Stream: Internet Engineering Task Force (IETF)
RFC: 9297
Category: Standards Track
Published: August 2022
ISSN: 2070-1721
Authors: D. Schinazi, *Google LLC*; L. Pardue, *Cloudflare*

RFC 9297
HTTP Datagrams and the Capsule Protocol

Abstract

This document describes HTTP Datagrams, a convention for conveying multiplexed, potentially unreliable datagrams inside an HTTP connection.

In HTTP/3, HTTP Datagrams can be sent unreliably using the QUIC DATAGRAM extension. When the QUIC DATAGRAM frame is unavailable or undesirable, HTTP Datagrams can be sent using the Capsule Protocol, which is a more general convention for conveying data in HTTP connections.

HTTP Datagrams and the Capsule Protocol are intended for use by HTTP extensions, not applications.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9297.

Copyright Notice

Table of Contents

1. Introduction

HTTP extensions (as defined in Section 16 of [HTTP]) sometimes need to access underlying transport protocol features such as unreliable delivery (as offered by [QUIC-DGRAM]) to enable desirable features. For example, this could allow for the introduction of an unreliable version of the CONNECT method and the addition of unreliable delivery to WebSockets [WEBSOCKET].

In Section 2, this document describes HTTP Datagrams, a convention for conveying bidirectional and potentially unreliable datagrams inside an HTTP connection, with multiplexing when possible. While HTTP Datagrams are associated with HTTP requests, they are not a part of message content. Instead, they are intended for use by HTTP extensions (such as the CONNECT method) and are compatible with all versions of HTTP.

When HTTP is running over a transport protocol that supports unreliable delivery (such as when the QUIC DATAGRAM extension [QUIC-DGRAM] is available to HTTP/3 [HTTP/3]), HTTP Datagrams can use that capability.

In Section 3, this document describes the HTTP Capsule Protocol, which allows the conveyance of HTTP Datagrams using reliable delivery. This addresses HTTP/3 cases where use of the QUIC DATAGRAM frame is unavailable or undesirable or where the transport protocol only provides reliable delivery, such as with HTTP/1.1 [HTTP/1.1] or HTTP/2 [HTTP/2] over TCP [TCP].

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.

This document uses terminology from [QUIC].

Where this document defines protocol types, the definition format uses the notation from Section 1.3 of [QUIC]. Where fields within types are integers, they are encoded using the variable-length integer encoding from Section 16 of [QUIC]. Integer values do not need to be encoded on the minimum number of bytes necessary.

In this document, the term "intermediary" refers to an HTTP intermediary as defined in Section 3.7 of [HTTP].

2. HTTP Datagrams

HTTP Datagrams are a convention for conveying bidirectional and potentially unreliable datagrams inside an HTTP connection with multiplexing when possible. All HTTP Datagrams are associated with an HTTP request.

When HTTP Datagrams are conveyed on an HTTP/3 connection, the QUIC DATAGRAM frame can be used to provide demultiplexing and unreliable delivery; see Section 2.1. Negotiating the use of QUIC DATAGRAM frames for HTTP Datagrams is achieved via the exchange of HTTP/3 settings; see Section 2.1.1.

When running over HTTP/2, demultiplexing is provided by the HTTP/2 framing layer, but unreliable delivery is unavailable. HTTP Datagrams are negotiated and conveyed using the Capsule Protocol; see Section 3.5.

When running over HTTP/1.x, requests are strictly serialized in the connection; therefore, demultiplexing is not available. Unreliable delivery is likewise not available. HTTP Datagrams are negotiated and conveyed using the Capsule Protocol; see Section 3.5.

HTTP Datagrams **MUST** only be sent with an association to an HTTP request that explicitly supports them. For example, existing HTTP methods GET and POST do not define semantics for associated HTTP Datagrams; therefore, HTTP Datagrams associated with GET or POST request streams cannot be sent.

If an HTTP Datagram is received and it is associated with a request that has no known semantics for HTTP Datagrams, the receiver **MUST** terminate the request. If HTTP/3 is in use, the request stream **MUST** be aborted with H3_DATAGRAM_ERROR (0x33). HTTP extensions **MAY** override these requirements by defining a negotiation mechanism and semantics for HTTP Datagrams.

