

MASQUE  
Internet-Draft  
Intended status: Standards Track  
Expires: 28 February 2022

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

The CONNECT-IP HTTP Method  
draft-cms-masque-connect-ip-02

Abstract

This document describes the CONNECT-IP HTTP method. CONNECT-IP is similar to CONNECT-UDP, but allows transmitting IP packets, without being limited to just TCP like CONNECT or UDP like CONNECT-UDP.

Discussion Venues

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

Discussion of this document takes place on the Multiplexed Application Substrate over QUIC Encryption Working Group mailing list ([masque@ietf.org](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/DavidSchinazi/draft-cms-masque-connect-ip>.

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 . . . . .	2
1.1. Conventions and Definitions . . . . .	3
2. The CONNECT-IP Method . . . . .	3
3. Transmitting IP Packets using HTTP Datagrams . . . . .	4
4. Forwarding of IP Packets . . . . .	4
5. Capsules . . . . .	5
5.1. ADDRESS_ASSIGN Capsule . . . . .	5
5.2. ADDRESS_REQUEST Capsule . . . . .	6
5.3. SHUTDOWN Capsule . . . . .	6
6. Extensibility Considerations . . . . .	7
7. Security Considerations . . . . .	7
8. IANA Considerations . . . . .	7
8.1. HTTP Method . . . . .	7
8.2. Capsule Type Registrations . . . . .	8
9. References . . . . .	8
9.1. Normative References . . . . .	8
9.2. Informative References . . . . .	9
Appendix A. Examples . . . . .	9
A.1. Consumer VPN . . . . .	9
Acknowledgments . . . . .	10
Authors' Addresses . . . . .	10

## 1. Introduction

This document describes the CONNECT-IP HTTP method. CONNECT-IP is similar to CONNECT-UDP, but allows transmitting IP packets, without being limited to just TCP like CONNECT or UDP like CONNECT-UDP.

CONNECT-IP allows endpoints to set up an IP tunnel between one another. This can be used to implement a consumer VPN, point-to-point and point-to-network capabilities as described in [REQS].

### 1.1. Conventions and Definitions

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

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

### 2. The CONNECT-IP Method

The CONNECT-IP method establishes a stream to an endpoint server that then permits the exchange of control data, such as IP address information and other relevant information for successfully transmitting IP datagrams between hosts.

The request-target of a CONNECT-IP request is a URI [URI] which uses the "https" scheme and a client-specified path. When using HTTP/2 [H2] or later, CONNECT-IP requests use HTTP pseudo-headers with the following requirements:

- \* The ":method" pseudo-header field is set to "CONNECT-IP".
- \* The ":scheme" pseudo-header field is set to "https".
- \* The ":path" pseudo-header field is set to the value provided by the client. That value MUST NOT be empty.
- \* The ":authority" pseudo-header field contains the host and port of the proxy. The target of a CONNECT-IP request is the server providing the CONNECT-IP featureset, not an individual endpoint with which a connection is desired.

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

Any 2xx (Successful) response indicates that the proxy is willing to open an IP tunnel between it and the client. Any response other than a successful response indicates that the tunnel has not yet been formed.

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

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

Responses to the CONNECT-IP method are not cacheable.

The lifetime of the tunnel is tied to the CONNECT-IP stream. Closing the stream (via the FIN bit on a QUIC STREAM frame, or a QUIC RESET\_STREAM frame) closes the associated tunnel.

### 3. Transmitting IP Packets using HTTP Datagrams

IP packets are sent using HTTP Datagrams [HTTP-DGRAM]. The HTTP Datagram Payload contains a full IP packet, from the IP Version field until the last byte of the IP Payload. In order to use HTTP Datagrams, the CONNECT-IP client will first decide whether or not to use HTTP Datagram Contexts and then register its context ID (or lack thereof) using the corresponding registration capsule, see [HTTP-DGRAM].

Since HTTP Datagrams require prior negotiation (for example, in HTTP/3 it is necessary to both send and receive the H3\_DATAGRAM SETTINGS Parameter), clients MUST NOT send any HTTP Datagrams until they have established support on a given connection. If negotiation of HTTP Datagrams fails (for example if an HTTP/3 SETTINGS frame was received without the H3\_DATAGRAM SETTINGS Parameter), the client MUST consider its CONNECT-IP request as failed.

### 4. Forwarding of IP Packets

Since CONNECT-IP allows the transmission of IP packets over HTTP, CONNECT-IP endpoints will most often forward these packets to and from traditional IP interfaces. As such, CONNECT-IP endpoints act as IP routers. When a CONNECT-IP endpoint receives an HTTP Datagram containing an IP packet, it will parse the packet's IP header, perform any local policy checks (e.g., source address validation), check their routing table to pick an outbound interface, and then use an implementation-specific mechanism (such as raw sockets) to send the IP packet on that interface.

Conversely, when a CONNECT-IP endpoint receives an IP packet whose destination address does not match any local addresses, it consults its routing table to pick a forwarding destination, and if the table points to a CONNECT-IP tunnel, the endpoint performs the same forwarding checks before transmitting the packet inside the tunnel.

Note that CONNECT-IP endpoints will decrement the IP Hop Count (or TTL) upon encapsulation but not decapsulation. In other words, the Hop Count is decremented right before an IP packet is transmitted in an HTTP Datagram. This prevents infinite loops in the presence of routing loops, and matches the choices in IPsec [IPSEC].

Endpoints MAY implement additional filtering policies on the IP packets they forward.

## 5. Capsules

### 5.1. ADDRESS\_ASSIGN Capsule

The ADDRESS\_ASSIGN capsule allows an endpoint to inform its peer that it has assigned an IP address to it. It allows assigning a prefix which can contain multiple addresses. This capsule uses a Capsule Type of 0xffff100. Its value uses the following format:

```
ADDRESS_ASSIGN Capsule {  
    IP Version (8),  
    IP Address (32..128),  
    IP Prefix Length (8),  
}
```

Figure 1: ADDRESS\_ASSIGN Capsule Format

IP Version: IP Version of this address assignment. MUST be either 4 or 6.

IP Address: Assigned IP address. If the IP Version field has value 4, the IP Address field SHALL have a length of 32 bits. If the IP Version field has value 6, the IP Address field SHALL have a length of 128 bits.

IP Prefix Length: Length of the IP Prefix assigned, in bits. MUST be lesser or equal to the length of the IP Address field, in bits.

## 5.2. ADDRESS\_REQUEST Capsule

The ADDRESS\_REQUEST capsule allows an endpoint to request assignment of an IP address from its peer. It allows the endpoint to optionally indicate a preference for which address it would get assigned. This capsule uses a Capsule Type of 0xffff101. Its value uses the following format:

```
ADDRESS_REQUEST Capsule {  
    IP Version (8),  
    IP Address (32..128),  
    IP Prefix Length (8),  
}
```

Figure 2: ADDRESS\_REQUEST Capsule Format

**IP Version:** IP Version of this address request. MUST be either 4 or 6.

**IP Address:** Requested IP address. If the IP Version field has value 4, the IP Address field SHALL have a length of 32 bits. If the IP Version field has value 6, the IP Address field SHALL have a length of 128 bits.

