

Workgroup: MASQUE
Internet-Draft:
draft-ietf-masque-connect-ip-06
Updates: [9298](#) (if approved)
Published: 13 February 2023
Intended Status: Standards Track
Expires: 17 August 2023
Authors: T. Pauly, Ed. D. Schinazi A. Chernyakhovsky
 Apple Inc. Google LLC Google LLC
 M. Kuehlewind M. Westerlund
 Ericsson Ericsson

Proxying IP in HTTP

Abstract

This document describes how to proxy IP packets in HTTP. This protocol is similar to UDP proxying in HTTP, but allows transmitting arbitrary IP packets. More specifically, this document defines a protocol that allows an HTTP client to create an IP tunnel through an HTTP server that acts as a proxy. This document updates RFC 9298.

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-ip/draft-ietf-masque-connect-ip.html>. Status information for this document may be found at <https://datatracker.ietf.org/doc/draft-ietf-masque-connect-ip/>.

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/>. Subscribe at <https://www.ietf.org/mailman/listinfo/masque/>.

Source for this draft and an issue tracker can be found at <https://github.com/ietf-wg-masque/draft-ietf-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 17 August 2023.

Copyright Notice

Copyright (c) 2023 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. [Conventions and Definitions](#)
3. [Configuration of Clients](#)
4. [Tunnelling IP over HTTP](#)
 - 4.1. [IP Proxy Handling](#)
 - 4.2. [HTTP/1.1 Request](#)
 - 4.3. [HTTP/1.1 Response](#)
 - 4.4. [HTTP/2 and HTTP/3 Requests](#)
 - 4.5. [HTTP/2 and HTTP/3 Responses](#)
 - 4.6. [Limiting Request Scope](#)
 - 4.7. [Capsules](#)
 - 4.7.1. [ADDRESS ASSIGN Capsule](#)
 - 4.7.2. [ADDRESS REQUEST Capsule](#)
 - 4.7.3. [ROUTE ADVERTISEMENT Capsule](#)
5. [Context Identifiers](#)
6. [HTTP Datagram Payload Format](#)
7. [Error Signalling](#)
8. [Examples](#)
 - 8.1. [Remote Access VPN](#)
 - 8.2. [IP Flow Forwarding](#)
 - 8.3. [Proxied Connection Racing](#)
9. [Extensibility Considerations](#)
10. [Security Considerations](#)
11. [IANA Considerations](#)
 - 11.1. [HTTP Upgrade Token](#)

- [11.2. Creation of the MASQUE URI Suffixes Registry](#)
- [11.3. Updates to masque Well-Known URI](#)
- [11.4. Capsule Type Registrations](#)
- [12. References](#)
 - [12.1. Normative References](#)
 - [12.2. Informative References](#)
- [Acknowledgments](#)
- [Authors' Addresses](#)

1. Introduction

HTTP provides the CONNECT method (see [Section 9.3.6](#) of [\[HTTP\]](#)) for creating a TCP [\[TCP\]](#) tunnel to a destination and a similar mechanism for UDP [\[CONNECT-UDP\]](#). However, these mechanisms cannot tunnel other IP protocols [\[IANA-PN\]](#) nor convey fields of the IP header.

This document describes a protocol for tunnelling IP to an HTTP server acting as an IP-specific proxy over HTTP. This can be used for various use cases such as point-to-network VPN, secure point-to-point communication, or general-purpose packet tunnelling.

IP proxying operates similarly to UDP proxying [\[CONNECT-UDP\]](#), whereby the proxy itself is identified with an absolute URL, optionally containing the traffic's destination. Clients generate these URLs using a URI Template [\[TEMPLATE\]](#), as described in [Section 3](#).

This protocol supports all existing 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 [\[HTTP\]](#).

2. 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 "IP proxy" to refer to the HTTP server that responds to the IP proxying request. If there are HTTP intermediaries (as defined in [Section 3.7](#) of [\[HTTP\]](#)) between the client and the proxy, those are referred to as "intermediaries" in this document.

This document uses terminology from [\[QUIC\]](#). Where this document defines protocol types, the definition format uses the notation from [Section 1.3](#) of [\[QUIC\]](#). This specification uses the variable-length

integer encoding from [Section 16](#) of [\[QUIC\]](#). Variable-length integer values do not need to be encoded on the minimum number of bytes necessary.

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.

3. Configuration of Clients

Clients are configured to use IP proxying over HTTP via an URI Template [\[TEMPLATE\]](#). The URI template **MAY** contain two variables: "target" and "ipproto"; see [Section 4.6](#). The optionality of the variables needs to be considered when defining the template so that either the variable is self-identifying or it is possible to exclude it in the syntax.

