Workgroup: MASQUE
Internet-Draft:
draft-ietf-masque-h3-datagram-01
Published: 13 May 2021
Intended Status: Standards Track
Expires: 14 November 2021

Authors: D. Schinazi   L. Pardue
Google LLC   Cloudflare

Using QUIC Datagrams with HTTP/3

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 describes how to use QUIC DATAGRAM frames when the application protocol running over QUIC is HTTP/3. It associates datagrams with client-initiated bidirectional streams and defines an optional additional demultiplexing layer.

Discussion of this work is encouraged to happen on the MASQUE IETF mailing list (masque@ietf.org) or on the GitHub repository which contains the draft: https://github.com/ietf-wg-masque/draft-ietf-masque-h3-datagram.

Status of This Memo

Copyright Notice

Table of Contents

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 describes how to use QUIC DATAGRAM frames when the application protocol running over QUIC is HTTP/3 [H3]. It associates datagrams with client-initiated bidirectional streams and defines an optional additional demultiplexing layer.

Discussion of this work is encouraged to happen on the MASQUE IETF mailing list (masque@ietf.org) or on the GitHub repository which contains the draft: https://github.com/ietf-wg-masque/draft-ietf-masque-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. Multiplexing

In order to allow multiple exchanges of datagrams to coexist on a given QUIC connection, HTTP datagrams contain two layers of multiplexing. First, the QUIC DATAGRAM frame payload starts with an encoded stream identifier that associates the datagram with a given QUIC stream. Second, datagrams carry a context identifier (see Section 2.1) that allows multiplexing multiple datagram contexts related to a given HTTP request. Conceptually, the first layer of multiplexing is per-hop, while the second is end-to-end.

2.1. Datagram Contexts

Within the scope of a given HTTP request, contexts provide an additional demultiplexing layer. Contexts determine the encoding of datagrams, and can be used to implicitly convey metadata. For example, contexts can be used for compression to elide some parts of the datagram: the context identifier then maps to a compression context that the receiver can use to reconstruct the elided data.

Contexts are identified within the scope of a given request by a numeric value, referred to as the context ID. A context ID is a 62-bit integer (0 to 2^62-1).

While stream IDs are a per-hop concept, context IDs are an end-to-end concept. In other words, if a datagram travels through one or more intermediaries on its way from client to server, the stream ID will most likely change from hop to hop, but the context ID will remain the same. Context IDs are opaque to intermediaries.

2.2. Context ID Allocation

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

Even-numbered context IDs are client-initiated, while odd-numbered context IDs are server-initiated. This means that an HTTP/3 client implementation of the context ID allocation service MUST only provide even-numbered IDs, while a server implementation MUST only provide odd-numbered IDs. Note that, once allocated, any context ID can be used by both client and server - only allocation carries separate namespaces to avoid requiring synchronization. Additionally, note that the context ID namespace is tied to a given HTTP request: it is possible for the same numeral context ID to be used simultaneously in distinct requests.

3. HTTP/3 DATAGRAM Frame Format

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),
  Context ID (i),
  HTTP/3 Datagram Payload (..),
}
```

Figure 1: HTTP/3 DATAGRAM Frame 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.)

Context ID: A variable-length integer indicating the context ID of the datagram (see Section 2.1).

HTTP/3 Datagram Payload: The payload of the datagram, whose semantics are defined by individual applications. Note that this field can be empty.

Intermediaries parse the Quarter Stream ID field in order to associate the QUIC DATAGRAM frame with a stream. If an intermediary receives a QUIC DATAGRAM frame whose payload is too short to allow parsing the Quarter Stream ID field, the intermediary MUST treat it as an HTTP/3 connection error of type H3_GENERAL_PROTOCOL_ERROR. Intermediaries MUST ignore any HTTP/3 Datagram fields after the Quarter Stream ID.

Endpoints parse both the Quarter Stream ID field and the Context ID field in order to associate the QUIC DATAGRAM frame with a stream and context within that stream. If an endpoint receives a QUIC DATAGRAM frame whose payload is too short to allow parsing the Quarter Stream ID field, the endpoint MUST treat it as an HTTP/3 connection error of type H3_GENERAL_PROTOCOL_ERROR. If an endpoint receives a QUIC DATAGRAM frame whose payload is long enough to allow parsing the Quarter Stream ID field but too short to allow parsing the Context ID field, the endpoint MUST abruptly terminate the corresponding stream with a stream error of type H3_GENERAL_PROTOCOL_ERROR.

If a DATAGRAM frame is received and its Quarter Stream ID maps to a stream that has already been closed, the receiver MUST silently drop that frame. If a DATAGRAM frame is received and its Quarter Stream ID maps to a stream that has not yet been created, the receiver SHALL either drop that frame silently or buffer it temporarily while awaiting the creation of the corresponding stream.

4. CAPSULE HTTP/3 Frame Definition

CAPSULE allows reliably sending request-related information end-to-end, even in the presence of HTTP intermediaries.

CAPSULE is an HTTP/3 Frame (as opposed to a QUIC frame) which SHALL only be sent in client-initiated bidirectional streams. Intermediaries MUST forward all received CAPSULE frames in their unmodified entirety on the same stream where it would forward DATA frames. Intermediaries MUST NOT send any CAPSULE frames other than the ones it is forwarding.

This specification of CAPSULE currently uses HTTP/3 frame type 0xffcab5. If this document is approved, a lower number will be requested from IANA.