**IP Prefix Length:** Length of the IP Prefix requested, in bits. MUST be lesser or equal to the length of the IP Address field, in bits.

Upon receiving the ADDRESS\_REQUEST capsule, an endpoint SHOULD assign an IP address to its peer, and then respond with an ADDRESS\_ASSIGN capsule to inform the peer of the assignment.

## 5.3. SHUTDOWN Capsule

The SHUTDOWN capsule allows an endpoint to communicate to its peer that it is about to close the CONNECT-IP stream, with a string explaining the reason for the shutdown. This capsule uses a Capsule Type of 0xffff105. Its value uses the following format:

```
SHUTDOWN Capsule {  
    Reason Phrase (..),  
}
```

Figure 3: SHUTDOWN Capsule Format

**Reason Phrase:** Additional diagnostic information for the shutdown.

This SHOULD be a UTF-8 encoded string [UTF8], though the frame does not carry information, such as language tags, that would aid comprehension by any entity other than the one that created the text.

Note that the SHUTDOWN capsule is informational, the tunnel is only closed when its corresponding CONNECT-IP stream is closed. Endpoints MAY close the tunnel with a reason phrase by sending the SHUTDOWN capsule with the FIN bit set on the underlying QUIC STREAM frame that carried it.

## 6. Extensibility Considerations

CONNECT-IP can be extended via multiple mechanisms to increase functionality. There are three main ways to extend CONNECT-IP: HTTP headers, Capsule Types, and HTTP Datagram Registration Extension Data. For example, an authentication extension could define an HTTP header that allows endpoints to send authentication credentials to their peer during the creation of the tunnel. Alternatively, one could specify an extension that defines a new Capsule Type which allows exchanging DNS configuration between endpoints. Additionally, an extension to CONNECT-IP can use multiple HTTP Datagram Contexts [HTTP-DGRAM] simultaneously to compress some IP packets by associating the compression context with an HTTP Datagram Context ID.

## 7. Security Considerations

There are significant risks in allowing arbitrary clients to establish a tunnel to arbitrary servers, as that could allow bad actors to send traffic and have it attributed to the proxy. Proxies that support CONNECT-IP SHOULD restrict its use to authenticated users. The HTTP Authorization header [AUTH] MAY be used to authenticate clients. More complex authentication schemes are out of scope for this document but can be implemented using CONNECT-IP extensions.

Since CONNECT-IP endpoints can proxy IP packets sent by their peer, they SHOULD follow the guidance in [BCP38] to help prevent denial of service attacks.

## 8. IANA Considerations

### 8.1. HTTP Method

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

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

## 8.2. Capsule Type Registrations

This document will request IANA to add the following values to the "HTTP Capsule Types" registry created by [HTTP-DGRAM]:

Value	Type	Description	Reference
0xffff100	ADDRESS_ASSIGN	Address Assignment	This document
0xffff101	ADDRESS_REQUEST	Address Request	This document
0xffff105	SHUTDOWN	Shutdown Reason	This document

## 9. References

### 9.1. Normative References

- [BCP38] Ferguson, P. and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing", BCP 38, RFC 2827, DOI 10.17487/RFC2827, May 2000, <<https://www.rfc-editor.org/rfc/rfc2827>>.
- [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>>.
- [HTTP-DGRAM] Schinazi, D. and L. Pardue, "Using Datagrams with HTTP", Work in Progress, Internet-Draft, draft-ietf-masque-h3-datagram-03, 12 July 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-masque-h3-datagram-03>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/rfc/rfc2119>>.



- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/rfc/rfc7230>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/rfc/rfc8174>>.
- [URI] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/rfc/rfc3986>>.
- [UTF8] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <<https://www.rfc-editor.org/rfc/rfc3629>>.

## 9.2. Informative References

- [AUTH] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, <<https://www.rfc-editor.org/rfc/rfc7235>>.
- [IPSEC] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <<https://www.rfc-editor.org/rfc/rfc4301>>.
- [REQS] Chernyakhovsky, A., McCall, D., and D. Schinazi, "Requirements for a MASQUE Protocol to Proxy IP Traffic", Work in Progress, Internet-Draft, draft-ietf-masque-ip-proxy-reqs-03, 27 August 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-masque-ip-proxy-reqs-03>>.

## Appendix A. Examples

### A.1. Consumer VPN

In this scenario, the client will typically receive a single IP address that the proxy has picked from a pool of addresses it maintains. The client will route all traffic through the tunnel. The exchange could look as follows:

Client

Server

```
ADDRESS_REQUEST      ----->
  IP Version = 4
  IP Address = 0.0.0.0
  IP Prefix Length = 0

<----- ADDRESS_ASSIGN
  IP Version = 4
  IP Address = 192.0.2.42
  IP Prefix Length = 32
```

#### Acknowledgments

The design of CONNECT-IP was inspired by discussions in the MASQUE working group around [REQS]. The authors would like to thank participants in those discussions for their feedback.

#### 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: 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

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

M. Kuehlewind  
M. Westerlund  
M. Ihlar  
Z. Sarker  
Ericsson  
12 July 2021

The CONNECT-IP HTTP method for proxying IP traffic  
draft-kuehlewind-masque-connect-ip-01

## Abstract

This draft specifies a new HTTP method CONNECT-IP to proxy IP traffic. CONNECT-IP uses HTTP/3 Datagrams to use QUIC Datagrams for efficient transport of proxied IP packets, with the possibility to fallback to HTTP/3 over reliable QUIC streams, or even HTTP 1.x and 2.

CONNECT-IP supports two modes: a tunneling mode where IP packets are forwarded without modifications and flow forwarding mode which supports optimization for individual IP flows forwarded to the targeted peer. To request tunneling or flow forwarding, a client connects to a proxy server by initiating a HTTP/3 connection and sends a CONNECT-IP request which either indicates the address of the proxy or the target peer. The proxy then forwards payload received on that stream or in an HTTP datagram with a certain stream ID.

## Discussion Venues

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

Discussion of this document takes place on the MASQUE Working Group mailing list ([masque@ietf.org](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/mirjak/draft-kuehlewind-masque-connect-ip>.

