entries is registered:

Name: H3_WEBTRANSPORT_APPLICATION_00 ...
H3_WEBTRANSPORT_APPLICATION_FF

Value: 0x52e4a40fa8db to 0x52e4a40fa9e2 inclusive, with the exception of 0x52e4a40fa8f9, 0x52e4a40fa918, 0x52e4a40fa937, 0x52e4a40fa956, 0x52e4a40fa975, 0x52e4a40fa994, 0x52e4a40fa9b3, and 0x52e4a40fa9d2.

Description: WebTransport application error codes.

Specification: This document.

8.6. Datagram Format Type

This document will request IANA to register WEB_TRANSPORT in the "HTTP Datagram Format Types" registry established by [HTTP-DATAGRAM].

Type	Value	Specification
WEB_TRANSPORT	0xff7c00	This Document

Table 1: Registered Datagram Format Type

9. References

9.1. Normative References

[HTTP-DATAGRAM]

Schinazi, D. and L. Pardue, "Using Datagrams with HTTP", Work in Progress, Internet-Draft, draft-ietf-masque-h3-datagram-05, 25 October 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-masque-h3-datagram-05>>.

[HTTP3]

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

[OVERVIEW]

Vasiliev, V., "The WebTransport Protocol Framework", Work in Progress, Internet-Draft, draft-ietf-webtrans-overview-latest, <<https://datatracker.ietf.org/doc/html/draft-ietf-webtrans-overview-latest>>.

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://doi.org/10.17487/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://doi.org/10.17487/RFC3986>>.
- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <<https://doi.org/10.17487/RFC6454>>.
- [RFC6585] Nottingham, M. and R. Fielding, "Additional HTTP Status Codes", RFC 6585, DOI 10.17487/RFC6585, April 2012, <<https://doi.org/10.17487/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://doi.org/10.17487/RFC7230>>.
- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<https://doi.org/10.17487/RFC7231>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://doi.org/10.17487/RFC8174>>.
- [RFC8441] McManus, P., "Bootstrapping WebSockets with HTTP/2", RFC 8441, DOI 10.17487/RFC8441, September 2018, <<https://doi.org/10.17487/RFC8441>>.
- [RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <<https://doi.org/10.17487/RFC9000>>.
- [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>>.

9.2. Informative References

[I-D.ietf-webtrans-http2]

Frindell, A., Kinnear, E., Pauly, T., Vasiliev, V., and G. Xie, "WebTransport using HTTP/2", Work in Progress, Internet-Draft, draft-ietf-webtrans-http2-01, 30 July 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-webtrans-http2-01>>.

Authors' Addresses

Alan Frindell
Facebook

Email: afrind@fb.com

Eric Kinnear
Apple Inc.

Email: ekinnear@apple.com

Victor Vasiliev
Google

Email: vasilvv@google.com

webtrans
Internet-Draft
Intended status: Standards Track
Expires: 26 August 2021

A. Frindell
Facebook Inc.
E. Kinnear
T. Pauly
Apple Inc.
V. Vasiliev
Google
G. Xie
Facebook Inc.
22 February 2021

WebTransport using HTTP/2
draft-kinnear-webtransport-http2-02

Abstract

WebTransport [OVERVIEW] is a protocol framework that enables clients constrained by the Web security model to communicate with a remote server using a secure multiplexed transport. This document describes a WebTransport protocol that is based on HTTP/2 [RFC7540] and provides support for unidirectional streams, bidirectional streams and datagrams, all multiplexed within the same HTTP/2 connection.

Note to Readers

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

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

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 26 August 2021.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	3
2. Protocol Overview	3
3. Session Establishment	4
3.1. Establishing a Transport-Capable HTTP/2 Connection	4
3.2. Extended CONNECT in HTTP/2	4
3.3. Creating a New Session	4
3.4. Limiting the Number of Simultaneous Sessions	5
4. WebTransport Features	5
4.1. WT_STREAM Frame	6
4.2. WT_DATAGRAM Frame	7
5. Session Termination	8
6. Transport Properties	8
7. Security Considerations	9
8. IANA Considerations	9
8.1. HTTP/2 SETTINGS Parameter Registration	9
8.2. Frame Type Registration	9
8.3. HTTP/2 Error Code Registry	10
8.4. Examples	10
9. References	12
9.1. Normative References	12
9.2. Informative References	13
Acknowledgments	14
Authors' Addresses	14

