Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 9 July 2021

D. Schinazi
Google LLC

5 January 2021

The CONNECT-UDP HTTP Method
draft-ietf-masque-connect-udp-03

Abstract

This document describes the CONNECT-UDP HTTP method. CONNECT-UDP is similar to the HTTP CONNECT method, but it uses UDP instead of TCP.

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-connect-udp.

Discussion Venues

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

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

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 9 July 2021.

Copyright Notice

Table of Contents

1. Introduction

This document describes the CONNECT-UDP HTTP method. CONNECT-UDP is similar to the HTTP CONNECT method (see section 4.3.6 of [RFC7231]), but it uses UDP [UDP] instead of TCP [TCP].

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-connect-udp.

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.

In this document, we use the term "proxy" to refer to the HTTP server that opens the UDP socket and responds to the CONNECT-UDP request. If there are HTTP intermediaries (as defined in Section 2.3 of [RFC7230]) between the client and the proxy, those are referred to as "intermediaries" in this document.

2. Supported HTTP Versions

The CONNECT-UDP method is defined for all versions of HTTP. When the HTTP version used runs over QUIC [QUIC], UDP payloads can be sent over QUIC DATAGRAM frames [DGRAM]. Otherwise they are sent on the stream where the CONNECT-UDP request was made. Note that, when the HTTP version in use does not support multiplexing streams (such as HTTP/1.1), then any reference to "stream" in this document is meant to represent the entire connection.

3. The CONNECT-UDP Method

The CONNECT-UDP method requests that the recipient establish a tunnel over a single HTTP stream to the destination origin server identified by the request-target and, if successful, thereafter restrict its behavior to blind forwarding of packets, in both directions, until the tunnel is closed. Tunnels are commonly used to create an end-to-end virtual connection, which can then be secured using QUIC or another protocol running over UDP.

The request-target of a CONNECT-UDP request is a URI [RFC3986] which uses the "masque" scheme and an immutable path of "/". For example:

```
CONNECT-UDP masque://target.example.com:443/ HTTP/1.1
Host: target.example.com:443
```

When using HTTP/2 [H2] or later, CONNECT-UDP requests use HTTP pseudo-headers with the following requirements:

* The ":method" pseudo-header field is set to "CONNECT-UDP".

* The ":scheme" pseudo-header field is set to "masque".

* The ":path" pseudo-header field is set to "/".

* The ":authority" pseudo-header field contains the host and port to connect to (similar to the authority-form of the request-target of CONNECT requests; see [RFC7230], Section 5.3).

A CONNECT-UDP request that does not conform to these restrictions is malformed (see [H2], Section 8.1.2.6).

The recipient proxy establishes a tunnel by directly opening a UDP socket to the request-target. Any 2xx (Successful) response indicates that the proxy has opened a socket to the request-target and is willing to proxy UDP payloads. Any response other than a successful response indicates that the tunnel has not yet been formed.

A proxy MUST NOT send any Transfer-Encoding or Content-Length header fields in a 2xx (Successful) response to CONNECT-UDP. A client MUST treat a response to CONNECT-UDP containing any Content-Length or Transfer-Encoding header fields as malformed.

A payload within a CONNECT-UDP request message has no defined semantics; a CONNECT-UDP request with a non-empty payload is malformed. Note that the CONNECT-UDP stream is used to convey UDP packets, but they are not semantically part of the request or response themselves.

Responses to the CONNECT-UDP method are not cacheable.

4. Datagram Encoding of Proxied UDP Packets

When the HTTP connection supports HTTP/3 datagrams [H3DGRAM], UDP packets can be encoded using QUIC DATAGRAM frames. This support is ascertained by checking the received value of the H3_DATAGRAM SETTINGS Parameter.

If the client has both sent and received the H3_DATAGRAM SETTINGS Parameter with value 1 on this connection, it SHOULD attempt to use HTTP/3 datagrams. This is accomplished by requesting a datagram flow identifier from the flow identifier allocation service [H3DGRAM]. That service generates an even flow identifier, and the client sends it to the proxy by using the unnamed element in a "Datagram-Flow-Id" header; see [H3DGRAM]. A CONNECT-UDP request with an odd flow identifier is malformed.

The proxy that is creating the UDP socket to the destination responds to the CONNECT-UDP request with a 2xx (Successful) response, and indicates it supports datagram encoding by sending a "Datagram-Flow-Id" header with the same unnamed element from the "Datagram-Flow-Id" header it received. Once the client has received the "Datagram-Flow-Id" header on the successful response, it knows that it can use the HTTP/3 datagram encoding to send proxied UDP packets for this particular request. It then encodes the payload of UDP datagrams into the payload of HTTP/3 datagrams. If the CONNECT-UDP response does not carry the "Datagram-Flow-Id" header, then the datagram encoding is not available for this request. A CONNECT-UDP response that carries the "Datagram-Flow-Id" header but with a different unnamed flow identifier than the one sent on the request is malformed.

When the proxy processes a new CONNECT-UDP request, it MUST ensure that the unnamed datagram flow identifier is not equal to flow identifiers from other requests: if it is, the proxy MUST reject the request with a 4xx (Client Error) status code. Extensions MAY weaken or remove this requirement.

Clients MAY optimistically start sending proxied UDP packets before receiving the response to its CONNECT-UDP request, noting however that those may not be processed by the proxy if it responds to the CONNECT-UDP request with a failure or without echoing the "Datagram-Flow-Id" header, or if the datagrams arrive before the CONNECT-UDP request.

Note that a proxy can send the H3_DATAGRAM SETTINGS Parameter with a value of 1 while disabling datagrams on a particular request by not echoing the "Datagram-Flow-Id" header. If the proxy does this, it MUST NOT treat receipt of datagrams as an error, because the client could have sent them optimistically before receiving the response. In this scenario, the proxy MUST discard those datagrams.

Extensions to CONNECT-UDP MAY leverage named elements or parameters in the "Datagram-Flow-Id" header (named elements are defined in [H3DGRAM] and parameters are defined in Section 3.1.2 of [STRUCT-HDR]). Proxies MUST NOT echo named elements or parameters on the "Datagram-Flow-Id" header if they do not understand their semantics.

5. Stream Chunks

The bidirectional stream that the CONNECT-UDP request was sent on is a sequence of CONNECT-UDP Stream Chunks, which are defined as a sequence of type-length-value tuples using the following format (using the notation from the "Notational Conventions" section of [QUIC]):

```
CONNECT-UDP Stream {
  CONNECT-UDP Stream Chunk (..) ...,
}
```

Figure 1: CONNECT-UDP Stream Format

```
CONNECT-UDP Stream Chunk {
  CONNECT-UDP Stream Chunk Type (i),
  CONNECT-UDP Stream Chunk Length (i),
  CONNECT-UDP Stream Chunk Value (..),
}
```

Figure 2: CONNECT-UDP Stream Chunk Format

CONNECT-UDP Stream Chunk Type: A variable-length integer indicating the Type of the CONNECT-UDP Stream Chunk. Endpoints that receive a chunk with an unknown CONNECT-UDP Stream Chunk Type MUST silently skip over that chunk.

CONNECT-UDP Stream Chunk Length: The length of the CONNECT-UDP Stream Chunk Value field following this field. Note that this field can have a value of zero.

CONNECT-UDP Stream Chunk Value: The payload of this chunk. Its semantics are determined by the value of the CONNECT-UDP Stream Chunk Type field.

6. Stream Encoding of Proxied UDP Packets

CONNECT-UDP Stream Chunks can be used to convey UDP payloads, by using a CONNECT-UDP Stream Chunk Type of UDP_PACKET (value 0x00). The payload of UDP packets is encoded in its unmodified entirety in the CONNECT-UDP Stream Chunk Value field. This is necessary when the version of HTTP in use does not support QUIC DATAGRAM frames, but MAY also be used when datagrams are supported. Note that empty UDP payloads are allowed.