## 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 January 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
1.1.	Tunnel mode
1.2.	Flow Forwarding mode
1.2.1.	Motivation of IP flow model for flow forwarding
1.3.	Definitions
2.	The CONNECT-IP method
2.1.	Data encapsulation
2.2.	Datagram Formats
2.2.1.	Tunnel Mode IPv4 Format
2.2.2.	Tunnel Mode IPv6 Format
2.2.3.	Flow Forwarding Format
2.2.4.	ICMP Message Format
3.	HTTP Headers
3.1.	IP-Protocol Header for CONNECT-IP
3.2.	IP-Version header for CONNECT-IP
3.3.	IP-Address header for CONNECT-IP
3.4.	IP-Address-Handling Header for CONNECT-IP
3.5.	Conn-ID Header for CONNECT-IP
4.	Client Connect-IP Request
4.1.	Requesting flow forwarding
4.2.	Requesting tunnel mode
5.	MASQUE server behavior
5.1.	Error handling
5.2.	IP address selection in flow forwarding mode
5.3.	Constructing the IP header in flow forwarding mode
5.4.	Decapsulation of tunnel mode IP Packets
5.5.	Receiving an IP packet
6.	Additional signalling
6.1.	ECN
6.2.	ICMP handling
6.3.	MTU considerations
7.	Examples
8.	Security considerations
9.	IANA considerations
9.1.	HTTP Method
9.2.	HTTP Header
	Acknowledgments
	References
	Normative References
	Informative References
	Authors' Addresses

## 1. Introduction

This document specifies the CONNECT-IP method for IPv4 [RFC0791] and IPv6 [RFC8200] tunneling and flow forwarding over HTTP/3.

CONNECT-IP supports two modes: a tunneling mode where IP packets are

forwarded without modifications and flow forwarding mode which supports optimization for individual IP flows forwarded to the targeted peer.

### 1.1. Tunnel mode

In tunnel mode the client requests to tunnel IP packets to and from one or more servers via the proxy. The Connect-IP request to the proxy establishes such a tunnel and optionally indicates the IP address or IP address range that will be allowed to be used by and forwarded to the client.

The tunnel mode is indicated by the ":authority" pseudo-header field of the CONNECT-IP request contain the host and listing port of the proxy itself. In this mode the proxy just blindly forwards all payload on its external interface without any modification and also forwards all incoming traffic to registered clients as payload within the respective tunnel association. That means all incoming traffic, where the destination address matches an by the client indicated IP address or range of IP addresses, is forwarded to the client over the tunnel association, except a more specific flow forwarding association exists where both destination and source IP address as well as any additionally used identifier match (see section Section 5.5).

However, a proxy MUST offer this service only for known clients and clients MUST be authenticated during connection establishment. The proxy SHOULD inspect the source IP address of the IP packet in the tunnel payload and only forward if the IP address matches the set of client IP addresses. Optionally, a proxy also MAY offer this service only for a limited set of target addresses. In such a case the proxy SHOULD also inspect the destination IP address of the tunnel payload as well as the source address of incoming packets from target servers and reject packets with unknown addresses with an error.

### 1.2. Flow Forwarding mode

In flow forwarding mode the CONNECT-IP method establishes an outgoing IP flow, from the MASQUE server's external address to the target server's address specified by the client for a particular upper layer protocol. This mode also enables reception and relaying of the reverse IP flow from the target address to the MASQUE server to ensure that return traffic can be received by the client. However, it does not support flow establishment by an external peer. This specification supports forwarding of incoming traffic to one of the clients only if an active mapping has previously been created based on an IP-CONNECT request. Clients that need to support reception of flows established by external peer need to use tunnel mode.

This mode covers the point-to-point use case [I-D.ietf-masque-ip-proxy-reqs] and allows for flow-based optimizations and a larger effective maximum packet size through the tunnel. The target IP address is provided by the client as part of the CONNECT-IP request. The sources address is either independently selected by the proxy or can be requested to be either the same as used in a previous and currently active CONNECT-IP request or different from currently requests by the same client. The client also indicates the upper layer protocol, thus defining the three tuple used as primary selector for the flow.

In this mode the payload between the client and proxy does not contain the IP header in order to reduce overhead. Any additional

information (other than the source and destination IP addresses and ports as well as the upper layer protocol identifier) that is needed to construct the IP header or to inform the client about information from received IP packets can be signalled as part of the CONNECT-IP request or using HTTP/3 Datagram [I-D.ietf-masque-h3-datagram] later.

In flow forwarding mode, usually one upper-layer end-to-end connection is associated to one CONNECT-IP forwarding association. While it would be possible for a client to use the same forwarding association for multiple end-to-end connections to the same target server, as long as they all require the same Protocol (IPv4) / Next Header (IPv6) value, this would lead to the use of the same flow ID for all connections. As such, this is not recommended for connection-oriented transmissions. In order to enable multiple flow forwarding associations to the same server, the flow forwarding mode supports a way to specify some additional upper layer protocol selectors, e.g. TCP source and destination port, to enable multiple CONNECT-IP request for the same three tuple, see CONN-ID header Section 3.5.

The default model for address handling in this specification is that the proxy (Masque Server) will have a pool of one or more IP addresses that it can lend to the MASQUE client and routable over its external interface. Other potential use cases and address handling are possible, potentially requiring further extensions.

This proposal is based on the analysis provided in [I-D.westerlund-masque-transport-issues] indicating that most information in the IP header is either IP flow related or can or even should be provided by the proxy as the IP communication endpoint without the need for input from the client. The most crucial information identified that requires client interaction is ECN [RFC3168] and ICMP [RFC0792] [RFC4443] handling.

This document defines the following IP header field treatment.

Required to be determined in Connect-IP request and response:

- \* IP version
- \* IP Source Address
- \* IP Destination Address (target address)
- \* Upper Layer Protocol (IPv4 Protocol field / IPv6 Next Header field)

Can be chosen by Proxy on transmission:

- \* IPv6 Flow label (per Connect-IP flow mode request)
- \* IPv4 Time to live / IPv6 Hop Limit (proxy configured)
- \* Diffserv Codepoint, default is set to 0 (Best Effort)

May optionally be provided on a per packet basis

- \* Explicit Congestion Notification in both directions.

The consequence of this is certain limitations that future extension can address. For packets that are sent from the target server to the client, the client will not get any information on the actual value

of TTL/Hop Count, DSCP, or flow label when received by the proxy. Instead these field are set and consumed by the proxy only.

Signalling of other dedicated values may be desired in certain deployments, e.g for DCSP [RFC2474]. However, DSCP is in any case a challenge due to local domain dependency of the used DSCP values and the forwarding behavior and traffic treatment they represent. Future use cases for DSCP, as well as new IPv6 extension headers or destination header options [RFC8200] may require additional signaling. Therefore, it is important that the signaling is extensible.

#### 1.2.1. Motivation of IP flow model for flow forwarding

The chosen IP flow model is selected due to several advantages:

- \* Minimized per packet overhead: The per packet overhead is reduced to basic framing of the IP payload for each IP packet and flow identifiers. This enables a larger effective Maximum Transmission Unit (MTU) than tunnel mode.
- \* Shared functionality with CONNECT-UDP: The UDP flow proxying functionality of CONNECT-UDP will need to establish, store and process the same IP header related fields and state. So this can be accomplished by simply removing the UDP specific processing of packets.
- \* CONNECT-IP can establish a new IP flow in 0-RTT: No network related latencies in establishing new flow.

