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.

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 psuedo-header field :protocol which can indicate the protocol intended to be used on the tunnel established by the CONNECT method. Values for the :protocol psuedo-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.

- Value: bytestream
- Description: Arbitrary bidirectional bytestream data
- Expected Version Tokens:
- 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