```
CAPSULE HTTP/3 Frame {
  Type (i) = 0xffcab5,
  Length (i),
  Capsule Type (i),
  Capsule Data (..),
}
```

Figure 2: CAPSULE HTTP/3 Frame Format

The Type and Length fields follows the definition of HTTP/3 frames from [H3]. The payload consists of:

Capsule Type: The type of this capsule.

Capsule Data: Data whose semantics depends on the Capsule Type.

Endpoints which receive a Capsule with an unknown Capsule Type MUST silently drop that Capsule. Intermediaries MUST forward Capsules, even if they do not know the Capsule Type or cannot parse the Capsule Data.

4.1. The REGISTER_DATAGRAM_CONTEXT Capsule

The REGISTER_DATAGRAM_CONTEXT capsule (type=0x00) allows an endpoint to inform its peer of the encoding and semantics of datagrams associated with a given context ID. Its Capsule Data field consists of:

```
REGISTER_DATAGRAM_CONTEXT Capsule {
  Context ID (i),
  Extension String (..),
}
```

Figure 3: REGISTER_DATAGRAM_CONTEXT Capsule Format

Context ID: The context ID to register.

Extension String: A string of comma-separated key-value pairs to enable extensibility. Keys are registered with IANA, see Section 8.4.

The ABNF for the Extension String field is as follows (using syntax from Section 3.2.6 of [RFC7230]):

```
extension-string = [ ext-member *( "," ext-member ) ]
ext-member       = ext-member-key "=" ext-member-value
ext-member-key   = token
ext-member-value = token
```

Note that these registrations are unilateral and bidirectional: the sender of the frame unilaterally defines the semantics it will apply to the datagrams it sends and receives using this context ID. Once a context ID is registered, it can be used in both directions.

Endpoints MUST NOT send DATAGRAM frames using a Context ID until they have either sent or received a REGISTER_DATAGRAM_CONTEXT Capsule with the same Context ID. However, due to reordering, an endpoint that receives a DATAGRAM frame with an unknown Context ID

MUST NOT treat it as an error, it SHALL instead drop the DATAGRAM frame silently, or buffer it temporarily while awaiting the corresponding REGISTER_DATAGRAM_CONTEXT Capsule.

Endpoints MUST NOT register the same Context ID twice on the same stream. This also applies to Context IDs that have been closed using a CLOSE_DATAGRAM_CONTEXT capsule. Clients MUST NOT register server-initiated Context IDs and servers MUST NOT register client-initiated Context IDs. If an endpoint receives a REGISTER_DATAGRAM_CONTEXT capsule that violates one or more of these requirements, the endpoint MUST abruptly terminate the corresponding stream with a stream error of type H3_GENERAL_PROTOCOL_ERROR.

4.2. The CLOSE_DATAGRAM_CONTEXT Capsule

The CLOSE_DATAGRAM_CONTEXT capsule (type=0x01) allows an endpoint to inform its peer that it will no longer send or parse received datagrams associated with a given context ID. Its Capsule Data field consists of:

```
CLOSE_DATAGRAM_CONTEXT Capsule {
  Context ID (i),
  Extension String (..),
}
```

Figure 4: CLOSE_DATAGRAM_CONTEXT Capsule Format

Context ID: The context ID to close.

Extension String: A string of comma-separated key-value pairs to enable extensibility, see the definition of the same field in Section 4.1 for details.

Note that this close is unilateral and bidirectional: the sender of the frame unilaterally informs its peer of the closure. Endpoints can use CLOSE_DATAGRAM_CONTEXT capsules to close a context that was initially registered by either themselves, or by their peer. Endpoints MAY use the CLOSE_DATAGRAM_CONTEXT capsule to immediately reject a context that was just registered using a REGISTER_DATAGRAM_CONTEXT capsule if they find its Extension String to be unacceptable.

After an endpoint has either sent or received a CLOSE_DATAGRAM_CONTEXT frame, it MUST NOT send any DATAGRAM frames with that Context ID. However, due to reordering, an endpoint that receives a DATAGRAM frame with a closed Context ID MUST NOT treat it as an error, it SHALL instead drop the DATAGRAM frame silently.

Endpoints MUST NOT close a Context ID that was not previously registered. Endpoints MUST NOT close a Context ID that has already been closed. If an endpoint receives a CLOSE_DATAGRAM_CONTEXT capsule that violates one or more of these requirements, the endpoint MUST abruptly terminate the corresponding stream with a stream error of type H3_GENERAL_PROTOCOL_ERROR.

4.3. The DATAGRAM Capsule

The DATAGRAM capsule (type=0x02) allows an endpoint to send a datagram frame over an HTTP stream. This is particularly useful when using a version of HTTP that does not support QUIC DATAGRAM frames. Its Capsule Data field consists of:

```
DATAGRAM Capsule {
  Context ID (i),
  HTTP/3 Datagram Payload (..),
}
```

Figure 5: DATAGRAM Capsule Format

Context ID: A variable-length integer indicating the context ID of the datagram (see Section 2.1).

HTTP/3 Datagram Payload: The payload of the datagram, whose semantics are defined by individual applications. Note that this field can be empty.

Datagrams sent using the DATAGRAM Capsule have the exact same semantics as datagrams sent in QUIC DATAGRAM frames.

5. 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.

An 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 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.

6. HTTP/1.x and HTTP/2 Support

We can provide DATAGRAM support in HTTP/2 by defining the CAPSULE frame in HTTP/2.

We can provide DATAGRAM support in HTTP/1.x by defining its data stream format to a sequence of length-value capsules.

TODO: Refactor this document into "HTTP Datagrams" with definitions for HTTP/1.x, HTTP/2, and HTTP/3.

7. Security Considerations

Since this feature requires sending an HTTP/3 Settings parameter, it "sticks out". In other words, probing clients can learn whether a server supports this feature. Implementations that support this feature SHOULD always send this Settings parameter to avoid leaking the fact that there are applications using HTTP/3 datagrams enabled on this endpoint.

8. IANA Considerations

8.1. HTTP/3 CAPSULE Frame

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

Frame Type	Value	Specification
CAPSULE	0xffcab5	This Document

8.2. HTTP SETTINGS Parameter

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

Setting Name	Value	Specification	Default
H3_DATAGRAM	0xffd276	This Document	0

8.3. Capsule Types

This document establishes a registry for HTTP/3 frame 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 4) is a 62bit 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 entries:

Capsule Type	Value	Specification
REGISTER_DATAGRAM_CONTEXT	0x00	This Document
CLOSE_DATAGRAM_CONTEXT	0x01	This Document
DATAGRAM	0x02	This Document

8.4. Context Extension Keys

REGISTER_DATAGRAM_CONTEXT capsules carry key-value pairs, see Section 4.1. This document will request IANA to create an "HTTP Datagram Context Extension Keys" registry. Registrations in this registry MUST include the following fields:

Key: The key (see Section 4.1). Keys MUST be valid tokens as defined in Section 3.2.6 of [RFC7230].

Description: A brief description of the key semantics, which MAY be a summary if a specification reference is provided.

Reference:
An optional reference to a specification for the parameter. 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 Key. This registry is initially empty.

9. 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-02, 16 February 2021, <https://tools.ietf.org/html/draft-ietf-quic-datagram-02>.

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

[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. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", Work in Progress, Internet-Draft, draft-ietf-quic-transport-34, 14 January 2021, <https://tools.ietf.org/html/draft-ietf-quic-transport-34>.

[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>.

[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/rfc/rfc7230>.

[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>.

Appendix A. Examples

A.1. CONNECT-UDP

