

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 7, 2020

E. Kinnear
T. Pauly
Apple Inc.
November 4, 2019

Using HTTP/2 as a Transport for Arbitrary Bytestreams
draft-kinnear-httpbis-http2-transport-02

Abstract

HTTP/2 provides multiplexing of HTTP requests over a single underlying transport connection. HTTP/2 Transport defines the use of the bidirectional extended CONNECT handshake to negotiate the use of application protocols using streams of an HTTP/2 connection as transport.

Status of This Memo

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

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

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 7, 2020.

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. Notational Conventions	2
2. The SETTINGS_ENABLE_BIDIRECTIONAL_CONNECT Parameter	3
3. Negotiating Bidirectional Transport	3
3.1. Initiating the Extended CONNECT Handshake	4
3.2. Responding to the Extended CONNECT Handshake	4
4. Using Tunnels Established via the Extended CONNECT Handshake	5
4.1. Example	5
5. IANA Considerations	6
6. Security Considerations	7
7. Acknowledgments	7
8. Normative References	7
Authors' Addresses	8

1. Introduction

HTTP/2 [RFC7540] provides a framing layer that describes the exchange of HTTP messages. This framing layer includes multiplexing of multiple streams on a single underlying transport connection, flow control, stream dependencies and priorities, and exchange of configuration information between endpoints.

Section 8.3 of [RFC7540] defines the HTTP CONNECT method for HTTP/2, which converts a HTTP/2 stream into a tunnel for arbitrary data. [RFC8441] describes the use of the extended CONNECT method to negotiate the use of the WebSocket Protocol [RFC6455] on an HTTP/2 stream.

This document extends the CONNECT handshake to allow both endpoints of an HTTP/2 connection to establish streams that tunnel data. It also defines a protocol name for use in the extended CONNECT handshake that allow negotiation of HTTP/2 streams that transport arbitrary bytestreams. Being able to transport application protocol data on individual HTTP/2 streams allows an underlying connection to be shared by multiple protocols and allows all protocols to benefit from the features provided by HTTP/2 framing.

1.1. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. The SETTINGS_ENABLE_BIDIRECTIONAL_CONNECT Parameter

As described in Section 5.5 of [RFC7540], SETTINGS parameters allow endpoints to negotiate use of protocol extensions that would otherwise generate protocol errors. Use of the CONNECT method extension defined in [RFC6455] requires the SETTINGS_ENABLE_CONNECT_PROTOCOL parameter to be received by a client prior to its use.

This document introduces another SETTINGS parameter, SETTINGS_ENABLE_BIDIRECTIONAL_CONNECT, which MUST have a value of 0 or 1.

Once a SETTINGS_ENABLE_BIDIRECTIONAL_CONNECT parameter has been sent with a value of 1, an endpoint MUST NOT send the parameter with a value of 0.

Upon receipt of SETTINGS_ENABLE_BIDIRECTIONAL_CONNECT with a value of 1, an endpoint MAY use the extended CONNECT defined in [RFC6455] with the protocol values defined in this document. An endpoint that supports receiving the extended CONNECT method SHOULD send this setting with a value of 1.

Note that [RFC6455] restricts SETTINGS_ENABLE_CONNECT_PROTOCOL to have no effect if received by a server. This document modifies that restriction and allows both SETTINGS_ENABLE_CONNECT_PROTOCOL and SETTINGS_ENABLE_BIDIRECTIONAL_CONNECT to take effect if received by either endpoint of an HTTP/2 connection.

3. Negotiating Bidirectional Transport

[RFC6455] defines the pseudo-header field :protocol which can indicate the protocol intended to be used on the tunnel established by the CONNECT method. Values for the :protocol pseudo-header field are maintained in an Upgrade Token Registry established by [RFC7230] for protocol-name tokens.

After receiving both SETTINGS_ENABLE_CONNECT_PROTOCOL and SETTINGS_ENABLE_BIDIRECTIONAL_CONNECT, either endpoint of an HTTP/2 connection can send a request in HEADERS frames to establish a new stream via the extended CONNECT method. Similarly, either endpoint may be required to respond to an incoming CONNECT request seeking to establish such a stream.

3.1. Initiating the Extended CONNECT Handshake

Endpoints using this mechanism to establish bidirectional transport over HTTP/2 streams follow the CONNECT handshake procedure defined in [RFC6455]. However, instead of supplying "websocket" for the :protocol psuedo-header field to indicate a WebSocket connection, they negotiate the use of a specific application protocol by specifying an appropriate value. This document registers "bytestream" as a value to be used when an out-of-band negotiation has already occurred and an application protocol wishes to transport arbitrary bytes on an HTTP/2 stream. Any endpoint supplying "bytestream" as a value for the :protocol psuedo-header MUST have previously negotiated the use of this value via another mechanism.

The :scheme and :path psuedo-headers are required by [RFC6455]. The scheme of the target URI MUST be set to "https" for all :protocol values. The path is used in the same manner as for the WebSocket protocol, and MAY be set to "/" (an empty path component) if not desired for use.

Implementations should note that the Origin, Sec-WebSocket-Version, Sec-WebSocket-Protocol, and Sec-WebSocket-Extensions header fields are not included in the CONNECT request and response header fields, since this handshake mechanism is not being used to negotiate a WebSocket connection.

If the response to the extended CONNECT request indicates success of the handshake, then all further data sent or received on the new HTTP/2 stream is considered to be that of the supplied :protocol value and follows the semantics defined by that protocol.

3.2. Responding to the Extended CONNECT Handshake

A recipient of the extended CONNECT method follows the same procedure outlined by [RFC8441].

If the recipient encounters a :protocol psuedo-header with an unknown value or a value corresponding to a protocol they do not support, or if the recipient encounters violations of the extended CONNECT handshake protocol, they MUST return an HTTP response with an appropriate error code, such as 400 Bad Request. Otherwise, unknown header fields are ignored.

Once the handshake has been validated and is considered successful, the responder sends a HTTP response with status 200. After that response, all further data sent or received on the new HTTP/2 stream is considered to be of the supplied :protocol value.

4. Using Tunnels Established via the Extended CONNECT Handshake

DATA frames are used as usual on the stream established by the CONNECT handshake to transmit data.

If the application negotiated the "bytestream" protocol, then individual DATA frames represent segments of an in-order byte stream and are delivered to the application as a stream of bytes. Implementations can deliver data to the application as soon as it becomes available, since there are no message boundaries to preserve.

The same considerations around intermediaries as defined in Section 7 of [RFC6455] apply to the extended CONNECT method. A client that connects via HTTP/2 to an HTTP proxy SHOULD use a traditional CONNECT request to tunnel through that proxy to the destination server.

Streams created via the extended CONNECT method participate in flow control, stream prioritization, and other HTTP/2 features in the same manner as request and response streams defined in [RFC7540]. Stream closure continues to be interpreted as defined in Section 5 of [RFC8441].

Note that the frame type restrictions defined in Section 8.3 of [RFC7540] remain in effect: only DATA, RST_STREAM, WINDOW_UPDATE, and PRIORITY frames are allowed on the connected streams and any other frame types MUST be treated as a stream error (Section 5.4.2 of [RFC7540]) if received.

4.1. Example

An example of negotiating a "bytestream" stream on an HTTP/2 connection follows. This example is intended to closely follow the example in Section 5.1 of [RFC8441] to help illustrate the minor differences defined in this document.

```
[[ From Client ]]
```

```
[[ From Server ]]
```

```
SETTINGS
```

```
SETTINGS_ENABLE_CONNECT[..] = 1
```

```
SETTINGS_ENABLE_BIDIRECTIONAL[..] = 1
```

```
SETTINGS
```

```
SETTINGS_ENABLE_CONNECT[..] = 1
```

```
SETTINGS_ENABLE_BIDIRECTIONAL[..] = 1
```

```
HEADERS + END_HEADERS
```

```
:method = CONNECT
```

```
:protocol = bytestream
```

```
:scheme = https
```

```
:path = /
```

```
:authority = server.example.com
```

```
HEADERS + END_HEADERS
```

```
:status = 200
```

```
DATA
```

```
Bytestream Data
```

```
DATA + END_STREAM
```

```
Bytestream Data
```

```
DATA + END_STREAM
```

```
Bytestream Data
```

5. IANA Considerations

This specification registers an entry in the "HTTP Upgrade Tokens" registry that was established by [RFC7230].

A new token, "bytestream", for arbitrary bytestream data.

- o Value: bytestream
- o Description: Arbitrary bidirectional bytestream data
- o Expected Version Tokens:
- o References: [[RFC Editor: Please fill in this value with the RFC number for this document.]]

6. Security Considerations

The tunnels established by the CONNECT handshake are expected to be protected with a TLS connection. They inherit the security properties of this cryptographic context.