1. Introduction

Currently, the only mechanism in HTTP/2 for server to client communication is server push. That is, servers can initiate unidirectional push promised streams to clients, but clients cannot respond to them; they can only accept them or discard them. Additionally, intermediaries along the path may have different server push policies and may not forward push promised streams to the downstream client. This best effort mechanism is not sufficient to reliably deliver messages from servers to clients, limiting server to client use-cases such as chat messages or notifications.

Several techniques have been developed to workaroud these limitations: long polling [RFC6202], WebSocket [RFC8441], and tunneling using the CONNECT method. All of these approaches have limitations.

This document defines a mechanism for multiplexing non-HTTP data with HTTP/2 in a manner that conforms with the WebTransport protocol requirements and semantics [OVERVIEW]. Using the mechanism described here, multiple WebTransport instances can be multiplexed simultaneously with regular HTTP traffic on the same HTTP/2 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/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 via an HTTP/2 server.

2. Protocol Overview

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

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

WebTransport sessions are initiated inside a given HTTP/2 connection by the client, who sends an extended CONNECT request [RFC8441]. If the server accepts the request, an 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 peers can exchange data using the following mechanisms:

- * Both client and server can create a bidirectional or unidirectional stream using a new HTTP/2 extension frame (`WT_STREAM`)
- * A datagram can be sent using a new HTTP/2 extension frame `WT_DATAGRAM`.

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

3. Session Establishment

3.1. Establishing a Transport-Capable HTTP/2 Connection

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

3.2. Extended CONNECT in HTTP/2

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

3.3. Creating a New Session

As WebTransport sessions are established over HTTP/2, 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 [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. An "Origin" header [RFC6454] MUST be provided within the request.

Upon receiving an extended CONNECT request with a ":protocol" field set to "webtransport", the HTTP/2 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 6.5.4, [RFC7231]). If it does, it MAY accept the session by replying with status code 200. The WebTransport server MUST verify the "Origin" header to ensure that the specified origin is allowed to access the server in question.

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

3.4. Limiting the Number of Simultaneous Sessions

From the flow control perspective, WebTransport sessions count against the stream flow control just like regular HTTP requests, since they are established via an HTTP CONNECT request. This document does not make any effort to introduce a separate flow control mechanism for sessions, nor to separate HTTP requests from WebTransport data streams. If the server needs to limit the rate of incoming requests, it has alternative mechanisms at its disposal:

- * "HTTP_STREAM_REFUSED" error code defined in [RFC7540] indicates to the receiving HTTP/2 stack that the request was not processed in any way.
- * HTTP status code 429 indicates that the request was rejected due to rate limiting [RFC6585]. Unlike the previous method, this signal is directly propagated to the application.

4. WebTransport Features

WebTransport over HTTP/2 provides the following features described in [OVERVIEW]: unidirectional streams, bidirectional streams and datagrams, initiated by either endpoint.

Session IDs are used to demultiplex streams and datagrams belonging to different WebTransport sessions. On the wire, session IDs are encoded using a 31-bit integer field.

4.1. WT_STREAM Frame

A new HTTP/2 frame called WT_STREAM is introduced for either endpoint to establish WebTransport streams. WT_STREAM frames can be sent on a stream in the "idle", "reserved (local)", "open", or "half-closed (remote)" state.

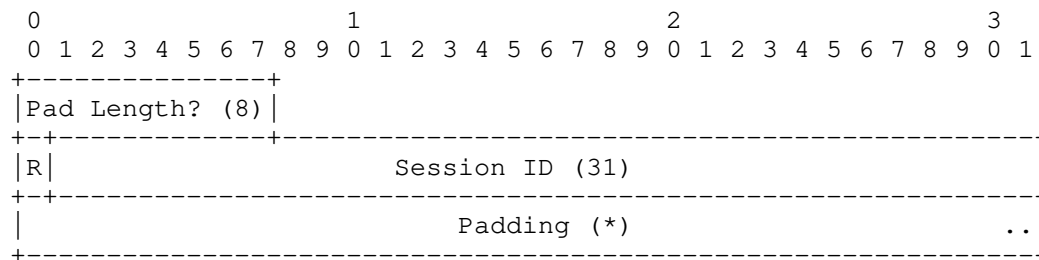


Figure 1: WT_STREAM Frame Format

The WT_STREAM frame define the following fields:

Pad Length: An 8-bit field containing the length of the frame padding in units of octets. This field is conditional (as signified by a "?" in the diagram) and is only present if the PADDED flag is set.