7. Proxy Handling

Unlike TCP, UDP is connection-less. The proxy that opens the UDP socket has no way of knowing whether the destination is reachable. Therefore it needs to respond to the CONNECT-UDP request without waiting for a TCP SYN-ACK.

Proxies can use connected UDP sockets if their operating system supports them, as that allows the proxy to rely on the kernel to only send it UDP packets that match the correct 5-tuple. If the proxy uses a non-connected socket, it MUST validate the IP source address and UDP source port on received packets to ensure they match the client's CONNECT-UDP request. Packets that do not match MUST be discarded by the proxy.

The lifetime of the socket is tied to the CONNECT-UDP stream. The proxy MUST keep the socket open while the CONNECT-UDP stream is open. Proxies MAY choose to close sockets due to a period of inactivity, but they MUST close the CONNECT-UDP stream before closing the socket.

8. HTTP Intermediaries

HTTP/3 DATAGRAM flow identifiers are specific to a given HTTP/3 connection. However, in some cases, an HTTP request may travel across multiple HTTP connections if there are HTTP intermediaries involved; see Section 2.3 of [RFC7230].

Intermediaries that support both CONNECT-UDP and HTTP/3 datagrams MUST negotiate flow identifiers separately on the client-facing and server-facing connections. This is accomplished by having the intermediary parse the unnamed element of the "Datagram-Flow-Id" header on all CONNECT-UDP requests it receives, and sending the same unnamed element in the "Datagram-Flow-Id" header on the response. The intermediary then ascertains whether it can use datagrams on the server-facing connection. If they are supported (as indicated by the H3_DATAGRAM SETTINGS parameter), the intermediary uses its own flow identifier allocation service to allocate a flow identifier for the server-facing connection, and waits for the server's reply to see if the server sent the "Datagram-Flow-Id" header on the response. The intermediary then translates datagrams between the two connections by using the flow identifier specific to that connection. An intermediary MAY also choose to use datagrams on only one of the two connections, and translate between datagrams and streams.

Intermediaries MUST NOT echo nor forward named elements or parameters on the "Datagram-Flow-Id" header if they do not understand their semantics. Extensions to CONNECT-UDP that leverage named elements or parameters in the "Datagram-Flow-Id" header MUST specify how they are handled by intermediaries.

9. Performance Considerations

Proxies SHOULD strive to avoid increasing burstiness of UDP traffic: they SHOULD NOT queue packets in order to increase batching.

When the protocol running over UDP that is being proxied uses congestion control (e.g., [QUIC]), the proxied traffic will incur at least two nested congestion controllers. This can reduce performance but the underlying HTTP connection MUST NOT disable congestion control unless it has an out-of-band way of knowing with absolute certainty that the inner traffic is congestion-controlled.

When the protocol running over UDP that is being proxied uses loss recovery (e.g., [QUIC]), and the underlying HTTP connection runs over TCP, the proxied traffic will incur at least two nested loss recovery mechanisms. This can reduce performance as both can sometimes independently retransmit the same data. To avoid this, HTTP/3 datagrams SHOULD be used.

10. Security Considerations

There are significant risks in allowing arbitrary clients to establish a tunnel to arbitrary servers, as that could allow bad actors to send traffic and have it attributed to the proxy. Proxies that support CONNECT-UDP SHOULD restrict its use to authenticated users.

Because the CONNECT method creates a TCP connection to the target, the target has to indicate its willingness to accept TCP connections by responding with a TCP SYN-ACK before the proxy can send it application data. UDP doesn't have this property, so a CONNECT-UDP proxy could send more data to an unwilling target than a CONNECT proxy. However, in practice denial of service attacks target open TCP ports so the TCP SYN-ACK does not offer much protection in real scenarios. Proxies MUST NOT introspect the contents of UDP payloads as that would lead to ossification of UDP-based protocols by proxies.

11. IANA Considerations

11.1. HTTP Method

This document will request IANA to register "CONNECT-UDP" in the HTTP Method Registry (IETF review) maintained at <https://www.iana.org/assignments/http-methods>.

Method Name	Safe	Idempotent	Reference
CONNECT-UDP	no	no	This document

11.2. URI Scheme Registration

This document will request IANA to register the URI scheme "masque".

The syntax definition below uses Augmented Backus-Naur Form (ABNF) [RFC5234]. The definitions of "host" and "port" are adopted from [RFC3986]. The syntax of a MASQUE URI is:

```
masque-URI = "masque:" "//" host ":" port "/"
```

The "host" and "port" component MUST NOT be empty, and the "port" component MUST NOT be 0.

11.3. Stream Chunk Type Registration

This document will request IANA to create a "CONNECT-UDP Stream Chunk Type" registry. This registry governs a 62-bit space, and follows the registration policy for QUIC registries as defined in [QUIC]. In addition to the fields required by the QUIC policy, registrations in this registry MUST include the following fields:

Type: A short mnemonic for the type.

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

The initial contents of this registry are:

Value	Type	Description	Reference
0x00	UDP_PACKET	Payload of UDP packet	This document

Each value of the format "37 * N + 23" for integer values of N (that is, 23, 60, 97, ...) are reserved; these values MUST NOT be assigned by IANA and MUST NOT appear in the listing of assigned values.

12. 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-01, 24 August 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-datagram-01.txt>.

[H2] 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>.

[H3DGRAM] Schinazi, D. and L. Pardue, "Using QUIC Datagrams with HTTP/3", Work in Progress, Internet-Draft, draft-schinazi-masque-h3-datagram-02, 14 December 2020, <http://www.ietf.org/internet-drafts/draft-schinazi-masque-h3-datagram-02.txt>.

[QUIC] Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", Work in Progress, Internet-Draft, draft-ietf-quic-transport-33, 13 December 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-33.txt>.

[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>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[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>.

[STRUCT-HDR] Nottingham, M. and P. Kamp, "Structured Field Values for HTTP", Work in Progress, Internet-Draft, draft-ietf-httpbis-header-structure-19, 3 June 2020, <http://www.ietf.org/internet-drafts/draft-ietf-httpbis-header-structure-19.txt>.

[TCP] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[UDP] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/info/rfc768>.

Acknowledgments

This document is a product of the MASQUE Working Group, and the author thanks all MASQUE enthusiasts for their contibutions. This proposal was inspired directly or indirectly by prior work from many people. In particular, the author would like to thank Eric Rescorla for suggesting to use an HTTP method to proxy UDP. Thanks to Lucas Pardue for their inputs on this document.

Author's Address

David Schinazi
Google LLC
1600 Amphitheatre Parkway
Mountain View, California 94043,
United States of America

Email: dschinazi.ietf@gmail.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 2 August 2021

D. Schinazi
Google LLC
L. Pardue
Cloudflare

29 January 2021

Using QUIC Datagrams with HTTP/3
draft-ietf-masque-h3-datagram-00

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 defines how to use QUIC DATAGRAM frames when the application protocol running over QUIC is HTTP/3 by adding an identifier at the start of the frame payload. This allows HTTP messages to convey related information using unreliable DATAGRAM frames, ensuring those frames are properly associated with an HTTP message.

Discussion of this work is encouraged to happen on the MASQUE IETF mailing list (masque@ietf.org (mailto: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

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 2 August 2021.

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 defines how to use QUIC DATAGRAM frames when the application protocol running over QUIC is HTTP/3 [H3] by adding an identifier at the start of the frame payload. This allows HTTP messages to convey related information using unreliable DATAGRAM frames, ensuring those frames are properly associated with an HTTP message.