Examples are shown below:

```
https://example.org/.well-known/masque/ip/{target}/{ipproto}/
https://proxy.example.org:4443/masque/ip?t={target}&i={ipproto}
https://proxy.example.org:4443/masque/ip{?target,ipproto}
https://masque.example.org/?user=bob
```

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 **MAY** contain the two variables "target" and "ipproto" and **MAY** contain other variables. If the "target" or "ipproto" variables are included, their values **MUST NOT** be empty. Clients can instead use "*" to indicate wildcard or no-preference values; see [Section 4.6](#).
- *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; see Section 2.1 of [\[URI\]](#)).

*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.

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

As with UDP proxying, some client configurations for IP proxies will only allow the user to configure the proxy host and proxy port. Clients with such limitations **MAY** attempt to access IP proxying capabilities using the default template, which is defined as: "https://\$PROXY_HOST:\$PROXY_PORT/.well-known/masque/ip/{target}/{ipproto}/", where \$PROXY_HOST and \$PROXY_PORT are the configured host and port of the IP proxy, respectively. IP proxy deployments **SHOULD** offer service at this location if they need to interoperate with such clients.

4. Tunnelling IP over HTTP

To allow negotiation of a tunnel for IP over HTTP, this document defines the "connect-ip" HTTP Upgrade Token. The resulting IP tunnels use the Capsule Protocol (see [Section 3.2](#) of [[HTTP-DGRAM](#)]) with HTTP Datagrams in the format defined in [Section 6](#).

To initiate an IP tunnel associated with a single HTTP stream, a client issues a request containing the "connect-ip" upgrade token.

When sending its IP proxying request, the client **SHALL** perform URI template expansion to determine the path and query of its request, see [Section 3](#).

By virtue of the definition of the Capsule Protocol (see [Section 3.2](#) of [[HTTP-DGRAM](#)]), IP proxying requests do not carry any message content. Similarly, successful IP proxying responses also do not carry any message content.

4.1. IP Proxy Handling

Upon receiving an IP proxying request:

*if the recipient is configured to use another HTTP proxy, it will act as an intermediary by forwarding the request to another HTTP server. Note that such intermediaries may need to re-encode 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 an IP proxy. It extracts the optional "target" and "ipproto" variables from the URI it has reconstructed from the request headers, decodes their percent-encoding, and establishes an IP tunnel.

IP proxies **MUST** validate whether the decoded "target" and "ipproto" variables meet the requirements in [Section 4.6](#). If they do not, the IP proxy **MUST** treat the request as malformed; see [Section 8.1.1](#) of [HTTP/2] and [Section 4.1.2](#) of [HTTP/3]. If the "target" variable is a DNS name, the IP proxy **MUST** perform DNS resolution before replying to the HTTP request. If errors occur during this process, the IP proxy **MUST** reject 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](#)].

The lifetime of the IP forwarding tunnel is tied to the IP proxying request stream. The IP proxy **MUST** maintain all IP address and route assignments associated with the IP forwarding tunnel while the request stream is open. IP proxies **MAY** choose to tear down the tunnel due to a period of inactivity, but they **MUST** close the request stream when doing so.

A successful response (as defined in Sections [4.3](#) and [4.5](#)) indicates that the IP proxy has established an IP tunnel and is willing to proxy IP payloads. Any response other than a successful response indicates that the request has failed; thus, the client **MUST** abort the request.

Along with a successful response, the IP proxy can send capsules to assign addresses and advertise routes to the client ([Section 4.7](#)). The client can also assign addresses and advertise routes to the IP proxy for network-to-network routing.

4.2. HTTP/1.1 Request

When using HTTP/1.1 [[HTTP/1.1](#)], an IP proxying request will meet the following requirements:

*the method **SHALL** be "GET".

*the request **SHALL** include a single Host header field containing the host and optional port of the IP 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-ip".

An IP proxying request that does not conform to these restrictions is malformed. The recipient of such a malformed request **MUST** respond with an error and **SHOULD** use the 400 (Bad Request) status code.

For example, if the client is configured with URI Template "https://example.org/.well-known/masque/ip/{target}/{ipproto}/" and wishes to open an IP forwarding tunnel with no target or protocol limitations, it could send the following request:

```
GET https://example.org/.well-known/masque/ip/*/* HTTP/1.1
Host: example.org
Connection: Upgrade
Upgrade: connect-ip
Capsule-Protocol: ?1
```

Figure 2: Example HTTP/1.1 Request

4.3. HTTP/1.1 Response

The IP proxy indicates 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 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-ip".

*the response **SHALL** meet the requirements of HTTP responses that start the Capsule Protocol; see [Section 3.2](#) of [\[HTTP-DGRAM\]](#).

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

For example, the IP proxy could respond with:

```
HTTP/1.1 101 Switching Protocols
Connection: Upgrade
Upgrade: connect-ip
Capsule-Protocol: ?1
```

Figure 3: Example HTTP/1.1 Response

4.4. HTTP/2 and HTTP/3 Requests

When using HTTP/2 [[HTTP/2](#)] or HTTP/3 [[HTTP/3](#)], IP proxying requests use HTTP 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-ip".

