

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 4, 2019

V. Vasiliev
Google
May 3, 2019

WebTransport over HTTP/3
draft-vvv-webtransport-http3-00

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.

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 November 4, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Terminology	2
2. Protocol Overview	3
3. Session IDs	4
4. Session Establishment	4
4.1. Establishing a Transport-Capable HTTP/3 Connection	4
4.2. Extended CONNECT in HTTP/3	4
4.3. Creating a New Session	5
4.4. Limiting Number of Simultaneous Sessions	5
5. WebTransport Features	6
5.1. Unidirectional streams	6
5.2. Client-Initiated Bidirectional Streams	6
5.3. Server-Initiated Bidirectional Streams	7
5.4. Datagrams	7
6. Session Termination	8
7. Transport Properties	8
8. Security Considerations	8
9. IANA Considerations	9
9.1. Upgrade Token Registration	9
9.2. QUIC Transport Parameter Registration	9
9.3. Frame Type Registration	10
9.4. Stream Type Registration	10
10. References	10
10.1. Normative References	10
10.2. Informative References	12
Author's Address	12

1. Introduction

HTTP/3 [HTTP3] is a protocol defined on top of QUIC [QUIC-TRANSPORT] that can provide multiplexed HTTP requests within the same QUIC connection. This document defines Http3Transport, a mechanism for embedding arbitrary streams of non-HTTP data into HTTP/3 in a manner that it can be used within WebTransport model [OVERVIEW]. Using the mechanism described here, multiple Http3Transport can be transmitted 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 connection; a WebTransport is one of potentially many applications that accepts WebTransport sessions, which HTTP/3 server can multiplex using the mechanisms defined in this document.

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 the server have to negotiate a specific set of QUIC transport parameters that would allow HTTP/3 connection to back an Http3Transport later, notably, the "http3_transport_support" parameter that signals Http3Transport support to the peer.

Http3Transport session begins with the client sending 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 in following ways:

- o A client can create a bidirectional stream using a special indefinite-length HTTP/3 frame that transfers ownership of the stream to Http3Transport.
- o A server can create a bidirectional stream, which is possible since HTTP/3 does not define any semantics for server-initiated bidirectional streams.
- o Both client and server can create a unidirectional stream using a special stream type.
- o A datagram can be sent using QUIC DATAGRAM frame [QUIC-DATAGRAM].

Http3Transport is terminated when the corresponding CONNECT stream 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 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 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 session ID space.

Session ID is a hop-by-hop property: if Http3Transport is proxied, the same session can have different IDs from client's and from 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, the client MAY send an empty "http3_transport_support" transport parameter, and the server MAY echo it in response. The peers MUST NOT use any Http3Transport-related functionality unless the parameter is negotiated. The negotiation is done through a QUIC transport parameter instead of HTTP/3-level setting in order to ensure that the server is aware of the connection being Http3Transport-capable when deciding which server transport parameters to send.

If "http3_transport_support" is negotiated, support for QUIC DATAGRAM frame 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 SETTINGS_ENABLE_CONNECT_PROTOCOL parameter. That parameter is only defined for HTTP/2. This document does not create a new parameter to support extended CONNECT in general HTTP/3 context; instead, "http3_transport_support" transport parameter implies that a peer understands extended CONNECT.

4.3. Creating a New Session

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 perspective, an Http3Transport session is established when the client receives a 200 response. From the server perspective, a session is established once it sends a 200 response. Both endpoints MUST NOT open any streams or send any datagrams before the session is established. Http3Transport does not support 0-RTT.

4.4. Limiting 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:

- o "HTTP_REQUEST_REJECTED" error code defined in [HTTP3] indicates the receiving HTTP/3 stack that the request was not processed in any way.
- o 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 a full set of 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, those are encoded using 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).

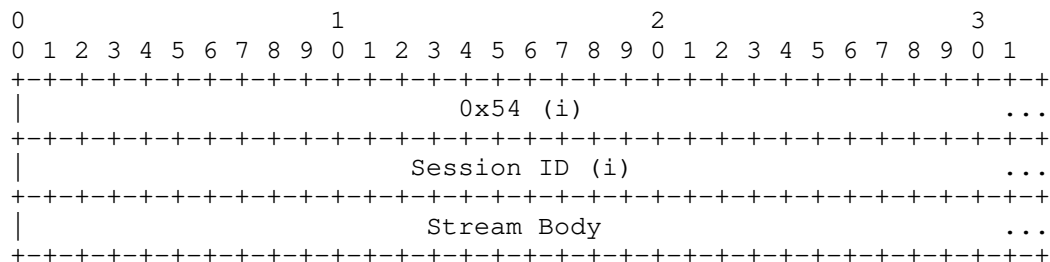


Figure 1: Unidirectional Http3Transport stream format

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.