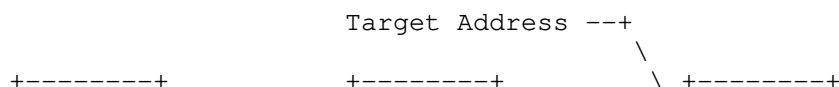
Disadvantages of this model are the following:

- \* Client to server focused solution: Accepting non-solicited peer-initiated traffic is not supported.

#### 1.3. Definitions

- \* Proxy: This document uses proxy as synonym for the MASQUE Server or an HTTP proxy, depending on context.
- \* Client: The endpoint initiating a MASQUE tunnel and IP relaying with the proxy.
- \* Target host: A remote endpoint the client wishes to establish bi-directional communication with via tunnelling over the proxy.
- \* IP proxying: A proxy forwarding IP payloads to a target for an IP flow. Data is decapsulate at the proxy and amended by a IP header before forwarding to the target. Packet boundaries need to be preserved or signalled between the client and proxy.
- \* IP flow: A flow of IP packets between two hosts as identified by their IP addresses, and where all the packets share some properties. These properties include source/destination address, protocol / next header field, flow label (IPv6 only), and DSCP per direction.

Address = IP address



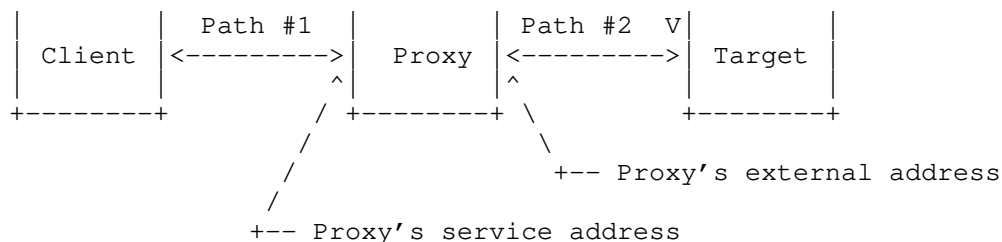


Figure 1: The nodes and their addresses

Figure 1 provides an overview figure of the involved nodes, i.e. client, proxy, and target host. There are also two network paths. Path #1 is the client to proxy path, where IP proxying is provided over an HTTP/3 session, usually over QUIC, to tunnel IP flow(s). Path #2 is the path between the proxy and the target.

The client will use the proxy's service address to establish a transport connection on which to request IP proxying using HTTP/3 CONNECT-IP. The proxy will then relay the client's IP flows to the target host. The IP header from the proxy to the target carries the proxy's external address as source address and the target's address as destination address.

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. The CONNECT-IP method

This document defines a new HTTP [I-D.ietf-httpbis-semantics] method CONNECT-IP to convert streams into tunnels or initialize HTTP datagram flows [I-D.ietf-masque-h3-datagram] to a forwarding proxy. Each stream can be used separately to establish forwarding to potentially different remote hosts. Unlike the HTTP CONNECT method, CONNECT-IP does not request the proxy to establish a TCP connection to the remote target host. Instead the tunnel payload will be forwarded as individual IP packets (tunnel mode) or right on top of the IP layer (flow forwarding), meaning the proxy has to identify messages boundaries to each message before forwarding (see section Section 5).

This document specifies CONNECT-IP for HTTP following the same semantics as the CONNECT method. As such a CONNECT-IP request MUST be constructed as follows:

- \* The ":method" pseudo-header field is set to "CONNECT-IP"
- \* The ":scheme" and ":path" pseudo-header fields are omitted
- \* The ":authority" pseudo-header field contains either the host address to connect to (equivalent to the authority-form of the request-target of CONNECT-UDP [I-D.ietf-masque-connect-udp] or CONNECT requests; see Section 3.2.3 of [I-D.ietf-httpbis-messaging]) or the host and port of the proxy if tunnel mode is requested

A CONNECT request that does not conform to these restrictions is malformed; see Section 4.1.3 of [I-D.ietf-quic-http].



Unlike the CONNECT method, CONNECT-IP does not sequentially trigger a connection establishment process from the proxy to the target host. Therefore, the client does not need to wait for an HTTP response in order to send forwarding data, unless in tunnel mode and requesting assignment of an external IP address. However, the client, especially on tunnel mode, SHOULD limit the amount of traffic sent to the proxy before a 2xx (Successful) response is received.

The forwarding stays active as long as the respective stream is open. A Forwarded IP packet can be either an encapsulated HTTP datagram on the same HTTP stream as the CONNECT-IP request, or as a HTTP datagram sent over QUIC datagram.

## 2.1. Data encapsulation

Once the CONNECT-IP method has completed, only CAPSULE [I-D.ietf-masque-h3-datagram] frames are permitted to be sent on that stream. Extension frames MAY be used if specifically permitted by the definition of the extension. Receipt of any other known frame type MUST be treated as a connection error of type H3\_FRAME\_UNEXPECTED.

Each HTTP Datagram frame contains one of the below specified data formats (Section 2.2) depending on request forwarding mode and given headers and parameters.

Stream based forwarding provides in-order and reliable delivery but may introduce Head of Line (HoL) Blocking if independent messages are sent over the same CONNECT-IP association. On streams payload data is encapsulated in the CAPSULE Frame using the DATAGRAM capsule (type=0x02) [I-D.ietf-masque-h3-datagram].

The client can, in addition to stream-based forwarding, request use of HTTP/3 datagrams [I-D.ietf-masque-h3-datagram].

To request datagram support the client sends H3\_DATAGRAM SETTINGS parameter with a value of 1 [I-D.ietf-masque-h3-datagram]. Datagram support MUST only be requested when the QUIC datagram extension [I-D.ietf-quic-datagram] was successfully negotiated during the QUIC handshake.

Datagrams provide un-ordered and unreliable delivery. In theory both, stream- as well as datagram-based forwarding, can be used in parallel, however, for most transmissions it is expected to only use one.

While IP packets sent over streams only have to respect the end-to-end MTU between the client and the target server, packets sent in datagrams are further restricted by the QUIC packet size of the QUIC tunnel and any overhead within the QUIC tunnel packet. Ideally, the proxy can provide MTU and overhead information to the client. The client MUST take the estimated overhead into account when indicating the MTU to the application (see section Section 6.3).

## 2.2. Datagram Formats

This section defines the different datagram formats used by Connect-IP. Even if only one format is currently used it is expected that for some usages future extension may require the flexibility to use multiple different formats for a given CONNECT-IP request.

### 2.2.1. Tunnel Mode IPv4 Format

The Datagram contains one full IPv4 Packet per [RFC0791]. Used in tunnel mode and when the IP Version is 4 per the IP-Version header or explicit given target address.