The security considerations of [RFC8441] Section 8 and [RFC7540] Section 10, and Section 10.5.2 especially, apply to this use of the CONNECT method.

7. Acknowledgments

Thanks to Anthony Chivetta, Joshua Otto, and Valentin Pistol for their contributions in the design and implementation of this work.

8. 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>>.
- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <<https://www.rfc-editor.org/info/rfc6455>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <<https://www.rfc-editor.org/info/rfc7540>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8441] McManus, P., "Bootstrapping WebSockets with HTTP/2", RFC 8441, DOI 10.17487/RFC8441, September 2018, <<https://www.rfc-editor.org/info/rfc8441>>.

Authors' Addresses

Eric Kinnear
Apple Inc.
One Apple Park Way
Cupertino, California 95014
United States of America

Email: ekinnear@apple.com

Tommy Pauly
Apple Inc.
One Apple Park Way
Cupertino, California 95014
United States of America

Email: tpauly@apple.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 7, 2020

T. Pauly
E. Kinnear
Apple Inc.
D. Schinazi
Google LLC
November 04, 2019

An Unreliable Datagram Extension to QUIC
draft-pauly-quic-datagram-05

Abstract

This document defines an extension to the QUIC transport protocol to add support for sending and receiving unreliable datagrams over a QUIC connection.

Discussion of this work is encouraged to happen on the QUIC IETF mailing list quic@ietf.org [1] or on the GitHub repository which contains the draft: <https://github.com/tfpaully/draft-pauly-quic-datagram> [2].

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 May 7, 2020.

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. Specification of Requirements	3
2. Motivation	3
3. Transport Parameter	4
4. Datagram Frame Type	5
5. Behavior and Usage	5
5.1. Acknowledgement Handling	6
5.2. Flow Control	6
5.3. Congestion Control	6
6. Security Considerations	7
7. IANA Considerations	7
8. Acknowledgments	7
9. References	8
9.1. Normative References	8
9.2. Informative References	8
9.3. URIs	8
Authors' Addresses	9

1. Introduction

The QUIC Transport Protocol [I-D.ietf-quic-transport] provides a secure, multiplexed connection for transmitting reliable streams of application data. Reliability within QUIC is performed on a per-stream basis, so some frame types are not eligible for retransmission.

Some applications, particularly those that need to transmit real-time data, prefer to transmit data unreliably. These applications can build directly upon UDP [RFC0768] as a transport, and can add security with DTLS [RFC6347]. Extending QUIC to support transmitting unreliable application data would provide another option for secure datagrams, with the added benefit of sharing a cryptographic and authentication context used for reliable streams.

This document defines four new DATAGRAM QUIC frame types, which carry application data without requiring retransmissions.

Discussion of this work is encouraged to happen on the QUIC IETF mailing list quic@ietf.org [3] or on the GitHub repository which

contains the draft: <https://github.com/tfpauly/draft-pauly-quic-datagram> [4].

1.1. Specification of Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Motivation

Transmitting unreliable data over QUIC provides benefits over existing solutions:

- o Applications that open both a reliable TLS stream and an unreliable DTLS flow to the same peer can benefit by sharing a single handshake and authentication context between a reliable QUIC stream and flow of unreliable QUIC datagrams. This can reduce the latency required for handshakes.
- o QUIC uses a more nuanced loss recovery mechanism than the DTLS handshake, which has a basic packet loss retransmission timer. This may allow loss recovery to occur more quickly for QUIC data.
- o QUIC datagrams, while unreliable, can support acknowledgements, allowing applications to be aware of whether a datagram was successfully received.
- o QUIC datagrams are subject to QUIC congestion control, allowing applications to avoid implementing their own.

These reductions in connection latency, and application insight into the delivery of datagrams, can be useful for optimizing audio/video streaming applications, gaming applications, and other real-time network applications.

Unreliable QUIC datagrams can also be used to implement an IP packet tunnel over QUIC, such as for a Virtual Private Network (VPN). Internet-layer tunneling protocols generally require a reliable and authenticated handshake, followed by unreliable secure transmission of IP packets. This can, for example, require a TLS connection for the control data, and DTLS for tunneling IP packets. A single QUIC connection could support both parts with the use of unreliable datagrams.

3. Transport Parameter

Support for receiving the DATAGRAM frame types is advertised by means of a QUIC Transport Parameter (name=max_datagram_frame_size, value=0x0020). The max_datagram_frame_size transport parameter is an integer value (represented as a variable-length integer) that represents the maximum size of a DATAGRAM frame (including the frame type, length, and payload) the endpoint is willing to receive, in bytes. An endpoint that includes this parameter supports the DATAGRAM frame types and is willing to receive such frames on this connection. Endpoints MUST NOT send DATAGRAM frames until they have sent and received the max_datagram_frame_size transport parameter. Endpoints MUST NOT send DATAGRAM frames of size strictly larger than the value of max_datagram_frame_size the endpoint has received from its peer. An endpoint that receives a DATAGRAM frame when it has not sent the max_datagram_frame_size transport parameter MUST terminate the connection with error `PROTOCOL_VIOLATION`. An endpoint that receives a DATAGRAM frame that is strictly larger than the value it sent in its max_datagram_frame_size transport parameter MUST terminate the connection with error `PROTOCOL_VIOLATION`. Endpoints that wish to use DATAGRAM frames need to ensure they send a max_datagram_frame_size value sufficient to allow their peer to use them. It is RECOMMENDED to send the value 65536 in the max_datagram_frame_size transport parameter as that indicates to the peer that this endpoint will accept any DATAGRAM frame that fits inside a QUIC packet.

When clients use 0-RTT, they MAY store the value of the server's max_datagram_frame_size transport parameter. Doing so allows the client to send DATAGRAM frames in 0-RTT packets. When servers decide to accept 0-RTT data, they MUST send a max_datagram_frame_size transport parameter greater or equal to the value they sent to the client in the connection where they sent them the `NewSessionTicket` message. If a client stores the value of the max_datagram_frame_size transport parameter with their 0-RTT state, they MUST validate that the new value of the max_datagram_frame_size transport parameter sent by the server in the handshake is greater or equal to the stored value; if not, the client MUST terminate the connection with error `PROTOCOL_VIOLATION`.

Application protocols that use datagrams MUST define how they react to the max_datagram_frame_size transport parameter being missing. If datagram support is integral to the application, the application protocol can fail the handshake if the max_datagram_frame_size transport parameter is not present.

4. Datagram Frame Type

DATAGRAM frames are used to transmit application data in an unreliable manner. The DATAGRAM frame type takes the form 0b0011000X (or the values 0x30 and 0x31). The least significant bit of the DATAGRAM frame type is the LEN bit (0x01). It indicates that there is a Length field present. If this bit is set to 0, the Length field is absent and the Datagram Data field extends to the end of the packet. If this bit is set to 1, the Length field is present.

The DATAGRAM frame is structured as follows:

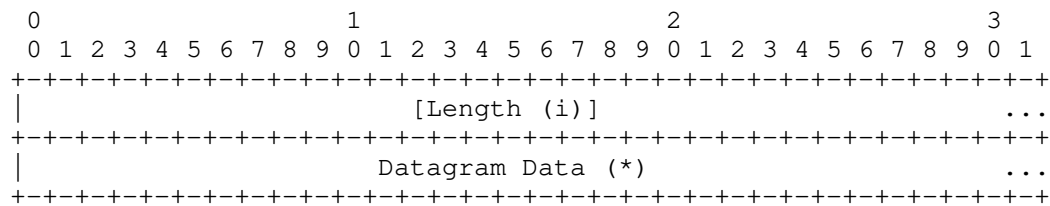


Figure 1: DATAGRAM Frame Format

DATAGRAM frames contain the following fields:

Length: A variable-length integer specifying the length of the datagram in bytes. This field is present only when the LEN bit is set. If the LEN bit is not set, the datagram data extends to the end of the QUIC packet. Note that empty (i.e., zero-length) datagrams are allowed.

Datagram Data: The bytes of the datagram to be delivered.

5. Behavior and Usage

When an application sends an unreliable datagram over a QUIC connection, QUIC will generate a new DATAGRAM frame and send it in the first available packet. This frame SHOULD be sent as soon as possible, and MAY be coalesced with other frames.

When a QUIC endpoint receives a valid DATAGRAM frame, it SHOULD deliver the data to the application immediately, as long as it is able to process the frame and can store the contents in memory.

DATAGRAM frames MUST be protected with either 0-RTT or 1-RTT keys.

Application protocols using datagrams are responsible for defining the semantics of the Datagram Data field, and how it is parsed. If the application protocol supports the coexistence of multiple

entities using datagrams inside a single QUIC connection, it may need a mechanism to allow demultiplexing between them. For example, using datagrams with HTTP/3 involves prepending a flow identifier to all datagrams, see [I-D.schinazi-quic-h3-datagram].