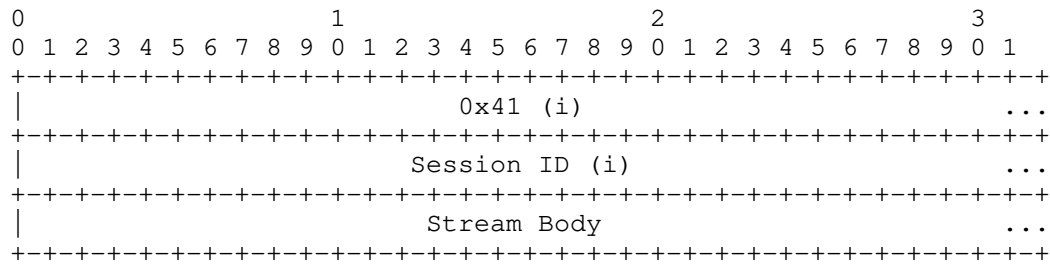


Figure 2: WEBTRANSPORT_STREAM frame format

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).

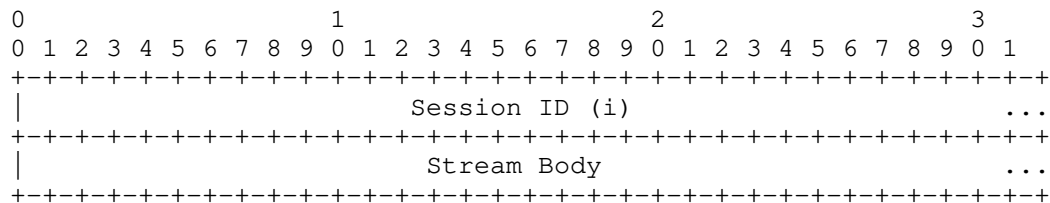


Figure 3: Server-initiated bidirectional stream format

5.4. Datagrams

Datagrams can be sent using the DATAGRAM frame as defined in [QUIC-DATAGRAM]. Just as with server-initiated bidirectional streams, the HTTP/3 specification does not assign any semantics to the datagrams, hence making this document a normative reference for all HTTP/3 connections in which the "http3_transport_support" option is negotiated. The format of those datagrams SHALL be the session ID, followed by the user-specified payload (Figure 4).

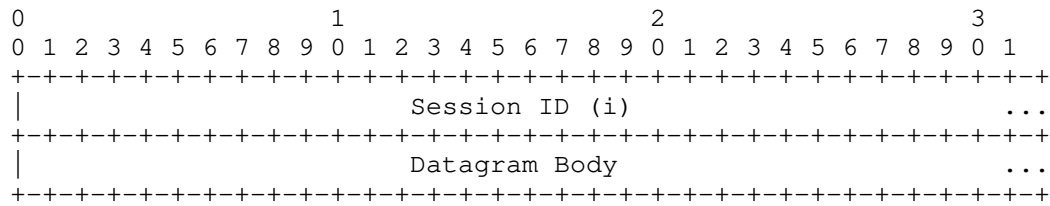


Figure 4: Datagram format

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 peer 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 WebTransport features as 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 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

[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-vvv-webtransport-overview-00 (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-00 (work in progress).

Author's Address

Victor Vasiliev
Google

Email: vasilvv@google.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 4, 2019

V. Vasiliev
Google
May 3, 2019

The WebTransport Protocol Framework
draft-vvv-webtransport-overview-00

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.

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 November 4, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Background	2
1.2. Definitions	3
2. Common Transport Requirements	5
3. Session Establishment	5
4. Transport Features	6
4.1. Datagrams	6
4.2. Streams	6
4.3. Protocol-Specific Features	7
4.4. Bandwidth Prediction	7
5. Buffering and Prioritization	8
6. Transport Properties	8
7. Security Considerations	8
8. IANA Considerations	9
9. References	9
9.1. Normative References	9
9.2. Informative References	10
Author's Address	11

1. Introduction

1.1. Background

Historically, web applications that needed bidirectional data stream between a client and a server could rely on WebSockets [RFC6455], a message-based protocol compatible with 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 the 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 general, it is possible to use it for the use cases addressed by this specification; however, in practice, its adoption in a non-browser-to-browser by the web developers has been quite low due to dependency on ICE (which fits poorly with the Web model) and userspace SCTP (which has very few implementations available).