*The `:authority` pseudo-header field **SHALL** contain the authority of the IP 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; see [Section 3](#). Variables in the URI template can determine the scope of the request, such as requesting full-tunnel IP packet forwarding, or a specific proxied flow; see [Section 4.6](#).

An IP proxying request that does not conform to these restrictions is malformed; see [Section 8.1.1](#) of [[HTTP/2](#)] and [Section 4.1.2](#) of [[HTTP/3](#)].

For example, if the client is configured with URI Template "https://example.org/.well-known/masque/ip/{target}/{ipproto}/" and wishes to open an IP forwarding tunnel with no target or protocol limitations, it could send the following request:

```
HEADERS
:method = CONNECT
:protocol = connect-ip
:scheme = https
:path = /.well-known/masque/ip/*/*/
:authority = example.org
capsule-protocol = ?1
```

Figure 4: Example HTTP/2 or HTTP/3 Request

4.5. HTTP/2 and HTTP/3 Responses

The IP proxy indicates a successful response by replying with the following requirements:

- *the HTTP status code on the response **SHALL** be in the 2xx (Successful) range.

- *the response **SHALL** meet the requirements of HTTP responses that start the Capsule Protocol; see [Section 3.2](#) of [[HTTP-DGRAM](#)].

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

For example, the IP proxy could respond with:

```
HEADERS
:status = 200
capsule-protocol = ?1
```

Figure 5: Example HTTP/2 or HTTP/3 Response

4.6. Limiting Request Scope

Unlike UDP proxying requests, which require specifying a target host, IP proxying requests can allow endpoints to send arbitrary IP packets to any host. The client can choose to restrict a given request to a specific IP prefix or IP protocol by adding parameters to its request. When the IP proxy knows that a request is scoped to a target prefix or protocol, it can leverage this information to optimize its resource allocation; for example, the IP proxy can assign the same public IP address to two IP proxying requests that are scoped to different prefixes and/or different protocols.

The scope of the request is indicated by the client to the IP proxy via the "target" and "ipproto" variables of the URI Template; see [Section 3](#). Both the "target" and "ipproto" variables are optional; if they are not included, they are considered to carry the wildcard value "*".

target: The variable "target" contains a hostname or IP prefix of a specific host to which the client wants to proxy packets. If the "target" variable is not specified or its value is "*", the client is requesting to communicate with any allowable host. "target" supports using DNS names, IPv6 prefixes and IPv4 prefixes. Note that IPv6 scoped addressing zone identifiers are not supported. If the target is an IP prefix (IP address optionally followed by a percent-encoded slash followed by the prefix length in bits), the request will only support a single IP

version. If the target is a hostname, the IP proxy is expected to perform DNS resolution to determine which route(s) to advertise to the client. The IP proxy **SHOULD** send a ROUTE_ADVERTISEMENT capsule that includes routes for all addresses that were resolved for the requested hostname, that are accessible to the IP proxy, and belong to an address family for which the IP proxy also sends an Assigned Address.

ipproto: The variable "ipproto" contains an IP protocol number, as defined in the "Assigned Internet Protocol Numbers" IANA registry maintained at <<https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml>>. If present, it specifies that a client only wants to proxy a specific IP protocol for this request. If the value is "*", or the variable is not included, the client is requesting to use any IP protocol.

Using the terms IPv6address, IPv4address, and reg-name from [URI], the "target" and "ipproto" variables **MUST** adhere to the format in [Figure 6](#), using notation from [ABNF]. Additionally:

*if "target" contains an IPv6 literal or prefix, the colons (":") **MUST** be percent-encoded. For example, if the target host is "2001:db8::42", it will be encoded in the URI as "2001%3Adb8%3A%3A42".

*If present, the IP prefix length in "target" **SHALL** be preceded by a percent-encoded slash ("/"): "%2F". The IP prefix length **MUST** represent a decimal integer between 0 and the length of the IP address in bits, inclusive.

*"ipproto" **MUST** represent a decimal integer between 0 and 255 inclusive, or the wildcard value "*".

```
target = IPv6prefix / IPv4prefix / reg-name / "*"
IPv6prefix = IPv6address ["%2F" 1*3DIGIT]
IPv4prefix = IPv4address ["%2F" 1*2DIGIT]
ipproto = 1*3DIGIT / "*"
```

Figure 6: URI Template Variable Format

IP proxies **MAY** perform access control using the scoping information provided by the client: if the client is not authorized to access any of the destinations included in the scope, then the IP proxy can immediately fail the request.

4.7. Capsules

This document defines multiple new capsule types that allow endpoints to exchange IP configuration information. Both endpoints **MAY** send any number of these new capsules.