2.1. HTTP/3 Datagrams

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

```
HTTP/3 Datagram {
  Quarter Stream ID (i),
  HTTP Datagram Payload (..),
}
```

Figure 1: HTTP/3 Datagram Format

Quarter Stream ID: A variable-length integer that contains the value of the client-initiated bidirectional stream that this datagram is associated with divided by four (the division by four stems from the fact that HTTP requests are sent on client-initiated bidirectional streams, which have stream IDs that are divisible by four). The largest legal QUIC stream ID value is 2^{62}-1, so the largest legal value of the Quarter Stream ID field is 2^{60}-1. Receipt of an HTTP/3 Datagram that includes a larger value **MUST** be treated as an HTTP/3 connection error of type H3_DATAGRAM_ERROR (0x33).

HTTP Datagram Payload: The payload of the datagram, whose semantics are defined by the extension that is using HTTP Datagrams. Note that this field can be empty.

Receipt of a QUIC DATAGRAM frame whose payload is too short to allow parsing the Quarter Stream ID field **MUST** be treated as an HTTP/3 connection error of type H3_DATAGRAM_ERROR (0x33).

HTTP/3 Datagrams **MUST NOT** be sent unless the corresponding stream's send side is open. If a datagram is received after the corresponding stream's receive side is closed, the received datagrams **MUST** be silently dropped.

If an HTTP/3 Datagram is received and its Quarter Stream ID field maps to a stream that has not yet been created, the receiver **SHALL** either drop that datagram silently or buffer it temporarily (on the order of a round trip) while awaiting the creation of the corresponding stream.

If an HTTP/3 Datagram is received and its Quarter Stream ID field maps to a stream that cannot be created due to client-initiated bidirectional stream limits, it **SHOULD** be treated as an HTTP/3 connection error of type H3_ID_ERROR. Generating an error is not mandatory because the QUIC stream limit might be unknown to the HTTP/3 layer.

Prioritization of HTTP/3 Datagrams is not defined in this document. Future extensions **MAY** define how to prioritize datagrams and **MAY** define signaling to allow communicating prioritization preferences.

2.1.1. The SETTINGS_H3_DATAGRAM HTTP/3 Setting

An endpoint can indicate to its peer that it is willing to receive HTTP/3 Datagrams by sending the SETTINGS_H3_DATAGRAM (0x33) setting with a value of 1.

The value of the SETTINGS_H3_DATAGRAM setting **MUST** be either 0 or 1. A value of 0 indicates that the implementation is not willing to receive HTTP Datagrams. If the SETTINGS_H3_DATAGRAM setting is received with a value that is neither 0 nor 1, the receiver **MUST** terminate the connection with error H3_SETTINGS_ERROR.

QUIC DATAGRAM frames **MUST NOT** be sent until the SETTINGS_H3_DATAGRAM setting has been both sent and received with a value of 1.

When clients use 0-RTT, they **MAY** store the value of the server's SETTINGS_H3_DATAGRAM setting. Doing so allows the client to send QUIC DATAGRAM frames in 0-RTT packets. When servers decide to accept 0-RTT data, they **MUST** send a SETTINGS_H3_DATAGRAM setting greater than 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 SETTINGS_H3_DATAGRAM setting with their 0-RTT state, they **MUST** validate that the new value of the SETTINGS_H3_DATAGRAM setting sent by the server in the handshake is greater than or equal to the stored value; if not, the client **MUST** terminate the connection with error H3_SETTINGS_ERROR. In all cases, the maximum permitted value of the SETTINGS_H3_DATAGRAM setting parameter is 1.

It is **RECOMMENDED** that implementations that support receiving HTTP/3 Datagrams always send the SETTINGS_H3_DATAGRAM setting with a value of 1, even if the application does not intend to use HTTP/3 Datagrams. This helps to avoid "sticking out"; see Section 4.

2.2. HTTP Datagrams Using Capsules

When HTTP/3 Datagrams are unavailable or undesirable, HTTP Datagrams can be sent using the Capsule Protocol; see Section 3.5.

3. Capsules

One mechanism to extend HTTP is to introduce new HTTP upgrade tokens; see Section 16.7 of [HTTP]. In HTTP/1.x, these tokens are used via the Upgrade mechanism; see Section 7.8 of [HTTP]. In HTTP/2 and HTTP/3, these tokens are used via the Extended CONNECT mechanism; see [EXT-CONNECT2] and [EXT-CONNECT3].