```
Client                                             Server

STREAM(44): HEADERS             -------->
  :method = CONNECT-UDP
  :scheme = https
  :path = /
  :authority = target.example.org:443

STREAM(44): CAPSULE             -------->
  Capsule Type = REGISTER_DATAGRAM_CONTEXT
  Context ID = 0
  Extension String = ""

DATAGRAM                        -------->
  Quarter Stream ID = 11
  Context ID = 0
  Payload = Encapsulated UDP Payload

           <--------  STREAM(44): HEADERS
                        :status = 200

/* Wait for target server to respond to UDP packet. */

           <--------  DATAGRAM
                        Quarter Stream ID = 11
                        Context ID = 0
                        Payload = Encapsulated UDP Payload
```

A.2. CONNECT-UDP with Timestamp Extension

```
Client                                             Server

STREAM(44): HEADERS            -------->
  :method = CONNECT-UDP
  :scheme = https
  :path = /
  :authority = target.example.org:443

STREAM(44): CAPSULE            -------->
  Capsule Type = REGISTER_DATAGRAM_CONTEXT
  Context ID = 0
  Extension String = ""

DATAGRAM                       -------->
  Quarter Stream ID = 11
  Context ID = 0
  Payload = Encapsulated UDP Payload

           <--------  STREAM(44): HEADERS
                        :status = 200

/* Wait for target server to respond to UDP packet. */

           <--------  DATAGRAM
                        Quarter Stream ID = 11
                        Context ID = 0
                        Payload = Encapsulated UDP Payload


STREAM(44): CAPSULE            -------->
  Capsule Type = REGISTER_DATAGRAM_CONTEXT
  Context ID = 2
  Extension String = "timestamp"

DATAGRAM                       -------->
  Quarter Stream ID = 11
  Context ID = 2
  Payload = Encapsulated UDP Payload With Timestamp
```

A.3. CONNECT-IP with IP compression

```
Client                                             Server

STREAM(44): HEADERS            -------->
  :method = CONNECT-IP
  :scheme = https
  :path = /
  :authority = proxy.example.org:443

           <--------  STREAM(44): HEADERS
                        :status = 200

/* Exchange CONNECT-IP configuration information. */

STREAM(44): CAPSULE             -------->
  Capsule Type = REGISTER_DATAGRAM_CONTEXT
  Context ID = 0
  Extension String = ""

DATAGRAM                       -------->
  Quarter Stream ID = 11
  Context ID = 0
  Payload = Encapsulated IP Packet

/* Endpoint happily exchange encapsulated IP packets */
/* using Quarter Stream ID 11 and Context ID 0.      */

DATAGRAM                       -------->
  Quarter Stream ID = 11
  Context ID = 0
  Payload = Encapsulated IP Packet

/* After performing some analysis on traffic patterns, */
/* the client decides it wants to compress a 5-tuple.  */


STREAM(44): CAPSULE             -------->
  Capsule Type = REGISTER_DATAGRAM_CONTEXT
  Context ID = 2
  Extension String = "ip=192.0.2.42,port=443"

DATAGRAM                       -------->
  Quarter Stream ID = 11
  Context ID = 2
  Payload = Compressed IP Packet
```

A.4. WebTransport

```
Client                                             Server

STREAM(44): HEADERS            -------->
  :method = CONNECT
  :scheme = https
  :method = webtransport
  :path = /hello
  :authority = webtransport.example.org:443
  Origin = https://www.example.org:443

STREAM(44): CAPSULE             -------->
  Capsule Type = REGISTER_DATAGRAM_CONTEXT
  Context ID = 0
  Extension String = ""

           <--------  STREAM(44): HEADERS
                        :status = 200

/* Both endpoints can now send WebTransport datagrams. */
```

Acknowledgments

The DATAGRAM context identifier was previously part of the 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.

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