WEBTRANS
Internet-Draft
Intended status: Standards Track
Expires: 23 April 2026

E. Kinnear
Apple Inc.
V. Vasiliev
Google

20 October 2025

The WebTransport Protocol Framework
draft-ietf-webtrans-overview-11

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 an abstract 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.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://ietf-wg-webtrans.github.io/draft-ietf-webtrans-overview/draft-ietf-webtrans-overview.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-webtrans-overview/.

Discussion of this document takes place on the WebTransport Working Group mailing list (mailto:webtransport@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/webtransport/. Subscribe at https://www.ietf.org/mailman/listinfo/webtransport/.

Source for this draft and an issue tracker can be found at https://github.com/ietf-wg-webtrans/draft-ietf-webtrans-overview.

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

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 an abstract 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 reliable stream of messages, it suffers from head-of-line blocking, meaning that all messages must be sent and received in order even if they could be processed independently of each other, and some messages may no longer be relevant. 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 [RFC8831], 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, it has not seen wide adoption outside of browser-to-browser settings due to its dependency on ICE (which fits poorly with the Web model) and userspace SCTP (which has a limited number of implementations available due to not being used in other contexts).

An alternative design would be to open multiple WebSocket connections over HTTP/3 [RFC9220]. That would avoid head-of-line blocking and provide an ability to cancel a stream by closing the corresponding WebSocket session. 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 [RFC9221]) 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).

WebTransport 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 also be 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:

WebTransport session: A WebTransport 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. WebTransport sessions are logically independent from one another even if some sessions can share an underlying transport-layer connection.

WebTransport protocol: A WebTransport protocol is a specific protocol that can be used to establish a WebTransport session.

Datagram: A datagram is a unit of transmission that is limited in size (typically to the path MTU), does not have an expectation of being delivered reliably, and is treated atomically by the transport.

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. WebTransport protocols and APIs are expected 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.

Application: A WebTransport application refers to executable code that is provided by a developer to perform some, often user-visible, function, such as sending and receiving data. For example, a JavaScript application using WebTransport that is running inside a browser or code running within an executable that makes outgoing or accepts incoming WebTransport sessions.

Server: A WebTransport server is an application that accepts incoming WebTransport sessions. In cases when WebTransport is served over a multiplexed protocol (such as HTTP/2 or HTTP/3), "WebTransport server" refers to a handler for a specific multiplexed endpoint (e.g. an application handling specific HTTP resource), rather than the application listening on a given TCP or UDP socket.

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.

Endpoint: An endpoint refers to either a Server or a Client.

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.

Event: An event is a notification, callback, or signal that a WebTransport endpoint can provide to a WebTransport application to notify it that some change of interest to the application has occurred.

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 protocol used.

All WebTransport protocols MUST use TLS [RFC8446] or a semantically equivalent security protocol (for instance, DTLS [RFC9147]). The protocols SHOULD use TLS version 1.3 or later, unless they aim for backwards compatibility with legacy systems.

All WebTransport protocols 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. STUN Consent Freshness [RFC7675] is another example of a mechanism satisfying this requirement.

All WebTransport protocols 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 [RFC9002]).

All WebTransport protocols MUST support simultaneously establishing multiple sessions between the same client and server.

All WebTransport protocols MUST prevent clients from establishing transport sessions to network endpoints that are not WebTransport servers.

All WebTransport protocols MUST provide a way for the user agent to indicate the origin [RFC6454] of the client to the server.

All WebTransport protocols 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].

All WebTransport protocols MUST provide a way for the session initiator to negotiate a subprotocol with the peer when establishing a WebTransport session. The session initiator provides an optional list of subprotocols to the peer. The peer selects one and responds indicating the selected subprotocol or rejects the session establishment request if none of the subprotocols are supported. Note that the semantics of individual subprotocol token values is determined by the WebTransport resource in question and are not registered in IANA’s "ALPN Protocol IDs" registry.

3. Session Establishment

WebTransport session establishment is an asynchronous process. A session is considered _ready_ from the client's perspective when the server has confirmed that it is willing to accept the session with the provided origin and URI. WebTransport protocols MAY allow clients to send data before the session is ready; however, they MUST NOT use mechanisms that are unsafe against replay attacks without an explicit indication from the client.