Note that while the `max_datagram_frame_size` transport parameter places a limit on the maximum size of DATAGRAM frames, that limit can be further reduced by the `max_packet_size` transport parameter, and by the Maximum Transmission Unit (MTU) of the path between endpoints. DATAGRAM frames cannot be fragmented, therefore application protocols need to handle cases where the maximum datagram size is limited by other factors.

5.1. Acknowledgement Handling

Although DATAGRAM frames are not retransmitted upon loss detection, they are ack-eliciting ([I-D.ietf-quic-recovery]). Receivers SHOULD support delaying ACK frames (within the limits specified by `max_ack_delay`) in response to receiving packets that only contain DATAGRAM frames, since the timing of these acknowledgements is not used for loss recovery.

If a sender detects that a packet containing a specific DATAGRAM frame might have been lost, the implementation MAY notify the application that it believes the datagram was lost. Similarly, if a packet containing a DATAGRAM frame is acknowledged, the implementation MAY notify the application that the datagram was successfully transmitted and received. Note that, due to reordering, a DATAGRAM frame that was thought to be lost could at a later point be received and acknowledged.

5.2. Flow Control

DATAGRAM frames do not provide any explicit flow control signaling, and do not contribute to any per-flow or connection-wide data limit.

The risk associated with not providing flow control for DATAGRAM frames is that a receiver may not be able to commit the necessary resources to process the frames. For example, it may not be able to store the frame contents in memory. However, since DATAGRAM frames are inherently unreliable, they MAY be dropped by the receiver if the receiver cannot process them.

5.3. Congestion Control

DATAGRAM frames employ the QUIC connection's congestion controller. As a result, a connection may be unable to send a DATAGRAM frame generated by the application until the congestion controller allows

it [I-D.ietf-quic-recovery]. The sender implementation **MUST** either delay sending the frame until the controller allows it or drop the frame without sending it (at which point it **MAY** notify the application).

Implementations can optionally support allowing the application to specify a sending expiration time, beyond which a congestion-controlled DATAGRAM frame ought to be dropped without transmission.

6. Security Considerations

The DATAGRAM frame shares the same security properties as the rest of the data transmitted within a QUIC connection. All application data transmitted with the DATAGRAM frame, like the STREAM frame, **MUST** be protected either by 0-RTT or 1-RTT keys.

7. IANA Considerations

This document registers a new value in the QUIC Transport Parameter Registry:

Value: 0x0020 (if this document is approved)

Parameter Name: max_datagram_frame_size

Specification: Indicates that the connection should enable support for unreliable DATAGRAM frames. An endpoint that advertises this transport parameter can receive datagrams frames from the other endpoint, up to and including the length in bytes provided in the transport parameter.

This document also registers a new value in the QUIC Frame Type registry:

Value: 0x30 and 0x31 (if this document is approved)

Frame Name: DATAGRAM

Specification: Unreliable application data

8. Acknowledgments

Thanks to Ian Swett, who inspired this proposal.

9. References

9.1. Normative References

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

9.2. Informative References

- [I-D.schinazi-quic-h3-datagram]
Schinazi, D., "Using QUIC Datagrams with HTTP/3", draft-schinazi-quic-h3-datagram-01 (work in progress), October 2019.
- [RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <<https://www.rfc-editor.org/info/rfc768>>.
- [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>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [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>>.

9.3. URIs

- [1] <mailto:quic@ietf.org>
- [2] <https://github.com/tfpaully/draft-paully-quic-datagram>
- [3] <mailto:quic@ietf.org>
- [4] <https://github.com/tfpaully/draft-paully-quic-datagram>

Authors' Addresses

Tommy Pauly
Apple Inc.
One Apple Park Way
Cupertino, California 95014
United States of America

Email: tpauly@apple.com

Eric Kinnear
Apple Inc.
One Apple Park Way
Cupertino, California 95014
United States of America

Email: ekinnear@apple.com

David Schinazi
Google LLC
1600 Amphitheatre Parkway
Mountain View, California 94043
United States of America

Email: dschinazi.ietf@gmail.com

Network Working Group
Internet-Draft
Intended status: Experimental
Expires: 15 April 2021

D. Schinazi
Google LLC
12 October 2020

Using QUIC Datagrams with HTTP/3
draft-schinazi-quic-h3-datagram-05

Abstract

The QUIC DATAGRAM extension provides application protocols running over QUIC with a mechanism to send unreliable data while leveraging the security and congestion-control properties of QUIC. However, QUIC DATAGRAM frames do not provide a means to demultiplex application contexts. This document defines how to use QUIC DATAGRAM frames when the application protocol running over QUIC is HTTP/3 by adding an identifier at the start of the frame payload.

Discussion of this work is encouraged to happen on the QUIC IETF mailing list (quic@ietf.org (<mailto:quic@ietf.org>)) or on the GitHub repository which contains the draft:
<https://github.com/DavidSchinazi/draft-h3-datagram>.

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 15 April 2021.

Copyright Notice

Copyright (c) 2020 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. Conventions and Definitions	3
2. HTTP/3 DATAGRAM Frame Format	3
2.1. Flow Identifiers	3
3. Flow Identifier Allocation	4
4. The H3_DATAGRAM HTTP/3 SETTINGS Parameter	4
5. Security Considerations	5
6. IANA Considerations	5
7. Normative References	5
Acknowledgments	6
Author's Address	6

1. Introduction

The QUIC DATAGRAM extension [DGRAM] provides application protocols running over QUIC [QUIC] with a mechanism to send unreliable data while leveraging the security and congestion-control properties of QUIC. However, QUIC DATAGRAM frames do not provide a means to demultiplex application contexts. This document defines how to use QUIC DATAGRAM frames when the application protocol running over QUIC is HTTP/3 [H3] by adding an identifier at the start of the frame payload.

This design mimics the use of Stream Types in HTTP/3, which provide a demultiplexing identifier at the start of each unidirectional stream.

Discussion of this work is encouraged to happen on the QUIC IETF mailing list (quic@ietf.org (<mailto:quic@ietf.org>)) or on the GitHub repository which contains the draft:
<https://github.com/DavidSchinazi/draft-h3-datagram>.

1.1. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. HTTP/3 DATAGRAM Frame Format

When used with HTTP/3, the Datagram Data field of QUIC DATAGRAM frames uses the following format:

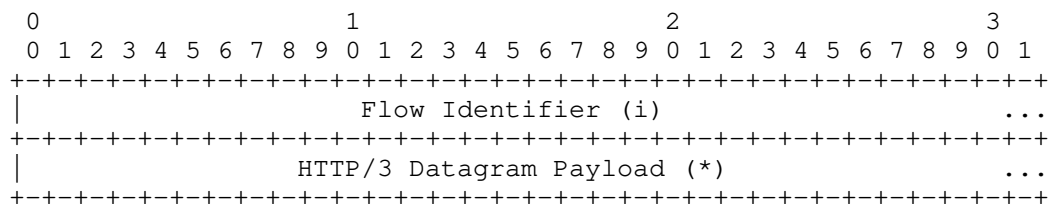


Figure 1: HTTP/3 DATAGRAM Frame Format

Flow Identifier: A variable-length integer indicating the Flow Identifier of the datagram (see Section 2.1).

HTTP/3 Datagram Payload: The payload of the datagram, whose semantics are defined by individual applications.

2.1. Flow Identifiers

Flow identifiers represent bidirectional flows of datagrams within a single QUIC connection. These are conceptually similar to streams in the sense that they allow multiplexing of application data. Of course flows lack any of the ordering or reliability guarantees of streams.

Beyond this, a sender SHOULD ensure that DATAGRAM frames within a single flow are transmitted in order relative to one another. If multiple DATAGRAM frames can be packed into a single QUIC packet, the sender SHOULD group them by flow identifier to promote fate-sharing within a specific flow and improve the ability to process batches of datagram messages efficiently on the receiver.

3. Flow Identifier Allocation

Implementations of HTTP/3 that support the DATAGRAM extension MUST provide a flow identifier allocation service. That service will allow applications co-located with HTTP/3 to request a unique flow identifier that they can subsequently use for their own purposes. The HTTP/3 implementation will then parse the flow identifier of incoming DATAGRAM frames and use it to deliver the frame to the appropriate application.

Even flow identifiers are client-initiated, while odd flow identifiers are server-initiated. This means that an HTTP/3 client implementation of the flow identifier allocation service MUST only provide even identifiers, while a server implementation MUST only provide odd identifiers. Note that, once allocated, any flow identifier can be used by both client and server - only allocation carries separate namespaces to avoid requiring synchronization.