This specification introduces the Capsule Protocol. The Capsule Protocol is a sequence of type-length-value tuples that definitions of new HTTP upgrade tokens can choose to use. It allows endpoints to reliably communicate request-related information end-to-end on HTTP request streams, even in the presence of HTTP intermediaries. The Capsule Protocol can be used to exchange HTTP Datagrams, which is necessary when HTTP is running over a transport that does not support the QUIC DATAGRAM frame. The Capsule Protocol can also be used to communicate reliable and bidirectional control messages associated with a datagram-based protocol even when HTTP/3 Datagrams are in use.

3.1. HTTP Data Streams

This specification defines the "data stream" of an HTTP request as the bidirectional stream of bytes that follows the header section of the request message and the final response message that is either successful (i.e., 2xx) or upgraded (i.e., 101).

In HTTP/1.x, the data stream consists of all bytes on the connection that follow the blank line that concludes either the request header section or the final response header section. As a result, only the last HTTP request on an HTTP/1.x connection can start the Capsule Protocol.

In HTTP/2 and HTTP/3, the data stream of a given HTTP request consists of all bytes sent in DATA frames with the corresponding stream ID.

The concept of a data stream is particularly relevant for methods such as CONNECT, where there is no HTTP message content after the headers.

Data streams can be prioritized using any means suited to stream or request prioritization. For example, see Section 11 of [PRIORITY].

Data streams are subject to the flow control mechanisms of the underlying layers; examples include HTTP/2 stream flow control, HTTP/2 connection flow control, and TCP flow control.

3.2. The Capsule Protocol

Definitions of new HTTP upgrade tokens can state that their associated request's data stream uses the Capsule Protocol. If they do so, the contents of the associated request's data stream uses the following format:

```
Capsule Protocol {
  Capsule (..) ...,
}
```

Figure 2: Capsule Protocol Stream Format

```
Capsule {
  Capsule Type (i),
  Capsule Length (i),
  Capsule Value (..),
}
```

Figure 3: Capsule Format

Capsule Type: A variable-length integer indicating the type of the capsule. An IANA registry is used to manage the assignment of Capsule Types; see Section 5.4.

Capsule Length: The length, in bytes, of the Capsule Value field, which follows this field, encoded as a variable-length integer. Note that this field can have a value of zero.

Capsule Value: The payload of this Capsule. Its semantics are determined by the value of the Capsule Type field.

An intermediary can identify the use of the Capsule Protocol either through the presence of the Capsule-Protocol header field (Section 3.4) or by understanding the chosen HTTP Upgrade token.

Because new protocols or extensions might define new Capsule Types, intermediaries that wish to allow for future extensibility **SHOULD** forward Capsules without modification unless the definition of the Capsule Type in use specifies additional intermediary processing. One such Capsule Type is the DATAGRAM Capsule; see Section 3.5. In particular, intermediaries **SHOULD** forward Capsules with an unknown Capsule Type without modification.

Endpoints that receive a Capsule with an unknown Capsule Type **MUST** silently drop that Capsule and skip over it to parse the next Capsule.

By virtue of the definition of the data stream:

- The Capsule Protocol is not in use unless the response includes a 2xx (Successful) or 101 (Switching Protocols) status code.
- When the Capsule Protocol is in use, the associated HTTP request and response do not carry HTTP content. A future extension **MAY** define a new Capsule Type to carry HTTP content.

The Capsule Protocol only applies to definitions of new HTTP upgrade tokens; thus, in HTTP/2 and HTTP/3, it can only be used with the CONNECT method. Therefore, once both endpoints agree to use the Capsule Protocol, the frame usage requirements of the stream change as specified in Section 8.5 of [HTTP/2] and Section 4.4 of [HTTP/3].

The Capsule Protocol **MUST NOT** be used with messages that contain Content-Length, Content-Type, or Transfer-Encoding header fields. Additionally, HTTP status codes 204 (No Content), 205 (Reset Content), and 206 (Partial Content) **MUST NOT** be sent on responses that use the Capsule Protocol. A receiver that observes a violation of these requirements **MUST** treat the HTTP message as malformed.

