Workgroup: webtrans
Internet-Draft:
draft-kinnear-webtransport-http2-00
Published: 9 March 2020
Intended Status: Standards Track
Expires: 10 September 2020
Authors: A. Frindell, Facebook Inc.; E. Kinnear, Apple Inc.; T. Pauly, Apple Inc.; V. Vasiliev, Google; G. Xie, Facebook Inc.

WebTransport using HTTP/2

Abstract

WebTransport 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 Http2Transport, a WebTransport protocol that is based on HTTP/2 and provides support for bidirectional streams 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/erickinnear/draft-http-transport/issues>. The web API draft corresponding to this document can be found at <https://wicg.github.io/web-transport/>.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

HTTP/2 [RFC7540] transports HTTP messages via a framing layer that includes many optimizations designed to make communication more efficient between clients and servers. These include multiplexing of multiple streams on a single underlying transport connection, flow control, priorities, header compression, and exchange of configuration information between endpoints.

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 workaround these limitations: long polling [RFC6202], WebSocket [RFC8441], and tunneling using the CONNECT method. All of these approaches layer an application protocol on top of HTTP/2, using HTTP/2 streams as transport connections. This layering defeats the optimizations provided by HTTP/2. For example, application metadata is encapsulated in DATA frames rather than HEADERS frames, bypassing the advantages of HPACK header compression. Further, application data might be framed multiple times at different protocol layers, reducing the wire efficiency of the protocol.

This document defines Http2Transport, a mechanism for multiplexing non-request/response streams with HTTP/2 in a manner that conforms with the WebTransport protocol framework [I-D.vvv-webtransport-overview]. Using the mechanism described, multiple Http2Transport instances can be multiplexed simultaneously with regular HTTP traffic on the same HTTP/2 connection.

Section 8.3 of [RFC7540] defines the HTTP CONNECT method for HTTP/2, which converts an HTTP/2 stream into a tunnel for arbitrary data. [RFC8441] describes the use of the extended CONNECT method to negotiate the use of the WebSocket Protocol [RFC6455] on an HTTP/2 stream. Http2Transport uses the extended CONNECT handshake to allow WebTransport endpoints to multiplex arbitrary data streams on HTTP/2 connections.

In this draft, a new HTTP/2 frame is introduced which carries structured metadata like the HEADERS and PUSH_PROMISE frames, but without the constraints of the request/response state machine and semantics.

The WebTransport over HTTP/2 extension:

1. Enables bidirectional and symmetric communication over HTTP/2. After a WebTransport session is established, a server can initiate a WebTransport stream to the client at any time, and the client can respond to server-initiated streams.

2. Allows WebTransport streams to take advantage of HTTP/2 features such as header compression, prioritization and flow-control.

3. Provides a mechanism for intermediaries to route server initiated messages to the correct client.

4. Allows clients and servers to group streams and route them together.

2. Conventions and Definitions

The key words "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 [I-D.vvv-webtransport-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.

3. Http2Transport Overview

3.1. WebTransport Connect Streams

After negotiating the use of this extension, clients initiate one or more WebTransport Connect Streams to a Http2Transport Server. Http2Transport servers are identified by a pair of authority value and path value (defined in [RFC3986] Sections 3.2 and 3.3 respectively). The client uses the extended CONNECT method with a :protocol token "webtransport" to establish a WebTransport Connect Stream. This stream is only used to establish a WebTransport session and is not intended for data exchange.

3.2. WebTransport Streams

Following the establishment of a WebTransport Connect stream, either the client or the server can initiate a WebTransport Stream by sending the WTHEADERS frame, defined in Section 3.5. This frame references an open WebTransport Connect stream which is used by any intermediaries to correctly forward the stream to the destination endpoint. The only frames allowed on WebTransport Streams are WTHEADERS, CONTINUATION, DATA and any negotiated extension frames.

3.3. Negotiation

Clients negotiate the use of WebTransport ove HTTP/2 using both the SETTINGS frame and one or more extended CONNECT requests as defined in [RFC8441].

Use of the extended CONNECT method extension requires the SETTINGS_ENABLE_CONNECT_PROTOCOL parameter to be received by a client prior to its use. An endpoint that supports receiving the extended CONNECT method SHOULD send this setting with a value of 1.

The extended CONNECT method extension uses the :protocol psuedo-header field to negotiate the protocol that will be used on a given stream in an HTTP/2 connection. This document registers a new token, "webtransport", in the "Hypertext Transfer Protocol (HTTP) Upgrade Token Registry" established by [RFC7230] and located at https://www.iana.org/assignments/http-upgrade-tokens/.

This token is used in the :protocol psuedo-header field to indicate that the endpoint wishes to use the WebTransport protocol on the new stream.