This design mimics the use of Stream Types in HTTP/3, which provide a demultiplexing identifier at the start of each unidirectional stream.

Discussion of this work is encouraged to happen on the MASQUE IETF mailing list (masque@ietf.org (mailto: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. Flow Identifiers

Flow identifiers represent bidirectional flows of datagrams within a single QUIC connection. These are conceptually similar to streams in the sense that they allow multiplexing of application data. Flows lack any of the ordering or reliability guarantees of streams.

Beyond this, a sender SHOULD ensure that DATAGRAM frames within a single flow are transmitted in order relative to one another. If multiple DATAGRAM frames can be packed into a single QUIC packet, the sender SHOULD group them by flow identifier to promote fate-sharing within a specific flow and improve the ability to process batches of datagram messages efficiently on the receiver.

3. Flow Identifier Allocation

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

Even-numbered flow identifiers are client-initiated, while odd-numbered flow identifiers are server-initiated. This means that an HTTP/3 client implementation of the flow identifier allocation service MUST only provide even-numbered identifiers, while a server implementation MUST only provide odd-numbered identifiers. Note that, once allocated, any flow identifier can be used by both client and server - only allocation carries separate namespaces to avoid requiring synchronization.

The flow allocation service SHOULD also provide a mechanism for applications to indicate they have completed their usage of a flow identifier and will no longer be using that flow identifier, this process is called "retiring" a flow identifier. Applications MUST NOT retire a flow identifier until after they have received confirmation that the peer has also stopped using that flow identifier. The flow identifier allocation service MAY reuse previously retired flow identifiers once they have ascertained that there are no packets with DATAGRAM frames using that flow identifier still in flight. Reusing flow identifiers can improve performance by transmitting the flow identifier using a shorter variable-length integer encoding.

4. 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 Frame {
  Flow Identifier (i),
  HTTP/3 Datagram Payload (..),
}
```

Figure 1: HTTP/3 DATAGRAM Frame Format

Flow Identifier: A variable-length integer indicating the Flow Identifier of the datagram (see Section 2).

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

Endpoints MUST treat receipt of a DATAGRAM frame whose payload is too short to parse the flow identifier as an HTTP/3 connection error of type H3_GENERAL_PROTOCOL_ERROR.

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. Datagram-Flow-Id Header Field Definition

"Datagram-Flow-Id" is a List Structured Field [STRUCT-FIELD], whose members MUST all be Items of type Integer. Its ABNF is:

```
Datagram-Flow-Id = sf-list
```

The "Datagram-Flow-Id" header field is used to associate one or more datagram flow identifiers with an HTTP message. As a simple example using a single identifier, the definition of an HTTP method could instruct the client to use its flow identifier allocation service to allocate a new flow identifier, and then the client will add the "Datagram-Flow-Id" header field to its request to communicate that value to the server. In this example, the resulting header field could look like:

```
Datagram-Flow-Id = 2
```

List members are flow identifier elements, which can be named or unnamed. One element in the list is allowed to be unnamed, but all but one elements MUST carry a name. The name of an element is encoded in the key of the first parameter of that element (parameters are defined in Section 3.1.2 of [STRUCT-FIELD]). Each name MUST NOT appear more than once in the list. The value of the first parameter of each named element (whose corresponding key conveys the element name) MUST be of type Boolean and equal to true. The value of the first parameter of the unnamed element MUST NOT be of type Boolean. The ordering of the list does not carry any semantics. For example, an HTTP method that wishes to use four datagram flow identifiers for the lifetime of its request stream could look like this:

```
Datagram-Flow-Id = 42, 44; ecn-ect0, 46; ecn-ect1, 48; ecn-ce
```

In this example, 42 is the unnamed flow identifier, 44 represents the name "ecn-ect0", 46 represents "ecn-ect1", and 48 represents "ecn-ce". Note that, since the list ordering does not carry semantics, this example can be equivalently encoded as:

```
Datagram-Flow-Id = 44; ecn-ect0, 42, 48; ecn-ce, 46; ecn-ect1
```

Even if a sender attempts to communicate the meaning of a flow identifier before it uses it in an HTTP/3 datagram, it is possible that its peer will receive an HTTP/3 datagram with a flow identifier that it does not know as it has not yet received the corresponding "Datagram-Flow-Id" header field. (For example, this could happen if the QUIC STREAM frame that contains the "Datagram-Flow-Id" header field is reordered and arrives afer the DATAGRAM frame.) Endpoints MUST NOT treat that scenario as an error; they MUST either silently discard the datagram or buffer it until they receive the "Datagram-Flow-Id" header field.

Distinct HTTP requests MAY refer to the same flow identifier in their respective "Datagram-Flow-Id" header fields.

Note that integer structured fields can only encode values up to 10^15-1, therefore the maximum possible value of an element of the "Datagram-Flow-Id" header field is lower then the theoretical maximum value of a flow identifier which is 2^62-1 due to the QUIC variable length integer encoding. If the flow identifier allocation service of an endpoint runs out of values lower than 10^15-1, the endpoint MUST fail the flow identifier allocation. An HTTP message that carries a "Datagram-Flow-Id" header field with a flow identifier value above 10^15-1 is malformed (see Section 8.1.2.6 of [H2]).

7. HTTP Intermediaries

HTTP/3 DATAGRAM flow identifiers are specific to a given HTTP/3 connection. However, in some cases, an HTTP request may travel across multiple HTTP connections if there are HTTP intermediaries involved; see Section 2.3 of [RFC7230].

If an intermediary has sent the H3_DATAGRAM SETTINGS parameter with a value of 1 on its client-facing connection, it MUST inspect all HTTP requests from that connection and check for the presence of the "Datagram-Flow-Id" header field. If the HTTP method of the request is not supported by the intermediary, it MUST remove the "Datagram-Flow-Id" header field before forwarding the request. If the intermediary supports the method, it MUST either remove the header field or adhere to the requirements leveraged by that method on intermediaries.

If an intermediary has sent the H3_DATAGRAM SETTINGS parameter with a value of 1 on its server-facing connection, it MUST inspect all HTTP responses from that connection and check for the presence of the "Datagram-Flow-Id" header field. If the HTTP method of the request is not supported by the intermediary, it MUST remove the "Datagram-Flow-Id" header field before forwarding the response. If the intermediary supports the method, it MUST either remove the header field or adhere to the requirements leveraged by that method on intermediaries.

If an intermediary processes distinct HTTP requests that refer to the same flow identifier in their respective "Datagram-Flow-Id" header fields, it MUST ensure that those requests are routed to the same backend.

8. Security Considerations

This document does not have additional security considerations beyond those defined in [QUIC] and [DGRAM].

9. IANA Considerations

9.1. 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	0x276	This Document	0

9.2. HTTP Header Field

This document will request IANA to register the "Datagram-Flow-Id" header field in the "Permanent Message Header Field Names" registry maintained at <https://www.iana.org/assignments/message-headers>.

Header Field Name	Protocol	Status	Reference
Datagram-Flow-Id	http	std	This document

9.3. Flow Identifier Parameters

This document will request IANA to create an "HTTP Datagram Flow Identifier Parameters" registry. Registrations in this registry MUST include the following fields:

Key: The key of a parameter that is associated with a datagram flow identifier list member (see Section 6). Keys MUST be valid structured field parameter keys (see Section 3.1.2 of [STRUCT-FIELD]).

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

Is Name: This field MUST be either Yes or No. Yes indicates that this parameter is the name of a named element (see Section 6). No indicates that it is a parameter that is not a name.

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.

10. 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-01, 24 August 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-datagram-01.txt>.

[H2] 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>.

[H3] Bishop, M., "Hypertext Transfer Protocol Version 3 (HTTP/3)", Work in Progress, Internet-Draft, draft-ietf-quic-http-33, 15 December 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-http-33.txt>.

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

[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>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[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>.

[STRUCT-FIELD] Nottingham, M. and P. Kamp, "Structured Field Values for HTTP", Work in Progress, Internet-Draft, draft-ietf-httpbis-header-structure-19, 3 June 2020, <http://www.ietf.org/internet-drafts/draft-ietf-httpbis-header-structure-19.txt>.

Acknowledgments

The DATAGRAM flow identifier was previously part of the DATAGRAM frame definition itself, the author would like to acknowledge the authors of that document and the members of the IETF MASQUE working group for their suggestions. Additionally, the author would like to thank Martin Thomson for suggesting the use of an HTTP/3 SETTINGS parameter.

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

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 1 November 2021

A. Chernyakhovsky
D. McCall
D. Schinazi
Google LLC

30 April 2021

Requirements for a MASQUE Protocol to Proxy IP Traffic
draft-ietf-masque-ip-proxy-reqs-02

Abstract

There is interest among MASQUE working group participants in designing a protocol that can proxy IP traffic over HTTP. This document describes the set of requirements for such a protocol.

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-ip-proxy-reqs.

Discussion Venues

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

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

Status of This Memo

Table of Contents

1. Introduction

There exist several IETF standards for proxying IP in a way that is authenticated and confidential, such as IKEv2/IPsec [IKEV2]. However, those are distinguishable from common Internet traffic and often blocked. Additionally, large server deployments have expressed interest in using a VPN solution that leverages existing security protocols such as QUIC [QUIC] or TLS [TLS] to avoid adding another protocol to their security posture.

This document describes the set of requirements for a protocol that can proxy IP traffic over HTTP. The requirements outlined below are similar to the considerations made in designing the CONNECT-UDP method [CONNECT-UDP], additionally including IP-specific requirements, such as a means of negotiating the routes that should be advertised on either end of the connection.

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-ip-proxy-reqs.

1.1. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.2. Definitions

* Data Transport: The mechanism responsible for transmitting IP packets over HTTP. This can involve streams or datagrams.

* IP Session: An association between client and server whereby both agree to proxy IP traffic given certain configuration properties. This is similar to a Child Security Association in IKEv2 terminology. An IP Session uses Data Transports to transmit packets.

2. Use Cases

There are multiple reasons to deploy an IP proxying protocol. This section discusses some examples of use cases that MUST be supported by the protocol. Note that while the protocol needs to support these use cases, the protocol elements that allow them may be optional.

2.1. Consumer VPN

Consumer VPNs refer to network applications that allow a user to hide some properties of their traffic from some network observers. In particular, it can hide the identity of servers the client is connecting to from the client's network provider, and can hide the client's IP address (and derived geographical information) from the servers they are communicating with. Note that this hidden information is now available to the VPN service provider, so is only beneficial for clients who trust the VPN service provider more than other entities.

2.2. Point to Point Connectivity

Point-to-point connectivity creates a private, encrypted and authenticated network between two IP addresses. This is useful, for example, with container networking to provide a virtual (overlay) network with addressing separate from the physical transport. An example of this is Wireguard.

2.3. Point to Network Connectivity

Point-to-Network connectivity is the more traditional remote-access "VPN" use case, frequently used when a user needs to connect to a different network (such as an enterprise network) for access to resources that are not exposed to the public Internet.

2.4. Network to Network Connectivity

Network-to-Network connectivity is also called a site-to-site VPN. Similar to the point-to-network use case, the goal is to connect two networks that are not exposed publicly. The site-to-site aspects make this transparent to the user; the entire networks are connected to each other and route packets transparently without a VPN client installed on the user's device. This style of connectivity can also be used to connect devices that cannot run VPN clients through to the network.

3. Requirements

This section lists requirements for a protocol that can proxy IP over an HTTP connection.

3.1. IP Session Establishment

The protocol will allow the client to request establishment of an IP Session, along with configuration options and one or more associated Data Transports. The server will have the ability to accept or deny the client's request.

3.2. Proxying of IP packets

The protocol will establish Data Transports, which will be able to forward IP packets. The Data Transports MUST be able to take IP datagrams input on one side and egress them unmodified in their entirety on the other side, although extensions may enable IP packets to be modified in transit. The protocol will support both IPv6 [IPV6] and IPv4 [IPV4].

3.3. Maximum Transmission Unit

The protocol will allow endpoints to inform each other of the Maximum Transmission Unit (MTU) they are willing to forward. This will allow avoiding IP fragmentation, especially as IPv6 does not allow IP fragmentation by nodes along the path.

3.4. IP Assignment

The client will be able to request to be assigned an IP address range, optionally specifying a preferred range. In response to that request, the server will either assign a range of its choosing to the client, or decline the request. For symmetry, the server may request assignment of an IP address range from the client, and the client will either assign a range or decline the request.

3.5. Route Negotiation

At any point in an IP Session (not limited to its initial negotiation), the protocol will allow both client and server to inform its peer that it can route a set of IP prefixes. Both endpoints can also request a route to a given prefix, and the peer can choose to provide that route or not. This can be used to inform peers of a default route for all prefixes.

Note that if an endpoint provides its peer with a route, the peer is in no way obligated to route its traffic through the endpoint.

3.6. Identity

When negotiating the creation of an IP Session, the protocol will allow both endpoints to exchange an identifier. As examples, the identity could be a user name, an email address, a token, or a fully-qualified domain name. Note that this requirement does not cover authenticating the identifier.

3.7. Transport Security

The protocol MUST be run over a protocol that provides mutual authentication, confidentiality and integrity. Using QUIC or TLS would meet this requirement.

3.8. Flow Control

The protocol will allow the ability to proxy IP packets without flow control, at least when HTTP/3 is in use. QUIC DATAGRAM frames are not flow controlled and would meet this requirement. The document defining the protocol will provide guidance on how best to use flow control to improve IP Session performance.

3.9. Indistinguishability

A passive network observer not participating in the encrypted connection should not be able to distinguish IP proxying from regular encrypted HTTP Web traffic by only observing non-encrypted parts of the traffic. Specifically, any data sent unencrypted (such as headers, or parts of the handshake) should look like the same unencrypted data that would be present for Web traffic. Traffic analysis is out of scope for this requirement.

3.10. Support HTTP/2 and HTTP/3