4.7.1. ADDRESS_ASSIGN Capsule

The ADDRESS_ASSIGN capsule (see [Section 11.4](#) for the value of the capsule type) allows an endpoint to inform its peer of the list of IP addresses or prefixes it has assigned to it. Every capsule contains the full list of IP prefixes currently assigned to the receiver. Any of these addresses can be used as the source address on IP packets originated by the receiver of this capsule.

```
ADDRESS_ASSIGN Capsule {  
  Type (i) = ADDRESS_ASSIGN,  
  Length (i),  
  Assigned Address (..) ...,  
}
```

Figure 7: ADDRESS_ASSIGN Capsule Format

The ADDRESS_ASSIGN capsule contains a sequence of zero or more Assigned Addresses.

```
Assigned Address {  
  Request ID (i),  
  IP Version (8),  
  IP Address (32..128),  
  IP Prefix Length (8),  
}
```

Figure 8: Assigned Address Format

Each Assigned Address contains the following fields:

- Request ID:** Request identifier, encoded as a variable-length integer. If this address assignment is in response to an Address Request (see [Section 4.7.2](#)), then this field **SHALL** contain the value of the corresponding field in the request. Otherwise, this field **SHALL** be zero.
- IP Version:** IP Version of this address assignment, encoded as an unsigned 8-bit integer. **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:** The number of bits in the IP Address that are used to define the prefix that is being assigned, encoded as an unsigned 8-bit integer. This **MUST** be less than or equal to the length of the IP Address field, in bits. If the prefix length is equal to the length of the IP Address, the receiver of this capsule is only allowed to send packets from a single source

address. If the prefix length is less than the length of the IP address, the receiver of this capsule is allowed to send packets from any source address that falls within the prefix.

If any of the capsule fields are malformed upon reception, the receiver of the capsule **MUST** follow the error handling procedure defined in [Section 3.3](#) of [[HTTP-DGRAM](#)].

If an ADDRESS_ASSIGN capsule does not contain an address that was previously transmitted in another ADDRESS_ASSIGN capsule, that indicates that the address has been removed. An ADDRESS_ASSIGN capsule can also be empty, indicating that all addresses have been removed.

In some deployments of IP proxying in HTTP, an endpoint needs to be assigned an address by its peer before it knows what source address to set on its own packets. For example, in the Remote Access case ([Section 8.1](#)) the client cannot send IP packets until it knows what address to use. In these deployments, the endpoint that is expecting an address assignment **MUST** send an ADDRESS_REQUEST capsule. This isn't required if the endpoint does not need any address assignment, for example when it is configured out-of-band with static addresses.

While ADDRESS_ASSIGN capsules are commonly sent in response to ADDRESS_REQUEST capsules, endpoints **MAY** send ADDRESS_ASSIGN capsules unprompted.

4.7.2. ADDRESS_REQUEST Capsule

The ADDRESS_REQUEST capsule (see [Section 11.4](#) for the value of the capsule type) allows an endpoint to request assignment of IP addresses from its peer. The capsule allows the endpoint to optionally indicate a preference for which address it would get assigned.

```
ADDRESS_REQUEST Capsule {  
  Type (i) = ADDRESS_REQUEST,  
  Length (i),  
  Requested Address (..) ...,  
}
```

Figure 9: ADDRESS_REQUEST Capsule Format

The ADDRESS_REQUEST capsule contains a sequence of one or more Requested Addresses.

```

Requested Address {
  Request ID (i),
  IP Version (8),
  IP Address (32..128),
  IP Prefix Length (8),
}

```

Figure 10: Requested Address Format

Each Requested Address contains the following fields:

Request ID: Request identifier, encoded as a variable-length integer. This is the identifier of this specific address request. Each request from a given endpoint carries a different identifier. Request IDs **MUST NOT** be reused by an endpoint, and **MUST NOT** be zero.

IP Version: IP Version of this address request, encoded as an unsigned 8-bit integer. **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, encoded as an unsigned 8-bit integer. **MUST** be less than or equal to the length of the IP Address field, in bits.

If the IP Address is all-zero (0.0.0.0 or ::), this indicates that the sender is requesting an address of that address family but does not have a preference for a specific address. In that scenario, the prefix length still indicates the sender's preference for the prefix length it is requesting.

If any of the capsule fields are malformed upon reception, the receiver of the capsule **MUST** follow the error handling procedure defined in [Section 3.3](#) of [[HTTP-DGRAM](#)].