Another option potentially available is layering 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:

- o Each new stream would require a WebSocket handshake to agree on application protocol used, meaning that it would take at least one RTT for each new stream before the client can write to it.
- o Only clients can initiate streams. Server-initiated streams and other alternative modes of communication (such as QUIC DATAGRAM frame) are not available.
- o While the streams would normally be pooled by the user agent, this is not guaranteed, and the general process of mapping a WebSocket to the end 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 impossible 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. 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 the specific transport-layer aspects of it to the application developers. It is structured around the following concepts:

Transport: A transport is a session established between a client and a server. It may correspond to a specific physical connection on the transport layer, or it may be a logical entity within an existing multiplexed connection. Each instance of a transport is

logically independent of each other even if some transports can share same connection underneath.

Transport protocol: A transport protocol (WebTransport protocol in context where this might be ambiguous) is a protocol that can be used to back a transport on the wire. It can provide the transport features described in this document, and is expected to fulfill all of the requirements described herein.

Datagram: A datagram is a unit of transmission that is generally treated as a single PDU by underlying network layer protocols, and, absent fragmentation, is expected to be treated atomically by the queues in the network.

Stream: A stream is a sequence of bytes that is delivered to the receiving application in the same order as it is transmitted by the sender. Streams are assumed to be sufficiently long that they cannot be buffered entirely into memory, thus requiring the transport protocol and the API to provide stream data before the stream is finished.

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 feature: A transport feature refers to the ability of a specific transport to provide a specific way of communicating data, such as supporting datagrams or streams.

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 the 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, for instance, the web browser.

2. Common Transport Requirements

Since the clients are potentially untrusted 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 an 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] may 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 the network endpoints that are not WebTransport servers.

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

3. Session Establishment

WebTransport session establishment is in general 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.

The client **MUST NOT** open streams or send datagrams until the session is established. In some situations, it might be possible for the client to be able to start sending data earlier, notably using TLS 0-RTT. In those situations, the user agent **MAY** provide client with ability to send a limited amount of data (using either streams or datagrams). The client **MUST** explicitly request for 0-RTT to be used.

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 **SHOULD** 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 reliable. 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 maximum 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.

As streams are reliable, the data sent on a stream has to be flow controlled by the transport protocol. In addition to the flow control for the stream data, the creation of new streams has to be 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 a lot of WebTransport use cases; for instance, real time media applications adapt the video bitrate to be a fraction of throughput they expect to be available. While not all transport protocols can provide low-level transport details, any protocol SHOULD provide a way to estimate the bandwidth available to the client.

5. Buffering and Prioritization

TODO: expand this outline into a full summary.

- o Datagrams are intended for low-latency communications, so the buffers for them should be small, and prioritized over stream data.
- o In general, the transport should not use any aggregation algorithms, e.g. Nagle's algorithm [RFC0896].

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 nature of transport without being aware of a specific implementation, thus simplifying introducing new transports as a drop-in replacement.

The following properties are defined in this specification:

- o Stream independence. Indicates that there is no head of line blocking between different streams.
- o Partial reliability. Indicates that if stream is reset, none of the data sent on it will be retransmitted. Indicates that datagrams will not be retransmitted.
- o Pooling support. Indicates that multiple transports using this transport protocol may end up sharing the same transport layer connection, and thus share the congestion control and other context.
- o 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 a lot of 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 that is reachable but is not a WebTransport server.

WebTransport does not support any traditional means of browser-based authentication. It is not based on HTTP, and hence does not support HTTP cookies or HTTP authentication. Since it uses TLS, individual transport protocols MAY expose TLS-based authentication capabilities such as client certificates. However, since in some of those protocols, multiple transports can be pooled within the same TLS connection, such features would not be universally available.

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>>.
- [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

- [I-D.ietf-quic-http]
Bishop, M., "Hypertext Transfer Protocol Version 3 (HTTP/3)", draft-ietf-quic-http-20 (work in progress), April 2019.
- [I-D.ietf-quic-recovery]
Iyengar, J. and I. Swett, "QUIC Loss Detection and Congestion Control", draft-ietf-quic-recovery-20 (work in progress), April 2019.
- [I-D.ietf-quic-transport]
Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-20 (work in progress), April 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-dtls13-31 (work in progress), March 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: November 4, 2019

V. Vasiliev
Google
May 3, 2019

WebTransport over QUIC
draft-vvv-webtransport-quic-00

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-TRANSPORT] connection and provides support for unidirectional streams, bidirectional streams and datagrams.