When processing Capsules, a receiver might be tempted to accumulate the full length of the Capsule Value field in the data stream before handling it. This approach **SHOULD** be avoided because it can consume flow control in underlying layers, and that might lead to deadlocks if the Capsule data exhausts the flow control window.

3.3. Error Handling

When a receiver encounters an error processing the Capsule Protocol, the receiver **MUST** treat it as if it had received a malformed or incomplete HTTP message. For HTTP/3, the handling of malformed messages is described in Section 4.1.2 of [HTTP/3]. For HTTP/2, the handling of malformed messages is described in Section 8.1.1 of [HTTP/2]. For HTTP/1.x, the handling of incomplete messages is described in Section 8 of [HTTP/1.1].

Each Capsule's payload **MUST** contain exactly the fields identified in its description. A Capsule payload that contains additional bytes after the identified fields or a Capsule payload that terminates before the end of the identified fields **MUST** be treated as it if were a malformed or incomplete message. In particular, redundant length encodings **MUST** be verified to be self-consistent.

If the receive side of a stream carrying Capsules is terminated cleanly (for example, in HTTP/3 this is defined as receiving a QUIC STREAM frame with the FIN bit set) and the last Capsule on the stream was truncated, this **MUST** be treated as if it were a malformed or incomplete message.

3.4. The Capsule-Protocol Header Field

The "Capsule-Protocol" header field is an Item Structured Field; see Section 3.3 of [STRUCTURED-FIELDS]. Its value **MUST** be a Boolean; any other value type **MUST** be handled as if the field were not present by recipients (for example, if this field is included multiple times, its type will

become a List and the field will be ignored). This document does not define any parameters for the Capsule-Protocol header field value, but future documents might define parameters. Receivers **MUST** ignore unknown parameters.

Endpoints indicate that the Capsule Protocol is in use on a data stream by sending a Capsule-Protocol header field with a true value. A Capsule-Protocol header field with a false value has the same semantics as when the header is not present.

Intermediaries **MAY** use this header field to allow processing of HTTP Datagrams for unknown HTTP upgrade tokens. Note that this is only possible for HTTP Upgrade or Extended CONNECT.

The Capsule-Protocol header field **MUST NOT** be used on HTTP responses with a status code that is both different from 101 (Switching Protocols) and outside the 2xx (Successful) range.

When using the Capsule Protocol, HTTP endpoints **SHOULD** send the Capsule-Protocol header field to simplify intermediary processing. Definitions of new HTTP upgrade tokens that use the Capsule Protocol **MAY** alter this recommendation.

3.5. The DATAGRAM Capsule

This document defines the DATAGRAM (0x00) Capsule Type. This Capsule allows HTTP Datagrams to be sent on a stream using the Capsule Protocol. This is particularly useful when HTTP is running over a transport that does not support the QUIC DATAGRAM frame.

```
Datagram Capsule {
  Type (i) = 0x00,
  Length (i),
  HTTP Datagram Payload (..),
}
```

Figure 4: DATAGRAM Capsule Format

HTTP Datagram Payload: The payload of the datagram, whose semantics are defined by the extension that is using HTTP Datagrams. Note that this field can be empty.

HTTP Datagrams sent using the DATAGRAM Capsule have the same semantics as those sent in QUIC DATAGRAM frames. In particular, the restrictions on when it is allowed to send an HTTP Datagram and how to process them (from Section 2.1) also apply to HTTP Datagrams sent and received using the DATAGRAM Capsule.

An intermediary can re-encode HTTP Datagrams as it forwards them. In other words, an intermediary **MAY** send a DATAGRAM Capsule to forward an HTTP Datagram that was received in a QUIC DATAGRAM frame and vice versa. Intermediaries **MUST NOT** perform this re-encoding unless they have identified the use of the Capsule Protocol on the corresponding request stream; see Section 3.2.

Note that while DATAGRAM Capsules, which are sent on a stream, are reliably delivered in order, intermediaries can re-encode DATAGRAM Capsules into QUIC DATAGRAM frames when forwarding messages, which could result in loss or reordering.

