webtrans
Internet-Draft
Intended status: Standards Track
Expires: 14 January 2021

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

13 July 2020

WebTransport using HTTP/2
draft-kinnear-webtransport-http2-01

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 either endpoint to initiate 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

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

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

Copyright Notice

Table of Contents

1. Introduction

HTTP/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.

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

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.

Http2Transport introduces a new HTTP/2 frame 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.

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 over 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" pseudo-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/ (https://www.iana.org/assignments/http-upgrade-tokens/).

This token is used in the ":protocol" pseudo-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 either endpoint to establish streams. 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.

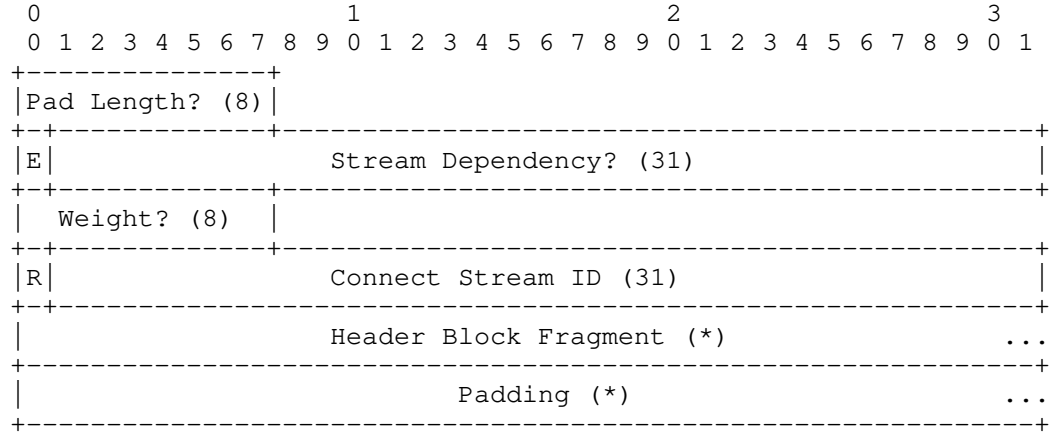

Figure 1: WTHEADERS Frame Format

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 pseudo-header field.

The :scheme and :path pseudo-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 limits 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 intermediary 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.

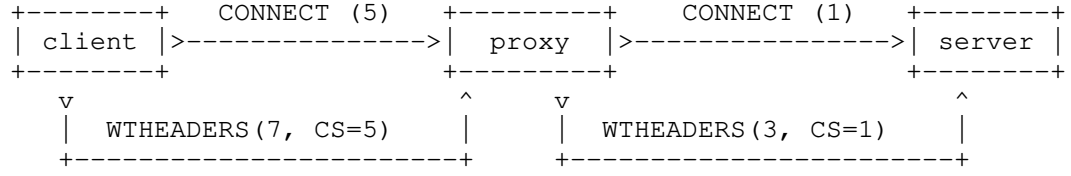

Figure 2: A client initiates a WebTransport Stream to a server.

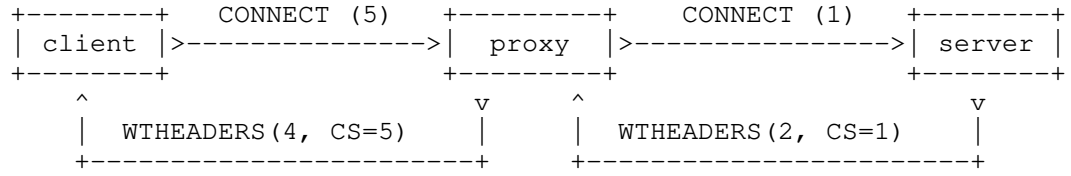

Figure 3: A server initiates a WebTransport Stream to a client.

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. Transport Properties

The WebTransport framework [I-D.vvv-webtransport-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.

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.

6. 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.

7. 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].

7.1. 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_

7.2. 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_

7.3. 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_

7.4. 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]

8. References

8.1. 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>.

8.2. Informative References

[I-D.vvv-webtransport-http3] Vasiliev, V., "WebTransport over HTTP/3", Work in Progress, Internet-Draft, draft-vvv-webtransport-http3-02, 30 June 2020, <http://www.ietf.org/internet-drafts/draft-vvv-webtransport-http3-02.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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 1, 2021

V. Vasiliev
Google

June 30, 2020

WebTransport over HTTP/3
draft-vvv-webtransport-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 Http3Transport, 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/vasilvv/webtransport/issues>. The web API draft corresponding to this document can be found at <https://wicg.github.io/web-transport/>.

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 January 1, 2021.

Table of Contents

1. Introduction

HTTP/3 [HTTP3] is a protocol defined on top of QUIC [QUIC-TRANSPORT] that can multiplex HTTP requests over a QUIC connection. This document defines Http3Transport, a mechanism for multiplexing non-HTTP data with HTTP/3 in a manner that conforms with the WebTransport protocol framework [OVERVIEW]. Using the mechanism described here, multiple Http3Transport 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

Http3Transport 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/3 connection is established, the client and server have to negotiate a specific set of QUIC transport parameters that indicate support for various Http3Transport features. Most notably, the "http3_transport_support" parameter signals Http3Transport support to the peer.

Http3Transport 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 Http3Transport session is established. As a part of this process, the client proposes, and the server confirms, a session ID. A session ID (SID) is unique within a given HTTP/3 connection, and is used to associate all of the streams and datagrams with the specific session.