The IP proxying protocol discussed in this document will run over HTTP. The protocol SHOULD strongly prefer to use HTTP/3 [H3] and SHOULD use the QUIC DATAGRAM frames [DGRAM] when available to improve performance. The protocol MUST also support HTTP/2 [H2] as a fallback when UDP is blocked on the network path. Proxying IP over HTTP/2 MAY result in lower performance than over HTTP/3.

3.11. Multiplexing

Since recent HTTP versions support concurrently running multiple requests over the same connection, the protocol SHOULD support multiple independent instances of IP proxying over a given HTTP connection.

3.12. Statefulness

The protocol should limit the amount of state a MASQUE client or server needs to operate. Keeping minimal state simplifies reconnection in the presence of failures and can facilitate extensibility.

4. Extensibility

The protocol will provide a mechanism by which clients and servers can add extension information to the exchange that establishes the IP Session. If the solution uses an HTTP request and response, this could be accomplished using HTTP headers.

Once the IP Session is established, the protocol will provide a mechanism that allows reliably exchanging extension messages in both directions at any point in the lifetime of the IP Session.

The subsections below list possible extensions that designers of the protocol will keep in mind to ensure it will be possible to design such extensions.

4.1. Authentication

Since the protocol will offer a way to convey identity, extensions will allow authenticating that identity, from both the client and server, during the establishment of the IP Session. For example, an extension could allow a client to offer an OAuth Access Token [OAUTH] when requesting an IP Session. As another example, another extension could allow an endpoint to demonstrate knowledge of a cryptographic secret.

4.2. Reliable Transmission of IP Packets

While it is desirable to transmit IP packets unreliably in most cases, an extension could provide a mechanism to allow forwarding some packets reliably. For example, when using HTTP/3, this can be accomplished by allowing Data Transports to run over both DATAGRAM and STREAM frames.

4.3. Configuration of Congestion and Flow Control

An extension will allow exchanging congestion and flow control parameters to improve performance. For example, an extension could disable congestion control for non-retransmitted Data Transports if it knows that the proxied traffic is itself congestion-controlled.

4.4. Data Transport Compression

While the core protocol Data Transports will transmit IP packets in their unmodified entirety, an extension can allow compressing these packets.

5. Non-requirements

This section discusses topics that are explicitly out of scope for the IP Proxying protocol. These topics MAY be handled by implementers or future extensions.

5.1. Addressing Architecture

This document only describes the requirements for a protocol that allows IP proxying. It does not discuss how the IPs assigned are determined, managed, or translated. While these details are important for producing a functional system, they do not need to be handled by the protocol beyond the ability to convey those assignments.

Similarly, "ownership" of an IP range is out of scope. If an endpoint communicates to its peer that it can allocate addresses from a range, or route traffic to a range, the peer has no obligation to trust that information. Whether or not to trust this information is left to individual implementations and extensions: the protocol will be extensible enough to allow the development of extensions that assist in assessing this trust.

5.2. Translation

Some servers may wish to perform Network Address Translation (NAT) or any other modification to packets they forward. Doing so is out of scope for the proxying protocol. In particular, the ability to discover the presence of a NAT, negotiate NAT bindings, or check connectivity through a NAT is explicitly out of scope and left to future extensions.

Servers that do not perform NAT will commonly forward packets similarly to how a traditional IP router would, but the specific of that are considered out of scope. In particular, decrementing the Hop Limit (or TTL) field of the IP header is out of scope for MASQUE and expected to be performed by a router behind the MASQUE server, or collocated with it.

5.3. IP Packet Extraction

How packets are forwarded between the IP proxying connection and the physical network is out of scope. For example, this can be accomplished on some operating systems using a TUN interface. How this is done is deliberately not specified and will be left to individual implementations.

6. Security Considerations

This document only discusses requirements on a protocol that allows IP proxying. That protocol will need to document its security considerations.

7. IANA Considerations

This document requests no actions from IANA.

Acknowledgments

The authors would like to thank participants of the MASQUE working group for their feedback.

References

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>.

[H2] 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/rfc/rfc7540>.

[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>.

[IPV4] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <https://www.rfc-editor.org/rfc/rfc791>.

[IPV6] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/rfc/rfc8200>.

[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>.

[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>.

[TLS] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/rfc/rfc8446>.

Informative References

[CONNECT-UDP] Schinazi, D., "The CONNECT-UDP HTTP Method", Work in Progress, Internet-Draft, draft-ietf-masque-connect-udp-03, 5 January 2021, <https://tools.ietf.org/html/draft-ietf-masque-connect-udp-03>.

[IKEV2] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <https://www.rfc-editor.org/rfc/rfc7296>.

[OAUTH] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/rfc/rfc6749>.

Authors' Addresses

Alex Chernyakhovsky
Google LLC
1600 Amphitheatre Parkway
Mountain View, California 94043,
United States of America

Email: achernya@google.com

Dallas McCall
Google LLC
1600 Amphitheatre Parkway
Mountain View, California 94043,
United States of America

Email: dallasmccall@google.com

David Schinazi
Google LLC
1600 Amphitheatre Parkway
Mountain View, California 94043,
United States of America

Email: dschinazi.ietf@gmail.com

MASQUE
Internet-Draft
Intended status: Experimental
Expires: 15 October 2021

T. Pauly
Apple Inc.
D. Schinazi
Google LLC

13 April 2021

QUIC-Aware Proxying Using CONNECT-UDP
draft-pauly-masque-quic-proxy-01

Abstract

This document defines an extension to the CONNECT-UDP HTTP method that adds specific optimizations for QUIC connections that are proxied. This extension allows a proxy to reuse UDP 4-tuples for multiple connections. It also defines a mode of proxying in which QUIC short header packets can be forwarded through the proxy rather than being re-encapsulated and re-encrypted.

Discussion Venues

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

Source for this draft and an issue tracker can be found at https://github.com/tfpauly/quic-proxy (https://github.com/tfpauly/quic-proxy).

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 15 October 2021.

Copyright Notice

Table of Contents

1. Introduction

The CONNECT-UDP HTTP method [CONNECT-UDP] defines a way to send datagrams through an HTTP proxy, where UDP is used to communicate between the proxy and a target server. This can be used to proxy QUIC connections [QUIC], since QUIC runs over UDP datagrams.

This document uses the term "target" to refer to the server that a client is accessing via a proxy. This target may be an origin hosting content, or another proxy.

This document extends the CONNECT-UDP HTTP method to add signalling about QUIC Connection IDs. QUIC Connection IDs are used to identify QUIC connections in scenarios where there is not a strict bidirectional mapping between one QUIC connection and one UDP 4-tuple (pairs of IP addresses and ports). A proxy that is aware of Connection IDs can reuse UDP 4-tuples between itself and a target for multiple proxied QUIC connections.

This extension is only defined for HTTP/3 [HTTP3] and not any earlier versions.

Awareness of Connection IDs also allows a proxy to avoid re-encapsulation and re-encryption of proxied QUIC packets once a connection has been established. When this functionality is present, the proxy can support two modes for handling QUIC packets:

1. Tunnelled, in which client <-> target QUIC packets are encapsulated inside client <-> proxy QUIC packets. These packets use multiple layers of encryption and congestion control. QUIC long header packets MUST use this mode. QUIC short header packets MAY use this mode. This is the default mode for CONNECT-UDP.

2. Forwarded, in which client <-> target QUIC packets are sent directly over the client <-> proxy UDP socket. These packets are only encrypted using the client-target keys, and use the client-target congestion control. This mode MUST only be used for QUIC short header packets.

Forwarding is defined as an optimization to reduce CPU processing on clients and proxies, as well as avoiding MTU overhead for packets on the wire. This makes it suitable for deployment situations that otherwise relied on cleartext TCP proxies, which cannot support QUIC and have inferior security and privacy properties.

The properties provided by the forwarding mode are as follows:

* All packets sent between the client and the target traverse through the proxy device.

* The target server cannot know the IP address of the client solely based on the proxied packets the target receives.

* Observers of either or both of the client <-> proxy link and the proxy <-> target are not able to learn more about the client <-> target communication than if no proxy was used.

It is not a goal of forwarding mode to prevent correlation between client <-> proxy and the proxy <-> target packets from an entity that can observe both links. See Section 9 for further discussion.

Both clients and proxies can unilaterally choose to disable forwarded mode for any client <-> target connection.

QUIC proxies only need to understand the Header Form bit, and the connection ID fields from packets in client <-> target QUIC connections. Since these fields are all in the QUIC invariants header [INVARIANTS], QUIC proxies can proxy all versions of QUIC.

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.

1.2. Terminology

This document uses the following terms:

* Client: the client of all QUIC connections discussed in this document.

* Proxy: the endpoint that responds to the CONNECT-UDP request.

* Target: the server that a client is accessing via a proxy.

* Client <-> Proxy QUIC connection: a single QUIC connection established from the client to the proxy.

* Datagram flow ID: represents a flow of HTTP/3 DATAGRAMs [H3DGRAM] specific to a single client <-> proxy QUIC connection.

* Socket: a UDP 4-tuple (local IP address, local UDP port, remote IP address, remote UDP port). In some implementations, this is referred to as a "connected" socket.

* Client-facing socket: the socket used to communicate between the client and the proxy.

* Server-facing socket: the socket used to communicate between the proxy and the target.

* Client Connection ID: a QUIC Connection ID that is chosen by the client, and is used in the Destination Connection ID field of packets from the target to the client.

* Server Connection ID: a QUIC Connection ID that is chosen by the target, and is used in the Destination Connection ID field of packets from the client to the target.

2. Required Proxy State

In the methods defined in this document, the proxy is aware of the QUIC Connection IDs being used by proxied connections, along with the sockets used to communicate with the client and the target. Tracking Connection IDs in this way allows the proxy to reuse server-facing sockets for multiple connections and support the forwarding mode of proxying.

QUIC packets can be either tunnelled within an HTTP/3 proxy connection using QUIC DATAGRAM frames [DGRAM], or be forwarded directly alongside an HTTP/3 proxy connection on the same set of IP addresses and UDP ports. The use of forwarded mode requires the consent of both the client and the proxy.

In order to correctly route QUIC packets in both tunnelled and forwarded modes, the proxy needs to maintain mappings between several items. There are three required unidirectional mappings, described below.

2.1. Datagram Flow ID Mapping

Each pair of client <-> proxy QUIC connection and datagram flow ID MUST be mapped to a single server-facing socket.