If an intermediary receives an HTTP Datagram in a QUIC DATAGRAM frame and is forwarding it on a connection that supports QUIC DATAGRAM frames, the intermediary **SHOULD NOT** convert that HTTP Datagram to a DATAGRAM Capsule. If the HTTP Datagram is too large to fit in a DATAGRAM frame (for example, because the Path MTU (PMTU) of that QUIC connection is too low or if the maximum UDP payload size advertised on that connection is too low), the intermediary **SHOULD** drop the HTTP Datagram instead of converting it to a DATAGRAM Capsule. This preserves the end-to-end unreliability characteristic that methods such as Datagram Packetization Layer PMTU Discovery (DPLPMTUD) depend on [DPLPMTUD]. An intermediary that converts QUIC DATAGRAM frames to DATAGRAM Capsules allows HTTP Datagrams to be arbitrarily large without suffering any loss. This can misrepresent the true path properties, defeating methods such as DPLPMTUD.

While DATAGRAM Capsules can theoretically carry a payload of length 2^{62}-1, most HTTP extensions that use HTTP Datagrams will have their own limits on what datagram payload sizes are practical. Implementations **SHOULD** take those limits into account when parsing DATAGRAM Capsules. If an incoming DATAGRAM Capsule has a length that is known to be so large as to not be usable, the implementation **SHOULD** discard the Capsule without buffering its contents into memory.

Since QUIC DATAGRAM frames are required to fit within a QUIC packet, implementations that re-encode DATAGRAM Capsules into QUIC DATAGRAM frames might be tempted to accumulate the entire Capsule in the stream before re-encoding it. This **SHOULD** be avoided, because it can cause flow control problems; see Section 3.2.

Note that it is possible for an HTTP extension to use HTTP Datagrams without using the Capsule Protocol. For example, if an HTTP extension that uses HTTP Datagrams is only defined over transports that support QUIC DATAGRAM frames, it might not need a stream encoding. Additionally, HTTP extensions can use HTTP Datagrams with their own data stream protocol. However, new HTTP extensions that wish to use HTTP Datagrams **SHOULD** use the Capsule Protocol, as failing to do so will make it harder for the HTTP extension to support versions of HTTP other than HTTP/3 and will prevent interoperability with intermediaries that only support the Capsule Protocol.

4. Security Considerations

Since transmitting HTTP Datagrams using QUIC DATAGRAM frames requires sending the HTTP/3 SETTINGS_H3_DATAGRAM setting, it "sticks out". In other words, probing clients can learn whether a server supports HTTP Datagrams over QUIC DATAGRAM frames. As some servers might wish to obfuscate the fact that they offer application services that use HTTP Datagrams, it's best for all implementations that support this feature to always send this setting; see Section 2.1.1.

Since use of the Capsule Protocol is restricted to new HTTP upgrade tokens, it is not directly accessible from Web Platform APIs (such as those commonly accessed via JavaScript in web browsers).

Definitions of new HTTP upgrade tokens that use the Capsule Protocol need to include a security analysis that considers the impact of HTTP Datagrams and Capsules in the context of their protocol.

5. IANA Considerations

5.1. HTTP/3 Setting

IANA has registered the following entry in the "HTTP/3 Settings" registry maintained at <https://www.iana.org/assignments/http3-parameters>:

Value: 0x33

Setting Name: SETTINGS_H3_DATAGRAM

Default: 0

Status: permanent

Reference: RFC 9297

Change Controller: IETF

Contact: HTTP_WG; HTTP working group; ietf-http-wg@w3.org

Notes: None

5.2. HTTP/3 Error Code

IANA has registered the following entry in the "HTTP/3 Error Codes" registry maintained at <https://www.iana.org/assignments/http3-parameters>:

Value: 0x33

Name: H3_DATAGRAM_ERROR

Description: Datagram or Capsule Protocol parse error

Status: permanent

Reference: RFC 9297

Change Controller: IETF

Contact: HTTP_WG; HTTP working group; ietf-http-wg@w3.org

Notes: None

5.3. HTTP Header Field Name

IANA has registered the following entry in the "Hypertext Transfer Protocol (HTTP) Field Name Registry" maintained at <https://www.iana.org/assignments/http-fields>:

Field Name: Capsule-Protocol

Template: None

Status: permanent

Reference: RFC 9297

Comments: None

5.4. Capsule Types

This document establishes a registry for HTTP Capsule Type codes. The "HTTP Capsule Types" registry governs a 62-bit space and operates under the QUIC registration policy documented in Section 22.1 of [QUIC]. This new registry includes the common set of fields listed in Section 22.1.1 of [QUIC]. In addition to those common fields, all registrations in this registry **MUST** include a "Capsule Type" field that contains a short name or label for the Capsule Type.