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 Http3Transport.
- 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 a QUIC DATAGRAM frame [QUIC-DATAGRAM].

An Http3Transport session is terminated when the CONNECT stream that created it is closed.

3. Session IDs

In order to allow multiple Http3Transport sessions to occur within the same HTTP/3 connection, Http3Transport assigns every session a unique ID, further referred to as session ID. A session ID is a 62-bit number that is unique within the scope of an HTTP/3 connection, and is never reused even after the session is closed. The client unilaterally picks the session ID. As the IDs are encoded using QUIC variable length integers, the client SHOULD start with zero and then sequentially increment the IDs. A session ID is considered to be used, and thus ineligible for new transports, as soon as the client sends a request proposing it. These reuse requirements guarantee that both HTTP/3 endpoints have a consistent view of the session ID space.

The Session ID is a hop-by-hop property: if Http3Transport is proxied, the same session can have different IDs from the client's and server's perspective. Because of that, session IDs SHOULD NOT be exposed to the application.

4. Session Establishment

4.1. Establishing a Transport-Capable HTTP/3 Connection

In order to indicate support for Http3Transport, both the client and server MUST send an empty "http3_transport_support" transport parameter. Endpoints MUST NOT use any Http3Transport-related functionality unless the parameter has been negotiated. The negotiation is done through a QUIC transport parameter instead of an HTTP/3-level setting as it allows the server to customize the

transport parameters it intends to send based on whether the client has indicated support for Http3Transport.

If "http3_transport_support" is negotiated, support for the QUIC DATAGRAM extension MUST be negotiated. The "initial_max_bidi_streams" MUST be greater than zero, overriding the existing requirement in [HTTP3].

4.2. 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 "http3_transport_support" transport parameter implies that an endpoint supports extended CONNECT.

4.3. Creating a New Session

As Http3Transport sessions are established over HTTP/3, they are identified using the "https" URI scheme [RFC7230].

In order to create a new Http3Transport session, a client can send an HTTP CONNECT request. The ":protocol" pseudo-header field 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. The client MUST pick a new session ID as described in Section 3 and send it encoded as a hexadecimal literal in ":sessionid" header. 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 status code 200. Before accepting it, the HTTP/3 server MUST verify that the proposed session ID does not conflict with any currently open sessions, and it MAY verify that it was not used ever before on this connection. 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, an Http3Transport 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. Http3Transport does not support 0-RTT.

4.4. Limiting the Number of Simultaneous Sessions

From the flow control perspective, Http3Transport 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.

5. WebTransport Features

Http3Transport 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 Http3Transport sessions. On the wire, session IDs are encoded using the QUIC variable length integer scheme described in [QUIC-TRANSPORT].

5.1. Unidirectional streams

Once established, both endpoints can open unidirectional streams. The HTTP/3 control 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).

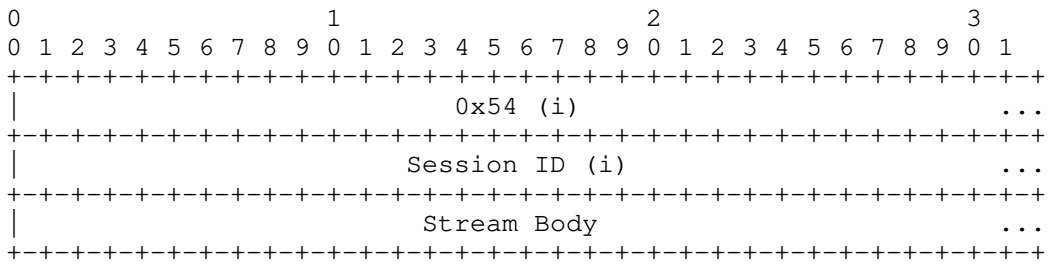

Figure 1: Unidirectional Http3Transport stream format

5.2. Client-Initiated Bidirectional Streams

Http3Transport clients 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.

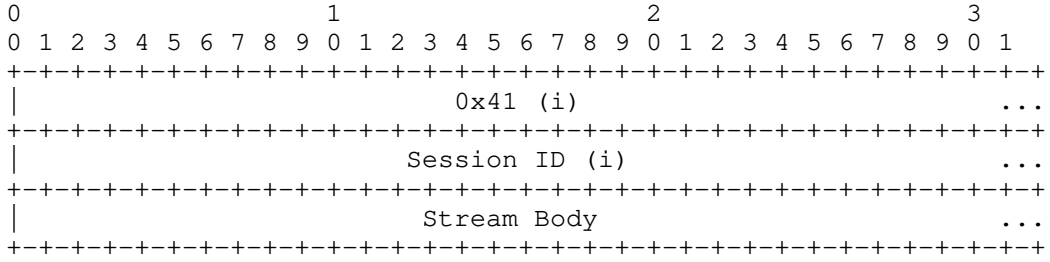

Figure 2: WEBTRANSPORT_STREAM frame format

5.3. Server-Initiated Bidirectional Streams

Http3Transport servers can initiate bidirectional streams by opening a bidirectional stream within the HTTP/3 connection. Note that since HTTP/3 does not define any semantics for server-initiated bidirectional streams, this document is a normative reference for the semantics of such streams for all HTTP/3 connections in which the "http3_transport_support" option is negotiated. The format of those streams SHALL be the session ID, encoded as a variable-length integer, followed by the user-specified stream data (Figure 3).

