Workgroup: MASQUE
Internet-Draft:
draft-ietf-masque-h3-datagram-08
Published: 28 March 2022
Intended Status: Standards Track
Expires: 29 September 2022
Authors: D. Schinazi L. Pardue
Google LLC Cloudflare

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 conveyed natively using the QUIC DATAGRAM extension. When the QUIC DATAGRAM frame is unavailable or undesirable, they can be sent using the Capsule Protocol, a more general convention for conveying data in HTTP connections.

Both are intended for use by HTTP extensions, not applications.

Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the MASQUE WG mailing list (masque@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/masque/.

Source for this draft and an issue tracker can be found at https://github.com/ietf-wg-masque/draft-ietf-masque-h3-datagram.

Status of This Memo

This Internet-Draft will expire on 29 September 2022.

Copyright Notice

Table of Contents

1. Introduction

HTTP extensions sometimes need to access underlying transport protocol features such as unreliable delivery (as offered by [DGRAM]) to enable desirable features like an unreliable version of the CONNECT method, and unreliable delivery in WebSockets [RFC6455] (or its successors).

In Section 2, this document describes HTTP Datagrams, a convention that supports the bidirectional and possibly multiplexed exchange of data inside an HTTP connection. While HTTP datagrams are associated with HTTP requests, they are not 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 the underlying transport protocol supports unreliable delivery (such as when the QUIC DATAGRAM extension is available in HTTP/3), they can use that capability.

This document also describes the HTTP Capsule Protocol in Section 3, to allow conveyance of HTTP Datagrams when the QUIC DATAGRAM frame is unavailable or undesirable, such as when earlier versions of HTTP are in use.

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 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 achieve these goals, including unreliable delivery; see Section 2.1. Negotiation is achieved using a setting; 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, requests are strictly serialized in the connection, and 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 a stream whose HTTP semantics explicitly supports HTTP Datagrams. For example, existing HTTP methods GET and POST do not define semantics for associated HTTP Datagrams; therefore, HTTP Datagrams cannot be sent associated with GET or POST request streams.

If an HTTP Datagram associated with a method that has no known semantics for HTTP Datagrams is received, the receiver **MUST** abort the corresponding stream; if HTTP/3 is in use, the stream **MUST** be aborted with H3_DATAGRAM_ERROR. HTTP extensions can 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 (using the notation from the "Notational Conventions" section of [QUIC]):

```
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, and those 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 Quarter Stream ID 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.

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.

HTTP/3 Datagrams **MUST NOT** be sent unless the corresponding stream's send side is open. Once the receive side of a stream is closed, incoming datagrams for this stream are no longer expected so related state can be released. State **MAY** be kept for a short time to account for reordering. Once the state is released, the received associated datagrams **MUST** be silently dropped.

If an HTTP/3 datagram is received and its Quarter Stream ID 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 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 in this case because HTTP/3 implementations might have practical barriers to determining the active stream concurrency limit that is applied by the QUIC 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 H3_DATAGRAM HTTP/3 SETTINGS Parameter

Implementations of HTTP/3 that support HTTP Datagrams 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 HTTP Datagrams are not supported. If the H3_DATAGRAM SETTINGS parameter is received with a value that is neither 0 or 1, the receiver **MUST** terminate the connection with error H3_SETTINGS_ERROR.

QUIC DATAGRAM frames **MUST NOT** be sent until the H3_DATAGRAM SETTINGS parameter 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 H3_DATAGRAM SETTINGS parameter. 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 H3_DATAGRAM SETTINGS parameter 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 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 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 H3_DATAGRAM SETTINGS parameter is 1.

It is **RECOMMENDED** that implementations that support receiving HTTP Datagrams using QUIC always send the H3_DATAGRAM SETTINGS parameter with a value of 1, even if the application does not intend to use HTTP Datagrams. This helps to avoid "sticking out"; see Section 4.

2.1.1.1. Note About Draft Versions

[[RFC editor: please remove this section before publication.]]

Some revisions of this draft specification use a different value (the Identifier field of a Setting in the HTTP/3 SETTINGS frame) for the H3_DATAGRAM Settings Parameter. This allows new draft revisions to make incompatible changes. Multiple draft versions **MAY** be supported by sending multiple values for H3_DATAGRAM. Once SETTINGS have been sent and received, an implementation that supports multiple drafts **MUST** compute the intersection of the values it has sent and received, and then it **MUST** select and use the most recent draft version from the intersection set. This ensures that both peers negotiate the same draft version.

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.

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, successful (i.e., 2xx) response message.

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 response header section. As a result, only a single HTTP request starting the capsule protocol can be sent on HTTP/1.x connections.

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].

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, that means that the contents of the associated request's data stream uses the following format (using the notation from the "Notational Conventions" section of [QUIC]):

```
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.

