

MASQUE  
Internet-Draft  
Intended status: Standards Track  
Expires: 4 November 2022

D. Schinazi  
Google LLC  
3 May 2022

Proxying UDP in HTTP  
draft-ietf-masque-connect-udp-12

## Abstract

This document describes how to proxy UDP in HTTP, similar to how the HTTP CONNECT method allows proxying TCP in HTTP. More specifically, this document defines a protocol that allows HTTP clients to create a tunnel for UDP communications through an HTTP server that acts as a proxy.

## About This Document

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

The latest revision of this draft can be found at <https://ietf-wg-masque.github.io/draft-ietf-masque-connect-udp/draft-ietf-masque-connect-udp.html>. Status information for this document may be found at <https://datatracker.ietf.org/doc/draft-ietf-masque-connect-udp/>.

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

Source for this draft and an issue tracker can be found at <https://github.com/ietf-wg-masque/draft-ietf-masque-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 4 November 2022.

## Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

1. Introduction . . . . .	2
1.1. Conventions and Definitions . . . . .	3
2. Client Configuration . . . . .	3
3. Tunnelling UDP over HTTP . . . . .	4
3.1. UDP Proxy Handling . . . . .	5
3.2. HTTP/1.1 Request . . . . .	6
3.3. HTTP/1.1 Response . . . . .	7
3.4. HTTP/2 and HTTP/3 Requests . . . . .	8
3.5. HTTP/2 and HTTP/3 Responses . . . . .	8
3.6. Note About Draft Versions . . . . .	9
4. Context Identifiers . . . . .	9
5. HTTP Datagram Payload Format . . . . .	10
6. Performance Considerations . . . . .	11
6.1. MTU Considerations . . . . .	11
6.2. Tunneling of ECN Marks . . . . .	12
7. Security Considerations . . . . .	12
8. IANA Considerations . . . . .	13
8.1. HTTP Upgrade Token . . . . .	13
8.2. Well-Known URI . . . . .	13
9. References . . . . .	13
9.1. Normative References . . . . .	13
9.2. Informative References . . . . .	15
Acknowledgments . . . . .	16
Author's Address . . . . .	16

## 1. Introduction

While HTTP provides the CONNECT method (see Section 9.3.6 of [HTTP]) for creating a TCP [TCP] tunnel to a proxy, it lacks a method for doing so for UDP [UDP] traffic.

This document describes a protocol for tunnelling UDP to a server acting as a UDP-specific proxy over HTTP. UDP tunnels are commonly used to create an end-to-end virtual connection, which can then be secured using QUIC [QUIC] or another protocol running over UDP. Unlike CONNECT, the UDP proxy itself is identified with an absolute URL containing the traffic's destination. Clients generate those URLs using a URI Template [TEMPLATE], as described in Section 2.

This protocol supports all versions of HTTP by using HTTP Datagrams [HTTP-DGRAM]. When using HTTP/2 [HTTP/2] or HTTP/3 [HTTP/3], it uses HTTP Extended CONNECT as described in [EXT-CONNECT2] and [EXT-CONNECT3]. When using HTTP/1.x [HTTP/1.1], it uses HTTP Upgrade as defined in Section 7.8 of [HTTP].

### 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 "UDP proxy" to refer to the HTTP server that acts upon the client's UDP tunnelling request to open a UDP socket to a target server, and generates the response to this request. If there are HTTP intermediaries (as defined in Section 3.7 of [HTTP]) between the client and the UDP proxy, those are referred to as "intermediaries" in this document.

Note that, when the HTTP version in use does not support multiplexing streams (such as HTTP/1.1), any reference to "stream" in this document represents the entire connection.

## 2. Client Configuration

HTTP clients are configured to use a UDP proxy with a URI Template [TEMPLATE] that has the variables "target\_host" and "target\_port". Examples are shown below:

```
https://masque.example.org/.well-known/masque/udp/{target_host}/{target_port}/
https://proxy.example.org:4443/masque?h={target_host}&p={target_port}
https://proxy.example.org:4443/masque{?target_host,target_port}
```

Figure 1: URI Template Examples

The following requirements apply to the URI Template:

- \* The URI Template MUST be a level 3 template or lower.

- \* The URI Template MUST be in absolute form, and MUST include non-empty scheme, authority and path components.
- \* The path component of the URI Template MUST start with a slash `"/"`.
- \* All template variables MUST be within the path or query components of the URI.
- \* The URI template MUST contain the two variables `"target_host"` and `"target_port"` and MAY contain other variables.
- \* The URI Template MUST NOT contain any non-ASCII unicode characters and MUST only contain ASCII characters in the range 0x21-0x7E inclusive (note that percent-encoding is allowed).
- \* The URI Template MUST NOT use Reserved Expansion (`"+"` operator), Fragment Expansion (`"#"` operator), Label Expansion with Dot-Prefix, Path Segment Expansion with Slash-Prefix, nor Path-Style Parameter Expansion with Semicolon-Prefix.

If the client detects that any of the requirements above are not met by a URI Template, the client MUST reject its configuration and fail the request without sending it to the UDP proxy. While clients SHOULD validate the requirements above, some clients MAY use a general-purpose URI Template implementation that lacks this specific validation.

Since the original HTTP CONNECT method allowed conveying the target host and port but not the scheme, proxy authority, path, nor query, there exist proxy configuration interfaces that only allow the user to configure the proxy host and the proxy port. Client implementations of this specification that are constrained by such limitations MAY attempt to access UDP proxying capabilities using the default template, which is defined as:

`"https://{PROXY_HOST}:{PROXY_PORT}/.well-known/masque/udp/{target_host}/{target_port}/"` where `{PROXY_HOST}` and `{PROXY_PORT}` are the configured host and port of the UDP proxy respectively. UDP proxy deployments SHOULD offer service at this location if they need to interoperate with such clients.

### 3. Tunneling UDP over HTTP

To allow negotiation of a tunnel for UDP over HTTP, this document defines the `"connect-udp"` HTTP Upgrade Token. The resulting UDP tunnels use the Capsule Protocol (see Section 3.2 of [HTTP-DGRAM]) with HTTP Datagram in the format defined in Section 5.

To initiate a UDP tunnel associated with a single HTTP stream, clients issue a request containing the "connect-udp" upgrade token. The target of the tunnel is indicated by the client to the UDP proxy via the "target\_host" and "target\_port" variables of the URI Template, see Section 2. If the request is successful, the UDP proxy commits to converting received HTTP Datagrams into UDP packets and vice versa until the tunnel is closed.

When sending its UDP proxying request, the client SHALL perform URI Template expansion to determine the path and query of its request. target\_host supports using DNS names, IPv6 literals and IPv4 literals. Note that this URI Template expansion requires using pct-encoding, so for example if the target\_host is "2001:db8::42", it will be encoded in the URI as "2001%3Adb8%3A%3A42".

By virtue of the definition of the Capsule Protocol (see [HTTP-DGRAM]), UDP proxying requests do not carry any message content. Similarly, successful UDP proxying responses also do not carry any message content.