3.1. Application Protocol Negotiation

WebTransport sessions offer a protocol negotiation mechanism, similar to TLS Application-Layer Protocol Negotiation Extension (ALPN) [RFC7301].

When establishing a session, a WebTransport client can offer the server a list of protocols that it would like to use on that session, in preference order. When the server receives such a list, it selects a single choice from that list and communicates that choice to the client. A server that does not wish to use any of the protocols offered by the client can reject the WebTransport session establishment attempt.

4. Transport Features

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

4.1. Session-Wide Features

Any WebTransport protocol SHALL provide the following operations on the session:

establish a session Create a new WebTransport session given a URI [RFC3986] of the requester. An origin [RFC6454] MUST be given if the WebTransport session is coming from a browser client; otherwise, it is OPTIONAL.

terminate a session Terminate the session while communicating to the peer an unsigned 32-bit error code and an error reason string of at most 1024 bytes. As soon as the session is terminated, no further application data will be exchanged on it. The error code and string are optional; the default values are 0 and "". The delivery of the error code and string MAY be best-effort.

drain a session Indicate to the peer that it expects the session to

be gracefully terminated as soon as possible. Either endpoint MAY continue using the session and MAY open new streams. This signal is intended to allow intermediaries and endpoints to request a session be drained of traffic without enforcement.

export keying material Derive a TLS keying material exporter (Section 7.5 of [RFC8446]) to provide keying material specific to the WebTransport session.

Any WebTransport protocol SHALL provide the following events:

session terminated event Indicates that the WebTransport session has been terminated, either by the peer or by the local networking stack, and no user data can be exchanged on it any further. If the session has been terminated as a result of the peer performing the "terminate a session" operation above, a corresponding error code and an error string can be provided.

session draining event Indicates that the WebTransport session has been asked to drain as soon as possible. Continued use of the session, including opening new streams is discouraged, but allowed.

4.2. 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.

A WebTransport sender is not expected to retransmit datagrams, though it may end up doing so if it is using TCP or some other underlying protocol that only provides 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.

In the WebTransport model, all of the outgoing and incoming datagrams are placed into a size-bound queue (similar to a network interface card queue).

Any WebTransport protocol SHALL provide the following operations on the session:

send a datagram Enqueues a datagram to be sent to the peer. This can potentially result in the datagram being dropped if the queue is full.

receive a datagram Dequeues an incoming datagram, if one is available.

get maxiumum datagram size Returns the largest size of the datagram that a WebTransport session is expected to be able to send.

4.3. 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 ([RFC9000], Sections 2 and 3).

A WebTransport stream can be reset, indicating that the endpoint is not interested in either sending or receiving any data related to the stream. The sender of a stream can indicate an offset in the stream (possibly zero) after which data that was already sent will not be retransmitted.

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. From the perspective of the client, this is presented as a size-bounded queue of incoming streams.

Any WebTransport protocol SHALL provide the following operations on the session:

create a unidirectional stream Creates an outgoing unidirectional stream; this operation may block until the flow control of the underlying protocol allows for it to be completed.

create a bidirectional stream Creates an outgoing bidirectional stream; this operation may block until the flow control of the underlying protocol allows for it to be completed.

receive a unidirectional stream Removes a stream from the queue of

incoming unidirectional streams, if one is available.

receive a bidirectional stream Removes a stream from the queue of incoming bidirectional streams, if one is available.

Any WebTransport protocol SHALL provide the following operations on an individual stream:

send bytes Add bytes into the stream send buffer. The sender can also indicate a FIN, signalling the fact that no new data will be send on the stream. Not applicable for incoming unidirectional streams.

receive bytes Removes bytes from the stream receive buffer. FIN can be received together with the stream data. Not applicable for outgoing unidirectional streams.

abort send side Sends a signal to the peer that the write side of the stream has been aborted, including an offset in the stream that is reliably delivered. Discards the send buffer after that offset; if possible, no currently outstanding data after the provided send offset is transmitted or retransmitted. If omitted, the offset is presumed to be 0. An unsigned 32-bit error code can be supplied as a part of the signal to the peer; if omitted, the error code is presumed to be 0.

abort receive side Sends a signal to the peer that the read side of the stream has been aborted. Discards the receive buffer; the peer is typically expected to abort the corresponding send side in response. An unsigned 32-bit error code can be supplied as a part of the signal to the peer.