4. The H3_DATAGRAM HTTP/3 SETTINGS Parameter

Implementations of HTTP/3 that support this mechanism can indicate that to their peer by sending the H3_DATAGRAM SETTINGS parameter with a value of 1. The value of the H3_DATAGRAM SETTINGS parameter MUST be either 0 or 1. A value of 0 indicates that this mechanism is not supported. An endpoint that receives the H3_DATAGRAM SETTINGS parameter with a value that is neither 0 or 1 MUST terminate the connection with error H3_SETTINGS_ERROR.

And endpoint that sends the H3_DATAGRAM SETTINGS parameter with a value of 1 MUST send the max_datagram_frame_size QUIC Transport Parameter [DGRAM]. An endpoint that receives the H3_DATAGRAM SETTINGS parameter with a value of 1 on a QUIC connection that did not also receive the max_datagram_frame_size QUIC Transport Parameter MUST terminate the connection with error H3_SETTINGS_ERROR.

When clients use 0-RTT, they MAY store the value of the server's H3_DATAGRAM SETTINGS parameter. Doing so allows the client to use HTTP/3 datagrams in 0-RTT packets. When servers decide to accept 0-RTT data, they MUST send a H3_DATAGRAM SETTINGS parameter greater or equal to the value they sent to the client in the connection where they sent them the NewSessionTicket message. If a client stores the value of the H3_DATAGRAM SETTINGS parameter with their 0-RTT state, they MUST validate that the new value of the H3_DATAGRAM SETTINGS parameter sent by the server in the handshake is greater or equal to the stored value; if not, the client MUST terminate the connection with error H3_SETTINGS_ERROR.

5. Security Considerations

This document currently does not have additional security considerations beyond those defined in [QUIC] and [DGRAM].

6. IANA Considerations

This document will request IANA to register the following entry in the "HTTP/3 Settings" registry:

Setting Name	Value	Specification	Default
H3_DATAGRAM	0x276	This Document	0

7. Normative References

- [DGRAM] Pauly, T., Kinnear, E., and D. Schinazi, "An Unreliable Datagram Extension to QUIC", Work in Progress, Internet-Draft, draft-ietf-quic-datagram-01, 24 August 2020, <<http://www.ietf.org/internet-drafts/draft-ietf-quic-datagram-01.txt>>.
- [H3] Bishop, M., "Hypertext Transfer Protocol Version 3 (HTTP/3)", Work in Progress, Internet-Draft, draft-ietf-quic-http-31, 24 September 2020, <<http://www.ietf.org/internet-drafts/draft-ietf-quic-http-31.txt>>.
- [QUIC] Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", Work in Progress, Internet-Draft, draft-ietf-quic-transport-31, 24 September 2020, <<http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-31.txt>>.
- [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>>.
- [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>>.

Acknowledgments

The DATAGRAM frame identifier was previously part of the DATAGRAM frame definition itself, the author would like to acknowledge the authors of that document and the members of the IETF QUIC working group for their suggestions. Additionally, the author would like to thank Martin Thomson for suggesting the use of an HTTP/3 SETTINGS parameter.

Author's Address

David Schinazi
Google LLC
1600 Amphitheatre Parkway
Mountain View, California 94043,
United States of America

Email: dschinazi.ietf@gmail.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 31 July 2021

V. Vasiliev
Google
27 January 2021

WebTransport over HTTP/3
draft-vvv-webtransport-http3-03

Abstract

WebTransport [OVERVIEW] is a protocol framework that enables clients constrained by the Web security model to communicate with a remote server using a secure multiplexed transport. This document describes a WebTransport protocol that is based on HTTP/3 [HTTP3] and provides support for unidirectional streams, bidirectional streams and datagrams, all multiplexed within the same HTTP/3 connection.

Note to Readers

Discussion of this draft takes place on the WebTransport mailing list (webtransport@ietf.org), which is archived at https://mailarchive.ietf.org/arch/search/?email_list=webtransport.

The repository tracking the issues for this draft can be found at <https://github.com/vasilvv/webtransport/issues>. The web API draft corresponding to this document can be found at <https://w3c.github.io/webtransport/>.

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 31 July 2021.

Copyright Notice

Copyright (c) 2021 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	3
1.1. Terminology	3
2. Protocol Overview	3
3. Session Establishment	4
3.1. Establishing a Transport-Capable HTTP/3 Connection . . .	4
3.2. Extended CONNECT in HTTP/3	4
3.3. Creating a New Session	5
3.4. Limiting the Number of Simultaneous Sessions	5
4. WebTransport Features	6
4.1. Unidirectional streams	6
4.2. Client-Initiated Bidirectional Streams	6
4.3. Server-Initiated Bidirectional Streams	7
4.4. Datagrams	7
5. Session Termination	8
6. Security Considerations	8
7. IANA Considerations	9
7.1. Upgrade Token Registration	9
7.2. HTTP/3 SETTINGS Parameter Registration	9
7.3. Frame Type Registration	9
7.4. Stream Type Registration	10
8. References	10
8.1. Normative References	10
8.2. Informative References	12
Author's Address	12

1. Introduction

HTTP/3 [HTTP3] is a protocol defined on top of QUIC [QUIC-TRANSPORT] that can multiplex HTTP requests over a QUIC connection. This document defines a mechanism for multiplexing non-HTTP data with HTTP/3 in a manner that conforms with the WebTransport protocol requirements and semantics [OVERVIEW]. Using the mechanism described here, multiple WebTransport instances can be multiplexed simultaneously with regular HTTP traffic on the same HTTP/3 connection.

1.1. Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document follows terminology defined in Section 1.2 of [OVERVIEW]. Note that this document distinguishes between a WebTransport server and an HTTP/3 server. An HTTP/3 server is the server that terminates HTTP/3 connections; a WebTransport server is an application that accepts WebTransport sessions, which can be accessed via an HTTP/3 server.

2. Protocol Overview

WebTransport servers in general 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, both the client and server have to send a `SETTINGS_ENABLE_WEBTRANSPORT` setting in order to indicate that they both support WebTransport over HTTP/3.

WebTransport sessions are initiated inside a given HTTP/3 connection by the client, who sends an extended `CONNECT` request [RFC8441]. If the server accepts the request, a WebTransport session is established. The resulting stream will be further referred to as a `_CONNECT stream_`, and its stream ID is used to uniquely identify a given WebTransport session within the connection. The ID of the `CONNECT` stream that established a given WebTransport session will be further referred to as a `_Session ID_`.

After the session is established, the peers can exchange data using the following mechanisms:

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

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

3. Session Establishment

3.1. Establishing a Transport-Capable HTTP/3 Connection

In order to indicate support for WebTransport, both the client and the server MUST send a SETTINGS_ENABLE_WEBTRANSPORT value set to "1" in their SETTINGS frame. Endpoints MUST NOT use any WebTransport-related functionality unless the parameter has been negotiated.

If SETTINGS_ENABLE_WEBTRANSPORT is negotiated, support for the QUIC DATAGRAMs within HTTP/3 MUST be negotiated as described in [HTTP3-DATAGRAM]; negotiating WebTransport support without negotiating QUIC DATAGRAM extension SHALL result in a H3_SETTINGS_ERROR error.

[HTTP3] requires client's "initial_max_bidi_streams" transport parameter to be set to zero. Existing implementation might enforce this requirement before negotiating settings; thus, the client MUST send a non-zero MAX_STREAMS for client-initiated bidirectional streams after receiving an appropriate SETTINGS frame from the server.

3.2. Extended CONNECT in HTTP/3

[RFC8441] defines an extended CONNECT method in Section 4, enabled by the SETTINGS_ENABLE_CONNECT_PROTOCOL parameter. That parameter is only defined for HTTP/2. This document does not create a new multi-purpose parameter to indicate support for extended CONNECT in HTTP/3; instead, the SETTINGS_ENABLE_WEBTRANSPORT setting implies that an endpoint supports extended CONNECT.

3.3. Creating a New Session

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

In order to create a new WebTransport session, a client can send an HTTP CONNECT request. The ":protocol" pseudo-header field ([RFC8441]) MUST be set to "webtransport". The ":scheme" field MUST be "https". Both the ":authority" and the ":path" value MUST be set; those fields indicate the desired WebTransport server. An "Origin" header [RFC6454] MUST be provided within the request.

Upon receiving an extended CONNECT request with a ":protocol" field set to "webtransport", the HTTP/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. The WebTransport server MUST verify the "Origin" header to ensure that the specified origin is allowed to access the server in question.

From the client's perspective, a WebTransport session is established when the client receives a 200 response. From the server's perspective, a session is established once it sends a 200 response. Both endpoints MUST NOT open any streams or send any datagrams on a given session before that session is established. WebTransport over HTTP/3 does not support 0-RTT.