### 3.1. UDP Proxy Handling

Upon receiving a UDP proxying request:

- \* if the recipient is configured to use another HTTP proxy, it will act as an intermediary: it forwards the request to another HTTP server. Note that such intermediaries may need to reencode the request if they forward it using a version of HTTP that is different from the one used to receive it, as the request encoding differs by version (see below).
- \* otherwise, the recipient will act as a UDP proxy: it extracts the "target\_host" and "target\_port" variables from the URI it has reconstructed from the request headers, and establishes a tunnel by directly opening a UDP socket to the requested target.

Unlike TCP, UDP is connection-less. The UDP proxy that opens the UDP socket has no way of knowing whether the destination is reachable. Therefore it needs to respond to the request without waiting for a packet from the target. However, if the target\_host is a DNS name, the UDP proxy MUST perform DNS resolution before replying to the HTTP request. If errors occur during this process, the UDP proxy MUST fail the request and SHOULD send details using an appropriate "Proxy-Status" header field [PROXY-STATUS] (for example, if DNS resolution returns an error, the proxy can use the dns\_error Proxy Error Type from Section 2.3.2 of [PROXY-STATUS]).

UDP proxies can use connected UDP sockets if their operating system supports them, as that allows the UDP proxy to rely on the kernel to only send it UDP packets that match the correct 5-tuple. If the UDP 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 request. Packets that do not match **MUST** be discarded by the UDP proxy.

The lifetime of the socket is tied to the request stream. The UDP proxy **MUST** keep the socket open while the request stream is open. If a UDP proxy is notified by its operating system that its socket is no longer usable (for example, this can happen when an ICMP "Destination Unreachable" message is received, see Section 3.1 of [ICMP6]), it **MUST** close the request stream. UDP proxies **MAY** choose to close sockets due to a period of inactivity, but they **MUST** close the request stream when closing the socket. UDP proxies that close sockets after a period of inactivity **SHOULD NOT** use a period lower than two minutes, see Section 4.3 of [BEHAVE].

A successful response (as defined in Section 3.3 and Section 3.5) indicates that the UDP proxy has opened a socket to the requested target and is willing to proxy UDP payloads. Any response other than a successful response indicates that the request has failed, and the client **MUST** therefore abort the request.

UDP proxies **MUST NOT** introduce fragmentation at the IP layer when forwarding HTTP Datagrams onto a UDP socket. In IPv4, the Don't Fragment (DF) bit **MUST** be set if possible, to prevent fragmentation on the path. Future extensions **MAY** remove these requirements.

### 3.2. HTTP/1.1 Request

When using HTTP/1.1 [HTTP/1.1], a UDP proxying request will meet the following requirements:

- \* the method **SHALL** be "GET".
- \* the request **SHALL** include a single "Host" header field containing the origin of the UDP proxy.
- \* the request **SHALL** include a "Connection" header field with value "Upgrade" (note that this requirement is case-insensitive as per Section 7.6.1 of [HTTP]).
- \* the request **SHALL** include an "Upgrade" header field with value "connect-udp".

For example, if the client is configured with URI Template "https://proxy.example.org/.well-known/masque/udp/{target\_host}/{target\_port}/" and wishes to open a UDP proxying tunnel to target 192.0.2.42:443, it could send the following request:

```
GET https://proxy.example.org/.well-known/masque/udp/192.0.2.42/443/ HTTP/1.1
Host: proxy.example.org
Connection: Upgrade
Upgrade: connect-udp
```

Figure 2: Example HTTP/1.1 Request

In HTTP/1.1, this protocol uses the GET method to mimic the design of the WebSocket Protocol [WEBSOCKET].

### 3.3. HTTP/1.1 Response

The UDP proxy SHALL indicate a successful response by replying with the following requirements:

- \* the HTTP status code on the response SHALL be 101 (Switching Protocols).
- \* the response SHALL include a single "Connection" header field with value "Upgrade" (note that this requirement is case-insensitive as per Section 7.6.1 of [HTTP]).
- \* the response SHALL include a single "Upgrade" header field with value "connect-udp".
- \* the response SHALL NOT include any "Transfer-Encoding" or "Content-Length" header fields.

If any of these requirements are not met, the client MUST treat this proxying attempt as failed and abort the connection.

For example, the UDP proxy could respond with:

```
HTTP/1.1 101 Switching Protocols
Connection: Upgrade
Upgrade: connect-udp
```

Figure 3: Example HTTP/1.1 Response

### 3.4. HTTP/2 and HTTP/3 Requests

When using HTTP/2 [HTTP/2] or HTTP/3 [HTTP/3], UDP proxying requests use Extended CONNECT. This requires that servers send an HTTP Setting as specified in [EXT-CONNECT2] and [EXT-CONNECT3], and that requests use HTTP pseudo-header fields with the following requirements:

- \* The ":method" pseudo-header field SHALL be "CONNECT".
- \* The ":protocol" pseudo-header field SHALL be "connect-udp".
- \* The ":authority" pseudo-header field SHALL contain the authority of the UDP proxy.
- \* The ":path" and ":scheme" pseudo-header fields SHALL NOT be empty. Their values SHALL contain the scheme and path from the URI Template after the URI template expansion process has been completed.

A UDP proxying request that does not conform to these restrictions is malformed (see Section 8.1.1 of [HTTP/2]).

For example, if the client is configured with URI Template "https://proxy.example.org/{target\_host}/{target\_port}/" and wishes to open a UDP proxying tunnel to target 192.0.2.42:443, it could send the following request:

```
HEADERS
:method = CONNECT
:protocol = connect-udp
:scheme = https
:path = /.well-known/masque/udp/192.0.2.42/443/
:authority = proxy.example.org
```

Figure 4: Example HTTP/2 Request

### 3.5. HTTP/2 and HTTP/3 Responses

The UDP proxy SHALL indicate a successful response by replying with any 2xx (Successful) HTTP status code, without any "Transfer-Encoding" or "Content-Length" header fields.

If any of these requirements are not met, the client MUST treat this proxying attempt as failed and abort the request.

For example, the UDP proxy could respond with:



```
HEADERS
:status = 200
```

Figure 5: Example HTTP/2 Response

### 3.6. Note About Draft Versions

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

In order to allow implementations to support multiple draft versions of this specification during its development, we introduce the "connect-udp-version" header field. When sent by the client, it contains a list of draft numbers supported by the client (e.g., "connect-udp-version: 0, 2"). When sent by the UDP proxy, it contains a single draft number selected by the UDP proxy from the list provided by the client (e.g., "connect-udp-version: 2"). Sending this header field is RECOMMENDED but not required. The "connect-udp-version" header field is a List Structured Field, see Section 3.1 of [STRUCT-FIELD]. Each list member MUST be an Integer.

## 4. Context Identifiers

This protocol allows future extensions to exchange HTTP Datagrams which carry different semantics from UDP payloads. Some of these extensions can augment UDP payloads with additional data, while others can exchange data that is completely separate from UDP payloads. In order to accomplish this, all HTTP Datagrams associated with UDP Proxying request streams start with a context ID, see Section 5.