3.4. The SETTINGS_ENABLE_WEBTRANSPORT SETTINGS parameter

As described in Section 5.5 of [RFC7540], SETTINGS parameters allow endpoints to negotiate use of protocol extensions that would otherwise generate protocol errors.

This document introduces a new SETTINGS parameter, SETTINGS_ENABLE_WEBTRANSPORT, which MUST have a value of 0 or 1.

Once a SETTINGS_ENABLE_WEBTRANSPORT parameter has been sent with a value of 1, an endpoint MUST NOT send the parameter with a value of 0.

Upon receipt of SETTINGS_ENABLE_WEBTRANSPORT with a value of 1, an endpoint MAY use the WTHEADERS frame type defined in this document. An endpoint that supports receiving the WTHEADERS as part of the WebTransport protocol SHOULD send this setting with a value of 1.

3.5. The WTHEADERS Frame

A new HTTP/2 frame called WTHEADERS is introduced for establishing streams in a bidirectional manner. A stream opened by a WTHEADERS frame is referred to as a WebTransport Stream, and it MAY be continued by CONTINUATION and DATA frames. WebTransport Streams can be initiated by either clients or servers via a WTHEADERS frame that refers to the corresponding WebTransport Connect Stream on which the WebTransport protocol was negotiated.

The WTHEADERS frame (type=0xfb) has all the fields and frame header flags defined by HEADERS frame in HEADERS [RFC7540], Section 6.2.

The WTHEADERS frame has one extra field, Connect Stream ID. WTHEADERS frames can be sent on a stream in the "idle", "open", or "half-closed (remote)" state, see Section 4.1.

Like HEADERS, the CONTINUATION frame (type=0x9) is used to continue a sequence of header block fragments, if the headers do not fit into one WTHEADERS frame.

The WTHEADERS frame is shown in Figure 1.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+
|Pad Length? (8)|
+-+-------------+-----------------------------------------------+
|E|                 Stream Dependency? (31)                     |
+-+-------------+-----------------------------------------------+
|  Weight? (8)  |
+-+-------------+-----------------------------------------------+
|R|                 Connect Stream ID (31)                      |
+-+-------------------------------------------------------------+
|                   Header Block Fragment (*)                 ...
+---------------------------------------------------------------+
|                           Padding (*)                       ...
+---------------------------------------------------------------+
```

Figure 1: WTHEADERS Frame Format

The Connect Stream specified in a WTHEADERS frame MUST be an open stream negotiated via the extended CONNECT protocol with a :protocol value of "webtransport".

The recipient MUST respond with a connection error of type WTHEADERS_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. This allows WebTransport Streams to participate in header compression and flow control.

3.6. Initiating the Extended CONNECT Handshake

An endpoint that wishes to establish a WebTransport session over an HTTP/2 stream follows the extended CONNECT handshake procedure defined in [RFC8441], specifying "webtransport" for the :protocol psuedo-header field.

The :scheme and :path psuedo-headers are required by [RFC6455]. The scheme of the target URI MUST be set to "https" for all :protocol values. The path is used to identify the specific WebTransport server instance for negotiation and MAY be set to "/" (an empty path component).

Implementations should note that the Origin, Sec-WebSocket-Version, Sec-WebSocket-Protocol, and Sec-WebSocket-Extensions header fields are not required to be included in the CONNECT request and response header fields, since this handshake mechanism is not being used to negotiate a WebSocket connection.

If the response to the extended CONNECT request indicates success of the handshake, then all further data sent or received on the new

HTTP/2 stream is considered to be that of the WebTransport protocol and follows the semantics defined by that protocol. If the response indicates failure of the handshake, any WebTransport Streams that reference the WebTransport Connect Stream that failed to establish MUST also be reset.

3.7. Examples

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

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

SETTINGS
SETTINGS_ENABLE_CONNECT_[..] = 1
SETTINGS_ENABLE_WEBTRANSPORT = 1

                                     SETTINGS
                                     SETTINGS_ENABLE_CONNECT_[..] = 1
                                     SETTINGS_ENABLE_WEBTRANSPORT = 1

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

                                     HEADERS + END_HEADERS
                                     + STREAM_ID = 3
                                     :status = 200

WTHEADERS + END_HEADERS
+ STREAM_ID = 5
+ CONNECT_STREAM = 3
:method = GET
:scheme = https
:path = /
:authority = server.example.com

                                     WTHEADERS + END_HEADERS
                                     + STREAM_ID = 5
                                     + CONNECT_STREAM = 3
                                     :status = 200

DATA + STREAM_ID = 5
WebTransport Data

                                     DATA + STREAM_ID = 5 + END_STREAM
                                     WebTransport Data

DATA + STREAM_ID = 5 + END_STREAM
WebTransport Data
```

