

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.

Internet-Draft

HTTP-PUBSUB

July 2019

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 workaround 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.

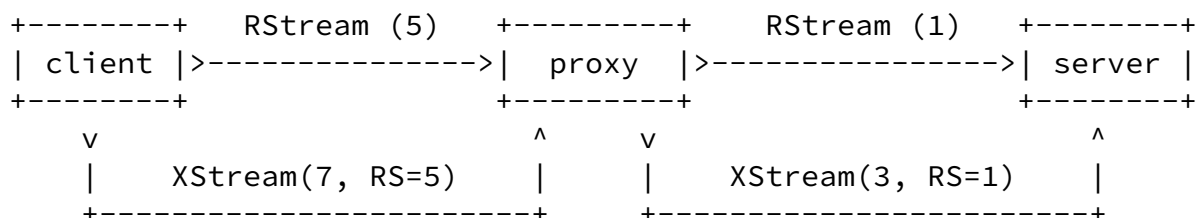


Figure 1: Client initiates an XStream to server.

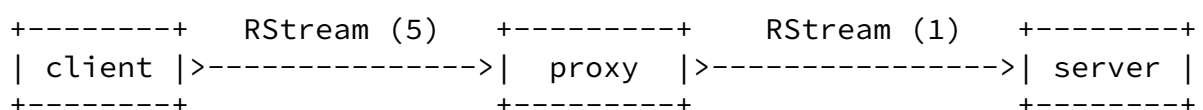




Figure 2: Server initiates an XStream to client.

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.

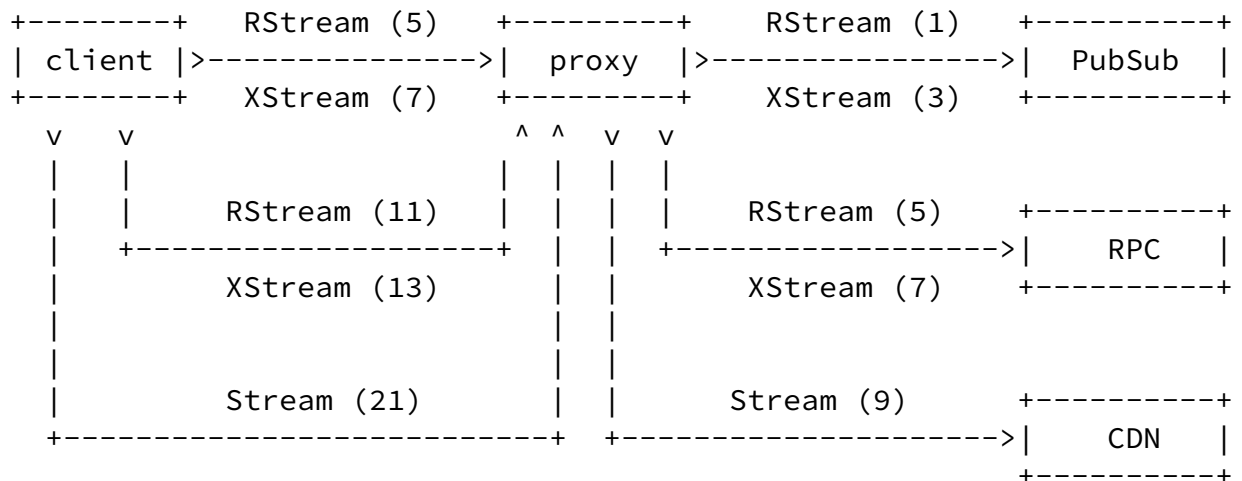


Figure 3: Client opens multiple RStreams, XStreams and an HTTP/2 stream within one HTTP/2 connection.

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

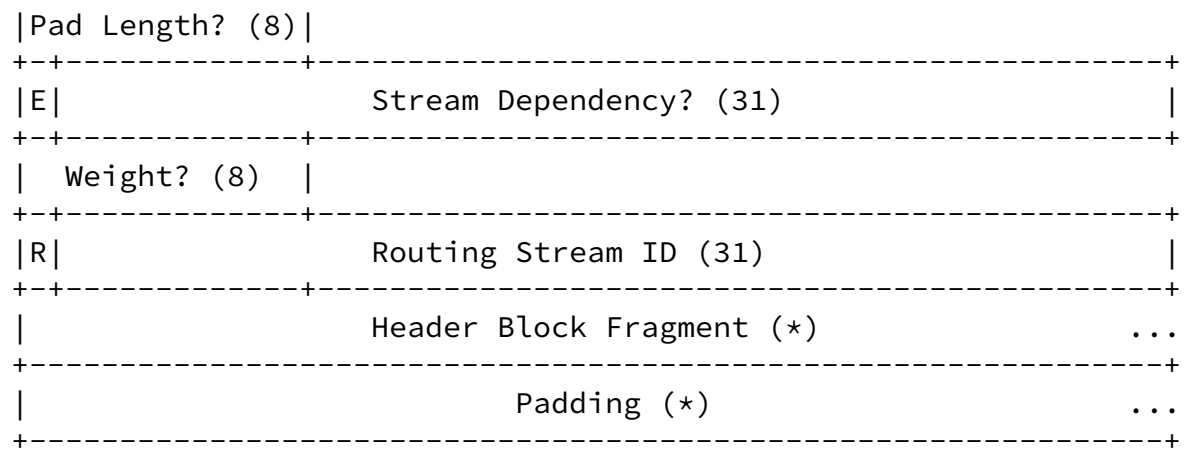


Figure 4: XHEADERS Frame Payload

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 | | HEADERS |
| Host: example.org | ==> | - 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
Host: example.org                             RStream_ID = 3
                                              - 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.

Internet-Draft

HTTP-PUBSUB

July 2019

| | | |
|------------|------|---------|
| 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>>.

Xie & Frindell

Expires January 9, 2020

[Page 10]

Internet-Draft

HTTP-PUBSUB

July 2019

[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

