webtrans   A. Frindell
Internet-Draft   Facebook Inc.
Intended status: Standards Track   E. Kinnear
Expires: 26 August 2021   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 will expire on 26 August 2021.

Copyright Notice

Table of Contents

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 workaround 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 not 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.

```
 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)|
+-+-------------+-----------------------------------------------+
|R|                    Session ID (31)                          |
+-+-------------------------------------------------------------+
|                          Padding (*)                        ...
+---------------------------------------------------------------+
```

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 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.

```
 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)|
+-+-------------+-----------------------------------------------+
|R|                 Session ID (31)                             |
+-+-------------------------------------------------------------+
|                            Data (*)                         ...
+---------------------------------------------------------------+
|                           Padding (*)                       ...
+---------------------------------------------------------------+
```

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 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 ]]                   [[ From Server ]]

SETTINGS
SETTINGS_ENABLE_WEBTRANSPORT = 1

                                    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

                                    HEADERS + END_HEADERS
                                    Stream ID = 3
                                    :status = 200

WT_STREAM
Stream ID = 5
Session ID = 3


DATA
Stream ID = 5
WebTransport Data

                                    DATA + END_STREAM
                                    Stream ID = 5
                                    WebTransport Data

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 ]]                    [[ From Server ]]

SETTINGS
SETTINGS_ENABLE_WEBTRANSPORT = 1

                                     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
                                     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