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

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

SETTINGS
SETTINGS_ENABLE_CONNECT_[..] = 1
SETTINGS_ENABLE_WEBTRANSPORT = 1

                                     SETTINGS
                                     SETTINGS_ENABLE_CONNECT_[..] = 1
                                     SETTINGS_ENABLE_WEBTRANSPORT = 1

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

                                     HEADERS + END_HEADERS
                                     + STREAM_ID = 3
                                     :status = 200

                                     WTHEADERS + END_HEADERS
                                     + STREAM_ID = 2
                                     + CONNECT_STREAM = 3
                                     :method = GET
                                     :scheme = https
                                     :path = /
                                     :authority = client.example.com

WTHEADERS + END_HEADERS
+ STREAM_ID = 2
+ CONNECT_STREAM = 3
:status = 200

                                     DATA + STREAM_ID = 2
                                     WebTransport Data

DATA + STREAM_ID = 2 + END_STREAM
WebTransport Data

                                     DATA + STREAM_ID = 2 + END_STREAM
                                     WebTransport Data
```

4. Using WebTransport Streams

Once the extended CONNECT handshake has completed and a WebTransport connect stream has been established, WTHEADERS frames can be sent that reference that stream in the Connect Stream ID field to establish WebTransport Streams. WebTransport Connect Streams are intended for exchanging metadata only and are RECOMMENDED to be long lived streams. Once a WebTransport Connect Stream is closed, all routing information it carries is lost, and subsequent WebTransport Streams cannot be created with WTHEADERS frames until the client completes another extended CONNECT handshake to establish a new WebTransport Connect Stream.

In contrast, WebTransport Streams established with WTHEADERS frames can be opened at any time by either endpoint and therefore need not remain open beyond their immediate usage as part of the WebTransport protocol.

An endpoint MUST NOT send DATA frames with a non-zero payload length on a WebTransport Connect Stream beyond the completion of the extended CONNECT handshake. If data is received by an endpoint on a WebTransport Connect Stream, it MUST reset that stream with a new error code, PROHIBITED_WT_CONNECT_DATA, indicating that additional data is prohibited on the Connect Stream when using "webtransport" as the :protocol value.

4.1. Stream States

WebTransport Connect Streams are regular HTTP/2 streams that follow the stream lifecycle described in Section 5.1 of [RFC7540]. WebTransport Streams established with the WTHEADERS frame also follow the same lifecycle as regular HTTP/2 streams, but have an additional dependency on the Connect Stream that they reference via their Connect Stream ID.

If the corresponding Connect Stream is reset, endpoints MUST reset the WebTransport Streams associated with that Connect Stream. If the Connect Stream is closed gracefully, endpoints SHOULD allow any existing WebTransport Streams to complete normally, however the Connect Stream SHOULD remain open while communication is expected to continue.

Endpoints SHOULD take measures to prevent a peer or intermediary from timing out the Connect Stream while its associated WebTransport Streams are expected to remain open. For example, an endpoint might choose to refresh a timeout on a Connect Stream any time a corresponding timeout is refreshed on a corresponding WebTransport Stream, such as when any data is sent or received on that WebTransport Stream.

An endpoint MUST NOT initiate new WebTransport Streams that reference a Connect Stream that is in the closed or half closed (remote) state. Endpoints process new WebTransport Streams only when the associated Connect Stream is in the open or half closed (local) state.

4.2. Interaction with HTTP/2 Features

WebTransport Streams are extended HTTP/2 streams, and all of the standard HTTP/2 features for streams still apply to WebTransport Streams. For example, WebTransport Streams are counted against the concurrent stream limit, which is defined in Section 5.1.2 of [RFC7540]. The connection level and stream level flow control principles are still valid for WebTransport Streams. Prioritizing the WebTransport Streams across different Connect Stream groupings does not make sense because they belong to different services.

Note that while HTTP/2 Stream IDs are used by WebTransport Streams to refer to their corresponding WebTransport Connect Streams, the Stream IDs themselves are an implementation detail and SHOULD NOT be vended to clients via a WebTransport API.

4.3. Intermediaries

WebTransport Connect Streams, and their corresponding WebTransport Streams, can be independently routed by intermediaries on the network path. The main purpose for a WebTransport Connect Stream is to facilitate imtermediary traversal by WebTransport Streams.

Any segment on which SETTINGS_ENABLE_WEBTRANSPORT has been negotiated MUST route all WebTransport Streams established by WTHEADERS frames on the same connection as their corresponding WebTransport Connect Streams.

If an intermediary cannot route WebTransport Streams on a subsequent segment of the path, it can fail the extended CONNECT handshake and prevent a WebTransport Connect Stream from being established for a given endpoint. In the event that additional WebTransport Streams reference that WebTransport Connect Stream, they will also be reset.

An example of such routing, for both client-initiated and server-initiated WebTransport streams, is shown in Figure 2 and in Figure 3. Note that "webtransport" is specified as the :protocol being negotiated by the CONNECT frame on both segments, and the corresponding stream is referenced by the Connect Stream ID field in the WTHEADERS frames.