#### 2.2.2. Tunnel Mode IPv6 Format

The Datagram contains one full IPv6 Packet per [RFC8200]. Used in tunnel mode and when the IP Version is 6 per the IP-Version header or explicit given target address.

#### 2.2.3. Flow Forwarding Format

The Datagram contains only the IP payload. This is defined as the payload following the IPv4 header and any options for IPv4, and for IPv6 as the payload following the IPv6 header and any extension header. Used for Flow Forwarding mode.

#### 2.2.4. ICMP Message Format

This datagram contains a summary message of the ICMP message received and validated for the respective IP flow. The message format carries the ICMP packet for ICMPv4 [RFC0792] or ICMPv6 [RFC4443]. This format is chosen for forward compatibility. From an implementation perspective the client don't need to verify the checksum or validate the header fields because that is done by the server. However, some type codes, like ICMPv4 type 2, (Packet Too Big) carries an MTU field that the implementation want to read beyond understanding the meaning of the type and code combination.

### 3. HTTP Headers

Note: This section should be improved by clarifying if headers are in request, response or both.

#### 3.1. IP-Protocol Header for CONNECT-IP

In order to construct the IP header the the proxy needs to fill the "Protocol" field in the IPv4 header or "Next header" field in the IPv6 header. As the IP payload is otherwise mostly opaque to the proxy, this information has to be provided by the client for each CONNECT-IP request for flow forwarding.

IP-Protocol is a Item Structured Header [RFC8941]. Its value MUST be an Integer. Its ABNF is:

IP-Protocol = sf-integer

#### 3.2. IP-Version header for CONNECT-IP

IP-Version is a Item Structured Header [RFC8941]. Its value MUST be an Integer and either 4 or 6. This information is used by the proxy to check if the requested IP version is supported by the network that the proxy is connected to, as well as to check the destination or source IP address for compliance.

IP-Version = sf-integer

#### 3.3. IP-Address header for CONNECT-IP

IP-Address is an Item Structured Header [RFC8941]. Its value MUST be an String contain an IP address or IP address range of the same IP

version as indicated in the IP-Version header. The address must be specified in the format specified by TBD.

This header is used to request the use of a certain IP address or IP address range by the client to be used as source IP address in tunnel mode. If the IP-Address header is not presented, the proxy is implicitly requested to assign an IP address or IP address range and provide this information to the client with the HTTP response.

If the the client does not provide an IP address or IP address range is has to wait for the proxy response before any payload data can be sent in tunnel mode. If the request is denied by the proxy, any sent payload data will be discarded and a new CONNECT-IP request has to be sent.

The header is also used as a response header from the proxy to the client to indicate the actual IP address or IP address range that should be used by the client in tunnel mode or will be used by the proxy in flow forwarding mode.

IP-Address = sf-string

#### 3.4. IP-Address-Handling Header for CONNECT-IP

This header can be used to request the use of a stable address for multiple active flow forwarding associations. The first association will be established with an IP selected by the proxy unless also the IP-Address header (Section 3.3) is provided and accepted by proxy. However, additional forwarding association can be requested by the client to use the same IP address as a previous request by specifying the stream ID as value in this header. This header can also be used to ensure that a "new", not yet for this client used address is selected by setting a value that is larger than the maximum stream ID.

IP-Address-Handling is a Item Structured Header [RFC8941]. Its value MUST be an Integer and indicates the stream ID of the corresponding active flow forwarding association. Its ABNF is:

IP-Address-Handling = sf-integer

#### 3.5. Conn-ID Header for CONNECT-IP

This document further defines a new header field to be used with CONNECT-IP "Conn-ID". The Conn-ID HTTP header field indicates the value, offset, and length of a field in the IP payload that can be used by the proxy as a connection identifier in addition to the IP address and protocol tuple when multiple connections are proxied to the same target server for incoming traffic on the service address.

Conn-ID is a Item Structured Header [RFC8941]. Its value MUST be a Byte Sequence. Its ABNF is:

Conn-ID = sf-binary

The following parameters are defined:

- \* A parameter whose name is "offset", and whose value is an Integer indicating the offset of the identifier field starting from the beginning of a datagram or HTTP frame on the forwarding stream.
- \* A parameter whose name is "length", and whose value is an Integer

indicating the length of the identifier field starting from the offset.

Both parameters MUST be present and the header MUST be ignored if these parameter are not present.

This function can be used to e.g. indicate the source port field in the IP payload when containing a TCP packet.

#### 4. Client Connect-IP Request

##### 4.1. Requesting flow forwarding

To request flow forwarding, the client sends a CONNECT-IP request to the forwarding proxy indicating the target host and port in the ":authority" pseudo-header field. The host portion is either an IP literal encapsulated within square brackets, an IPv4 address in dotted-decimal form, or a registered name. Further the CONNECT-IP request MUST contain the IP-Protocol header (Section 3.1) and MAY contain the IP-Address-Handling (Section 3.4) or the Conn-ID (Section 3.5) header.

##### 4.2. Requesting tunnel mode

In tunnel mode, the CONNECT-IP request MUST contain the IP-Version header to indicate if IPv4 or IPv6 is used for the IP packet in the tunnel payload. Further, the request MAY contain an IP-Address header to request use of an IP address or IP address range.

#### 5. MASQUE server behavior

Upon the establishment of a HTTP Connection with the proxy on its service addresses. HTTP level capabilities will be exchanged in the HTTP SETTINGS frame. This will determine if support of datagrams is indicated. If indicated by the client, the MASQUE server SHALL send a H3\_DATAGRAM SETTINGS parameter with a value of 1 to indicates is support.

A MASQUE server that receives an IP-CONNECT request examines the target URL to determine if this request is for tunnel or flow forwarding mode. Based on the mode it determines if the required headers are present and which of the optional headers that are included.

The proxy maintains a database with mappings between the HTTP connections and stream IDs and the IP level selectors and Conn-ID information. Using this database and the pool of available addresses and the requests IP-Address-Handling, Conn-ID, IP-Version, IP-Address headers (if included) to select a source IP address. This selection for flow forwarding mode is further discussed below in Section 5.2. For Tunnel Mode, the proxy determine if the proposed IP address per IP-Version and IP-Address headers is possible to use if included, else selects a otherwise unused address from its pool. For tunnel mode the IP selector for incoming traffic for this HTTP Connection and Stream ID is simply the IP destination address.

Once the mapping is successfully established, the proxy sends a HEADERS frame containing a 2xx series status code to the client. The response MUST contain an IP-Address header indicating the outgoing source IP address or IP address range selected by the proxy.

All Datagram capsules received on that stream as well as all HTTP/3

datagrams belonging to this CONNECT-IP association are processed for forwarding to the target server. For flow forwarding mode the Datagram is processed as specified in Section 5.3 to produce IP packets that can be forwarded. For tunnel-mode the complete IP packet are extracted from the Datagram and then forwarded as specified in Section 5.4.