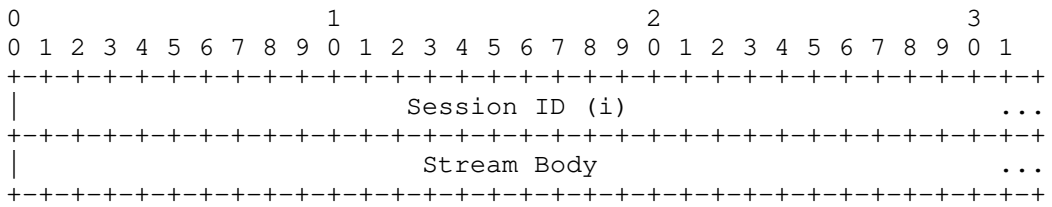

Figure 3: Server-initiated bidirectional stream format

5.4. Datagrams

Datagrams can be sent using the DATAGRAM frame as defined in [QUIC-DATAGRAM] and [H3-DATAGRAM]. For all HTTP/3 connections in which the "http3_transport_support" option is negotiated, the Flow Identifier is set to the session ID. In other words, the format of datagrams SHALL be the session ID, followed by the user-specified payload (Figure 4).

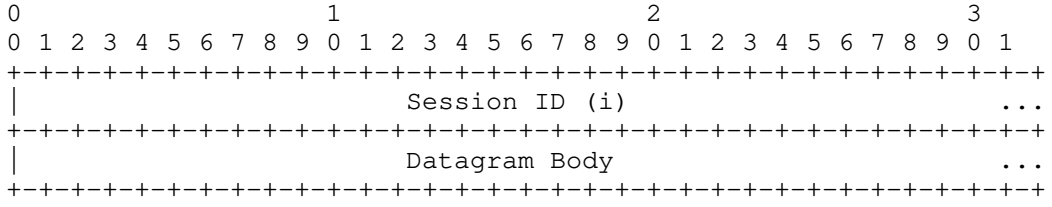

Figure 4: Datagram format

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.

6. Session Termination

An Http3Transport 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.

7. Transport Properties

Http3Transport supports most of the WebTransport features described in Table 1.

Property	Support
Stream independence	Always supported
Partial reliability	Always supported
Pooling support	Always supported
Connection mobility	Implementation-dependent

Table 1: Transport properties of Http3Transport

8. Security Considerations

Http3Transport satisfies all of the security requirements imposed by [QUIC-TRANSPORT] on WebTransport protocols, thus providing a secure framework for client-server communication in cases when the client is potentially untrusted. Since HTTP/3 is QUIC-based, a lot of the analysis in [WEBTRANSPORT-QUIC] applies here.

Http3Transport 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/3 itself, Http3Transport 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.

9. IANA Considerations

9.1. Upgrade Token Registration

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

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

Value: webtransport

Description: WebTransport over HTTP/3

Reference: This document

9.2. QUIC Transport Parameter Registration

The following entry is added to the "QUIC Transport Parameter Registry" registry established by [QUIC-TRANSPORT]:

The "http3_transport_support" parameter indicates that the specified HTTP/3 connection is Http3Transport-capable.

Value: 0x????

Parameter Name: http3_transport_support

Specification: This document

9.3. 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: 0x54

Frame Type: WEBTRANSPORT_STREAM

Specification: This document

9.4. Stream Type Registration

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

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

Code: 0x41

Stream Type: WebTransport stream

Specification: This document

Sender: Both

10. References

10.1. Normative References

[H3-DATAGRAM]
Schinazi, D., "Using QUIC Datagrams with HTTP/3", draft-schinazi-quic-h3-datagram-04 (work in progress), April 2020.

[HTTP3] Bishop, M., Ed., "Hypertext Transfer Protocol Version 3 (HTTP/3)", draft-ietf-quic-http (work in progress).

[OVERVIEW]
Vasiliev, V., "The WebTransport Protocol Framework", draft-ietf-webtrans-overview-latest (work in progress).

[QUIC-DATAGRAM]
Pauly, T., Kinnear, E., and D. Schinazi, "An Unreliable Datagram Extension to QUIC", draft-pauly-quic-datagram (work in progress).

[QUIC-TRANSPORT]
Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport (work in progress).

[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>.

[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>.

10.2. Informative References

[WEBTRANSPORT-QUIC]
Vasiliev, V., "WebTransport over QUIC", draft-vvv-webtransport-quic-02 (work in progress).

Author's Address

Victor Vasiliev
Google

Email: vasilvv@google.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 5, 2020

V. Vasiliev
Google

November 2, 2019

The WebTransport Protocol Framework
draft-vvv-webtransport-overview-01

Abstract

The WebTransport Protocol Framework enables clients constrained by the Web security model to communicate with a remote server using a secure multiplexed transport. It consists of a set of individual protocols that are safe to expose to untrusted applications, combined with a model that allows them to be used interchangeably.

This document defines the overall requirements on the protocols used in WebTransport, as well as the common features of the protocols, support for some of which may be optional.

Table of Contents

1. Introduction

The WebTransport Protocol Framework enables clients constrained by the Web security model to communicate with a remote server using a secure multiplexed transport. It consists of a set of individual protocols that are safe to expose to untrusted applications, combined with a model that allows them to be used interchangeably.

This document defines the overall requirements on the protocols used in WebTransport, as well as the common features of the protocols, support for some of which may be optional.