Any WebTransport protocol SHALL provide the following events for an individual stream:

send side aborted Indicates that the peer has aborted the corresponding receive side of the stream. An unsigned 32-bit error code from the peer may be available.

receive side aborted Indicates that the peer has aborted the corresponding send side of the stream. An unsigned 32-bit error code from the peer may be available.

all data committed Indicates that all of the outgoing data on the stream, including the end stream indication, is in the state where aborting the send side would have no further effect on any data being delivered.

For protocols, like HTTP/2, stream data might be passed to another component (like a kernel) for transmission. Once data is passed to that component it might not be possible to abort the sending of stream data without also aborting the entire connection. For these protocols, data is considered committed once it passes to the other component.

A protocol, like HTTP/3, that uses a more integrated stack might be able to retract data further into the process. For these protocols, sending on a stream might be aborted at any time until all data has been received and acknowledged by the peer, corresponding to the "Data Recvd" state in QUIC; see Section 3.1 of [QUIC].

5. Transport Properties

WebTransport defines common semantics for multiple protocols to allow them to be used interchangeably. Nevertheless, those protocols still have substantially different performance properties that an application may want to query.

The most notable property is support for unreliable data delivery. The protocol is defined to support unreliable delivery if:

* Resetting a stream results in the lost stream data no longer being retransmitted, and

* The datagrams are never retransmitted.

Another important property is pooling support. Pooling means that multiple transport sessions may end up sharing the same transport layer connection, and thus share a congestion controller and other contexts.

6. 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 provides confidentiality and integrity for the transport, protecting it from both potential attackers and ossification by intermediaries in the network.

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.

Since WebTransport requires TLS, individual transport protocols MAY expose TLS-based authentication capabilities such as client certificates and custom validation of server certificates, including validation using a client-specified set of server certificate hashes.

7. IANA Considerations

There are no requests to IANA in this document.

8. References

8.1. Normative References

[QUIC] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>.

[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/rfc/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/rfc/rfc3986>.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <https://www.rfc-editor.org/rfc/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/rfc/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/rfc/rfc8446>.

8.2. Informative References

[CSP] W3C, "Content Security Policy Level 3", October 2025, <https://www.w3.org/TR/CSP/>.

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

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

[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/rfc/rfc7301>.

[RFC7675] Perumal, M., Wing, D., Ravindranath, R., Reddy, T., and M. Thomson, "Session Traversal Utilities for NAT (STUN) Usage for Consent Freshness", RFC 7675, DOI 10.17487/RFC7675, October 2015, <https://www.rfc-editor.org/rfc/rfc7675>.

[RFC8831] Jesup, R., Loreto, S., and M. T端xen, "WebRTC Data Channels", RFC 8831, DOI 10.17487/RFC8831, January 2021, <https://www.rfc-editor.org/rfc/rfc8831>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>.

[RFC9002] Iyengar, J., Ed. and I. Swett, Ed., "QUIC Loss Detection and Congestion Control", RFC 9002, DOI 10.17487/RFC9002, May 2021, <https://www.rfc-editor.org/rfc/rfc9002>.

[RFC9147] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", RFC 9147, DOI 10.17487/RFC9147, April 2022, <https://www.rfc-editor.org/rfc/rfc9147>.

[RFC9220] Hamilton, R., "Bootstrapping WebSockets with HTTP/3", RFC 9220, DOI 10.17487/RFC9220, June 2022, <https://www.rfc-editor.org/rfc/rfc9220>.

[RFC9221] Pauly, T., Kinnear, E., and D. Schinazi, "An Unreliable Datagram Extension to QUIC", RFC 9221, DOI 10.17487/RFC9221, March 2022, <https://www.rfc-editor.org/rfc/rfc9221>.

Authors' Addresses

Eric Kinnear
Apple Inc.
Email: ekinnear@apple.com

Victor Vasiliev
Google
Email: vasilvv@google.com