IP packets received from the target server are mapped to an active forwarding connection and are respectively forwarded in an CAPSULE DATAGRAM frame or HTTP/3 datagram to the client (see section Section 5.5 below).

#### 5.1. Error handling

TBD (e.g. out of IP address, conn-id collision)

#### 5.2. IP address selection in flow forwarding mode

In flow forwarding mode the proxy constructs the IP header when sending the IP payload towards the target server and it selects an source IP address from its pool of IP addresses that are routed to the MASQUE server.

If no additional information about a payload field that can be used as an identifier based on Conn-ID header is provided by the client, the proxy uses the source/destination address and protocol ID 3-tuple in order to map an incoming IP packet to an active forwarding connection. The proxy MUST also consider if IP-Address-Handling header Section 3.4 is included and its value. If the IP-Address-Handling header is not included and there has been prior request the proxy SHOULD give the client the same source Address as the first flow forwarding request. Given these constraints the MASQUE proxy MUST select a source IP address that leads to a unique tuple, and if that is not possible an error response is generated. The same IP address MAY be used for different clients when those client connect to different target servers. However, this also means that potentially multiple IP address are used for the same client when multiple connection to one target server are needed. This can be problematic if the source address is used by the target as an identifier. Therefore it is RECOMMENDED that clients are given unique addresses unless a large fraction of the pool has been exhausted.

If the Conn-ID header is provided, the proxy should use that field as an connection identifier together with protocol ID, source and destination address, as a 4-tuple. In this case it is recommended to use a stable IP address for each client, while the same IP address might still be used for multiple clients, if not indicated differently by the client in the configuration file. Note that if the same IP address is used for multiple clients, this can still lead to an identifier collision and the IP-CONNECT request MUST be reject if such a collision is detect.

Note: Are we allowing multiple client's to share the same 3-tuple when using Conn-ID? It might be good for privacy reasons however, it significantly increases the collision risk.

#### 5.3. Constructing the IP header in flow forwarding mode

To retrieve the source and destination address the proxy looks up the mapping for the datagram flow ID or stream identifier. The IP version, flow label, DiffServ codepoint (DSCP), and hop limit/TTL is

selected by the proxy. The IPv4 Protocol or IPv6 Next Header field is set based on the information provided by the IP-Protocol header in the CONNECT-IP request.

The proxy MUST set the Don't Fragment (DF) flag in the IPv4 header. Payload that does not fit into one IP packet MUST be dropped. A dropping indication should be provided to the client. Further the proxy should provide MTU information.

The ECN field is by default set to non-ECN capable transport (non-ECT). Further ECN handling is described in Section Section 6.1.

#### 5.4. Decapsulation of tunnel mode IP Packets

On receiving an HTTP Datagram containing any of the tunnel mode formats for IPv4 or IPv6 the proxy extracts the full IP packet.

The proxy MUST verify that the extracted IP packet's source IP address matches any address associated with this CONNECTION-IP request, i.e. the assigned address or IP range. This is to prevent source address spoofing in tunnel mode.

Further the proxy should verify that the IP header length field correspond to the extracted packets length.

#### 5.5. Receiving an IP packet

When the proxy receives an incoming IP packet on the external interface(s), it checks the packet selectors to find the mappings that match the given packet.

If a client has a tunnel as well as multiple flow forwarding associations, the proxy need to check the mappings for the flow forwarding associations first, and only send it over the the tunnel association if no active flow forwarding is found.

If one or more mappings exists, it further checks if this mapping contains additional identifier information as provided by the Conn-ID Header of the CONNECT-IP request. If this field maps as well, the IP payload is forwarded to the client. If no active mapping is found, the IP packet is discarded.

The above is achieve by using the selector with the most number of fields that match the packet.

If both datagram and stream based forwarding is supported, it is recommended for the proxy to use the same encapsulation as most recently used by the client or datagrams as default. Further considerations might be needed here.

### 6. Additional signalling

Context ID as defined by [I-D.ietf-masque-h3-datagram] can be used to provide additional per association or per-payload signals. As [I-D.ietf-masque-h3-datagram] is still work in progress, registration and use of Context IDs is left for future work at this point.

#### 6.1. ECN

ECN requires coordination with the end-to-end communication points as it should only be used if the endpoints are also capable and willing to signal congestion notifications to the other end and react

accordingly if a congestion notification is received.

The probing and verification in the upper layer protocol of end-to-end ECN requires per packet control over what value is set on IP packet transmission as well as which of all values are received by the proxy. The QUIC specification is providing one such example in Section 13.4 of [RFC9000]. Thus in flow forwarding mode the proxy needs to be able to set and read the ECN values in sent and received IP packets respectively. This may motivate that this functionality is optional to implement, even if supporting CONNECT-IP implementations in general will need to handle IP packets and their fields with fine grained control. If optional some negotiation mechanism is needed.

Possible realizations are:

- a) always have two bits before payload in flow forwarding model, e.g. by including the whole Type of Service (TOS) byte, which would also enable DSCP setting and reading.
- b) use 4 different context IDs depending on what ECN field value was received or should be set.

This is work in process and will be further specified in a future version of this document.

## 6.2. ICMP handling

ICMP messages are directly forwarded in tunneling mode. In flow forwarding mode a ICMP datagram format (Section 2.2.4) is used to send the information from some ICMP message to the client.

The proxy upon receiving an ICMP message with a destination of an IP address it performs flow forwarding on it needs to process the ICMP message. First it should validate that the ICMP message and find if it matches any of its IP flow selectors (including Conn-ID). In case there are multiple matching use the IP selector with the most number of field that matches fully.

Some messages may be applicable both to the proxy and the client. For example an verified ICMPv6 Packet Too Big is applicable both to the proxy and the client. Others like ICMPv6 Destination Unreachable (Type=1), Code=3 (Address unreachable) and Code=4 (Port unreachable) is only possible to act on by the client.

QUESTION: Which ICMP messages should be suppressed by the proxy?

If a matching IP selector was chosen, then lookup the mapping for the HTTP connection and Stream ID which this message should be sent to. Encapsulate the received ICMP message in the ICMP datagram format and send it to the client.

## 6.3. MTU considerations

The use of QUIC as a encapsulation between the client and proxy introduces additional overhead. If datagrams are used to encapsulate packets between the proxy and client, the end-to-end packets must fit within one datagram but the size of the datagrams is limited by the tunneling encapsulation overhead.

In forwarding mode the client is usually also the tunnel endpoint that knows about the tunnel overhead and can therefore restrict the

size of the packets on the end-to-end connection accordingly. However, the target endpoint is usually not aware of the tunnel overhead. Additional signalling on the end-to-end connection from the client to the target endpoint might be needed to restrict the packet size. If QUIC is also used as end-to-end protocol, this could be realized by the transport parameter. In addition, signal from the proxy to the client could be provided as an extension to indicate the tunnel overhead more accurately and flexibly over time. Such signalling might be realized on the HTTP layer in order to take any additional limitations by HTTP intermediates into account.