Session ID: An unsigned 31-bit integer that identifies the stream Connect Stream for this Web Transport stream. The Session ID MUST be an open stream negotiated via the extended CONNECT protocol with a ":protocol" value of "webtransport".

The WT_STREAM frame defines the following flags:

UNIDIRECTIONAL (0x1): When set, the stream begins in the "half-closed (remote)" state at the sender, and in the "half-closed (local)" state at the receiver.

As with all HTTP/2 streams, WebTransport streams initiated by a client have odd stream IDs and those initiated by a server have even stream IDs.

The recipient MUST respond with a stream error of type WT_STREAM_ERROR if the specified WebTransport Connect Stream does not exist, is not a stream established via extended CONNECT to use the "webtransport" protocol, or if it is in the "closed" or "half-closed (remote)" stream state.

4.2. WT_DATAGRAM Frame

A new HTTP/2 frame called WT_DATAGRAM is introduced for either endpoint to transmit a datagram. WT_DATAGRAM frames are sent with Stream Identifier 0.

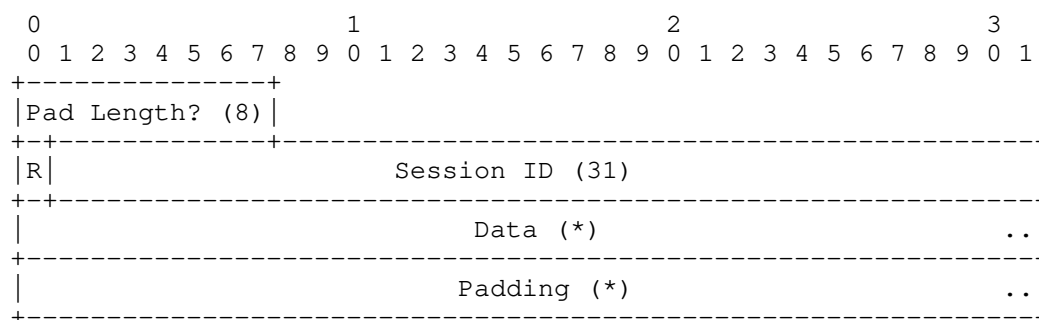


Figure 2: WT_DATAGRAM Frame Format

The WT_DATAGRAM frame define the following fields:

Pad Length: An 8-bit field containing the length of the frame padding in units of octets. This field is conditional (as signified by a "?" in the diagram) and is only present if the PADDED flag is set.

Session ID: An unsigned 31-bit integer that identifies the stream Connect Stream for this Web Transport stream. The Session ID MUST be an open stream negotiated via the extended CONNECT protocol with a ":protocol" value of "webtransport".

Data: Application data. The amount of data is the remainder of the frame payload after subtracting the length of the other fields that are present.

The WT_DATAGRAM frame does not define any flags.

The recipient MAY respond with a stream error of type WT_STREAM_ERROR if the specified WebTransport Connect Stream does not exist, is not a stream established via extended CONNECT to use the "webtransport" protocol, or if it is in the "closed" or "half-closed (remote)" stream state.

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

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

5. Session Termination

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

6. Transport Properties

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

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

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

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

Connection Mobility: Http2Transport does not support connection mobility, unless an underlying transport protocol that supports multipath or migration, such as MPTCP [RFC7540], is used underneath HTTP/2 and TLS. Without such support, Http2Transport connections cannot survive network transitions.

7. 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.