3.4. Limiting the Number of Simultaneous Sessions

From the flow control perspective, WebTransport sessions count against the stream flow control just like regular HTTP requests, since they are established via an HTTP CONNECT request. This document does not make any effort to introduce a separate flow control mechanism for sessions, nor to separate HTTP requests from WebTransport data streams. If the server needs to limit the rate of incoming requests, it has alternative mechanisms at its disposal:

- * "HTTP_REQUEST_REJECTED" error code defined in [HTTP3] indicates to the receiving HTTP/3 stack that the request was not processed in any way.
- * HTTP status code 429 indicates that the request was rejected due to rate limiting [RFC6585]. Unlike the previous method, this signal is directly propagated to the application.

4. WebTransport Features

WebTransport over HTTP/3 provides the following features described in [OVERVIEW]: unidirectional streams, bidirectional streams and datagrams, initiated by either endpoint.

Session IDs are used to demultiplex streams and datagrams belonging to different WebTransport sessions. On the wire, session IDs are encoded using the QUIC variable length integer scheme described in [QUIC-TRANSPORT].

4.1. Unidirectional streams

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

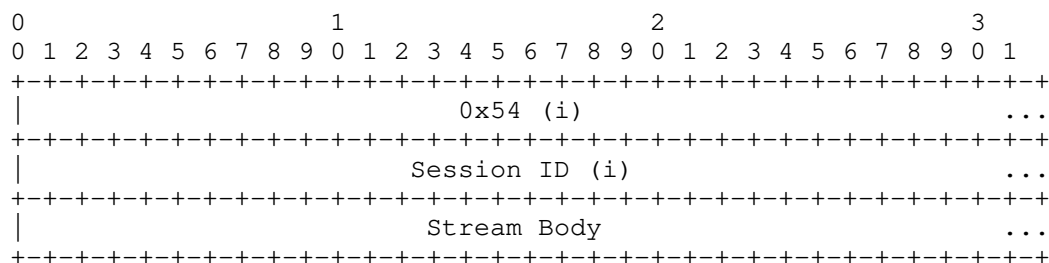


Figure 1: Unidirectional WebTransport stream format

4.2. Client-Initiated Bidirectional Streams

WebTransport 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.

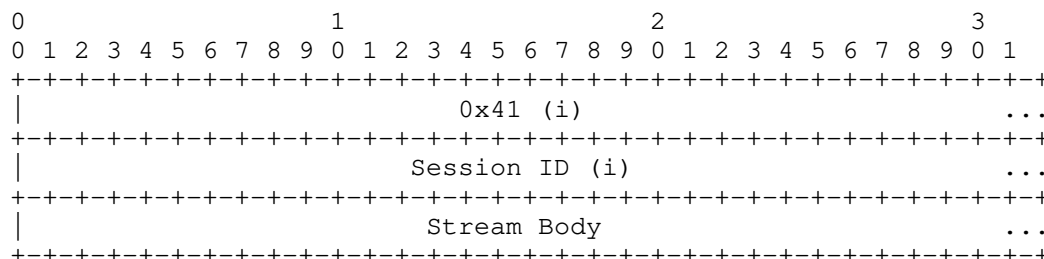


Figure 2: WEBTRANSPORT_STREAM frame format

4.3. Server-Initiated Bidirectional Streams

WebTransport 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 `SETTINGS_ENABLE_WEBTRANSPORT` 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).

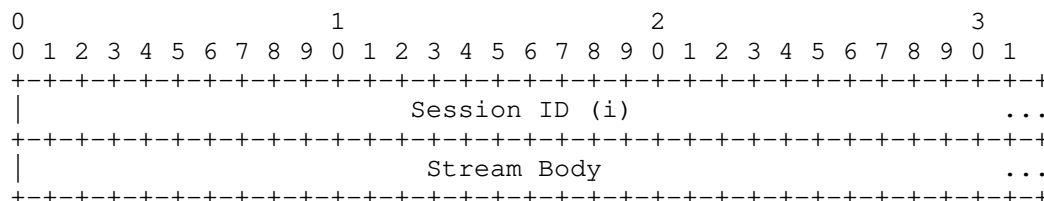


Figure 3: Server-initiated bidirectional stream format

4.4. Datagrams

Datagrams can be sent using the DATAGRAM frame as defined in [QUIC-DATAGRAM] and [HTTP3-DATAGRAM]. For all HTTP/3 connections in which the `SETTINGS_ENABLE_WEBTRANSPORT` option is negotiated, the Flow Identifier is set to the session ID. In other words, the format of datagrams SHALL be the session ID, followed by the user-specified payload (Figure 4).

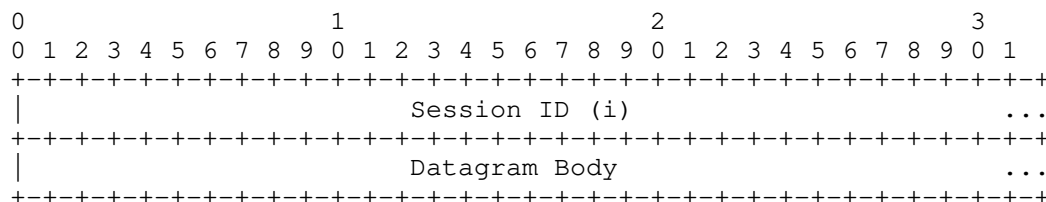


Figure 4: Datagram format

In QUIC, a datagram frame can span at most one packet. Because of that, the applications have to know the maximum size of the datagram they can send. However, when proxying the datagrams, the hop-by-hop MTUs can vary. TODO: Describe how the path MTU can be computed, specifically propagation across HTTP proxies.

5. Session Termination

An WebTransport session over HTTP/3 is terminated when either endpoint closes the stream associated with the CONNECT request that initiated the session. Upon learning about the session being terminated, the endpoint MUST stop sending new datagrams and reset all of the streams associated with the session.

6. Security Considerations

WebTransport over HTTP/3 satisfies all of the security requirements imposed by [OVERVIEW] on WebTransport protocols, thus providing a secure framework for client-server communication in cases when the client is potentially untrusted.

WebTransport over HTTP/3 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 traffic going over HTTP/3, WebTransport pools traffic to different origins within a single connection. Different origins imply different trust domains, meaning that the implementations have to treat each transport as potentially hostile towards others on the same connection. One potential attack is a resource exhaustion attack: since all of the transports share both congestion control and flow control context, a single client aggressively using up those resources can cause other transports to stall. The user agent thus SHOULD implement a fairness scheme that ensures that each transport within connection gets a reasonable share of controlled resources; this applies both to sending data and to opening new streams.

7. IANA Considerations

7.1. 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 and [I-D.kinnear-webtransport-http2]

7.2. HTTP/3 SETTINGS Parameter Registration

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

The "SETTINGS_ENABLE_WEBTRANSPORT" parameter indicates that the specified HTTP/3 connection is WebTransport-capable.

Setting Name: ENABLE_WEBTRANSPORT

Value: 0x2b603742

Default: 0

Specification: This document

7.3. 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

7.4. 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

8. References

8.1. Normative References

- [HTTP3] Bishop, M., Ed., "Hypertext Transfer Protocol Version 3 (HTTP/3)", Work in Progress, Internet-Draft, draft-ietf-quic-http, <<https://tools.ietf.org/html/draft-ietf-quic-http>>.
- [HTTP3-DATAGRAM] Schinazi, D. and L. Pardue, "Using QUIC Datagrams with HTTP/3", Work in Progress, Internet-Draft, draft-schinazi-masque-h3-datagram-04, 5 January 2021, <<http://www.ietf.org/internet-drafts/draft-schinazi-masque-h3-datagram-04.txt>>.
- [OVERVIEW] Vasiliev, V., "The WebTransport Protocol Framework", Work in Progress, Internet-Draft, draft-ietf-webtrans-overview-latest, <<https://tools.ietf.org/html/draft-ietf-webtrans-overview-latest>>.

[QUIC-DATAGRAM]

Pauly, T., Kinnear, E., and D. Schinazi, "An Unreliable Datagram Extension to QUIC", Work in Progress, Internet-Draft, draft-pauly-quic-datagram, <<https://tools.ietf.org/html/draft-pauly-quic-datagram>>.

[QUIC-TRANSPORT]

Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", Work in Progress, Internet-Draft, draft-ietf-quic-transport, <<https://tools.ietf.org/html/draft-ietf-quic-transport>>.

[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>>.

8.2. Informative References

[I-D.kinnear-webtransport-http2]

Frindell, A., Kinnear, E., Pauly, T., Vasiliev, V., and G. Xie, "WebTransport using HTTP/2", Work in Progress, Internet-Draft, draft-kinnear-webtransport-http2-01, 13 July 2020, <<http://www.ietf.org/internet-drafts/draft-kinnear-webtransport-http2-01.txt>>.

Author's Address

Victor Vasiliev
Google

Email: vasilvv@google.com

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

V. Vasiliev
Google
November 2, 2019

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

Abstract

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

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

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 May 5, 2020.

