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

Copyright Notice

Table of Contents

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:

- 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.

- Only clients can initiate streams. Server-initiated streams and other alternative modes of communication (such as QUIC DATAGRAM frame) 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 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 (WebTransprot protocol in context where this might be ambiguos) 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 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.

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