Upon receiving the ADDRESS_REQUEST capsule, an endpoint **SHOULD** assign one or more IP addresses to its peer, and then respond with an ADDRESS_ASSIGN capsule to inform the peer of the assignment. For each Requested Address, the receiver of the ADDRESS_REQUEST capsule **SHALL** respond with an Assigned Address with a matching Request ID. If the requested address was assigned, the IP Address and IP Prefix Length fields in the Assigned Address response **SHALL** be set to the assigned values. If the requested address was not assigned, the IP Address **SHALL** be all-zero and the IP Prefix Length **SHALL** be the maximum length (0.0.0.0/32 or ::/128) to indicate that no address was assigned. These address rejections **SHOULD NOT** be included in subsequent ADDRESS_ASSIGN capsules. Note that other Assigned Address

entries that do not correspond to any Request ID can also be contained in the same ADDRESS_ASSIGN response.

If an endpoint receives an ADDRESS_REQUEST capsule that contains zero Requested Addresses, it **MUST** abort the IP proxying request stream.

Note that the ordering of Requested Addresses does not carry any semantics. Similarly, the Request ID is only meant as a unique identifier, it does not convey any priority or importance.

4.7.3. ROUTE_ADVERTISEMENT Capsule

The ROUTE_ADVERTISEMENT capsule (see [Section 11.4](#) for the value of the capsule type) allows an endpoint to communicate to its peer that it is willing to route traffic to a set of IP address ranges. This indicates that the sender has an existing route to each address range, and notifies its peer that if the receiver of the ROUTE_ADVERTISEMENT capsule sends IP packets for one of these ranges in HTTP Datagrams, the sender of the capsule will forward them along its preexisting route. Any address which is in one of the address ranges can be used as the destination address on IP packets originated by the receiver of this capsule.

```
ROUTE_ADVERTISEMENT Capsule {  
  Type (i) = ROUTE_ADVERTISEMENT,  
  Length (i),  
  IP Address Range (..) ...,  
}
```

Figure 11: ROUTE_ADVERTISEMENT Capsule Format

The ROUTE_ADVERTISEMENT capsule contains a sequence of IP Address Ranges.

```
IP Address Range {  
  IP Version (8),  
  Start IP Address (32..128),  
  End IP Address (32..128),  
  IP Protocol (8),  
}
```

Figure 12: IP Address Range Format

Each IP Address Range contains the following fields:

IP Version: IP Version of this range, encoded as an unsigned 8-bit integer. **MUST** be either 4 or 6.

Start IP Address and End IP Address: Inclusive start and end IP address of the advertised range. If the IP Version field has value 4, these fields **SHALL** have a length of 32 bits. If the IP Version field has value 6, these fields **SHALL** have a length of 128 bits. The Start IP Address **MUST** be less than or equal to the End IP Address.

IP Protocol: The Internet Protocol Number for traffic that can be sent to this range, encoded as an unsigned 8-bit integer. If the value is 0, all protocols are allowed. ICMP traffic is always allowed, regardless of the value of this field.

If any of the capsule fields are malformed upon reception, the receiver of the capsule **MUST** follow the error handling procedure defined in [Section 3.3](#) of [[HTTP-DGRAM](#)].

Upon receiving the ROUTE_ADVERTISEMENT capsule, an endpoint **MAY** start routing IP packets in these ranges to its peer.

Each ROUTE_ADVERTISEMENT contains the full list of address ranges. If multiple ROUTE_ADVERTISEMENT capsules are sent in one direction, each ROUTE_ADVERTISEMENT capsule supersedes prior ones. In other words, if a given address range was present in a prior capsule but the most recently received ROUTE_ADVERTISEMENT capsule does not contain it, the receiver will consider that range withdrawn.

If multiple ranges using the same IP protocol were to overlap, some routing table implementations might reject them. To prevent overlap, the ranges are ordered; this places the burden on the sender and makes verification by the receiver much simpler. If an IP Address Range A precedes an IP address range B in the same ROUTE_ADVERTISEMENT capsule, they **MUST** follow these requirements:

- *IP Version of A **MUST** be less than or equal than IP Version of B
- *If the IP Version of A and B are equal, the IP Protocol of A **MUST** be less than or equal than IP Protocol of B.
- *If the IP Version and IP Protocol of A and B are both equal, the End IP Address of A **MUST** be strictly less than the Start IP Address of B.

If an endpoint receives a ROUTE_ADVERTISEMENT capsule that does not meet these requirements, it **MUST** abort the IP proxying request stream.

Since setting the IP protocol to zero indicates all protocols are allowed, the requirements above make it possible for two routes to overlap when one has IP protocol set to zero and the other set to non-zero. Endpoints **MUST** not send a ROUTE_ADVERTISEMENT capsule with routes that overlap in such a way. Validating this requirement is

OPTIONAL, but if an endpoint detects the violation, it **MUST** abort the IP proxying request stream.

5. Context Identifiers

The mechanism for proxying IP in HTTP defined in this document allows future extensions to exchange HTTP Datagrams that carry different semantics from IP payloads. Some of these extensions can augment IP payloads with additional data or compress IP header fields, while others can exchange data that is completely separate from IP payloads. In order to accomplish this, all HTTP Datagrams associated with IP proxying request streams start with a Context ID field; see [Section 6](#).