Permanent registrations in this registry are assigned using the Specification Required policy (Section 4.6 of [IANA-POLICY]), except for values between 0x00 and 0x3f (in hexadecimal; inclusive), which are assigned using Standards Action or IESG Approval as defined in Sections 4.9 and 4.10 of [IANA-POLICY].

Capsule Types with a value of the form 0x29 * N + 0x17 for integer values of N are reserved to exercise the requirement that unknown Capsule Types be ignored. These Capsules have no semantics and can carry arbitrary values. These values **MUST NOT** be assigned by IANA and **MUST NOT** appear in the listing of assigned values.

This registry initially contains the following entry:

Value: 0x00

Capsule Type: DATAGRAM

Status: permanent

Reference: RFC 9297

Change Controller: IETF

Contact: MASQUE Working Group masque@ietf.org

Notes: None

6. References

6.1. Normative References

[HTTP] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

[HTTP/1.1] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1", STD 99, RFC 9112, DOI 10.17487/RFC9112, June 2022, <https://www.rfc-editor.org/info/rfc9112>.

[HTTP/2] Thomson, M., Ed. and C. Benfield, Ed., "HTTP/2", RFC 9113, DOI 10.17487/RFC9113, June 2022, <https://www.rfc-editor.org/info/rfc9113>.

[HTTP/3] Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <https://www.rfc-editor.org/info/rfc9114>.

[IANA-POLICY] 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>.

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

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

[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>.

[STRUCTURED-FIELDS] Nottingham, M. and P-H. Kamp, "Structured Field Values for HTTP", RFC 8941, DOI 10.17487/RFC8941, February 2021, <https://www.rfc-editor.org/info/rfc8941>.

[TCP] Eddy, W., Ed., "Transmission Control Protocol (TCP)", STD 7, RFC 9293, DOI 10.17487/RFC9293, August 2022, <https://www.rfc-editor.org/info/rfc9293>.

6.2. Informative References

[DPLPMTUD] Fairhurst, G., Jones, T., Tüxen, M., Rüngeler, I., and T. Völker, "Packetization Layer Path MTU Discovery for Datagram Transports", RFC 8899, DOI 10.17487/RFC8899, September 2020, <https://www.rfc-editor.org/info/rfc8899>.

[EXT-CONNECT2] McManus, P., "Bootstrapping WebSockets with HTTP/2", RFC 8441, DOI 10.17487/RFC8441, September 2018, <https://www.rfc-editor.org/info/rfc8441>.

[EXT-CONNECT3] Hamilton, R., "Bootstrapping WebSockets with HTTP/3", RFC 9220, DOI 10.17487/RFC9220, June 2022, <https://www.rfc-editor.org/info/rfc9220>.

[PRIORITY] Oku, K. and L. Pardue, "Extensible Prioritization Scheme for HTTP", RFC 9218, DOI 10.17487/RFC9218, June 2022, <https://www.rfc-editor.org/info/rfc9218>.

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

Acknowledgments

Portions of this document were previously part of the QUIC DATAGRAM frame definition itself; the authors would like to acknowledge the authors of that document and the members of the IETF MASQUE working group for their suggestions. Additionally, the authors would like to thank Martin Thomson for suggesting the use of an HTTP/3 setting. Furthermore, the authors would like to thank Ben Schwartz for substantive input. The final design in this document came out of the HTTP Datagrams Design Team, whose members were Alan Frindell, Alex Chernyakhovsky, Ben Schwartz, Eric Rescorla, Marcus Ihlar, Martin Thomson, Mike Bishop, Tommy Pauly, Victor Vasiliev, and the authors of this document. The authors thank Mark Nottingham and Philipp Tiesel for their helpful comments.

Authors' Addresses

David Schinazi
Google LLC
1600 Amphitheatre Parkway
Mountain View, CA 94043
United States of America
Email: dschinazi.ietf@gmail.com

Lucas Pardue
Cloudflare
Email: lucaspardue.24.7@gmail.com