1.1. Background

Historically, web applications that needed a bidirectional data stream between a client and a server could rely on WebSockets [RFC6455], a message-based protocol compatible with the Web security model. However, since the abstraction it provides is a single ordered stream of messages, it suffers from head-of-line blocking (HOLB), meaning that all messages must be sent and received in order even if they are independent and some of them are no longer needed.

This makes it a poor fit for latency-sensitive applications which rely on partial reliability and stream independence for performance.

One existing option available to Web developers are WebRTC data channels [I-D.ietf-rtcweb-data-channel], which provide a WebSocket-like API for a peer-to-peer SCTP channel protected by DTLS. In theory, it is possible to use it for the use cases addressed by this specification. However, in practice, its user in non-browser-to-browser settings has been quite low due to its dependency on ICE (which fits poorly with the Web model) and userspace SCTP (which has very few implementations available).

An alternative design would be to layer WebSockets over HTTP/3 [I-D.ietf-quic-http] in a manner similar to how they are currently layered over HTTP/2 [RFC8441]. That would avoid head-of-line blocking and provide an ability to cancel a stream by closing the corresponding WebSocket object. However, this approach has a number of drawbacks, which all stem primarily from the fact that semantically each WebSocket is a completely independent entity:

- Each new stream would require a WebSocket handshake to agree on application protocol used, meaning that it would take at least one RTT to establish each new stream before the client can write to it.

- Only clients can initiate streams. Server-initiated streams and other alternative modes of communication (such as the QUIC DATAGRAM frame [I-D.pauly-quic-datagram]) are not available.

- While the streams would normally be pooled by the user agent, this is not guaranteed, and the general process of mapping a WebSocket to a server is opaque to the client. This introduces unpredictable performance properties into the system, and prevents optimizations which rely on the streams being on the same connection (for instance, it might be possible for the client to request different retransmission priorities for different streams, but that would be much more complex unless they are all on the same connection).

The WebTransport protocol framework avoids all of those issues by letting applications create a single transport object that can contain multiple streams multiplexed together in a single context (similar to SCTP, HTTP/2, QUIC and others), and can be also used to send unreliable datagrams (similar to UDP).

1.2. Conventions and Definitions

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.

WebTransport is a framework that aims to abstract away the underlying transport protocol while still exposing a few key transport-layer aspects to application developers. It is structured around the following concepts:

Transport session: A transport session is a single communication context established between a client and a server. It may correspond to a specific transport-layer connection, or it may be a logical entity within an existing multiplexed transport-layer connection. Transport sessions are logically independent from one another even if some sessions can share an underlying transport-layer connection.

Transport protocol: A transport protocol (WebTransport protocol in contexts where this might be ambiguous) is an instantiation of WebTransport over a given transport-layer protocol.

Datagram: A datagram is a unit of transmission that is treated atomically.

Stream: A stream is a sequence of bytes that is reliably delivered to the receiving application in the same order as it was transmitted by the sender. Streams can be of arbitrary length, and therefore cannot always be buffered entirely in memory. It is expected for transport protocols and APIs to provide partial stream data to the application before the stream has been entirely received.

Message: A message is a stream that is sufficiently small that it can be fully buffered before being passed to the application. WebTransport does not define messages as a primitive, since from the transport perspective they can be simulated by fully buffering a stream before passing it to the application. However, this distinction is important to highlight since some of the similar protocols and APIs (notably WebSocket [RFC6455]) use messages as a core abstraction.

Transport property: A transport property is a specific behavior that may or may not be exhibited by a transport. Some of those are inherent for all instances of a given transport protocol (TCP-

based transport cannot support unreliable delivery), while others can vary even within the same protocol (QUIC connections may or may not support connection migration).

Server: A WebTransport server is an application that accepts incoming transport sessions.

Client: A WebTransport client is an application that initiates the transport session and may be running in a constrained security context, for instance, a JavaScript application running inside a browser.

User agent: A WebTransport user agent is a software system that has an unrestricted access to the host network stack and can create transports on behalf of the client.

2. Common Transport Requirements

Since clients are not necessarily trusted and have to be constrained by the Web security model, WebTransport imposes certain requirements on any specific transport protocol used.

Any transport protocol used MUST use TLS [RFC8446] or a semantically equivalent security protocol (for instance, DTLS [I-D.ietf-tls-dtls13]). The protocols SHOULD use TLS version 1.3 or later, unless they aim for backwards compatibility with legacy systems.

Any transport protocol used MUST require the user agent to obtain and maintain explicit consent from the server to send data. For connection-oriented protocols (such as TCP or QUIC), the connection establishment and keep-alive mechanisms suffice. For other protocols, a mechanism such as ICE [RFC8445] can be used.

Any transport protocol used MUST limit the rate at which the client sends data. This SHOULD be accomplished via a feedback-based congestion control mechanism (such as [RFC5681] or [I-D.ietf-quic-recovery]).

Any transport protocol used MUST support simultaneously establishing multiple sessions between the same client and server.

Any transport protocol used MUST prevent the clients from establishing transport sessions to network endpoints that are not WebTransport servers.

Any transport protocol used MUST provide a way for servers to filter clients that can access it by checking the initiating origin [RFC6454].

Any transport protocol used MUST provide a way for a server endpoint location to be described using a URI [RFC3986]. This enables integration with various Web platform features that represent resources as URIs, such as Content Security Policy [CSP].