```
(Client <-> Proxy QUIC connection + Datagram flow ID)
    => Server-facing socket
```

Multiple datagram flow IDs can map to the same server-facing socket, but a single datagram flow ID cannot be mapped to multiple server-facing sockets.

This mapping guarantees that any QUIC packet using the datagram flow ID sent from the client to the proxy in tunnelled mode can be sent to the correct target.

2.2. Server Connection ID Mapping

Each pair of Server Connection ID and client-facing socket MUST map to a single server-facing socket.

```
(Client-facing socket + Server Connection ID)
    => Server-facing socket
```

Multiple pairs of Connection IDs and sockets can map to the same server-facing socket.

This mapping guarantees that any QUIC packet containing the Server Connection ID sent from the client to the proxy in forwarded mode can be sent to the correct target. Thus, a proxy that does not allow forwarded mode does not need to maintain this mapping.

2.3. Client Connection ID Mappings

Each pair of Client Connection ID and server-facing socket MUST map to a single datagram flow ID on a single client <-> proxy QUIC connection. Additionally, the pair of Client Connection ID and server-facing socket MUST map to a single client-facing socket.

```
(Server-facing socket + Client Connection ID)
    => (Client <-> Proxy QUIC connection + Datagram flow ID)
(Server-facing socket + Client Connection ID)
    => Client-facing socket
```

Multiple pairs of Connection IDs and sockets can map to the same datagram flow ID or client-facing socket.

These mappings guarantee that any QUIC packet sent from a target to the proxy can be sent to the correct client, in either tunnelled or forwarded mode. Note that this mapping becomes trivial if the proxy always opens a new server-facing socket for every client request with a unique datagram flow ID. The mapping is critical for any case where server-facing sockets are shared or reused.

2.4. Detecting Connection ID Conflicts

In order to be able to route packets correctly in both tunnelled and forwarded mode, proxies check for conflicts before creating a new mapping. If a conflict is detected, the proxy will reject the client's request, as described in Section 5.

Two sockets conflict if and only if all members of the 4-tuple (local IP address, local UDP port, remote IP address, and remote UDP port) are identical.

Two Connection IDs conflict if and only if one Connection ID is equal to or a prefix of another. For example, a zero-length Connection ID conflicts with all connection IDs. This definition of a conflict originates from the fact that QUIC short headers do not carry the length of the Destination Connection ID field, and therefore if two short headers with different Destination Connection IDs are received on a shared socket, one being a prefix of the other prevents the receiver from identifying which mapping this corresponds to.

The proxy treats two mappings as being in conflict when a conflict is detected for all elements on the left side of the mapping diagrams above.

Since very short Connection IDs are more likely to lead to conflicts, particularly zero-length Connection IDs, a proxy MAY choose to reject all requests for very short Connection IDs as conflicts, in anticipation of future conflicts. Note that a request that doesn't contain any Connection ID is equivalent to a request for a zero-length Connection ID, and similarly would cause conflicts when forwarding.

3. Connection ID Headers for CONNECT-UDP

This document defines two headers that can be used in CONNECT-UDP requests and responses. All other requirements defined for CONNECT-UDP [CONNECT-UDP] still apply.

Both "Client-Connection-Id" and "Server-Connection-Id" are Item Structured Headers [STRUCT]. Their value MUST be a Byte Sequence. The byte sequence MAY be zero-length. The byte sequence length MUST NOT exceed 255 bytes. The ABNF is:

```
Client-Connection-Id = sf-binary
Server-Connection-Id = sf-binary
```

"Client-Connection-Id" contains the client connection ID, whereas "Server-Connection-Id" contains the server connection ID.

Like the Datagram-Flow-Id header [CONNECT-UDP], the Client-Connection-Id and Server-Connection-Id headers can only be supported by an HTTP/3 proxy. If a proxy does not support HTTP/3 datagrams [H3DGRAM], or it does not support the extension defined in this document, it MUST NOT send the Client-Connection-Id and Server-Connection-Id headers on any responses. If a proxy does support this extension, it MUST echo the Client-Connection-Id and Server-Connection-Id headers on any 2xx (Successful) responses. Clients that do not receive an echoed Client-Connection-Id or Server-Connection-Id header MUST fall back to using CONNECT-UDP without the extended behavior defined in this document.

4. Client Request Behavior

A client sends new CONNECT-UDP requests when it wants to start a new QUIC connection to a target, when it has received a new Server Connection ID for the target, and before it advertises a new Client Connection ID to the target.

Each request MUST contain a Datagram-Flow-Id header and an authority pseudo-header identifying the target. All requests for the same QUIC Connection between a client and a target MUST contain the same authority, and SHOULD contain the same Datagram-Flow-Id. If there is Datagram-Flow-Id mismatch, the proxy will treat the requests as different proxied connections, which could appear as a migration or NAT rebinding event to the target.

Each request MUST also contain exactly one connection ID header, either Client-Connection-Id or Server-Connection-Id. Client-Connection-Id requests define paths for receiving packets from the target server to the client, and Server-Connection-Id requests define paths for sending packets from the client to target server.

4.1. New Proxied Connection Setup

The first time that a client uses a proxy for a given QUIC connection, it selects a new datagram flow ID with an even-numbered value [H3DGRAM].

The first request the clients makes MUST contain the authority pseudo-header, the Datagram-Flow-Id header, and the Client-Connection-Id header. These respectively contain the authority of the target, the selected datagram flow ID and the Client Connection ID that will be used in the initial QUIC packets sent through the proxy.

The client can start sending packets tunnelled within DATAGRAM frames as soon as this first CONNECT-UDP request for the datagram flow ID has been sent, even in the same QUIC packet to the proxy. That is, the QUIC packet sent from the client to the proxy MAY contain a STREAM frame containing the CONNECT-UDP request, as well as a DATAGRAM frame that contains a tunnelled QUIC packet to send to the target. This is particularly useful for reducing round trips on connection setup.

Since clients are always aware whether or not they are using a QUIC proxy, clients are expected to cooperate with proxies in selecting Client Connection IDs. A proxy detects a conflict when it is not able to create a unique mapping using the Client Connection ID (Section 2.4). It can reject requests that would cause a conflict and indicate this to the client by replying with a 409 (Conflict) status. In order to avoid conflicts, clients SHOULD select Client Connection IDs of at least 8 bytes in length with unpredictable values. A client also SHOULD NOT select a Client Connection ID that matches the ID used for the QUIC connection to the proxy, as this inherently creates a conflict.

Note that packets sent in DATAGRAM frames before the proxy has sent its CONNECT-UDP response might be dropped if the proxy rejects the request. Specifically, this can occur if the Client Connection ID causes a conflict and the proxy returns a 409 (Conflict) error. Any DATAGRAM frames that are sent in a separate QUIC packet from the STREAM frame that contains the CONNECT-UDP request might also be dropped in the case that the packet arrives at the proxy before the packet containing the STREAM frame.

If the server rejects the first request that uses a specific datagram flow ID, the client MUST retire that datagram flow ID. If the rejection indicated a conflict due to the Client Connection ID, the client MUST select a new Connection ID before sending a new request, and generate a new packet. For example, if a client is sending a QUIC Initial packet and chooses a Connection ID that conflicts with an existing mapping to the same target server, it will need to generate a new QUIC Initial.

4.2. Adding New Client Connection IDs

Since QUIC connection IDs are chosen by the receiver, an endpoint needs to communicate its chosen connection IDs to its peer before the peer can start using them. In QUICv1, this is performed using the NEW_CONNECTION_ID frame.

Prior to informing the target of a new chosen client connection ID, the client MUST send a CONNECT-UDP request with the Client-Connection-Id header to the proxy, and only inform the target of the new client connection ID once a 2xx (Successful) response is received.

4.3. Sending With Forwarded Mode

Once the client has learned the target server's Connection ID, such as in the response to a QUIC Initial packet, it can send a request containing the Server-Connection-Id header to request the ability to forward packets. The client MUST wait for a successful (2xx) response before using forwarded mode. Prior to receiving the server response, the client MUST send short header packets tunnelled in DATAGRAM frames. The client MAY also choose to tunnel some short header packets even after receiving the successful response.

If the client's request that included the Server-Connection-Id is rejected, for example with a 409 (Conflict) response, it MUST NOT forward packets to the requested Server Connection ID, but only use tunnelled mode. The request might also be rejected if the proxy does not support forwarded mode or has it disabled by policy. For any response code other than a 2xx success, the client MUST NOT retry a request for the same Server Connection ID. For errors other than 409 (Conflict), clients SHOULD stop sending requests for other Server Connection IDs in the future.

QUIC long header packets MUST NOT be forwarded. These packets can only be tunnelled within DATAGRAM frames to avoid exposing unnecessary connection metadata.

When forwarding, the client sends a QUIC packet with the target server's Connection ID in the QUIC short header, using the same socket between client and proxy that was used for the main QUIC connection between client and proxy.

4.4. Receiving With Forwarded Mode

A proxy MUST NOT forward packets from the target to the client until after the client has sent at least one packet in forwarded mode. Once this occurs, the proxy MAY use forwarded mode for any Client Connection ID for which it has a valid mapping.

If a client has started using forwarding mode, it MUST be prepared to receive forwarded short header packets on the socket between itself and the proxy for any Client Connection ID that has been accepted by the proxy. The client uses the received Connection ID to determine if a packet was originated by the proxy, or merely forwarded from the target.

4.5. Opting Out of Forwarded Mode

The use of forwarded mode is initiated by the client sending a request with the Server-Connection-Id header and sending at least one forwarded packet to the proxy. A client that does not wish to accept forwarded packets from the proxy when communicating with a specific target can simply not start forwarding packets to the proxy.

5. Proxy Response Behavior

Upon receipt of a CONNECT-UDP request that contains a Client-Connection-Id or Server-Connection-Id header, the proxy validates the request, tries to establish the appropriate mappings as described in Section 2, and establishes a new server-facing socket if necessary.

The proxy MUST validate that the request only contains one of either the Client-Connection-Id or the Server-Connection-Id header, along with a Datagram-Flow-Id header and an authority pseudo-header. If any of these conditions is not met, the proxy MUST reject the request with a 400 (Bad Request) response. The proxy also MUST reject the request if the requested datagram flow ID has already been used on that client <-> proxy QUIC connection with a different requested authority.

The proxy then determines the server-facing socket to associate with the client's datagram flow. This will generally involve performing a DNS lookup for the hostname in the request authority, or finding an existing server-facing socket to the authority. The server-facing socket might already be open due to a previous request from this client, or another. If the socket is not already created, the proxy creates a new one. Proxies can choose to reuse server-facing sockets across multiple datagram flows, or have a unique server-facing socket for every datagram flow.

If a proxy reuses server-facing sockets, it SHOULD store which authorities (which could be a domain name or IP address literal) are being accessed over a particular server-facing socket so it can avoid performing a new DNS query and potentially choosing a different server IP address which could map to a different server.

Server-facing sockets MUST NOT be reused across QUIC and non-QUIC CONNECT-UDP requests, since it might not be possible to correctly demultiplex or direct the traffic. Any packets received on a server-facing socket used for proxying QUIC that does not correspond to a known Connection ID MUST be dropped.

If the request includes a Client-Connection-Id header, the proxy is receiving a request to be able to route traffic back to the client using that Connection ID. If the pair of this Client Connection ID and the selected server-facing socket does not create a conflict, the proxy creates the mapping and responds with a 200 (OK) response. After this point, any packets received by the proxy from the server-facing socket that match the Client Connection ID can to be sent to the client. The proxy MUST use tunnelled mode (DATAGRAM frames) on the correct datagram flow for any long header packets. The proxy SHOULD forward directly to the client for any matching short header packets, but MAY tunnel them in DATAGRAM frames. If the pair is not unique, or the proxy chooses not to support zero-length Client Connection IDs, the proxy responds with a 409 (Conflict) response. If this occurs on the first request for a given datagram flow, the proxy removes any mapping for that datagram flow.

If the request includes a Server-Connection-Id header, the proxy is receiving a request to allow the client to forward packets to the target. If the pair of this Server Connection ID and the client-facing socket on which the request was received does not create a conflict, the proxy creates the mapping and responds with a 200 (OK) response. Once the successful response is sent, the proxy will forward any short header packets received on the client-facing socket that use the Server Connection ID using the correct server-facing socket. If the pair is not unique, the server responds with a 409 (Conflict) response. If this occurs, traffic for that Server Connection ID can only use tunnelled mode, not forwarded.

If the proxy does not support forwarded mode, or does not allow forwarded mode for a particular client or authority by policy, it can reject requests that include the Server-Connection-Id header with a response to indicate the error, such as 403 (Forbidden).

Any successful (2xx) response MUST also echo any Client-Connection-Id, Server-Connection-Id, and Datagram-Flow-Id headers included in the request.

The proxy MUST only forward non-tunnelled packets from the client that are QUIC short header packets (based on the Header Form bit) and have mapped Server Connection IDs. Packets sent by the client that are forwarded SHOULD be considered as activity for restarting QUIC's Idle Timeout [QUIC].

5.1. Removing Mapping State

Each CONNECT-UDP request consumes one bidirectional HTTP/3 stream [HTTP3]. For any stream on which the proxy has sent a response indicating success, any mappings for the request last as long as the stream is open.

A client that no longer wants a given Connection ID to be forwarded by the proxy, for either direction, MUST cancel its CONNECT-UDP HTTP/3 request by closing the corresponding stream.

If the proxy rejects a CONNECT-UDP request by sending a status code of 300 or higher, it MUST close the corresponding stream and remove any associated mappings.

If a client's connection to the proxy is terminated for any reason, all mappings associated with all requests are removed.

A proxy can close its server-facing socket once all datagram flows mapped to that socket have been removed.

5.2. Handling Connection Migration

If a proxy supports QUIC connection migration, it needs to ensure that a migration event does not end up sending too many tunnelled or proxied packets on a new path prior to path validation.

Specifically, the proxy MUST limit the number of packets that it will proxy to an unvalidated client address to the size of an initial congestion window. Proxies additionally SHOULD pace the rate at which packets are sent over a new path to avoid creating unintentional congestion on the new path.

6. Example

Consider a client that is establishing a new QUIC connection through the proxy. It has selected a Client Connection ID of 0x31323334. It selects the next open datagram flow ID (2). In order to inform a proxy of the new QUIC Client Connection ID, and binds that connection ID to datagram flow ID 2, the client sends the following CONNECT-UDP request:

```
HEADERS
:method = CONNECT-UDP
:authority = target.example.com:443
client-connection-id = :MTIzNA==:
datagram-flow-id = 2
```

The client will also send the initial QUIC packet with the Long Header form in a DATAGRAM frame with flow ID 2.

Once the proxy sends a 200 response indicating success, packets received by the proxy that match the Connection ID 0x31323334 will be directly forwarded to the client. The proxy will also forward the initial QUIC packet received on DATAGRAM flow ID 2 to target.example.com:443.

When the proxy receives a response from target.example.com:443 that has 0x31323334 as the Destination Connection ID, the proxy will forward that packet to the client on DATAGRAM flow ID 2.

Once the client learns which Connection ID has been selected by the target server, it can send a new request to the proxy to establish a mapping. In this case, that ID is 0x61626364. The client sends the following request:

```
HEADERS
:method = CONNECT-UDP
:authority = target.example.com:443
server-connection-id = :YWJjZA==:
datagram-flow-id = 2
```

The client also sends its reply to the target server in a DATAGRAM frame on flow ID 2 after sending the new request.

Once the proxy sends a 200 response indicating success, packets sent by the client that match the Connection ID 0x61626364 will be forwarded to the target server, i.e., without proxy decryption.

Upon receiving the response, the client starts sending Short Header packets with a Destination Connection ID of 0x61626364 directly to the proxy (not tunnelled), and these are forwarded directly to the target by the proxy. Similarly, Short Header packets from the target with a Destination Connection ID of 0x31323334 are forwarded directly to the client.

7. Interactions with Load Balancers

Some QUIC servers are accessed using load balancers, as described in [QUIC-LB]. These load balancers route packets to servers based on the server's Connection ID. These Connection IDs are generated in a way that can be coordinated between servers and their load balancers.

If a proxy that supports this extension is itself running behind a load balancer, extra complexity arises once clients start using forwarding mode and sending packets to the proxy that have

Destination Connection IDs that belong to the end servers, not the proxy. If the load balancer is not aware of these Connection IDs, or the Connection IDs conflict with other Connection IDs used by the load balancer, packets can be routed incorrectly.

QUIC-aware CONNECT-UDP proxies that use forwarding mode generally SHOULD NOT be run behind load balancers; and if they are, they MUST coordinate between the proxy and the load balancer to create mappings for proxied Connection IDs prior to the proxy sending 2xx (Successful) responses to clients.

QUIC-aware CONNECT-UDP proxies that do not allow forwarding mode can function unmodified behind QUIC load balancers.

8. Packet Size Considerations

Since Initial QUIC packets must be at least 1200 bytes in length, the DATAGRAM frames that are used for a QUIC-aware CONNECT-UDP proxy MUST be able to carry at least 1200 bytes.

Additionally, clients that connect to a proxy for purpose of proxying QUIC SHOULD start their connection with a larger packet size than 1200 bytes, to account for the overhead of tunnelling an Initial QUIC packet within a DATAGRAM frame. If the client does not begin with a larger packet size than 1200 bytes, it will need to perform Path MTU (Maximum Transmission Unit) discovery to discover a larger path size prior to sending any tunnelled Initial QUIC packets.

Once a proxied QUIC connections moves into forwarded mode, the client SHOULD initiate Path MTU discovery to increase its end-to-end MTU.

9. Security Considerations

Proxies that support this extension SHOULD provide protections to rate-limit or restrict clients from opening an excessive number of proxied connections, so as to limit abuse or use of proxies to launch Denial-of-Service attacks.

Sending QUIC packets by forwarding through a proxy without tunnelling exposes some QUIC header metadata to onlookers, and can be used to correlate packets flows if an attacker is able to see traffic on both sides of the proxy. Tunnelled packets have similar inference problems. An attacker on both sides of the proxy can use the size of ingress and egress packets to correlate packets belonging to the same connection. (Absent client-side padding, tunneled packets will typically have a fixed amount of overhead that is removed before their DATAGRAM contents are written to the target.)

Since proxies that forward QUIC packets do not perform any cryptographic integrity check, it is possible that these packets are either malformed, replays, or otherwise malicious. This may result in proxy targets rate limiting or decreasing the reputation of a given proxy.

10. IANA Considerations

10.1. HTTP Headers

This document registers the "Client-Connection-Id" and "Server-Connection-Id" headers in the "Permanent Message Header Field Names" <https://www.iana.org/assignments/message-headers>.

Header Field Name	Protocol	Status	Reference
Client-Connection-Id	http	exp	This document
Server-Connection-Id	http	exp	This document

11. References

11.1. Normative References

[CONNECT-UDP] Schinazi, D., "The CONNECT-UDP HTTP Method", Work in Progress, Internet-Draft, draft-ietf-masque-connect-udp-03, 5 January 2021, <http://www.ietf.org/internet-drafts/draft-ietf-masque-connect-udp-03.txt>.

[DGRAM] Pauly, T., Kinnear, E., and D. Schinazi, "An Unreliable Datagram Extension to QUIC", Work in Progress, Internet-Draft, draft-ietf-quic-datagram-01, 24 August 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-datagram-01.txt>.

[H3DGRAM] Schinazi, D., "Using QUIC Datagrams with HTTP/3", Work in Progress, Internet-Draft, draft-schinazi-quic-h3-datagram-05, 12 October 2020, <http://www.ietf.org/internet-drafts/draft-schinazi-quic-h3-datagram-05.txt>.

[HTTP3] Bishop, M., "Hypertext Transfer Protocol Version 3 (HTTP/3)", Work in Progress, Internet-Draft, draft-ietf-quic-http-33, 15 December 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-http-33.txt>.

[INVARIANTS]
Thomson, M., "Version-Independent Properties of QUIC", Work in Progress, Internet-Draft, draft-ietf-quic-invariants-13, 14 January 2021, <http://www.ietf.org/internet-drafts/draft-ietf-quic-invariants-13.txt>.

[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, <http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-34.txt>.

[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>.

[STRUCT] Nottingham, M. and P. Kamp, "Structured Field Values for HTTP", Work in Progress, Internet-Draft, draft-ietf-httpbis-header-structure-19, 3 June 2020, <http://www.ietf.org/internet-drafts/draft-ietf-httpbis-header-structure-19.txt>.

11.2. Informative References

[QUIC-LB] Duke, M. and N. Banks, "QUIC-LB: Generating Routable QUIC Connection IDs", Work in Progress, Internet-Draft, draft-ietf-quic-load-balancers-05, 30 October 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-load-balancers-05.txt>.

Acknowledgments

Thanks to Lucas Pardue, Ryan Hamilton, and Mirja Kuehlewind for their inputs on this document.

Authors' Addresses

Tommy Pauly
Apple Inc.
One Apple Park Way
Cupertino, California 95014,
United States of America

Email: tpauly@apple.com

David Schinazi
Google LLC
1600 Amphitheatre Parkway
Mountain View, California 94043,
United States of America

Email: dschinazi.ietf@gmail.com