```
+--------+   CONNECT (5)   +---------+    CONNECT (1)   +--------+
| client |>--------------->|  proxy  |>---------------->| server |
+--------+                 +---------+                  +--------+
   v                         ^     v                         ^
   |  WTHEADERS(7, CS=5)     |     |  WTHEADERS(3, CS=1)     |
   +-------------------------+     +-------------------------+
```

Figure 2: A client initiates a WebTransport Stream to a server.

```
+--------+   CONNECT (5)   +---------+    CONNECT (1)   +--------+
| client |>--------------->|  proxy  |>---------------->| server |
+--------+                 +---------+                  +--------+
    ^                         v     ^                        v
    |  WTHEADERS(4, CS=5)     |     |  WTHEADERS(2, CS=1)    |
    +-------------------------+     +------------------------+
```

Figure 3: A server initiates a WebTransport Stream to a client.

4.4. Session Termination

An Http2Transport session is terminated when either endpoint closes the stream associated with the CONNECT request that initiated the session. Upon learning about the session being terminated, both endpoints MUST stop sending new frames on the WebTransport Connect Stream associated with the CONNECT request and reset all WebTransport Streams associated with the session.

5. Security Considerations

WebTransport Streams established by the CONNECT handshake and the WTHEADERS frame are expected to be protected with a TLS connection. They inherit the security properties of this cryptographic context, as well as the security properties of client-server communication via HTTP/2 as described in [RFC7540].

The security considerations of [RFC8441] Section 8 and [RFC7540] Section 10, and Section 10.5.2 especially, apply to this use of the CONNECT method.

Http2Transport requires explicit opt-in through the use of an HTTP/2 SETTINGS parameter, avoiding potential protocol confusion attacks by ensuring the HTTP/2 server explicitly supports the WebTransport protocol. 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/2 itself, Http2Transport 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 WebTransport session within a connection is allocated a reasonable share of controlled resources, both when sending data and opening new streams.

6. IANA Considerations

This document adds an entry to the "HTTP/2 Frame Type" registry, the "HTTP/2 Settings" registry, and the "HTTP/2 Error Code" registry, all defined in [RFC7540]. It also registers an HTTP upgrade token in the registry established by [RFC7230].

6.1. HTTP/2 Frame Type Registry

The following entry is added to the "HTTP/2 Frame Type" registry established by Section 11.2 of [RFC7540].

Frame Type: WTHEADERS

Code: 0xFB

Specification: *RFC Editor: Please fill in this value with the RFC number for this document*

6.2. HTTP/2 Settings Registry

The following entry is added to the "HTTP/2 Settings" registry that was established by Section 11.3 of [RFC7540].

Code: 0xFB

Name: SETTINGS_ENABLE_WEBTRANSPORT

Initial Value: 0

Specification: *RFC Editor: Please fill in this value with the RFC number for this document*

6.3. 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: WTHEADERS_STREAM_ERROR

Code: 0xFB

Description: Invalid use of WTHEADERS frame

Specification: *RFC Editor: Please fill in this value with the RFC number for this document*

Name: PROHIBITED_WT_CONNECT_DATA

Code: 0xFC

Description:
Prohibited data sent on WebTransport Connect Stream

Specification: *RFC Editor: Please fill in this value with the RFC number for this document*

6.4. Upgrade Token Registration

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

Value: webtransport

Description: WebTransport over HTTP

Expected Version Tokens:

Reference: *RFC Editor: Please fill in this value with the RFC number for this document* and [I-D.vvv-webtransport-http3]

7. References

7.1. Normative References

[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>.

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

[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>.

[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>.

7.2. Informative References

[I-D.vvv-webtransport-http3]
Vasiliev, V., "WebTransport over HTTP/3", Work in Progress, Internet-Draft, draft-vvv-webtransport-http3-01, 3 November 2019, <http://www.ietf.org/internet-drafts/draft-vvv-webtransport-http3-01.txt>.

[I-D.vvv-webtransport-overview]
Vasiliev, V., "The WebTransport Protocol Framework", Work in Progress, Internet-Draft, draft-vvv-webtransport-overview-01, 3 November 2019, <http://www.ietf.org/internet-drafts/draft-vvv-webtransport-overview-01.txt>.

[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