3. Session Establishment

WebTransport session establishment is most often asynchronous, although in some transports it can succeed instantaneously (for instance, if a transport is immediately pooled with an existing connection). A session MUST NOT be considered established until it is secure against replay attacks. For instance, in protocols creating a new TLS 1.3 session [RFC8446], this would mean that the user agent MUST NOT treat the session as established until it received a Finished message from the server.

In some cases, the transport protocol might allow transmitting data before the session is established; an example is TLS 0-RTT data. Since this data can be replayed by attackers, it MUST NOT be used unless the client has explicitly requested 0-RTT for specific streams or datagrams it knows to be safely replayable.

4. Transport Features

The following transport features are defined in this document. This list is not meant to be comprehensive; future documents may define new features for both new and already existing transports.

All transport protocols MUST provide datagrams, unidirectional and bidirectional streams in order to make the transport protocols easily interchangeable.

4.1. Datagrams

A datagram is a sequence of bytes that is limited in size (generally to the path MTU) and is not expected to be transmitted reliably. The general goal for WebTransport datagrams is to be similar in behavior to UDP while being subject to common requirements expressed in Section 2.

The WebTransport sender is not expected to retransmit datagrams, though it may if it is using a TCP-based protocol or some other underlying protocol that requires reliable delivery. WebTransport datagrams are not expected to be flow controlled, meaning that the

receiver might drop datagrams if the application is not consuming them fast enough.

The application MUST be provided with the maxiumum datagram size that it can send. The size SHOULD be derived from the result of performing path MTU discovery.

4.2. Streams

A unidirectional stream is a one-way reliable in-order stream of bytes where the initiator is the only endpoint that can send data. A bidirectional stream allows both endpoints to send data and can be conceptually represented as a pair of unidirectional streams.

The streams are in general expected to follow the semantics and the state machine of QUIC streams ([I-D.ietf-quic-transport], Sections 2 and 3). TODO: describe the stream state machine explicitly.

A WebTransport stream can be reset, indicating that the endpoint is not interested in either sending or receiving any data related to the stream. In that case, the sender is expected to not retransmit any data that was already sent on that stream.

Streams SHOULD be sufficiently lightweight that they can be used as messages.

Data sent on a stream is flow controlled by the transport protocol. In addition to flow controlling stream data, the creation of new streams is flow controlled as well: an endpoint may only open a limited number of streams until the peer explicitly allows creating more streams.

Every stream within a transport has a unique 64-bit number identifying it. Both unidirectional and bidirectional streams share the number space. The client and the server have to agree on the numbering, so it can be referenced in the application payload. WebTransport does not impose any other specific restrictions on the structure of stream IDs, and they should be treated as opaque 64-bit blobs.

4.3. Protocol-Specific Features

In addition to features described above, there are some capabilities that may be provided by an individual protocol but are not universally applicable to all protocols. Those are allowed, but any protocol is expected to be useful without those features, and portable clients should not rely on them.

A notable class of protocol-specific features are features available only in non-pooled transports. Since those transports have a dedicated connection, a user agent can provide clients with an extensive amount of transport-level data that would be too noisy and difficult to interpret when the connection is shared with unrelated traffic. For instance, a user agent can provide the number of packets lost, or the number of times stream data was delayed due to flow control. It can also expose variables related to congestion control, such as the size of the congestion window or the current pacing rate.

4.4. Bandwidth Prediction

Using congestion control state and transport metrics, the client can predict the rate at which it can send data. That is essential for many WebTransport use cases; for instance, real time media applications adapt the video bitrate to be a fraction of the throughput they expect to be available. While not all transport protocols can provide low-level transport details, all protocols SHOULD provide the client with an estimate of the available bandwidth.

5. Buffering and Prioritization

TODO: expand this outline into a full summary.

- Datagrams are intended for low-latency communications, so the buffers for them should be small, and prioritized over stream data.

- In general, the transport should not apply aggregation algorithms (e.g., Nagle's algorithm [RFC0896]) to datagrams.

6. Transport Properties

In addition to common requirements, each transport can have multiple optional properties associated with it. Querying them allows the client to ascertain the presence of features it can use without requiring knowledge of all protocols. This allows introducing new transports as drop-in replacements for existing ones.

The following properties are defined in this specification:

- Stream independence. This indicates that there is no head of line blocking between different streams.

- Partial reliability. This indicates that if a stream is reset, none of the data sent on it will be retransmitted. This also indicates that datagrams will not be retransmitted.

- Pooling support. Indicates that multiple transports using this transport protocol may end up sharing the same transport layer connection, and thus share a congestion controller and other contexts.

- Connection mobility. Indicates that the transport may continue existing even if the network path between the client and the server changes.

7. Security Considerations

Providing untrusted clients with a reasonably low-level access to the network comes with risks. This document mitigates those risks by imposing a set of common requirements described in Section 2.

WebTransport mandates the use of TLS for all protocols implementing it. This has a dual purpose. On one hand, it protects the transport from the network, including both potential attackers and ossification by middleboxes. On the other hand, it protects the network elements from potential confusion attacks such as the one discussed in Section 10.3 of [RFC6455].

One potential concern is that even when a transport cannot be created, the connection error would reveal enough information to allow an attacker to scan the network addresses that would normally be inaccessible. Because of that, the user agent that runs untrusted clients MUST NOT provide any detailed error information until the server has confirmed that it is a WebTransport endpoint. For example, the client must not be able to distinguish between a network address that is unreachable and one that is reachable but is not a WebTransport server.