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 IP 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 request 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 synchronization 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 either the client or the IP 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 Context IDs. Depending on the method being used, it is possible for datagrams to be received with Context IDs that have not yet been registered. For instance, this can be due to reordering of the packet containing the datagram and the packet containing the registration message during transmission.

6. HTTP Datagram Payload Format

When associated with IP proxying request streams, the HTTP Datagram Payload field of HTTP Datagrams (see [HTTP-DGRAM]) has the format defined in [Figure 13](#). 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:


```

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

```

Figure 13: IP Proxying HTTP Datagram Format

The IP Proxying HTTP Datagram Payload contains the following fields:

Context ID: A variable-length integer 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.

IP 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 a full IP packet (from the IP Version field until the last byte of the IP Payload).

Clients **MAY** optimistically start sending proxied IP packets before receiving the response to its IP proxying request, noting however that those may not be processed by the IP proxy if it responds to the request with a failure, or if the datagrams are received by the IP proxy before the request. Since receiving addresses and routes is required in order to know that a packet can be sent through the tunnel, such optimistic packets might be dropped by the IP proxy if it chooses to provide different addressing or routing information than what the client assumed.

When an 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 send the IP packet on that interface or pass it to a local application.

In the other direction, when an endpoint receives an IP packet, it checks to see if the packet matches the routes mapped for an IP tunnel, and performs the same forwarding checks as above before transmitting the packet over HTTP Datagrams.

Note that 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](#)].

IPv6 requires that every link have an MTU of at least 1280 bytes [[IPv6](#)]. Since IP proxying in HTTP conveys IP packets in HTTP Datagrams and those can in turn be sent in QUIC DATAGRAM frames which cannot be fragmented [[DGRAM](#)], the MTU of an IP tunnel can be limited by the MTU of the QUIC connection that IP proxying is operating over. This can lead to situations where the IPv6 minimum link MTU is violated. IP proxying endpoints that support IPv6 **MUST** ensure that the IP tunnel link MTU is at least 1280 (i.e., that they can send HTTP Datagrams with payloads of at least 1280 bytes). This can be accomplished using various techniques:

- *if both IP proxying endpoints know for certain that HTTP intermediaries are not in use, the endpoints can pad the QUIC INITIAL packets of the underlying QUIC connection that IP proxying is running over. (Assuming QUIC version 1 is in use, the overhead is 1 byte type, 20 bytes maximal connection ID length, 4 bytes maximal packet number length, 1 byte DATAGRAM frame type, 8 bytes maximal quarter stream ID, one byte for the zero Context ID, and 16 bytes for the AEAD authentication tag, for a total of 51 bytes of overhead which corresponds to padding QUIC INITIAL packets to 1331 bytes or more.)

- *IP proxying endpoints can also send ICMPv6 echo requests with 1232 bytes of data to ascertain the link MTU and tear down the tunnel if they do not receive a response. Unless endpoints have an out of band means of guaranteeing that the previous techniques is sufficient, they **MUST** use this method. If an endpoint does not know an IPv6 address of its peer, it can send the ICMPv6 echo request to the link local all nodes multicast address (ff02::1).

If an endpoint is using QUIC DATAGRAM frames to convey IPv6 packets, and it detects that the QUIC MTU is too low to allow sending 1280 bytes, it **MUST** abort the IP proxying request stream.

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

7. Error Signalling

Since IP proxying endpoints often forward IP packets onwards to other network interfaces, they need to handle errors in the forwarding process. For example, forwarding can fail if the endpoint does not have a route for the destination address, or if it is configured to reject a destination prefix by policy, or if the MTU of the outgoing link is lower than the size of the packet to be forwarded. In such scenarios, IP proxying endpoints **SHOULD** use ICMP [[ICMP](#)] [[ICMPv6](#)] to signal the forwarding error to its peer.

Endpoints are free to select the most appropriate ICMP errors to send. Some examples that are relevant for IP proxying include:

- *For invalid source addresses, send Destination Unreachable [Section 3.1](#) of [\[ICMPv6\]](#) with code 5, "Source address failed ingress/egress policy".
- *For unroutable destination addresses, send Destination Unreachable [Section 3.1](#) of [\[ICMPv6\]](#) with a code 0, "No route to destination", or code 1, "Communication with destination administratively prohibited".
- *For packets that cannot fit within the MTU of the outgoing link, send Packet Too Big [Section 3.2](#) of [\[ICMPv6\]](#).

In order to receive these errors, endpoints need to be prepared to receive ICMP packets. If an endpoint does not send ROUTE_ADVERTISEMENT capsules, such as a client opening an IP flow through an IP proxy, it **SHOULD** process proxied ICMP packets from its peer in order to receive these errors. Note that ICMP messages can originate from a source address different from that of the IP proxying peer.