Context IDs are 62-bit integers (0 to  $2^{62}-1$ ). Context IDs are encoded as variable-length integers, see Section 16 of [QUIC]. The context ID value of 0 is reserved for UDP payloads, while non-zero values are dynamically allocated: non-zero even-numbered context IDs are client-allocated, and odd-numbered context IDs are proxy-allocated. The context ID namespace is tied to a given HTTP request: it is possible for a context ID with the same numeric value to be simultaneously allocated in distinct requests, potentially with different semantics. Context IDs MUST NOT be re-allocated within a given HTTP namespace but MAY be allocated in any order. The context ID allocation restrictions to the use of even-numbered and odd-numbered context IDs exist in order to avoid the need for synchronisation between endpoints. However, once a context ID has been allocated, those restrictions do not apply to the use of the context ID: it can be used by any client or UDP proxy, independent of which endpoint initially allocated it.

Registration is the action by which an endpoint informs its peer of the semantics and format of a given context ID. This document does not define how registration occurs. Future extensions MAY use HTTP header fields or capsules to register contexts. Depending on the method being used, it is possible for datagrams to be received with Context IDs which have not yet been registered, for instance due to reordering of the packet containing the datagram and the packet containing the registration message during transmission.

## 5. HTTP Datagram Payload Format

When HTTP Datagrams (see [HTTP-DGRAM]) are associated with UDP proxying request streams, the HTTP Datagram Payload field has the format defined in Figure 6. Note that when HTTP Datagrams are encoded using QUIC DATAGRAM frames, the Context ID field defined below directly follows the Quarter Stream ID field which is at the start of the QUIC DATAGRAM frame payload:

```
UDP Proxying HTTP Datagram Payload {  
    Context ID (i),  
    Payload (...),  
}
```

Figure 6: UDP Proxying HTTP Datagram Format

Context ID: A variable-length integer (see Section 16 of [QUIC]) that contains the value of the Context ID. If an HTTP/3 datagram which carries an unknown Context ID is received, the receiver SHALL either drop that datagram silently or buffer it temporarily (on the order of a round trip) while awaiting the registration of the corresponding Context ID.

Payload: The payload of the datagram, whose semantics depend on value of the previous field. Note that this field can be empty.

UDP packets are encoded using HTTP Datagrams with the Context ID set to zero. When the Context ID is set to zero, the Payload field contains the unmodified payload of a UDP packet (referred to as "data octets" in [UDP]).

Clients MAY optimistically start sending UDP packets in HTTP Datagrams before receiving the response to its UDP proxying request. However, implementors should note that such proxied packets may not be processed by the UDP proxy if it responds to the request with a failure, or if the proxied packets are received by the UDP proxy before the request.

By virtue of the definition of the UDP header [UDP], it is not possible to encode UDP payloads longer than 65527 bytes. Therefore, endpoints MUST NOT send HTTP Datagrams with a Payload field longer than 65527 using Context ID zero. An endpoint that receives a DATAGRAM capsule using Context ID zero whose Payload field is longer than 65527 MUST abort the stream. If a UDP proxy knows it can only send out UDP packets of a certain length due to its underlying link MTU, it SHOULD discard incoming DATAGRAM capsules using Context ID zero whose Payload field is longer than that limit without buffering the capsule contents.

## 6. Performance Considerations

UDP 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.

If a client or UDP proxy with a connection containing a UDP proxying request stream disables congestion control, it MUST NOT signal ECN support on that connection. That is, it MUST mark all IP headers with the Not-ECT codepoint. It MAY continue to report ECN feedback via ACK\_ECN frames, as the peer may not have disabled congestion control.

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, UDP proxying SHOULD be performed over HTTP/3 to allow leveraging the QUIC DATAGRAM frame.

### 6.1. MTU Considerations

When using HTTP/3 with the QUIC Datagram extension [DGRAM], UDP payloads are transmitted in QUIC DATAGRAM frames. Since those cannot be fragmented, they can only carry payloads up to a given length determined by the QUIC connection configuration and the path MTU. If a UDP proxy is using QUIC DATAGRAM frames and it receives a UDP payload from the target that will not fit inside a QUIC DATAGRAM frame, the UDP proxy SHOULD NOT send the UDP payload in a DATAGRAM capsule, as that defeats the end-to-end unreliability characteristic

that methods such as Datagram Packetization Layer Path MTU Discovery (DPLPMTUD) depend on [DPLPMTUD]. In this scenario, the UDP proxy SHOULD drop the UDP payload and send an ICMP "Packet Too Big" message to the target, see Section 3.2 of [ICMP6].

## 6.2. Tunneling of ECN Marks

UDP proxying does not create an IP-in-IP tunnel, so the guidance in [ECN-TUNNEL] about transferring ECN marks between inner and outer IP headers does not apply. There is no inner IP header in UDP proxying tunnels.

Note that UDP proxying clients do not have the ability in this specification to control the ECN codepoints on UDP packets the UDP proxy sends to the target, nor can UDP proxies communicate the markings of each UDP packet from target to UDP proxy.

A UDP proxy MUST ignore ECN bits in the IP header of UDP packets received from the target, and MUST set the ECN bits to Not-ECT on UDP packets it sends to the target. These do not relate to the ECN markings of packets sent between client and UDP proxy in any way.

## 7. Security Considerations

There are significant risks in allowing arbitrary clients to establish a tunnel to arbitrary targets, as that could allow bad actors to send traffic and have it attributed to the UDP proxy. HTTP servers that support UDP proxying ought to 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 CONNECT proxy can send it application data. UDP doesn't have this property, so a 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. While a UDP proxy could potentially limit the number of UDP packets it is willing to forward until it has observed a response from the target, that is unlikely to provide any protection against denial of service attacks because such attacks target open UDP ports where the protocol running over UDP would respond, and that would be interpreted as willingness to accept UDP by the UDP proxy.

UDP sockets for UDP proxying have a different lifetime than TCP sockets for CONNECT, therefore implementors would be well served to follow the advice in Section 3.1 if they base their UDP proxying implementation on a preexisting implementation of CONNECT.

The security considerations described in [HTTP-DGRAM] also apply here.

## 8. IANA Considerations

### 8.1. HTTP Upgrade Token

This document will request IANA to register "connect-udp" in the "HTTP Upgrade Tokens" registry maintained at <https://www.iana.org/assignments/http-upgrade-tokens>.

Value: connect-udp  
Description: Proxying of UDP Payloads  
Expected Version Tokens: None  
Reference: This document

### 8.2. Well-Known URI

This document will request IANA to register "masque/udp" in the "Well-Known URIs" registry maintained at <https://www.iana.org/assignments/well-known-uris>.

URI Suffix: masque/udp  
Change Controller: IETF  
Reference: This document  
Status: permanent (if this document is approved)  
Related Information: Includes all resources identified with the path prefix `"/.well-known/masque/udp/"`

## 9. References

### 9.1. Normative References

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

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

## [EXT-CONNECT3]

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

## [HTTP]

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

## [HTTP-DGRAM]

Schinazi, D. and L. Pardue, "HTTP Datagrams and the Capsule Protocol", Work in Progress, Internet-Draft, draft-ietf-masque-h3-datagram-09, 11 April 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-masque-h3-datagram-09>>.