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

1. Introduction

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

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

1.1. Background

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

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

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

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

- o 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.
- o Only clients can initiate streams. Server-initiated streams and other alternative modes of communication (such as the QUIC DATAGRAM frame [I-D.pauly-quic-datagram]) are not available.
- 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 a server is opaque to the client. This introduces unpredictable performance properties into the system, and prevents optimizations which rely on the streams being on the same connection (for instance, it might be possible for the client to request different retransmission priorities for different streams, but that would be much more complex unless they are all on the same connection).

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

1.2. Conventions and Definitions

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

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

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

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

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

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

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

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

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

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

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

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

2. Common Transport Requirements

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

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

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

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

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

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

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

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

3. Session Establishment

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

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

4. Transport Features

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

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

4.1. Datagrams

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

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

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

The application **MUST** be provided with the 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.

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

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

4.3. Protocol-Specific Features

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

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

4.4. Bandwidth Prediction

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

5. Buffering and Prioritization

TODO: expand this outline into a full summary.

- 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 apply aggregation algorithms (e.g., Nagle's algorithm [RFC0896]) to datagrams.

6. Transport Properties

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

The following properties are defined in this specification:

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

- o Partial reliability. This indicates that if a stream is reset, none of the data sent on it will be retransmitted. This also indicates that datagrams will not be retransmitted.
- o Pooling support. Indicates that multiple transports using this transport protocol may end up sharing the same transport layer connection, and thus share a congestion controller and other contexts.
- 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 risks. This document mitigates those risks by imposing a set of common requirements described in Section 2.

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

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

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

8. IANA Considerations

There are no requests to IANA in this document.

9. References

9.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <<https://www.rfc-editor.org/info/rfc6454>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

9.2. Informative References

- [CSP] W3C, "Content Security Policy Level 3", November 2019, <<https://www.w3.org/TR/CSP/>>.
- [I-D.ietf-quic-http] Bishop, M., "Hypertext Transfer Protocol Version 3 (HTTP/3)", draft-ietf-quic-http-23 (work in progress), September 2019.
- [I-D.ietf-quic-recovery] Iyengar, J. and I. Swett, "QUIC Loss Detection and Congestion Control", draft-ietf-quic-recovery-23 (work in progress), September 2019.
- [I-D.ietf-quic-transport] Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-23 (work in progress), September 2019.

- [I-D.ietf-rtcweb-data-channel]
Jesup, R., Loreto, S., and M. Tuexen, "WebRTC Data Channels", draft-ietf-rtcweb-data-channel-13 (work in progress), January 2015.
- [I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", draft-ietf-tls-dtls13-33 (work in progress), October 2019.
- [I-D.pauly-quic-datagram]
Pauly, T., Kinnear, E., and D. Schinazi, "An Unreliable Datagram Extension to QUIC", draft-pauly-quic-datagram-04 (work in progress), October 2019.
- [RFC0896] Nagle, J., "Congestion Control in IP/TCP Internetworks", RFC 896, DOI 10.17487/RFC0896, January 1984, <<https://www.rfc-editor.org/info/rfc896>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<https://www.rfc-editor.org/info/rfc5681>>.
- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <<https://www.rfc-editor.org/info/rfc6455>>.
- [RFC8441] McManus, P., "Bootstrapping WebSockets with HTTP/2", RFC 8441, DOI 10.17487/RFC8441, September 2018, <<https://www.rfc-editor.org/info/rfc8441>>.
- [RFC8445] Keranen, A., Holmberg, C., and J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal", RFC 8445, DOI 10.17487/RFC8445, July 2018, <<https://www.rfc-editor.org/info/rfc8445>>.

Author's Address

Victor Vasiliev
Google

Email: vasilvv@google.com

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

V. Vasiliev
Google
June 30, 2020

WebTransport over QUIC
draft-vvv-webtransport-quic-02

Abstract

WebTransport [OVERVIEW] is a protocol framework that enables clients constrained by the Web security model to communicate with a remote server using a secure multiplexed transport. This document describes QuicTransport, a transport protocol that uses a dedicated QUIC [QUIC] connection and provides support for unidirectional streams, bidirectional streams and datagrams.

Note to Readers

Discussion of this draft takes place on the WebTransport mailing list (webtransport@ietf.org), which is archived at https://mailarchive.ietf.org/arch/search/?email_list=webtransport.

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

Status of This Memo

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

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

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 1, 2021.

Copyright Notice

Copyright (c) 2020 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	3
2. Protocol Overview	3
3. Connection Establishment	4
3.1. Identifying as QuicTransport	4
3.2. Client Indication	4
3.2.1. Origin Field	5
3.2.2. Path Field	6
3.3. 0-RTT	6
4. Streams	6
5. Datagrams	7
6. QuicTransport URI Scheme	7
7. Transport Properties	8
8. Security Considerations	8
9. IANA Considerations	9
9.1. ALPN Value Registration	9
9.2. Client Indication Fields Registry	9
9.3. URI Scheme Registration	10
10. References	10
10.1. Normative References	10
10.2. Informative References	12
10.3. URIs	12
Author's Address	12

1. Introduction

WebTransport [OVERVIEW] is a protocol framework that enables clients constrained by the Web security model to communicate with a remote server using a secure multiplexed transport. This document describes QuicTransport, a transport protocol that uses a dedicated QUIC [QUIC]

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

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

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

1.1. Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document follows terminology defined in Section 1.2 of [OVERVIEW]. The diagrams describe encoding following the conventions described in Section 1.3 of [QUIC].

2. Protocol Overview

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

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

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

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

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

3. Connection Establishment

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

3.1. Identifying as QuicTransport

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

3.2. Client Indication

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

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

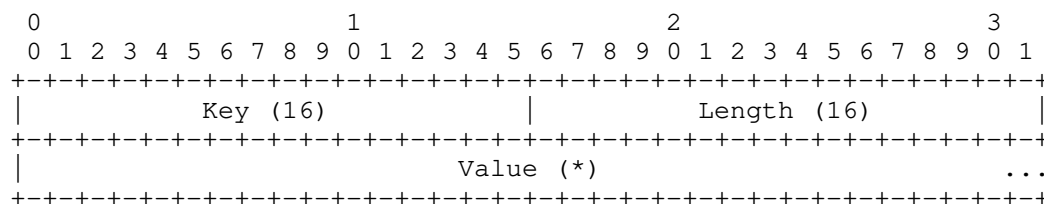


Figure 1: Client indication format

The pair includes the following fields:

Key: Indicates the field that is being expressed.

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

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

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

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

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

3.2.1. Origin Field

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

Name: Origin

Key: 0x0000

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

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

3.2.2. Path Field

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

Name: Path

Key: 0x0001

Description: The path component of the QuicTransport URI.

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

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

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

3.3. 0-RTT

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

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

4. Streams

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

5. Datagrams

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

6. QuicTransport URI Scheme

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

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

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

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

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

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

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

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

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

7. Transport Properties

QuicTransport supports most WebTransport features as described in Table 1.

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

Table 1: Transport properties of QuicTransport

8. Security Considerations

QuicTransport satisfies all of the security requirements imposed by [OVERVIEW] on WebTransport protocols, thus providing a secure framework for client-server communication in cases when the client is potentially untrusted.

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

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

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

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

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

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

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

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

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

9. IANA Considerations

9.1. ALPN Value Registration

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

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

Protocol: QuicTransport

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

Specification: This document

9.2. Client Indication Fields Registry

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

Name: The name of the field.

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

Description: A brief description of what the parameter does.

Reference: The document that describes the parameter.

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

9.3. URI Scheme Registration

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

Scheme name: quic-transport

Status: Permanent

Applications/protocols that use this scheme name: QuicTransport

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

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

Reference: Section 6 of this document.

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

10. References

10.1. Normative References

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

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

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

[QUIC-DATAGRAM]

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

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/info/rfc5234>>.
- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <<https://www.rfc-editor.org/info/rfc6454>>.
- [RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <<https://www.rfc-editor.org/info/rfc7301>>.
- [RFC7595] Thaler, D., Ed., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", BCP 35, RFC 7595, DOI 10.17487/RFC7595, June 2015, <<https://www.rfc-editor.org/info/rfc7595>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

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

10.2. Informative References

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

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

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

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

10.3. URIs

[1] <mailto:chair@ietf.org>

[2] <mailto:iesg@ietf.org>

Author's Address

Victor Vasiliev
Google

Email: vasilvv@google.com

httpbis Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 9, 2020

G. Xie
A. Frindell
Facebook Inc.
July 08, 2019

An HTTP/2 Extension for Bidirectional Message Communication
draft-xie-bidirectional-messaging-02

Abstract

This draft proposes an HTTP/2 protocol extension that enables bidirectional messaging communication between client and server.

Status of This Memo

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

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

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 9, 2020.

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.