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 November 4, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Terminology	2
2. Protocol Overview	3
3. Connection Establishment	3
3.1. Identifying as QuicTransport	3
3.2. Verifying the Origin	3
3.3. 0-RTT	4
4. Streams	4
5. Datagrams	4
6. Transport Properties	4
7. Security Considerations	5
8. IANA Considerations	6
8.1. ALPN Value Registration	6
8.2. QUIC Transport Parameter Registration	6
9. References	7
9.1. Normative References	7
9.2. Informative References	7
Author's Address	8

1. Introduction

QUIC [QUIC-TRANSPORT] 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, an adaptation of QUIC to WebTransport model. The protocol is designed to be low-overhead on the server side, meaning that server software that already has a working QUIC implementation available would not require a large amount 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].

2. Protocol Overview

Each 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 congestion control context with other potentially already existing connections. Http3Transport [I-D.vvv-webtransport-http3] can be used in situations where such pooling is beneficial.

When a client requests a QuicTransport 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 that the client is allowed to connect to the specified endpoint using "web_accepted_origins" transport parameter. Once the verification succeeds and the QUIC connection is ready, the client can send and receive streams and datagrams.

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 receives a TLS Finished message from the server.

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" and it MUST NOT establish the session unless that value is accepted.

3.2. Verifying the Origin

In order to verify that the client is authorized to access a specific WebTransport server, QuicTransport has a mechanism to verify the origin [RFC6454] associated with the client. The server MUST send a "web_accepted_origins" transport parameter which SHALL be one of the following:

- o A value "*", indicating that any origin is accepted.
- o A comma-separated list of accepted origins, serialized as described in Section 6 of [RFC6454].

In the latter case, the user agent MUST verify that one of the origins is identical (as defined in Section 5 of [RFC6454]) to the origin of the client; otherwise, it MUST abort the session establishment.

3.3. 0-RTT

QuicTransport provides applications with ability to use the 0-RTT feature described in [RFC8446] and [QUIC-TRANSPORT]. 0-RTT allows a client to send data before the TLS session is fully established. It provides a lower latency, but has the drawback of being vulnerable to replay attacks as a result. Since only the application can make the decision of 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.

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 corresponding type. All other operations (read, write, close) are also mapped directly to the operations as defined in [QUIC-TRANSPORT]. 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. The datagram ID SHALL be absent.

The datagrams sent using QuicTransport MUST be subject to congestion control.

6. Transport Properties

QuicTransport supports most of 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

7. 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 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 the user agent will prevent the client from sending data to network endpoints that are not QuicTransport endpoints. 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.

QuicTransport prevents the WebTransport clients connecting to arbitrary non-Web servers through the use of ALPN. Unlike TLS over TCP, successfully ALPN negotiation is mandatory in QUIC. Thus, unless the server explicitly picks "wq" as the ALPN value, the TLS handshake will fail. It will also fail unless the "web_accepted_origins" is present.

QuicTransport uses a QUIC transport parameter to provide the user agent with an origin whitelist. The origin is not sent explicitly,

as TLS ClientHello messages are sent in cleartext; instead, the server provides the user agent with a whitelist of origins that are allowed to connect to it.

In order to avoid the use of QuicTransport, 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, the user agents SHOULD limit the amount of simultaneous connections opened. The server MAY limit the amount of connections open by the same client.

8. IANA Considerations

8.1. ALPN Value Registration

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

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

Protocol: QuicTransport

Identification Sequence: 0x77 0x71 ("wq")

Specification: This document

8.2. QUIC Transport Parameter Registration

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

The "web_accepted_origins" parameter allows the server to indicate origins that are permitted to connect to it:

Value: 0x????

Parameter Name: web_accepted_origins

Specification: This document

9. References

9.1. Normative References

[OVERVIEW]

Vasiliev, V., "The WebTransport Protocol Framework", draft-vvv-webtransport-overview-00 (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).

[QUIC-TRANSPORT]

Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-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>>.

[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>>.

[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

[I-D.ietf-quic-http]

Bishop, M., "Hypertext Transfer Protocol Version 3 (HTTP/3)", draft-ietf-quic-http-20 (work in progress), April 2019.

[I-D.ietf-quic-recovery]

Iyengar, J. and I. Swett, "QUIC Loss Detection and
Congestion Control", draft-ietf-quic-recovery-20 (work in
progress), April 2019.

Author's Address

Victor Vasiliev
Google

Email: vasilvv@google.com