## [HTTP/1.1]

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

## [HTTP/2]

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

## [HTTP/3]

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

## [PROXY-STATUS]

Nottingham, M. and P. Sikora, "The Proxy-Status HTTP Response Header Field", Work in Progress, Internet-Draft, draft-ietf-httpbis-proxy-status-08, 13 October 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-proxy-status-08>>.

## [QUIC]

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

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/rfc/rfc2119>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/rfc/rfc8174>>.
- [STRUCT-FIELD] Nottingham, M. and P-H. Kamp, "Structured Field Values for HTTP", RFC 8941, DOI 10.17487/RFC8941, February 2021, <<https://www.rfc-editor.org/rfc/rfc8941>>.
- [TCP] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<https://www.rfc-editor.org/rfc/rfc793>>.
- [TEMPLATE] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <<https://www.rfc-editor.org/rfc/rfc6570>>.
- [UDP] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <<https://www.rfc-editor.org/rfc/rfc768>>.

## 9.2. Informative References

- [BEHAVE] Audet, F., Ed. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, DOI 10.17487/RFC4787, January 2007, <<https://www.rfc-editor.org/rfc/rfc4787>>.
- [DPLPMTUD] Fairhurst, G., Jones, T., Tüxen, M., Rüngeler, I., and T. Völker, "Packetization Layer Path MTU Discovery for Datagram Transports", RFC 8899, DOI 10.17487/RFC8899, September 2020, <<https://www.rfc-editor.org/rfc/rfc8899>>.
- [ECN-TUNNEL] Briscoe, B., "Tunnelling of Explicit Congestion Notification", RFC 6040, DOI 10.17487/RFC6040, November 2010, <<https://www.rfc-editor.org/rfc/rfc6040>>.

[ICMP6] Conta, A., Deering, S., and M. Gupta, Ed., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", STD 89, RFC 4443, DOI 10.17487/RFC4443, March 2006, <<https://www.rfc-editor.org/rfc/rfc4443>>.

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

#### Acknowledgments

This document is a product of the MASQUE Working Group, and the author thanks all MASQUE enthusiasts for their contributions. 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. The author is indebted to Mark Nottingham and Lucas Pardue for the many improvements they contributed to this document. The extensibility design in this document came out of the HTTP Datagrams Design Team, whose members were Alan Frindell, Alex Chernyakhovsky, Ben Schwartz, Eric Rescorla, Lucas Pardue, Marcus Ihlar, Martin Thomson, Mike Bishop, Tommy Pauly, Victor Vasiliev, and the author of this document.

#### Author's Address

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



MASQUE  
Internet-Draft  
Intended status: Standards Track  
Expires: 13 October 2022

D. Schinazi  
Google LLC  
L. Pardue  
Cloudflare  
11 April 2022

HTTP Datagrams and the Capsule Protocol  
draft-ietf-masque-h3-datagram-09

Abstract

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

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

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

About This Document

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

The latest revision of this draft can be found at <https://ietf-wg-masque.github.io/draft-ietf-masque-h3-datagram/draft-ietf-masque-h3-datagram.html>. Status information for this document may be found at <https://datatracker.ietf.org/doc/draft-ietf-masque-h3-datagram/>.

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

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

Status of This Memo

This Internet-Draft 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 13 October 2022.

#### Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

#### Table of Contents

1. Introduction . . . . .	3
1.1. Conventions and Definitions . . . . .	3
2. HTTP Datagrams . . . . .	3
2.1. HTTP/3 Datagrams . . . . .	4
2.1.1. The SETTINGS_H3_DATAGRAM HTTP/3 Setting . . . . .	5
2.2. HTTP Datagrams using Capsules . . . . .	6
3. Capsules . . . . .	7
3.1. HTTP Data Streams . . . . .	7
3.2. The Capsule Protocol . . . . .	8
3.3. Error Handling . . . . .	9
3.4. The Capsule-Protocol Header Field . . . . .	9
3.5. The DATAGRAM Capsule . . . . .	10
4. Security Considerations . . . . .	12
5. IANA Considerations . . . . .	12
5.1. HTTP/3 Setting . . . . .	12
5.2. HTTP/3 Error Code . . . . .	12
5.3. HTTP Header Field Name . . . . .	12
5.4. Capsule Types . . . . .	13
6. References . . . . .	13
6.1. Normative References . . . . .	13

6.2. Informative References . . . . .	15
Acknowledgments . . . . .	15
Authors' Addresses . . . . .	15

## 1. Introduction

HTTP extensions sometimes need to access underlying transport protocol features such as unreliable delivery (as offered by [DGRAM]) to enable desirable features. For example, this could allow introducing an unreliable version of the CONNECT method, or adding unreliable delivery to WebSockets [RFC6455].

In Section 2, this document describes HTTP Datagrams, a convention that supports the bidirectional and optionally multiplexed exchange of data inside an HTTP connection. While HTTP datagrams are associated with HTTP requests, they are not part of message content; instead, they are intended for use by HTTP extensions (such as the CONNECT method), and are compatible with all versions of HTTP.

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

This document also describes the HTTP Capsule Protocol in Section 3, to allow conveyance of HTTP Datagrams using reliable delivery. This addresses HTTP/3 cases where use of the QUIC DATAGRAM frame is unavailable or undesirable, or where the transport protocol only provides reliable delivery, such as with HTTP/1 or HTTP/2 over TCP.

### 1.1. Conventions and Definitions

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

## 2. HTTP Datagrams

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

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

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

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

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

If an HTTP Datagram is received and it is associated with a request that has no known semantics for HTTP Datagrams, the receiver MUST terminate the request; if HTTP/3 is in use, the request stream MUST be aborted with H3\_DATAGRAM\_ERROR (0x33). HTTP extensions can override these requirements by defining a negotiation mechanism and semantics for HTTP Datagrams.

## 2.1. HTTP/3 Datagrams

When used with HTTP/3, the Datagram Data field of QUIC DATAGRAM frames uses the following format (using the notation from the "Notational Conventions" section of [QUIC]):

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

Figure 1: HTTP/3 Datagram Format

Quarter Stream ID: A variable-length integer that contains the value of the client-initiated bidirectional stream that this datagram is associated with, divided by four (the division by four stems from the fact that HTTP requests are sent on client-initiated bidirectional streams, and those have stream IDs that are divisible by four). The largest legal QUIC stream ID value is

$2^{62}-1$ , so the largest legal value of Quarter Stream ID is  $2^{60}-1$ . Receipt of an HTTP/3 Datagram that includes a larger value MUST be treated as an HTTP/3 connection error of type `H3_DATAGRAM_ERROR` (0x33).

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

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

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

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

If an HTTP/3 datagram is received and its Quarter Stream ID maps to a stream that cannot be created due to client-initiated bidirectional stream limits, it SHOULD be treated as an HTTP/3 connection error of type `H3_ID_ERROR`. Generating an error is not mandatory in this case because HTTP/3 implementations might have practical barriers to determining the active stream concurrency limit that is applied by the QUIC layer.