8. IANA Considerations

8.1. HTTP/2 SETTINGS Parameter Registration

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

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

Setting Name: ENABLE_WEBTRANSPORT

Value: 0x2b603742

Default: 0

Specification: This document

8.2. Frame Type Registration

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

The "WT_STREAM" frame allows HTTP/2 client- and server-initiated unidirectional and bidirectional streams to be used by WebTransport:

Code: 0xTBD

Frame Type: WEBTRANSPORT_STREAM

Specification: This document

The "WT_DATAGRAM" frame allows HTTP/2 client and server to exchange datagrams used by WebTransport:

Code: 0xTBD

Frame Type: WEBTRANSPORT_DATAGRAM

Specification: This document

8.3. HTTP/2 Error Code Registry

The following entries are added to the "HTTP/2 Error Code" registry that was established by Section 11.2 of [RFC7540].

Name: WT_STREAM_ERROR

Code: 0xTBD

Description: Invalid use of WT_STREAM frame

Specification: _RFC Editor: Please fill in this value with the RFC number for this document_

8.4. 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 ]]
```

```
SETTINGS
SETTINGS_ENABLE_WEBTRANSPORT = 1
```

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

```
WT_STREAM
Stream ID = 5
Session ID = 3
```

```
DATA
Stream ID = 5
WebTransport Data
```

```
DATA + END_STREAM
Stream ID = 5
WebTransport Data
```

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

```
SETTINGS
SETTINGS_ENABLE_WEBTRANSPORT = 1
```

```
HEADERS + END_HEADERS
Stream ID = 3
:status = 200
```

```
DATA + END_STREAM
Stream ID = 5
WebTransport Data
```

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

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

```
SETTINGS
```

```
SETTINGS_ENABLE_WEBTRANSPORT = 1
```

```
HEADERS + END_HEADERS
```

```
Stream ID = 3
```

```
:method = CONNECT
```

```
:protocol = webtransport
```

```
:scheme = https
```

```
:path = /
```

```
:authority = server.example.com
```

```
origin: server.example.com
```

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

```
SETTINGS
```

```
SETTINGS_ENABLE_WEBTRANSPORT = 1
```

```
HEADERS + END_HEADERS
```

```
Stream ID = 3
```

```
:status = 200
```

```
WT_STREAM
```

```
Stream ID = 2
```

```
Session ID = 3
```

```
DATA
```

```
Stream ID = 2
```

```
WebTransport Data
```

```
DATA + END_STREAM
```

```
Stream ID = 2
```

```
WebTransport Data
```

```
DATA + END_STREAM
```

```
Stream ID = 2
```

```
WebTransport Data
```

9. References

9.1. Normative References

[OVERVIEW] Vasiliev, V., "The WebTransport Protocol Framework", Work in Progress, Internet-Draft, draft-ietf-webtrans-overview-latest, <<https://tools.ietf.org/html/draft-ietf-webtrans-overview-latest>>.

- [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/info/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/info/rfc3986>>.
- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <<https://www.rfc-editor.org/info/rfc6454>>.
- [RFC6585] Nottingham, M. and R. Fielding, "Additional HTTP Status Codes", RFC 6585, DOI 10.17487/RFC6585, April 2012, <<https://www.rfc-editor.org/info/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/info/rfc7230>>.
- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<https://www.rfc-editor.org/info/rfc7231>>.
- [RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <<https://www.rfc-editor.org/info/rfc7540>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8441] McManus, P., "Bootstrapping WebSockets with HTTP/2", RFC 8441, DOI 10.17487/RFC8441, September 2018, <<https://www.rfc-editor.org/info/rfc8441>>.
- [WEBTRANSPORT-H3] Vasiliev, V., "WebTransport over HTTP/3", Work in Progress, Internet-Draft, draft-ietf-webtrans-http3-latest, <<https://tools.ietf.org/html/draft-ietf-webtrans-http3-latest>>.

9.2. Informative References

[RFC6202] Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins,
"Known Issues and Best Practices for the Use of Long
Polling and Streaming in Bidirectional HTTP", RFC 6202,
DOI 10.17487/RFC6202, April 2011,
<<https://www.rfc-editor.org/info/rfc6202>>.

Acknowledgments

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

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

Victor Vasiliev
Google

Email: vasilvv@google.com

Guowu Xie
Facebook Inc.

Email: woo@fb.com