WebTransport does not support any traditional means of HTTP-based authentication. It is not necessarily based on HTTP, and hence does not support HTTP cookies or HTTP authentication. Since it requires TLS, individual transport protocols MAY expose TLS-based authentication capabilities such as client certificates.

8. IANA Considerations

There are no requests to IANA in this document.

9. References

9.1. 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>.

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

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

9.2. Informative References

[CSP] W3C, "Content Security Policy Level 3", November 2019, <https://www.w3.org/TR/CSP/>.

[I-D.ietf-quic-http] Bishop, M., "Hypertext Transfer Protocol Version 3 (HTTP/3)", draft-ietf-quic-http-23 (work in progress), September 2019.

[I-D.ietf-quic-recovery] Iyengar, J. and I. Swett, "QUIC Loss Detection and Congestion Control", draft-ietf-quic-recovery-23 (work in progress), September 2019.

[I-D.ietf-quic-transport] Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-23 (work in progress), September 2019.

[I-D.ietf-rtcweb-data-channel]
Jesup, R., Loreto, S., and M. Tuexen, "WebRTC Data Channels", draft-ietf-rtcweb-data-channel-13 (work in progress), January 2015.

[I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", draft-ietf-tls-dtls13-33 (work in progress), October 2019.

[I-D.pauly-quic-datagram]
Pauly, T., Kinnear, E., and D. Schinazi, "An Unreliable Datagram Extension to QUIC", draft-pauly-quic-datagram-04 (work in progress), October 2019.

[RFC0896] Nagle, J., "Congestion Control in IP/TCP Internetworks", RFC 896, DOI 10.17487/RFC0896, January 1984, <https://www.rfc-editor.org/info/rfc896>.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <https://www.rfc-editor.org/info/rfc5681>.

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

[RFC8441] McManus, P., "Bootstrapping WebSockets with HTTP/2", RFC 8441, DOI 10.17487/RFC8441, September 2018, <https://www.rfc-editor.org/info/rfc8441>.

[RFC8445] Keranen, A., Holmberg, C., and J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal", RFC 8445, DOI 10.17487/RFC8445, July 2018, <https://www.rfc-editor.org/info/rfc8445>.

Author's Address

Victor Vasiliev
Google

Email: vasilvv@google.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 1, 2021

V. Vasiliev
Google

June 30, 2020

WebTransport over QUIC
draft-vvv-webtransport-quic-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 QuicTransport, a transport protocol that uses a dedicated QUIC [QUIC] connection and provides support for unidirectional streams, bidirectional streams and datagrams.

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/vasilvv/webtransport/issues>. The web API draft corresponding to this document can be found at <https://wicg.github.io/web-transport/>.

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 January 1, 2021.

Table of Contents

1. Introduction

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 QuicTransport, a transport protocol that uses a dedicated QUIC [QUIC]

connection and provides support for unidirectional streams, bidirectional streams and datagrams.

QUIC [QUIC] is a UDP-based multiplexed secure transport. It is the underlying protocol for HTTP/3 [I-D.ietf-quic-http], and as such is reasonably expected to be available in web browsers and server-side web frameworks. This makes it a compelling transport to base a WebTransport protocol on.

This document defines QuicTransport, a protocol conforming to the WebTransport protocol framework. QuicTransport is an application protocol running directly over QUIC. The protocol is designed to have low implementation overhead on the server side, meaning that server software that already has a working QUIC implementation available would not require large amounts of code to implement QuicTransport. Where possible, WebTransport concepts are mapped directly to the corresponding QUIC concepts.

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]. The diagrams describe encoding following the conventions described in Section 1.3 of [QUIC].

2. Protocol Overview

Each instance of QuicTransport uses a single dedicated QUIC connection. This allows the peers to exercise a greater level of control over the way their data is being transmitted. However, this also means that multiple instances of QuicTransport cannot be pooled, and thus do not benefit from sharing a congestion controller with other connections.

QuicTransport is designed to be a minimal extension of QUIC, and as such does not provide much higher-level functionality, such as pooling, exchanging metadata at session establishment, redirects, and other similar capabilties not provided by QUIC itself. Http3Transport [I-D.vvv-webtransport-http3] can be used in situations where these features are desired.

When a client requests a QuicTransport session to be created, the user agent establishes a QUIC connection to the specified address.

It verifies that the the server is a QuicTransport endpoint using ALPN, and additionally sends a client indication containing the requested path and the origin of the initiating website to the server. At that point, the connection is ready from the client's perspective. The server MUST wait until the client indication is received before processing any application data.

WebTransport streams are provided by creating an individual unidirectional or bidirectional QUIC stream. WebTransport datagrams are provided through the QUIC datagram extension [QUIC-DATAGRAM].

3. Connection Establishment

In order to establish a QuicTransport session, a QUIC connection must be established. From the client perspective, the session becomes established when the client both have received a TLS Finished message from the server and has sent a client indication. From the server perspective, the session is established after the client indication has been successfully processed.

3.1. Identifying as QuicTransport

In order to identify itself as a WebTransport application, QuicTransport relies on TLS Application-Layer Protocol Negotiation [RFC7301]. The user agent MUST request the ALPN value of "wq-vvv-01" and it MUST close the connection unless the server confirms that ALPN value.