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

#### 2.1.1. The `SETTINGS_H3_DATAGRAM` HTTP/3 Setting

Endpoints can indicate to their peer that they are willing to receive HTTP/3 Datagrams by sending the `SETTINGS_H3_DATAGRAM` (0x33) setting with a value of 1.

The value of the `SETTINGS_H3_DATAGRAM` setting MUST be either 0 or 1. A value of 0 indicates that the implementation is not willing to receive HTTP Datagrams. If the `SETTINGS_H3_DATAGRAM` setting is received with a value that is neither 0 or 1, the receiver MUST terminate the connection with error `H3_SETTINGS_ERROR`.

QUIC DATAGRAM frames MUST NOT be sent until the SETTINGS\_H3\_DATAGRAM setting has been both sent and received with a value of 1.

When clients use 0-RTT, they MAY store the value of the server's SETTINGS\_H3\_DATAGRAM setting. Doing so allows the client to send QUIC DATAGRAM frames in 0-RTT packets. When servers decide to accept 0-RTT data, they MUST send a SETTINGS\_H3\_DATAGRAM setting greater than or equal to the value they sent to the client in the connection where they sent them the NewSessionTicket message. If a client stores the value of the SETTINGS\_H3\_DATAGRAM setting with their 0-RTT state, they MUST validate that the new value of the SETTINGS\_H3\_DATAGRAM setting sent by the server in the handshake is greater than or equal to the stored value; if not, the client MUST terminate the connection with error H3\_SETTINGS\_ERROR. In all cases, the maximum permitted value of the SETTINGS\_H3\_DATAGRAM setting parameter is 1.

It is RECOMMENDED that implementations that support receiving HTTP/3 Datagrams always send the SETTINGS\_H3\_DATAGRAM setting with a value of 1, even if the application does not intend to use HTTP/3 Datagrams. This helps to avoid "sticking out"; see Section 4.

#### 2.1.1.1. Note About Draft Versions

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

Some revisions of this draft specification use a different value (the Identifier field of a Setting in the HTTP/3 SETTINGS frame) for the SETTINGS\_H3\_DATAGRAM setting. This allows new draft revisions to make incompatible changes. Multiple draft versions MAY be supported by sending multiple values for SETTINGS\_H3\_DATAGRAM. Once SETTINGS have been sent and received, an implementation that supports multiple drafts MUST compute the intersection of the values it has sent and received, and then it MUST select and use the most recent draft version from the intersection set. This ensures that both peers negotiate the same draft version.

#### 2.2. HTTP Datagrams using Capsules

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

### 3. Capsules

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

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

#### 3.1. HTTP Data Streams

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

In HTTP/1.x, the data stream consists of all bytes on the connection that follow the blank line that concludes either the request header section, or the response header section. As a result, only a single HTTP request starting the capsule protocol can be sent on HTTP/1.x connections.

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

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

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

### 3.2. The Capsule Protocol

Definitions of new HTTP Upgrade Tokens can state that their associated request's data stream uses the Capsule Protocol. If they do so, that means that the contents of the associated request's data stream uses the following format (using the notation from the "Notational Conventions" section of [QUIC]):

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

Figure 2: Capsule Protocol Stream Format

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

Figure 3: Capsule Format

**Capsule Type:** A variable-length integer indicating the Type of the capsule.

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

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

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

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

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



By virtue of the definition of the data stream:

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

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

### 3.3. Error Handling

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

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

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

### 3.4. The Capsule-Protocol Header Field

The "Capsule-Protocol" header field is an Item Structured Field, see Section 3.3 of [STRUCT-FIELD]; its value MUST be a Boolean; any other value type MUST be handled as if the field were not present by recipients (for example, if this field is included multiple times, its type will become a List and the field will therefore be ignored). This document does not define any parameters for the Capsule-Protocol header field value, but future documents might define parameters.

Receivers MUST ignore unknown parameters.

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

Intermediaries MAY use this header field to allow processing of HTTP Datagrams for unknown HTTP Upgrade Tokens; note that this is only possible for HTTP Upgrade or Extended CONNECT.

The Capsule-Protocol header field MUST NOT be used on HTTP responses with a status code outside the 2xx range.

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

### 3.5. The DATAGRAM Capsule

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

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

Figure 4: DATAGRAM Capsule Format

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

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

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

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

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

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

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

#### 4. Security Considerations

Since transmitting HTTP Datagrams using QUIC DATAGRAM frames requires sending the HTTP/3 SETTINGS\_H3\_DATAGRAM setting, it "sticks out". In other words, probing clients can learn whether a server supports HTTP Datagrams over QUIC DATAGRAM frames. As some servers might wish to obfuscate the fact that they offer application services that use HTTP datagrams, it's best for all implementations that support this feature to always send this setting, see Section 2.1.1.

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

#### 5. IANA Considerations

##### 5.1. HTTP/3 Setting

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

Value: 0x33  
Setting Name: SETTINGS\_H3\_DATAGRAM  
Default: 0  
Status: provisional (permanent if this document is approved)  
Specification: This Document  
Change Controller: IETF  
Contact: HTTP\_WG; HTTP working group; ietf-http-wg@w3.org

##### 5.2. HTTP/3 Error Code

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

Value: 0x33  
Name: H3\_DATAGRAM\_ERROR  
Description: Datagram or capsule protocol parse error  
Status: provisional (permanent if this document is approved)  
Specification: This Document  
Change Controller: IETF  
Contact: HTTP\_WG; HTTP working group; ietf-http-wg@w3.org

##### 5.3. HTTP Header Field Name

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

Field Name: Capsule-Protocol

Template: None  
Status: provisional (permanent if this document is approved)  
Reference: This document  
Comments: None

#### 5.4. Capsule Types

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

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

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

This registry initially contains the following entry:

Value: 0x00  
Capsule Type: DATAGRAM  
Status: permanent  
Specification: This document  
Change Controller: IETF  
Contact: MASQUE Working Group masque@ietf.org  
(mailto:masque@ietf.org)  
Notes: None

#### 6. References

##### 6.1. Normative References

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

- [HTTP] Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP Semantics", Work in Progress, Internet-Draft, draft-ietf-httpbis-semantics-19, 12 September 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-semantics-19>>.
- [HTTP/1.1] Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP/1.1", Work in Progress, Internet-Draft, draft-ietf-httpbis-messaging-19, 12 September 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-messaging-19>>.
- [HTTP/2] Thomson, M. and C. Benfield, "HTTP/2", Work in Progress, Internet-Draft, draft-ietf-httpbis-http2bis-07, 24 January 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-http2bis-07>>.
- [HTTP/3] Bishop, M., "Hypertext Transfer Protocol Version 3 (HTTP/3)", Work in Progress, Internet-Draft, draft-ietf-quic-http-34, 2 February 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-34>>.
- [IANA-POLICY] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/rfc/rfc8126>>.
- [QUIC] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <<https://www.rfc-editor.org/rfc/rfc9000>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/rfc/rfc2119>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/rfc/rfc8174>>.
- [STRUCT-FIELD] Nottingham, M. and P-H. Kamp, "Structured Field Values for HTTP", RFC 8941, DOI 10.17487/RFC8941, February 2021, <<https://www.rfc-editor.org/rfc/rfc8941>>.