If the proxy receives an incoming packet from a target endpoint that is too big to fit within a datagram on the tunnel connection, the proxy MAY either forward the packet encapsulated in the CAPSULE frames on the respective stream or, if IPv4 with DF bit set or IPv6 is used, the proxy MAY reject the packet and send an ICMPv4 Packet type 3 code 4, or ICMPv6 Too Big (PTB) message.

## 7. Examples

TBD

## 8. Security considerations

This document does currently not discuss risks that are generic to the MASQUE approach.

Any CONNECT-IP specific risks need further consideration in future, especially when the handling of IP functions is defined in more detail.

## 9. IANA considerations

### 9.1. HTTP Method

This document (if published as RFC) registers "CONNECT-IP" in the HTTP Method Registry maintained at <<https://www.iana.org/assignments/http-methods>>.

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

### 9.2. HTTP Header

This document (if published as RFC) registers the following header in the "Permanent Message Header Field Names" registry maintained at <<https://www.iana.org/assignments/message-headers>>.

Header Field Name	Protocol	Status	Reference
Conn-ID	http	exp	This document
IP-Protocol	http	exp	This document
IP-Address	http	exp	This document
IP-Address-Handling	http	exp	This document



IP-Verison	http	exp	This document
+-----+	+-----+	+-----+	+-----+

## Acknowledgments

## References

### Normative References

- [I-D.ietf-httpbis-messaging]  
 Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP/1.1", Work in Progress, Internet-Draft, draft-ietf-httpbis-messaging-16, 27 May 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-messaging-16.txt>>.
- [I-D.ietf-httpbis-semantic]  
 Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP Semantics", Work in Progress, Internet-Draft, draft-ietf-httpbis-semantic-16, 27 May 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-semantic-16.txt>>.
- [I-D.ietf-masque-connect-udp]  
 Schinazi, D., "The CONNECT-UDP HTTP Method", Work in Progress, Internet-Draft, draft-ietf-masque-connect-udp-03, 5 January 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-masque-connect-udp-03.txt>>.
- [I-D.ietf-masque-h3-datagram]  
 Schinazi, D. and L. Pardue, "Using QUIC Datagrams with HTTP/3", Work in Progress, Internet-Draft, draft-ietf-masque-h3-datagram-02, 26 May 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-masque-h3-datagram-02.txt>>.
- [I-D.ietf-quic-datagram]  
 Pauly, T., Kinnear, E., and D. Schinazi, "An Unreliable Datagram Extension to QUIC", Work in Progress, Internet-Draft, draft-ietf-quic-datagram-02, 16 February 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-quic-datagram-02.txt>>.
- [I-D.ietf-quic-http]  
 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.txt>>.
- [RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <<https://datatracker.ietf.org/doc/html/rfc791>>.
- [RFC0792] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, DOI 10.17487/RFC0792, September 1981, <<https://datatracker.ietf.org/doc/html/rfc792>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997,

- <<https://datatracker.ietf.org/doc/html/rfc2119>>.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<https://datatracker.ietf.org/doc/html/rfc3168>>.
- [RFC4443] 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://datatracker.ietf.org/doc/html/rfc4443>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://datatracker.ietf.org/doc/html/rfc8174>>.
- [RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <<https://datatracker.ietf.org/doc/html/rfc8200>>.
- [RFC8941] Nottingham, M. and P-H. Kamp, "Structured Field Values for HTTP", RFC 8941, DOI 10.17487/RFC8941, February 2021, <<https://datatracker.ietf.org/doc/html/rfc8941>>.
- [RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <<https://datatracker.ietf.org/doc/html/rfc9000>>.

#### Informative References

- [I-D.ietf-masque-ip-proxy-reqs]  
Chernyakhovsky, A., McCall, D., and D. Schinazi,  
"Requirements for a MASQUE Protocol to Proxy IP Traffic",  
Work in Progress, Internet-Draft, draft-ietf-masque-ip-proxy-reqs-02, 30 April 2021,  
<<https://datatracker.ietf.org/doc/html/draft-ietf-masque-ip-proxy-reqs-02.txt>>.
- [I-D.westerlund-masque-transport-issues]  
Westerlund, M., Ihlar, M., Sarker, Z., and M. Kuehlewind,  
"Transport Considerations for IP and UDP Proxying in MASQUE",  
Work in Progress, Internet-Draft, draft-westerlund-masque-transport-issues-02, 12 July 2021,  
<<https://datatracker.ietf.org/doc/html/draft-westerlund-masque-transport-issues-02.txt>>.
- [RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black,  
"Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, DOI 10.17487/RFC2474, December 1998, <<https://datatracker.ietf.org/doc/html/rfc2474>>.

#### Authors' Addresses

Mirja Kuehlewind  
Ericsson

Email: [mirja.kuehlewind@ericsson.com](mailto:mirja.kuehlewind@ericsson.com)

Magnus Westerlund  
Ericsson

Email: [magnus.westerlund@ericsson.com](mailto:magnus.westerlund@ericsson.com)

Marcus Ihlar  
Ericsson

Email: [marcus.ihlar@ericsson.com](mailto:marcus.ihlar@ericsson.com)

Zaheduzzaman Sarker  
Ericsson

Email: [zaheduzzaman.sarker@ericsson.com](mailto:zaheduzzaman.sarker@ericsson.com)

MASQUE  
Internet-Draft  
Intended status: Experimental  
Expires: 27 January 2022

L. Pardue  
Cloudflare  
26 July 2021

HTTP Datagram Prioritization  
draft-pardue-masque-dgram-priority-01

Abstract

Application protocols using the QUIC transport protocol rely on streams, and optionally the DATAGRAM extension, to carry application data. Streams and datagrams can be multiplexed but QUIC provides no interoperable prioritization scheme or signaling mechanism itself. The HTTP Extensible Prioritization scheme describes how to prioritize streams in HTTP/2 and HTTP/3. This document adopts the scheme to support HTTP datagrams.

Note to Readers

\_RFC EDITOR: please remove this section before publication\_

Source code and issues list for this draft can be found at  
<https://github.com/LPardue/draft-pardue-masque-dgram-priority>  
(<https://github.com/LPardue/draft-pardue-masque-dgram-priority>).