8. Examples

IP proxying in HTTP enables many different use cases that can benefit from IP packet proxying and tunnelling. These examples are provided to help illustrate some of the ways in which IP proxying in HTTP can be used.

8.1. Remote Access VPN

The following example shows a point-to-network VPN setup, where a client receives a set of local addresses, and can send to any remote host through the IP proxy. Such VPN setups can be either full-tunnel or split-tunnel.

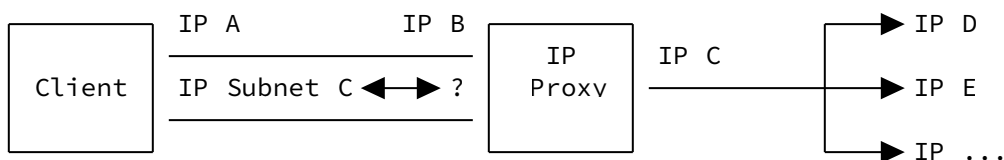


Figure 14: VPN Tunnel Setup

In this case, the client does not specify any scope in its request. The IP proxy assigns the client an IPv4 address (192.0.2.11) and a

full-tunnel route of all IPv4 addresses (0.0.0.0/0). The client can then send to any IPv4 host using a source address in its assigned prefix.

[[From Client]]

[[From IP Proxy]]

SETTINGS

H3_DATAGRAM = 1

SETTINGS

ENABLE_CONNECT_PROTOCOL = 1

H3_DATAGRAM = 1

STREAM(44): HEADERS

:method = CONNECT

:protocol = connect-ip

:scheme = https

:path = /vpn

:authority = proxy.example.com

capsule-protocol = ?1

STREAM(44): HEADERS

:status = 200

capsule-protocol = ?1

STREAM(44): DATA

Capsule Type = ADDRESS_REQUEST

(Request ID = 1

IP Version = 4

IP Address = 0.0.0.0

IP Prefix Length = 32)

STREAM(44): DATA

Capsule Type = ADDRESS_ASSIGN

(Request ID = 1

IP Version = 4

IP Address = 192.0.2.11

IP Prefix Length = 32)

STREAM(44): DATA

Capsule Type = ROUTE_ADVERTISEMENT

(IP Version = 4

Start IP Address = 0.0.0.0

End IP Address = 255.255.255.255

IP Protocol = 0) // Any

DATAGRAM

Quarter Stream ID = 11

Context ID = 0

Payload = Encapsulated IP Packet

DATAGRAM

Quarter Stream ID = 11

Context ID = 0

Payload = Encapsulated IP Packet

Figure 15: VPN Full-Tunnel Example

A setup for a split-tunnel VPN (the case where the client can only access a specific set of private subnets) is quite similar. In this case, the advertised route is restricted to 192.0.2.0/24, rather than 0.0.0.0/0.

```
[[ From Client ]]                [[ From IP Proxy ]]  
  
STREAM(44): DATA  
Capsule Type = ADDRESS_ASSIGN  
(Request ID = 0  
  IP Version = 4  
  IP Address = 192.0.2.42  
  IP Prefix Length = 32)  
  
STREAM(44): DATA  
Capsule Type = ROUTE_ADVERTISEMENT  
(IP Version = 4  
  Start IP Address = 192.0.2.0  
  End IP Address = 192.0.2.255  
  IP Protocol = 0) // Any
```

Figure 16: VPN Split-Tunnel Capsule Example

8.2. IP Flow Forwarding

The following example shows an IP flow forwarding setup, where a client requests to establish a forwarding tunnel to target.example.com using SCTP (IP protocol 132), and receives a single local address and remote address it can use for transmitting packets. A similar approach could be used for any other IP protocol that isn't easily proxied with existing HTTP methods, such as ICMP, ESP, etc.

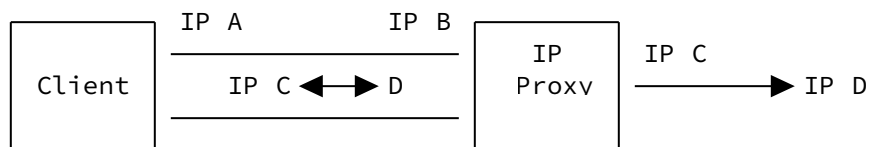


Figure 17: Proxied Flow Setup

In this case, the client specifies both a target hostname and an IP protocol number in the scope of its request, indicating that it only needs to communicate with a single host. The IP proxy is able to

perform DNS resolution on behalf of the client and allocate a specific outbound socket for the client instead of allocating an entire IP address to the client. In this regard, the request is similar to a traditional CONNECT proxy request.