## 6.2. Informative References

- [DPLPMTUD] Fairhurst, G., Jones, T., Tüxen, M., Rüngeler, I., and T. Völker, "Packetization Layer Path MTU Discovery for Datagram Transports", RFC 8899, DOI 10.17487/RFC8899, September 2020, <<https://www.rfc-editor.org/rfc/rfc8899>>.
- [EXT-CONNECT2] McManus, P., "Bootstrapping WebSockets with HTTP/2", RFC 8441, DOI 10.17487/RFC8441, September 2018, <<https://www.rfc-editor.org/rfc/rfc8441>>.
- [EXT-CONNECT3] Hamilton, R., "Bootstrapping WebSockets with HTTP/3", Work in Progress, Internet-Draft, draft-ietf-httpbis-h3-websockets-04, 8 February 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-h3-websockets-04>>.
- [PRIORITY] Oku, K. and L. Pardue, "Extensible Prioritization Scheme for HTTP", Work in Progress, Internet-Draft, draft-ietf-httpbis-priority-12, 17 January 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-priority-12>>.
- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <<https://www.rfc-editor.org/rfc/rfc6455>>.

## Acknowledgments

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

## Authors' Addresses

David Schinazi  
Google LLC  
1600 Amphitheatre Parkway  
Mountain View, CA 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: 28 February 2022

A. Chernyakhovsky  
D. McCall  
D. Schinazi  
Google LLC  
27 August 2021

Requirements for a MASQUE Protocol to Proxy IP Traffic  
draft-ietf-masque-ip-proxy-reqs-03

## 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](mailto: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

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 28 February 2022.

## Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
1.1. Conventions . . . . .	3
1.2. Definitions . . . . .	3
2. Use Cases . . . . .	4
2.1. Consumer VPN . . . . .	4
2.2. Point to Point Connectivity . . . . .	4
2.3. Point to Network Connectivity . . . . .	4
3. Requirements . . . . .	4
3.1. IP Session Establishment . . . . .	4
3.2. Proxying of IP packets . . . . .	5
3.3. Maximum Transmission Unit . . . . .	5
3.4. IP Assignment . . . . .	5
3.5. Identity . . . . .	5
3.6. Transport Security . . . . .	5
3.7. Flow Control . . . . .	6
3.8. Indistinguishability . . . . .	6
3.9. Support HTTP/2 and HTTP/3 . . . . .	6
3.10. Multiplexing . . . . .	6
3.11. Statefulness . . . . .	6
4. Extensibility . . . . .	6
4.1. Authentication . . . . .	7
4.2. Reliable Transmission of IP Packets . . . . .	7
4.3. Configuration of Congestion and Flow Control . . . . .	7
4.4. Data Transport Compression . . . . .	7
5. Non-requirements . . . . .	7
5.1. Addressing Architecture . . . . .	8
5.2. Translation . . . . .	8
5.3. IP Packet Extraction . . . . .	8
5.4. Trust . . . . .	9
6. Security Considerations . . . . .	9
7. IANA Considerations . . . . .	9
Acknowledgments . . . . .	9

References	9
Normative References	9
Informative References	10
Authors' Addresses	11

## 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](mailto: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.

## 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 tunnel endpoints to inform each other of the Maximum Transmission Unit (MTU) they are willing to forward. This will allow avoiding some IP fragmentation, especially as IPv6 does not allow IP fragmentation by nodes along the path. In cases where the tunnel endpoint is not the same as the communication endpoint, tunnel endpoints are expected to apply the guidance on UDP tunnels in [TUNNELS].

### 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. Endpoints will also have the ability to assign an IP address range to their peer, and to communicate that assignment to the peer, without having received a request.

### 3.5. 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.

### 3.6. 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.

### 3.7. 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.

### 3.8. 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.

### 3.9. 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.

### 3.10. 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.

### 3.11. 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 specifics 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.



#### 5.4. Trust

All the use-cases described in Section 2 require some level of trust between endpoints. However, how this trust is established and what decisions endpoints make based on this trust is considered out of scope. For example, if an endpoint doesn't sufficiently trust its peer, it would be well advised to validate the IP addresses used by that peer - however that is considered out of scope for the document that will describe an IP proxying protocol.

#### 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-03, 12 July 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-quic-datagram-03>>.
- [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://datatracker.ietf.org/doc/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://datatracker.ietf.org/doc/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-04, 12 July 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-masque-connect-udp-04>>.
- [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>>.

[TUNNELS] Touch, J. and M. Townsley, "IP Tunnels in the Internet Architecture", Work in Progress, Internet-Draft, draft-ietf-intarea-tunnels-10, 12 September 2019, <<https://datatracker.ietf.org/doc/html/draft-ietf-intarea-tunnels-10>>.

#### Authors' Addresses

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

Email: [achernya@google.com](mailto:achernya@google.com)

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

Email: [dallasmccall@google.com](mailto:dallasmccall@google.com)

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

Email: [dschinazi.ietf@gmail.com](mailto:dschinazi.ietf@gmail.com)

MASQUE  
Internet-Draft  
Intended status: Experimental  
Expires: 6 September 2022

T. Pauly  
Apple Inc.  
D. Schinazi  
Google LLC  
5 March 2022

QUIC-Aware Proxying Using HTTP  
draft-pauly-masque-quic-proxy-03

Abstract

This document defines an extension to UDP Proxying over HTTP that adds specific optimizations for proxied QUIC connections. 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 using an HTTP/3 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 6 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

1. Introduction . . . . .	2
1.1. Conventions and Definitions . . . . .	4
1.2. Terminology . . . . .	4
2. Required Proxy State . . . . .	5
2.1. Stream Mapping . . . . .	5
2.2. Target Connection ID Mapping . . . . .	5
2.3. Client Connection ID Mappings . . . . .	6
2.4. Detecting Connection ID Conflicts . . . . .	6
3. Connection ID Capsule Types . . . . .	7
4. Client Request Behavior . . . . .	8
4.1. New Proxied Connection Setup . . . . .	9
4.2. Adding New Client Connection IDs . . . . .	9
4.3. Sending With Forwarded Mode . . . . .	9
4.4. Receiving With Forwarded Mode . . . . .	10
5. Proxy Response Behavior . . . . .	11
5.1. Removing Mapping State . . . . .	12
5.2. Handling Connection Migration . . . . .	13
6. Example . . . . .	13
7. Interactions with Load Balancers . . . . .	15
8. Packet Size Considerations . . . . .	15
9. Security Considerations . . . . .	16
10. IANA Considerations . . . . .	16
10.1. HTTP Header . . . . .	16
10.2. Capsule Types . . . . .	17
11. References . . . . .	17
11.1. Normative References . . . . .	17
11.2. Informative References . . . . .	18
Acknowledgments . . . . .	18
Authors' Addresses . . . . .	18

## 1. Introduction

UDP Proxying over HTTP [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 UDP proxying protocol 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.