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 27 January 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 . . . . .	2
1.1. Notational Conventions . . . . .	3
2. Signalling Datagram Priority . . . . .	3
2.1. Datagram Urgency . . . . .	4
2.2. Prioritization of Contexts . . . . .	4
2.3. Reprioritization . . . . .	4
3. Client Scheduling . . . . .	4
4. Server Scheduling . . . . .	5
5. Retransmission Scheduling . . . . .	5
6. Security Considerations . . . . .	5
7. IANA Considerations . . . . .	5
8. References . . . . .	5
8.1. Normative References . . . . .	5
8.2. Informative References . . . . .	6
Appendix A. Acknowledgements . . . . .	6
Author's Address . . . . .	7

## 1. Introduction

Application protocols using the QUIC transport protocol [QUIC] rely on streams, and optionally the DATAGRAM extension [QUIC-DATAGRAM], to carry application data. Streams and datagrams can be multiplexed but QUIC provides no interoperable prioritization scheme or signaling mechanism itself. The HTTP Extensible Prioritization scheme [I-D.ietf-httpbis-priority] describes how to prioritize streams in HTTP/2 and HTTP/3. This document adopts the scheme to support HTTP datagrams [HTTP-DATAGRAM].

The Extensible Priorities scheme for HTTP describes how clients can send priority signals related to requests in order to suggest how a server allocates resources to serving responses. When the protocol is HTTP/2, responses are carried on streams. When the protocol is HTTP/3, responses are carried on QUIC streams.

While QUIC streams support multiplexing natively via use of a stream identifier, the QUIC DATAGRAM extension does not provide any such identifier. HTTP datagrams [HTTP-DATAGRAM] supports multiplexing

using a set of application-level identifiers that can be controlled and accessed by HTTP/3. One identifier relates to a request stream, the second, optional, identifier relates to an abstract context. [HTTP-DATAGRAM] does not, however, define any means for multiplexed datagram prioritization.

When the application protocol is HTTP/3, HTTP Datagrams can map directly to QUIC datagrams or they can be carried on streams using a DATAGRAM Capsule; see Section 4.4 of [HTTP-DATAGRAM].

This document describes how the Extensible Priorities scheme applies to HTTP datagrams. Priority signals sent by clients, related to requests, can also be considered input to server scheduling decisions for HTTP datagrams mapped to QUIC datagrams.

### 1.1. Notational 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.

The term sf-integer is imported from [STRUCTURED-FIELDS].

## 2. Signalling Datagram Priority

The Extensible Prioritization scheme [I-D.ietf-httpbis-priority] provides a framework for communicating and acting upon priority parameters, using [STRUCTURED-FIELDS] formats. It defines the urgency and incremental parameters and provides guidance to implementers about how to act on these parameters, in combination with other inputs, to make resource allocation and scheduling choices. Urgency communicates the client-view of request importance, and incremental communicates how the client intends to process response data as it arrives. Parameters are communicated in HTTP headers or version-specific frames. A client omitting the urgency or incremental parameters can be interpreted by the server as a signal to apply default priorities. The core scheme is extensible, new parameters can be defined to augment the base ones.

This specification defines the datagram-urgency ("du") extension parameter that operates in addition to the base urgency. There is no extension to the base incremental behavior; individual datagrams, even if belonging to the same identifier, are messages that are expected to be processed individually as they arrive.

### 2.1. Datagram Urgency

The datagram-urgency parameter ("du") takes an integer between 0 and 7, in descending order of priority. This range matches the base urgency ("u") parameter range; see Section 4.1 of [I-D.ietf-httpbis-priority].

The value is encoded as an sf-integer. There is no default value.

This parameter indicates the sender's recommendation, based on the expectation that the server would transmit HTTP datagrams in the order of their datagram-urgency values if possible. The smaller the value, the higher the precedence. Omitting the datagram-urgency parameter is a signal to apply the value of the urgency parameter.

The following example shows a request for a CSS file with the urgency set to "0", any associated datagrams have the lower urgency of "2":

```
:method = GET
:scheme = https
:authority = example.net
:path = /style.css
priority = u=0, du=2
```

Endpoints MUST NOT treat reception of the datagram-urgency parameter, even if HTTP datagram support is not enabled.

The datagram-urgency parameter applies only to HTTP datagrams mapped to QUIC datagrams. Datagram capsules are sent on streams, so the base urgency parameter applies to them.

### 2.2. Prioritization of Contexts

The datagram-urgency parameter applies to all HTTP datagram contexts related to a request stream. Prioritization of individual contexts is not supported.

### 2.3. Reprioritization

Reprioritization is supported using the existing mechanisms defined in Section 6 of [I-D.ietf-httpbis-priority].

## 3. Client Scheduling

Clients MAY use datagram-urgency to make local processing or scheduling choices about HTTP datagrams related to the requests it initiates.

#### 4. Server Scheduling

Priority signals are input to a prioritization process. Expressing priority is only a suggestion. The datagram-urgency parameter introduces new scheduling considerations on top of those presented in Section 10 of [I-D.ietf-httpbis-priority].

It is RECOMMENDED that, when possible, servers send higher urgency HTTP datagrams before lower urgency datagrams.

Where streams and datagrams have equal urgency and datagram-urgency, it is RECOMMENDED that servers alternate emitting HTTP datagrams and stream bytes. Where servers implement the recommendations in Section 10 of [I-D.ietf-httpbis-priority], alternating between datagram and stream data will result in fair scheduling. This recommendation holds whether stream are incremental or not.

It is RECOMMENDED that servers schedule DATAGRAM capsules the same as response data.

#### 5. Retransmission Scheduling

Section 12 of [I-D.ietf-httpbis-priority] provides guidance about scheduling of retransmission data vs. new data. Since QUIC datagrams are not retransmitted, endpoints that prioritize QUIC stream retransmission data could delay datagrams. Furthermore, since DATAGRAM capsules are sent as stream data, they *are* subject to retransmission and could also delay native QUIC datagrams.

#### 6. Security Considerations

There are believed to be no additional considerations to those presented in [I-D.ietf-httpbis-priority].

#### 7. IANA Considerations

This specification registers the following entry in the HTTP Priority Parameters Registry

Name: datagram-urgency

Description: Priority of HTTP datagrams

Reference: This document

#### 8. References

##### 8.1. Normative References



## [HTTP-DATAGRAM]

Schinazi, D. and L. Pardue, "Using Datagrams with HTTP", Work in Progress, Internet-Draft, draft-ietf-masque-h3-datagram-03, 12 July 2021, <<https://www.ietf.org/archive/id/draft-ietf-masque-h3-datagram-03.txt>>.

## [I-D.ietf-httpbis-priority]

Oku, K. and L. Pardue, "Extensible Prioritization Scheme for HTTP", Work in Progress, Internet-Draft, draft-ietf-httpbis-priority-04, 11 July 2021, <<https://www.ietf.org/archive/id/draft-ietf-httpbis-priority-04.txt>>.

## [QUIC-DATAGRAM]

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://www.ietf.org/archive/id/draft-ietf-quic-datagram-03.txt>>.

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

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

## [STRUCTURED-FIELDS]

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

## 8.2. Informative References

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

## Appendix A. Acknowledgements

This document is inspired by discussion by many people across HTTP, QUIC and MASQUE WGs.

Author's Address

Lucas Pardue  
Cloudflare

Email: [lucaspardue.24.7@gmail.com](mailto:lucaspardue.24.7@gmail.com)