3.2. Client Indication

In order to verify that the client's origin is allowed to connect to the server in question, the user agent has to communicate the origin to the server. This is accomplished by sending a special message, called client indication, on stream 2, which is the first client-initiated unidirectional stream.

The client indication is a sequence of key-value pairs that are formatted in the following way:

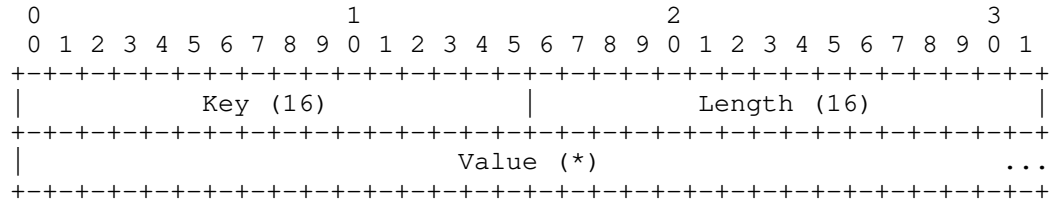

Figure 1: Client indication format

The pair includes the following fields:

Key: Indicates the field that is being expressed.

Length: Indicates the length of the value (the length of the key and the length itself are not included).

Value: The value of the field, the semantics of which are determined by the key.

A FIN on the stream 2 SHALL indicate that the message is complete. The client MUST send the entirety of the client indication and a FIN immediately after opening the connection. The server MUST NOT process any application data before receiving the entirety of the client indication. The total length of the client indication MUST NOT exceed 65,535 bytes.

In order to ensure that the user agent can send the client indication immediately, the server MUST set "initial_max_streams_uni" transport parameter to at least "1". The user agent MUST close the connection if the server sets "initial_max_streams_uni" to "0".

The server MUST ignore any field it does not recognize. All of the fields MUST be unique; the server MAY close the connection if any of the keys is used more than once.

3.2.1. Origin Field

In order to allow the server to enforce its origin policy, the user agent has to communicate the origin in the client indication. This can be accomplished using the "Origin" field:

Name: Origin

Key: 0x0000

Description: The origin [RFC6454] of the client initiating the connection.

The user agent MUST send the "Origin" field. The "Origin" field MUST be set to the origin of the client initiating the connection, serialized as described in the "serializing a request origin" section of [FETCH].

3.2.2. Path Field

In order to allow multiplexing multiple application on the same host-port tuple, QuicTransport allows specifying extra routing information in the path component of the URI. That component is communicated using the "Path" field in the client indication:

Name: Path

Key: 0x0001

Description: The path component of the QuicTransport URI.

The user agent MUST send a non-empty "Path" field. When the connection is initiated through a URI Section 6, that value SHALL be the "path-abempty" part, followed by a concatenation of the "?" literal and the "query" componenet if such is present. In case when "path-abempty" is empty, the value sent SHALL be "/".

Unlike HTTP, the "authority" portion of the URL is not communicated in the client indication. As QuicTransport has its own connection dedicated to it, the host name portion can be retrieved from the "server_name" TLS extension [RFC6066].

The server MAY use the value of the "Path" field in any way defined by the target application.

3.3. 0-RTT

QuicTransport provides applications with the ability to use the 0-RTT feature described in [RFC8446] and [QUIC]. 0-RTT allows a client to send data before the TLS session is fully established. It provides lower latency, but has the drawback of being vulnerable to replay attacks. Since only the application can make an informed decision as to whether some data is safe to send in that context, 0-RTT requires the client API to only send data over 0-RTT when specifically requested by the client.

0-RTT support in QuicTransport is OPTIONAL, as it is in QUIC and TLS 1.3.

4. Streams

QuicTransport unidirectional and bidirectional streams are created by creating a QUIC stream of the corresponding type. All other operations (read, write, close) are also mapped directly to the operations defined in [QUIC]. The QUIC stream IDs are the stream IDs that are exposed to the application.

5. Datagrams

QuicTransport uses the QUIC DATAGRAM frame [QUIC-DATAGRAM] to provide WebTransport datagrams. A QuicTransport endpoint MUST negotiate and support the DATAGRAM frame. The datagrams provided by the application are sent as-is.

6. QuicTransport URI Scheme

NOTE: the URI scheme definition in this section is provisional and subject to change, especially the name of the scheme.

QuicTransport uses the "quic-transport" URI scheme for identifying QuicTransport servers.

The syntax definition below uses Augmented Backus-Naur Form (ABNF) [RFC5234]. The definitions of "host", "port", "path-abempty", "query" and "fragment" are adopted from [RFC3986]. The syntax of a QuicTransport URI SHALL be:

```
quic-transport-URI = "quic-transport:" "//"
                              host [ ":" port ]
                              path-abempty
                              [ "?" query ]
                              [ "#" fragment ]
```

The "path-abempty" and the "query" portions of the URI are communicated to the server in the client indication as described in Section 3.2.2. The "quic-transport" URI scheme supports the "/.well-known/" path prefix defined in [RFC8615].

This document does not assign any semantics to the "fragment" portion of the URI. Any QuicTransport implementation MUST ignore those until a subsequent specification assigns semantics to those.

The "host" component MUST NOT be empty. If the "port" component is missing, the port SHALL be assumed to be 0.

In order to connect to a QuicTransport server identified by a given URI, the user agent SHALL establish a QUIC connection to the specified "host" and "port" as described in Section 3. It MUST immediately signal an error to the client if the port value is 0.