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 UDP proxying.
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.

The forwarding mode of this extension is only defined for HTTP/3 [HTTP3] and not any earlier versions of HTTP. The forwarding mode also requires special handling in order to be compatible with intermediaries or load balancers (see Section 7).

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 UDP proxying 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.
- \* 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.
- \* Target-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.
- \* Target 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 target-facing sockets for multiple connections and support the forwarding mode of proxying.

QUIC packets can be either tunnelled within an HTTP proxy connection using HTTP Datagram frames [HTTP-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. Stream Mapping

Each pair of client <-> proxy QUIC connection and an HTTP stream MUST be mapped to a single target-facing socket.

(Client <-> Proxy QUIC connection + Stream)  
=> Target-facing socket

Multiple streams can map to the same target-facing socket, but a single stream cannot be mapped to multiple target-facing sockets.

This mapping guarantees that any HTTP Datagram using a stream sent from the client to the proxy in tunnelled mode can be sent to the correct target.

### 2.2. Target Connection ID Mapping

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



(Client-facing socket + Target Connection ID)  
=> Target-facing socket

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

This mapping guarantees that any QUIC packet containing the Target 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 target-facing socket MUST map to a single stream on a single client <-> proxy QUIC connection. Additionally, the pair of Client Connection ID and target-facing socket MUST map to a single client-facing socket.

(Target-facing socket + Client Connection ID)  
=> (Client <-> Proxy QUIC connection + Stream)  
(Target-facing socket + Client Connection ID)  
=> Client-facing socket

Multiple pairs of Connection IDs and sockets can map to the same stream 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 target-facing socket for every client request with a unique stream. The mapping is critical for any case where target-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 Capsule Types

Proxy awareness of QUIC Connection IDs relies on using capsules ([HTTP-DGRAM]) to signal the addition and removal of client and Target Connection IDs.

Note that these capsules do not register contexts. QUIC packets are encoded using HTTP Datagrams with the context ID set to zero as defined in [CONNECT-UDP].

The capsules used for QUIC-aware proxying allow a client to register connection IDs with the proxy, and for the proxy to acknowledge or reject the connection ID mappings.

The REGISTER\_CLIENT\_CID and REGISTER\_TARGET\_CID capsule types (see Section 10.2 for the capsule type values) allow a client to inform the proxy about a new Client Connection ID or a new Target Connection ID, respectively. These capsule types MUST only be sent by a client.

The ACK\_CLIENT\_CID and ACK\_TARGET\_CID capsule types (see Section 10.2 for the capsule type values) are sent by the proxy to the client to indicate that a mapping was successfully created for a registered connection ID. These capsule types MUST only be sent by a proxy.

The CLOSE\_CLIENT\_CID and CLOSE\_TARGET\_CID capsule types (see Section 10.2 for the capsule type values) allow either a client or a proxy to remove a mapping for a connection ID. These capsule types MAY be sent by either a client or the proxy. If a proxy sends a CLOSE\_CLIENT\_CID without having sent an ACK\_CLIENT\_CID, or if a proxy sends a CLOSE\_TARGET\_CID without having sent an ACK\_TARGET\_CID, it is rejecting a Connection ID registration.

All Connection ID capsule types share the same format:

```
Connection ID Capsule {  
  Type (i) = 0xffe100..0xffe103,  
  Length (i),  
  Connection ID (0..2040),  
}
```

Figure 1: Connection ID Capsule Format

**Connection ID:** A connection ID being registered or acknowledged, which is between 0 and 255 bytes in length. The length of the connection ID is implied by the length of the capsule. Note that in QUICv1, the length of the Connection ID is limited to 20 bytes, but QUIC invariants allow up to 255 bytes.

#### 4. Client Request Behavior

A client initiates UDP proxying via a CONNECT request as defined in [CONNECT-UDP]. Within its request, it includes the "Proxy-QUIC-Forwarding" header to indicate whether or not the request should support forwarding. If this header is not included, the client **MUST NOT** send any connection ID capsules.

The "Proxy-QUIC-Forwarding" is an Item Structured Header [RFC8941]. Its value **MUST** be a Boolean. Its ABNF is:

```
Proxy-QUIC-Forwarding = sf-boolean
```

If the client wants to enable QUIC packet forwarding for this request, it sets the value to "?1". If it doesn't want to enable forwarding, but instead only provide information about QUIC Connection IDs for the purpose of allowing the proxy to share a target-facing socket, it sets the value to "?0".

If the proxy supports QUIC-aware proxying, it will include the "Proxy-QUIC-Forwarding" header in successful HTTP responses. The value indicates whether or not the proxy supports forwarding. If the client does not receive this header in responses, the client **SHALL** assume that the proxy does not understand how to parse Connection ID capsules, and **MUST NOT** send any Connection ID capsules.

The client sends a REGISTER\_CLIENT\_CID capsule whenever it advertises a new Client Connection ID to the target, and a REGISTER\_TARGET\_CID capsule when it has received a new Target Connection ID for the target. Note that the initial REGISTER\_CLIENT\_CID capsule **MAY** be sent prior to receiving an HTTP response from the proxy.

#### 4.1. New Proxied Connection Setup

To initiate QUIC-aware proxying, the client sends a REGISTER\_CLIENT\_CID capsule containing the initial Client Connection ID that the client has advertised to the target.

If the mapping is created successfully, the client will receive a ACK\_CLIENT\_CID capsule that contains the same connection ID that was requested.

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 CLOSE\_CLIENT\_CID capsule. 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.

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 REGISTER\_CLIENT\_CID capsule request containing the new Client Connection ID.

The client should only inform the target of the new Client Connection ID once an ACK\_CLIENT\_CID capsule is received that contains the echoed connection ID.

#### 4.3. Sending With Forwarded Mode

Support for forwarding mode is determined by the "Proxy-QUIC-Forwarding" header, see Section 5.

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 REGISTER\_TARGET\_CID capsule containing the Target Connection ID to request the ability to forward packets.

The client MUST wait for an ACK\_TARGET\_CID capsule that contains the echoed connection ID before using forwarded mode.

Prior to receiving the proxy server response, the client MUST send short header packets tunnelled in HTTP Datagram frames. The client MAY also choose to tunnel some short header packets even after receiving the successful response.

If the Target Connection ID registration is rejected, for example with a CLOSE\_TARGET\_CID capsule, it MUST NOT forward packets to the requested Target 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.

QUIC long header packets MUST NOT be forwarded. These packets can only be tunnelled within HTTP 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

If the client has indicated support for forwarding with the "Proxy-QUIC-Forwarding" header, the proxy MAY use forwarded mode for any Client Connection ID for which it has a valid mapping.

Once a client has sent "Proxy-QUIC-Forwarding" with a value of "?1", it MUST be prepared to receive forwarded short header packets on the socket between itself and the proxy for any Client Connection ID that it has registered with a REGISTER\_CLIENT\_CID capsule. The client uses the Destination Connection ID field of the received packet to determine if the packet was originated by the proxy, or merely forwarded from the target.