Capsule Length: The length of the Capsule Value field following 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 which 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 on responses unless the response includes a 2xx (Successful) status code.

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.

3.3. Error Handling

When an error occurs in processing the Capsule Protocol, the receiver **MUST** treat the message as malformed or incomplete, according to the underlying transport protocol. For HTTP/3, the handling of malformed messages is described in Section 4.1.3 of [H3]. For HTTP/2, the handling of malformed messages is described in Section 8.1.1 of [H2]. For HTTP/1.1, the handling of incomplete messages is described in Section 8 of [H1].

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 a malformed or incomplete message. In particular, redundant length encodings **MUST** be verified to be self-consistent.

When a stream carrying capsules terminates cleanly, if the last capsule on the stream was truncated, this **MUST** be treated as 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 [STRUCT-FIELD]; 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 therefore 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 outside the 2xx 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 capsule type (see Section 5.4 for the value of the 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) = DATAGRAM,
  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 reencode HTTP Datagrams as it forwards them. In other words, an intermediary **MAY** send a DATAGRAM capsule to forward an HTTP Datagram which was received in a QUIC DATAGRAM frame, and vice versa.

Note that while DATAGRAM capsules that are sent on a stream are reliably delivered in order, intermediaries can reencode 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 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 Path MTU 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.

Note that use of the Capsule Protocol is not required to use HTTP Datagrams. 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 unless they have a good reason not to.

4. Security Considerations

Since transmitting HTTP Datagrams using QUIC DATAGRAM frames requires sending an HTTP/3 Settings parameter, 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 Settings parameter, see Section 2.1.1.

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

5. IANA Considerations

5.1. HTTP/3 SETTINGS Parameter

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

Value: 0xffd277 (note that this will switch to a lower value before publication)

Setting Name: H3_DATAGRAM

Default: 0

Status: provisional (permanent if this document is approved)

Specification: This Document

Change Controller: IETF

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

5.2. HTTP/3 Error Code

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

Value: 0x4A1268 (note that this will switch to a lower value before publication)

Name: H3_DATAGRAM_ERROR

Description: Datagram or capsule protocol parse error

Status: provisional (permanent if this document is approved)

Specification: This Document

Change Controller: IETF

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

5.3. HTTP Header Field Name

This document will request IANA to register the following entry in the "HTTP Field Name" registry:

Field Name:
Capsule-Protocol

Template: None

Status: provisional (permanent if this document is approved)

Reference: This document

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. Registrations in this registry **MUST** include the following fields:

Type: A name or label for the capsule type.

Value: The value of the Capsule Type field (see Section 3.2) is a 62-bit integer.

Reference: An optional reference to a specification for the type. This field **MAY** be empty.

Registrations follow the "First Come First Served" policy (see Section 4.4 of [IANA-POLICY]) where two registrations **MUST NOT** have the same Type.

This registry initially contains the following entry:

Capsule Type: DATAGRAM

Value: 0xff37a5 (note that this will switch to a lower value before publication)

Reference: This document

Capsule types with a value of the form 41 * N + 23 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.

6. References

6.1. 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-10, 4 February 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-datagram-10>.

[H1] Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP/1.1", Work in Progress, Internet-Draft, draft-ietf-httpbis-messaging-19, 12 September 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-messaging-19>.

[H2] Thomson, M. and C. Benfield, "HTTP/2", Work in Progress, Internet-Draft, draft-ietf-httpbis-http2bis-07, 24 January 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-http2bis-07>.

[H3] Bishop, M., "Hypertext Transfer Protocol Version 3 (HTTP/3)", Work in Progress, Internet-Draft, draft-ietf-quic-http-34, 2 February 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-34>.

[HTTP] Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP Semantics", Work in Progress, Internet-Draft, draft-ietf-httpbis-semantics-19, 12 September 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-semantics-19>.

[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/rfc/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/rfc/rfc9000>.

[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/rfc/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/rfc/rfc8174>.

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

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/rfc/rfc8899>.

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

[EXT-CONNECT3] Hamilton, R., "Bootstrapping WebSockets with HTTP/3", Work in Progress, Internet-Draft, draft-ietf-httpbis-h3-websockets-04, 8 February 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-h3-websockets-04>.

[PRIORITY] Oku, K. and L. Pardue, "Extensible Prioritization Scheme for HTTP", Work in Progress, Internet-Draft, draft-ietf-httpbis-priority-12, 17 January 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-priority-12>.

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <https://www.rfc-editor.org/rfc/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 SETTINGS parameter. Furthermore, the authors would like to thank Ben Schwartz for writing the first proposal that used two layers of indirection. 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, California 94043,
United States of America

Email: dschinazi.ietf@gmail.com

Lucas Pardue
Cloudflare

Email: lucaspardue.24.7@gmail.com