1. Introduction

HTTP/2 [RFC7540] transports HTTP messages via a framing layer that includes many technologies and optimizations designed to make communication more efficient between clients and servers. These include multiplexing of multiple streams on a single underlying transport connection, flow control, stream dependencies and priorities, header compression, and exchange of configuration information between endpoints.

Many of these capabilities are generic and can be useful in applications beyond web browsing, such as Publish/Subscribe protocols or RPC. However, HTTP/2 framing's request/response client to server communication pattern prevents wider use in this type of application. This draft proposes an HTTP/2 protocol extension that enables bidirectional communication between client and server.

Currently, the only mechanism in HTTP/2 for server to client communication is server push. That is, servers can initiate unidirectional push promised streams to clients, but clients cannot respond to them and either accept or discard them silently. Additionally, intermediaries along the path may have different server push policies and may not forward push promised streams to the downstream client. This best effort mechanism is not sufficient to reliably deliver content from servers to clients, limiting additional use-cases, such as sending messages and notifications from servers to clients immediately when they become available.

Several techniques have been developed to workaroud these limitations: long polling [RFC6202], WebSocket [RFC8441], and tunneling using the CONNECT method. All of these approaches layer an application protocol on top of HTTP/2, using HTTP/2 streams as transport connections. This layering defeats the optimizations provided by HTTP/2. For example, multiplexing multiple parallel interactions onto one HTTP/2 stream reintroduces head of line blocking. Also, application metadata is encapsulated into DATA frames, rather than HEADERS frames, making header compression impossible. Further, user data is framed multiple times at different protocol layers, which offsets the wire efficiency of HTTP/2 binary framing. Take WebSocket over HTTP/2 as an example, user data is framed at the application protocol, WebSocket, and HTTP/2 layers. This not only introduces overhead on the wire, but also complicates data processing. Finally, intermediaries have no visibility to user interactions layered on a single HTTP/2 stream, and lose the capability to collect telemetry metrics (e.g., time to the first/last byte of request and response) for services.

These techniques also pose new operational challenges to intermediaries. Because all traffic from a user's session is encapsulated into one HTTP/2 stream, this stream can last a very long time. Intermediaries may take a long time to drain these streams. HTTP/2 GOAWAY only signals the remote endpoint to stop using the connection for new streams; additional work is required to prevent new application messages from being initiated on the long lived stream.

In this draft, a new HTTP/2 frame is introduced which has the routing properties of a PUSH_PROMISE frame and the bi-directionality of a HEADERS frame. The extension provides several benefits:

1. After a HTTP/2 connection is established, a server can initiate streams to the client at any time, and the client can respond to the incoming streams accordingly. That is, the communication over HTTP/2 is bidirectional and symmetric.
2. All of the HTTP/2 technologies and optimizations still apply. Intermediaries also have all the necessary metadata to properly handle the communication between the client and the server.
3. Clients are able to group streams together for routing purposes, such that each individual stream group can be used for a different service, within the same HTTP/2 connection.

2. Conventions and Terminology

The keywords **MUST**, **MUST NOT**, **REQUIRED**, **SHALL**, **SHALL NOT**, **SHOULD**, **SHOULD NOT**, **RECOMMENDED**, **MAY**, and **OPTIONAL**, when they appear in this document, are to be interpreted as described in [RFC2119].

All the terms defined in the Conventions and Terminology section in [RFC7540] apply to this document.

3. Solution Overview

3.1. RStream and XStream

A routing stream (RStream) is a regular HTTP/2 stream. It is opened by a HEADERS frame, and **MAY** be continued by CONTINUATION and DATA frames. RStreams are initiated by clients to servers, and can be independently routed by intermediaries on the network path. The main purpose for an RStream is to facilitate XStreams' intermediary traversal.

A new HTTP/2 stream called eXtended stream (XStream) is introduced for exchanging user data bidirectionally. An XStream is opened by an XHEADERS frame, and **MAY** be continued by CONTINUATION and DATA frames. XStreams can be initiated by either clients or servers. Unlike a regular stream, an XStream **MUST** be associated with an open RStream. In this way, XStreams can be routed according to their RStreams by intermediaries and servers. XStream **MUST NOT** be associated with any other XStream, or any closed RStream. Otherwise, it cannot be routed properly.

3.2. Bidirectional Communication

With RStreams and XStreams, HTTP/2 framing can be used natively for bidirectional communication. As shown in Figure 1 and Figure 2, as long as an RStream is open from client to server, either endpoint can initiate an XStream to its peer.

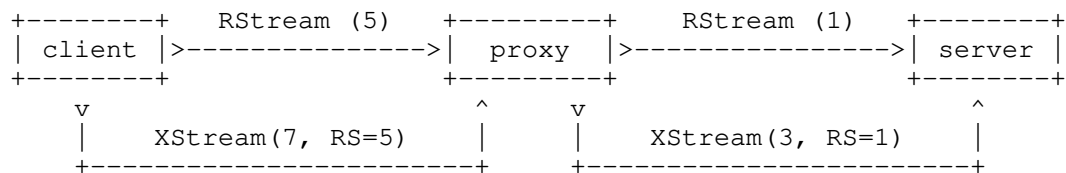


Figure 1: Client initiates an XStream to server.

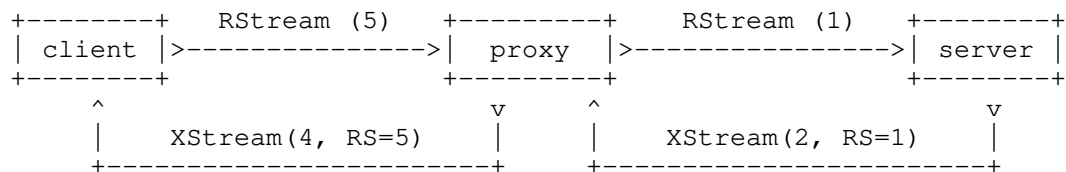


Figure 2: Server initiates an XStream to client.

3.3. XStream Grouping

A client can multiplex RStreams, XStreams and regular HTTP/2 streams into a single HTTP/2 connection. Additionally, all of the XStreams associated with the same RStream form a logical stream group, and are routed to the same endpoint. This enables clients to access different services without initiating new connections, or including routing metadata in every message. As shown in Figure 3, the client can exchange data with three different services (PubSub, RPC, and CDN) using one HTTP/2 connection.

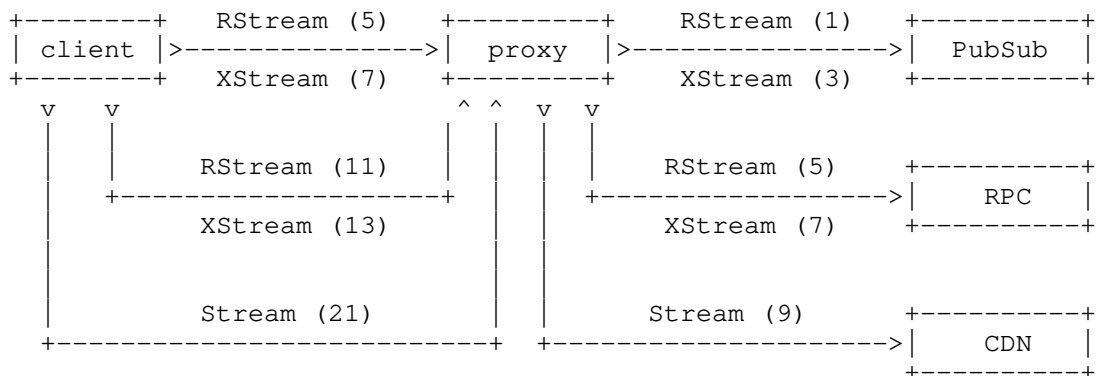


Figure 3: Client opens multiple RStreams, XStreams and an HTTP/2 stream within one HTTP/2 connection.

Reusing one connection for different purposes saves the latency of setting up new connections. This is especially desirable for mobile devices which often have higher latency network connectivity and tighter battery constraints. Multiplexing these services also allows them to share a single transport connection congestion control context. It also opens new optimization opportunities, like prioritizing interactive streams over streams used to fetch static content. It also reduces the number of connections that are adding load to intermediaries and servers in the network.

3.4. Recommended Usage

RStreams and XStreams are designed for different purposes. RStreams are **RECOMMENDED** for exchanging metadata only, and **SHOULD** be long lived, as once an RStream is closed any routing information it carried is lost. Unless a new RStream is re-established promptly, no new XStreams can be initiated. To keep an RStream open, endpoints **SHOULD NOT** send a HEADERS or DATA frame containing the END_STREAM flag. Implementations might require special logic to prevent RStreams from timing out. For example, refresh the timeouts on RStreams if a new XStream is exchanged.