## 5. Proxy Response Behavior

Upon receipt of a CONNECT request that includes the "Proxy-QUIC-Forwarding" header, the proxy indicates to the client that it supports QUIC-aware proxying by including a "Proxy-QUIC-Forwarding" header in a successful response. If it supports QUIC packet forwarding, it sets the value to "?1"; otherwise, it sets it to "?0".

Upon receipt of a REGISTER\_CLIENT\_CID or REGISTER\_TARGET\_CID capsule, the proxy validates the registration, tries to establish the appropriate mappings as described in Section 2.

The proxy MUST reply to each REGISTER\_CLIENT\_CID capsule with either an ACK\_CLIENT\_CID or CLOSE\_CLIENT\_CID capsule containing the Connection ID that was in the registration capsule.

Similarly, the proxy MUST reply to each REGISTER\_TARGET\_CID capsule with either an ACK\_TARGET\_CID or CLOSE\_TARGET\_CID capsule containing the Connection ID that was in the registration capsule.

The proxy then determines the target-facing socket to associate with the client's request. This will generally involve performing a DNS lookup for the target hostname in the CONNECT request, or finding an existing target-facing socket to the authority. The target-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 target-facing sockets across multiple UDP proxying requests, or have a unique target-facing socket for every UDP proxying request.

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

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

When the proxy receives a REGISTER\_CLIENT\_CID capsule, it 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 target-facing socket does not create a conflict, the proxy creates the mapping and responds with a ACK\_CLIENT\_CID capsule.

After this point, any packets received by the proxy from the target-facing socket that match the Client Connection ID can be sent to the client. The proxy MUST use tunnelled mode (HTTP Datagram frames) for any long header packets. The proxy SHOULD forward directly to the client for any matching short header packets if forwarding is supported by the client, but the proxy MAY tunnel these packets in HTTP Datagram frames instead. If the mapping would create a conflict, the proxy responds with a `CLOSE_CLIENT_CID` capsule.

When the proxy receives a `REGISTER_TARGET_CID` capsule, it is receiving a request to allow the client to forward packets to the target. If the pair of this Target 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 `ACK_TARGET_CID` capsule. Once the successful response is sent, the proxy will forward any short header packets received on the client-facing socket that use the Target Connection ID using the correct target-facing socket. If the pair is not unique, the proxy responds with a `CLOSE_TARGET_CID` capsule. If this occurs, traffic for that Target 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 all `REGISTER_TARGET_CID` requests with `CLOSE_TARGET_CID` capsule.

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 Target 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

For any connection ID for which the proxy has sent an acknowledgement, any mappings for the connection ID last until either endpoint sends a close capsule or the either side of the HTTP stream closes.

A client that no longer wants a given Connection ID to be forwarded by the proxy sends a `CLOSE_CLIENT_CID` or `CLOSE_TARGET_CID` capsule.

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 target-facing socket once all UDP proxying requests 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. In order to inform a proxy of the new QUIC Client Connection ID, the client also sends a REGISTER\_CLIENT\_CID capsule.

The client will also send the initial QUIC packet with the Long Header form in an HTTP datagram.



Client

Server

```
STREAM(44): HEADERS          ----->
  :method = CONNECT
  :protocol = connect-udp
  :scheme = https
  :path = /target.example.com/443/
  :authority = proxy.example.org
  proxy-quic-forwarding = ?1
  capsule-protocol = ?1
```

```
STREAM(44): DATA            ----->
  Capsule Type = REGISTER_CLIENT_CID
  Connection ID = 0x31323334
```

```
DATAGRAM                      ----->
  Quarter Stream ID = 11
  Context ID = 0
  Payload = Encapsulated QUIC initial
```

```
<-----  STREAM(44): HEADERS
           :status = 200
           proxy-quic-forwarding = ?1
           capsule-protocol = ?1
```

```
<-----  STREAM(44): DATA
           Capsule Type = ACK_CLIENT_CID
           Connection ID = 0x31323334
```

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

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

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 for forwarding. In this case, that ID is 0x61626364. The client sends the following capsule:

```
STREAM(44): DATA          ----->
  Capsule Type = REGISTER_TARGET_CID
  Connection ID = 0x61626364

<----- STREAM(44): DATA
  Capsule Type = ACK_TARGET_CID
  Connection ID = 0x61626364
```

Upon receiving an ACK\_TARGET\_CID capsule, 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 target 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 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 ACK\_CLIENT\_CID or ACK\_TARGET\_CID capsules to clients.

QUIC-aware 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 HTTP Datagram frames that are used for a QUIC-aware 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 an HTTP 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 packet 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 HTTP 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 Header

This document registers the "Proxy-QUIC-Forwarding" header in the "Permanent Message Header Field Names" <<https://www.iana.org/assignments/message-headers>>.

Header Field Name	Protocol	Status	Reference
Proxy-QUIC-Forwarding	http	exp	This document

Figure 2: Registered HTTP Header

## 10.2. Capsule Types

This document registers six new values in the "HTTP Capsule Types" registry established by [HTTP-DGRAM].

Capule Type	Value	Specification
REGISTER_CLIENT_CID	0xffe100	This Document
REGISTER_TARGET_CID	0xffe101	This Document
ACK_CLIENT_CID	0xffe102	This Document
ACK_TARGET_CID	0xffe103	This Document
CLOSE_CLIENT_CID	0xffe104	This Document
CLOSE_TARGET_CID	0xffe105	This Document

Table 1: Registered Capsule Types

## 11. References

### 11.1. Normative References

#### [CONNECT-UDP]

Schinazi, D., "UDP Proxying Support for HTTP", Work in Progress, Internet-Draft, draft-ietf-masque-connect-udp-07, 4 March 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-masque-connect-udp-07>>.

#### [HTTP-DGRAM]

Schinazi, D. and L. Pardue, "Using Datagrams with HTTP", Work in Progress, Internet-Draft, draft-ietf-masque-h3-datagram-06, 4 March 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-masque-h3-datagram-06>>.

#### [HTTP3]

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

## [INVARIANTS]

Thomson, M., "Version-Independent Properties of QUIC",  
RFC 8999, DOI 10.17487/RFC8999, May 2021,  
<<https://www.rfc-editor.org/rfc/rfc8999>>.

## [QUIC]

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

## [RFC2119]

Bradner, S., "Key words for use in RFCs to Indicate  
Requirement Levels", BCP 14, RFC 2119,  
DOI 10.17487/RFC2119, March 1997,  
<<https://www.rfc-editor.org/rfc/rfc2119>>.

## [RFC8174]

Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC  
2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174,  
May 2017, <<https://www.rfc-editor.org/rfc/rfc8174>>.

## [RFC8941]

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

## 11.2. Informative References

## [QUIC-LB]

Duke, M., Banks, N., and C. Huitema, "QUIC-LB: Generating  
Routable QUIC Connection IDs", Work in Progress, Internet-  
Draft, draft-ietf-quic-load-balancers-12, 11 February  
2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-quic-load-balancers-12>>.

## 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](mailto:tpauly@apple.com)

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