NOTE: this effectively requires the port number to be specified. This specification may include an actually usable default port number in the future.

7. Transport Properties

QuicTransport supports most WebTransport features as described in Table 1.

Property	Support
Stream independence	Always supported
Partial reliability	Always supported
Pooling support	Not supported
Connection mobility	Implementation-dependent

Table 1: Transport properties of QuicTransport

8. Security Considerations

QuicTransport 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 the client is potentially untrusted.

QuicTransport uses QUIC with TLS, and as such, provides the full range of security properties provided by TLS, including confidentiality, integrity and authentication of the server.

QUIC is a client-server protocol where a client cannot send data until either the handshake is complete or a previously established session is resumed. This ensures that clients cannot send data to a network endpoint that has not accepted an incoming connection. Furthermore, the QuicTransport session can be immediately aborted by the server through a connection close or a stateless reset, causing the user agent to stop the traffic from the client. This provides a defense against potential denial-of-service attacks on the network by untrusted clients.

QUIC provides a congestion control mechanism [I-D.ietf-quic-recovery] that limits the rate at which the traffic is sent. This prevents potentially malicious clients from overloading the network.

WebTransport requires user agents to continually verify that the server is still interested in talking to them. QuicTransport accomplishes that by virtue of QUIC being an acknowledgement-based protocol; if the client is attempting to send data, and the server

does not send any ACK frames in response, the client side of the QUIC connection will time out.

QuicTransport prevents WebTransport clients from connecting to arbitrary non-Web servers through the use of ALPN. Unlike TLS over TCP, successful ALPN negotiation is mandatory in QUIC. Thus, unless the server explicitly picks the QuicTransport ALPN value, the TLS handshake will fail.

QuicTransport uses a unidirectional QUIC stream to provide the server with the origin of the client.

In order to avoid the use of QuicTransport to scan internal networks, the user agents MUST NOT allow the clients to distinguish different connection errors before the correct ALPN is received from the server.

Since each instance of QuicTransport opens a new connection, a malicious client can cause resource exhaustion, both on the local system (through depleting file descriptor space or other per-connection resources) and on a given remote server. Because of that, user agents SHOULD limit the amount of simultaneous connections opened. The server MAY limit the amount of open connections from a given client.

9. IANA Considerations

9.1. ALPN Value Registration

The following entry is added to the "Application Layer Protocol Negotiation (ALPN) Protocol IDs" registry established by [RFC7301]:

The "wq-vvv-01" label identifies QUIC used as a protocol for WebTransport:

Protocol: QuicTransport

Identification Sequence: 0x77 0x71 0x2d 0x76 0x76 0x76 0x2d 0x30 0x31 ("wq-vvv-01")

Specification: This document

9.2. Client Indication Fields Registry

IANA SHALL add a registry for "QuicTransport Client Indication Fields" registry. Every entry in the registry SHALL include the following fields:

Name: The name of the field.

Key: The 16-bit unique identifier that is used on the wire.

Description: A brief description of what the parameter does.

Reference: The document that describes the parameter.

The IANA policy, as described in [RFC8126], SHALL be Standards Action for values between 0x0000 and 0x03ff; Specification Required for values between 0x0400 and 0xefff; and Private Use for values between 0xf000 and 0xffff.

9.3. URI Scheme Registration

This document contains the request for the registration of the URI scheme "quic-transport". The registration request is in accordance with [RFC7595].

Scheme name: quic-transport

Status: Permanent

Applications/protocols that use this scheme name: QuicTransport

Contact: IETF Chair chair@ietf.org [1]

Change controller: IESG iesg@ietf.org [2]

Reference: Section 6 of this document.

Well-Known URI Support: Section 6 of this document.

10. References

10.1. Normative References

[FETCH] WHATWG, "Fetch Standard", June 2020, <https://fetch.spec.whatwg.org/>.

[OVERVIEW] Vasiliev, V., "The WebTransport Protocol Framework", draft-ietf-webtrans-overview-latest (work in progress).

[QUIC] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-latest (work in progress).

[QUIC-DATAGRAM]
Pauly, T., Kinnear, E., and D. Schinazi, "An Unreliable Datagram Extension to QUIC", draft-pauly-quic-datagram-latest (work in progress).

[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>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

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

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <https://www.rfc-editor.org/info/rfc7301>.

[RFC7595] Thaler, D., Ed., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", BCP 35, RFC 7595, DOI 10.17487/RFC7595, June 2015, <https://www.rfc-editor.org/info/rfc7595>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[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>.

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

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://www.rfc-editor.org/info/rfc8615>.

10.2. Informative References

[I-D.ietf-quic-http]
Bishop, M., "Hypertext Transfer Protocol Version 3 (HTTP/3)", draft-ietf-quic-http-29 (work in progress), June 2020.

[I-D.ietf-quic-recovery]
Iyengar, J. and I. Swett, "QUIC Loss Detection and Congestion Control", draft-ietf-quic-recovery-29 (work in progress), June 2020.

[I-D.vvv-webtransport-http3]
Vasiliev, V., "WebTransport over HTTP/3", draft-vvv-webtransport-http3-01 (work in progress), November 2019.

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <https://www.rfc-editor.org/info/rfc6066>.

10.3. URIs

[1] mailto:chair@ietf.org

[2] mailto:iesg@ietf.org

Author's Address

Victor Vasiliev
Google

Email: vasilvv@google.com