By contrast, XStreams are **RECOMMENDED** for exchanging user data, and **SHOULD** be short lived. In long polling, WebSocket and tunneling solutions, streams have to be kept alive for a long time because servers need those streams for sending data to the client in the future. With this extension, servers are able to initiate new XStreams as long as RStreams are still open and no longer need to keep idle streams around for future use. This allows all parties involved in the connection to keep resource usage to a minimum. Moreover, short lived XStreams make graceful shutdown of a connection

easier for intermediaries and servers. After exchanging GOAWAY frames, short lived XStreams will naturally drain within a short period of time.

3.5. States of RStream and XStream

RStreams are regular HTTP/2 streams that follow the stream lifecycle described in [RFC7540], section 5.1. XStreams use the same lifecycle as regular HTTP/2 streams, but have extra dependency on their RStreams. If an RStream is reset, endpoints *MUST* reset the XStreams associated with that RStream. If the RStream is closed, endpoints *SHOULD* allow the existing XStreams to complete normally. The RStream *SHOULD* remain open while communication is ongoing. Endpoints *SHOULD* refresh any timeout on the RStream while its associated XStreams are open.

A sender *MUST NOT* initiate new XStreams with an RStream that is in the closed or half closed (remote) state.

Endpoints process new XStreams only when the associated RStream is in the open or half closed (local) state. If an endpoint receives an XHEADERS frame specifying an RStream in the closed or half closed (remote) state, it *MUST* respond with a connection error of type `ROUTING_STREAM_ERROR`.

3.6. Negotiating the Extension

The extension *SHOULD* be disabled by default. As noted in [RFC7540], section 5.5, HTTP/2 compliant implementations which do not support this extension *MUST* ignore the unknown `ENABLE_XHEADERS` setting and XHEADERS frame. Endpoints can negotiate the use of this extension through the SETTINGS frame, and once enabled, this extension *MUST NOT* be disabled over the lifetime of the connection.

This document introduces another SETTINGS parameter, `ENABLE_XHEADERS`, which *MUST* have a value of 0 or 1.

Once a `ENABLE_XHEADERS` parameter has been sent with a value of 1, an endpoint *MUST NOT* send the parameter with a value of 0.

If an implementation supports the extension, it is *RECOMMENDED* to include the `ENABLE_XHEADERS` setting in the initial SETTINGS frame, such that the remote endpoint can discover the support at the earliest possible time.

An endpoint can send XHEADERS frames immediately upon receiving a SETTINGS frame with `ENABLE_XHEADERS=1`. An endpoint *MUST NOT* send out XHEADERS before receiving a SETTINGS frame with the

ENABLE_XHEADERS=1. If a remote endpoint does not support this extension, the XHEADERS will be ignored, making the header compression context inconsistent between sender and receiver.

If an endpoint supports this extension, but receives XHEADERS frames before ENABLE_XHEADERS, it **SHOULD** to respond with a connection error XHEADER_NOT_ENABLED_ERROR. This helps the remote endpoint to implement this extension properly.

Intermediaries **SHOULD** send the ENABLE_XHEADERS setting to clients only if intermediaries and their upstream servers support this extension. If an intermediary receives an XStream but discovers the destination endpoint does not support the extension, it **MUST** reset the stream with XHEADER_NOT_ENABLED_ERROR.

3.7. Interaction with Standard HTTP/2 Features

XStreams are extended HTTP/2 streams, thus all the standard HTTP/2 features for streams still apply to XStreams. For example, like streams, XStreams are counted against the concurrent stream limit, defined in [RFC7540], Section 5.1.2. The connection level and stream level flow control principles are still valid for XStreams. However, for the stream priority and dependencies, XStreams have one extra constraint: a XStream can have a dependency on its RStream, or any XStream sharing with the same RStream. Prioritizing the XStreams across different RStream groups does not make sense, because they belong to different services.

4. HTTP/2 XHEADERS Frame

The XHEADERS frame (type=0xfb) has all the fields and frame header flags defined by HEADERS frame in HEADERS [RFC7540], section 6.2. The XHEADERS frame has one extra field, Routing Stream ID. It is used to open an XStream, and additionally carries a header block fragment. XHEADERS frames can be sent on a stream in the "idle", "open", or "half-closed (remote)" state.

Like HEADERS, the CONTINUATION frame (type=0x9) is used to continue a sequence of header block fragments, if the headers do not fit into one XHEADERS frame.

4.1. Definition

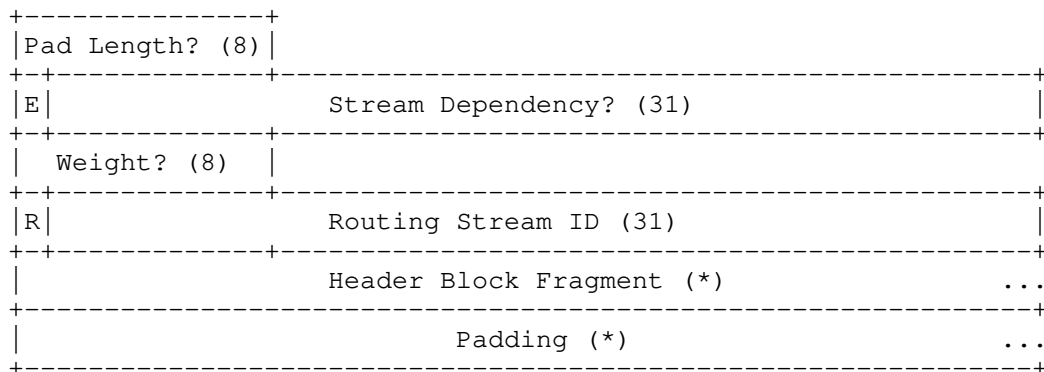


Figure 4: XHEADERS Frame Payload

The RStream specified in a XHEADERS frame **MUST** be an open stream. The recipient **MUST** respond with a connection error of type `ROUTING_STREAM_ERROR` `PROTOCOL_ERROR`, if the specified RStream is missing, is an XStream rather than a regular HTTP/2 stream, or is closed or half-closed (remote). Otherwise, the states maintained for header compression or flow control may be out of sync.

4.2. Examples

This section shows HTTP/1.1 request and response messages that are transmitted on an RStream with regular HEADERS frames, and on an XStream with HTTP/2 XHEADERS frames.

```

GET /login HTTP/1.1
Host: example.org                                ==>  HEADERS
                                                    - END_STREAM
                                                    + END_HEADERS
                                                    :method = GET
                                                    :scheme = https
                                                    :path = /login
                                                    host = example.org

{binary data .... }                             ==>  DATA
                                                    - END_STREAM
                                                    {binary data ... }
```

Figure 5: The request message and HEADERS frame on an RStream

```

HTTP/1.1 200 OK                                ==>  HEADERS
                                                - END_STREAM
                                                + END_HEADERS
                                                :status = 200

{binary data .... }                          ==>  DATA
                                                - END_STREAM
                                                {binary data...}

```

Figure 6: The response message and HEADERS frame on an RStream

The server initiates an XStream to this client.

```

POST /new_msg HTTP/1.1                        XHEADERS
                                                RStream_ID = 3
Host: example.org                            ==>  - END_STREAM
                                                + END_HEADERS
                                                :method = POST
                                                :scheme = https
                                                :path = /new_msg
                                                host = example.org

{binary data}                                ==>  DATA
                                                + END_STREAM
                                                {binary data}

```

Figure 7: The request message and XHEADERS frame on an XStream

```

HTTP/1.1 200 OK                                XHEADERS
                                                RStream_ID = 3
                                                ==>  + END_STREAM
                                                + END_HEADERS
                                                :status = 200

```

Figure 8: The response message and XHEADERS frame on an XStream

5. IANA Considerations

This specification adds an entry to the "HTTP/2 Frame Type" registry, the "HTTP/2 Settings" registry, and the "HTTP/2 Error Code" registry, all defined in [RFC7540].

5.1. FRAME TYPE Registry

The entry in the following table are registered by this document.

Frame Type	Code	Section
XHEADERS	0xfb	

5.2. Settings Registry

The entry in the following table are registered by this document.

Name	Code	Initial Value	Specification
ENABLE_XHEADERS	0xfbfb	0	

5.3. Error Code Registry

The entry in the following table are registered by this document.

Name	Code	Description	Specification
ROUTING_STREAM_ERROR	0xfb	Routing stream is not open	
XHEADERS_NOT_ENABLED_ERROR	0xfc	XHEADERS is not enabled yet	

6. 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>>.
- [RFC6202] Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins, "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP", RFC 6202, DOI 10.17487/RFC6202, April 2011, <<https://www.rfc-editor.org/info/rfc6202>>.
- [RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <<https://www.rfc-editor.org/info/rfc7540>>.

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

Authors' Addresses

Guowu Xie
Facebook Inc.

Email: woo@fb.com

Alan Frindell
Facebook Inc.

Email: afrind@fb.com