The IP proxy assigns a single IPv6 address to the client (2001:db8:1234::a) and a route to a single IPv6 host (2001:db8:3456::b), scoped to SCTP. The client can send and receive SCTP IP packets to the remote host.

<pre> [[From Client]] SETTINGS H3_DATAGRAM = 1 STREAM(44): HEADERS :method = CONNECT :protocol = connect-ip :scheme = https :path = /proxy?target=target.example.com&ipproto=132 :authority = proxy.example.com capsule-protocol = ?1 DATAGRAM Quarter Stream ID = 11 Context ID = 0 Payload = Encapsulated SCTP/IP Packet </pre>	<pre> [[From IP Proxy]] SETTINGS ENABLE_CONNECT_PROTOCOL = 1 H3_DATAGRAM = 1 STREAM(44): HEADERS :status = 200 capsule-protocol = ?1 STREAM(44): DATA Capsule Type = ADDRESS_ASSIGN (Request ID = 0 IP Version = 6 IP Address = 2001:db8:1234::a IP Prefix Length = 128) STREAM(44): DATA Capsule Type = ROUTE_ADVERTISEMENT (IP Version = 6 Start IP Address = 2001:db8:3456::b End IP Address = 2001:db8:3456::b IP Protocol = 132) DATAGRAM Quarter Stream ID = 11 Context ID = 0 Payload = Encapsulated SCTP/IP Packet </pre>
--	--

Figure 18: Proxied SCTP Flow Example

8.3. Proxied Connection Racing

The following example shows a setup where a client is proxying UDP packets through an IP proxy in order to control connection establishment racing through an IP proxy, as defined in Happy Eyeballs [HEv2]. This example is a variant of the proxied flow, but highlights how IP-level proxying can enable new capabilities even for TCP and UDP.

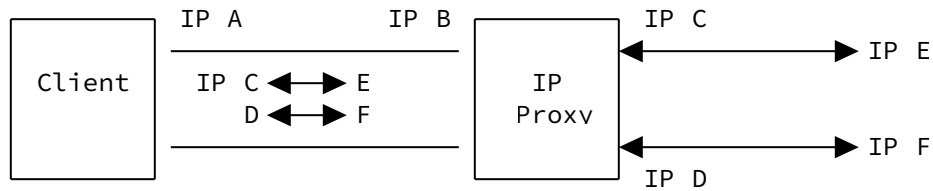


Figure 19: Proxied Connection Racing Setup

As with proxied flows, the client specifies both a target hostname and an IP protocol number in the scope of its request. When the IP proxy performs DNS resolution on behalf of the client, it can send the various remote address options to the client as separate routes. It can also ensure that the client has both IPv4 and IPv6 addresses assigned.

The IP proxy assigns the client both an IPv4 address (192.0.2.3) and an IPv6 address (2001:db8:1234::a) to the client, as well as an IPv4 route (198.51.100.2) and an IPv6 route (2001:db8:3456::b), which represent the resolved addresses of the target hostname, scoped to UDP. The client can send and receive UDP IP packets to either one of the IP proxy addresses to enable Happy Eyeballs through the IP proxy.

[[From Client]]

SETTINGS

H3_DATAGRAM = 1

STREAM(44): HEADERS

:method = CONNECT

:protocol = connect-ip

:scheme = https

:path = /proxy?ipproto=17

:authority = proxy.example.com

capsule-protocol = ?1

[[From IP Proxy]]

SETTINGS

ENABLE_CONNECT_PROTOCOL = 1

H3_DATAGRAM = 1

STREAM(44): HEADERS

:status = 200

capsule-protocol = ?1

STREAM(44): DATA

Capsule Type = ADDRESS_ASSIGN

(Request ID = 0

IP Version = 4

IP Address = 192.0.2.3

IP Prefix Length = 32),

(Request ID = 0

IP Version = 6

IP Address = 2001:db8::1234:1234

IP Prefix Length = 128)

STREAM(44): DATA

Capsule Type = ROUTE_ADVERTISEMENT

(IP Version = 4

Start IP Address = 198.51.100.2

End IP Address = 198.51.100.2

IP Protocol = 17),

(IP Version = 6

Start IP Address = 2001:db8:3456::b

End IP Address = 2001:db8:3456::b

IP Protocol = 17)

...

DATAGRAM

Quarter Stream ID = 11

Context ID = 0

Payload = Encapsulated IPv6 Packet

DATAGRAM

Quarter Stream ID = 11

Context ID = 0

Payload = Encapsulated IPv4 Packet

Figure 20: Proxied Connection Racing Example

9. Extensibility Considerations

Extensions to IP proxying in HTTP can define behavior changes to this mechanism. Such extensions **SHOULD** define new capsule types to exchange configuration information if needed. It is **RECOMMENDED** for extensions that modify addressing to specify that their extension capsules be sent before the ADDRESS_ASSIGN capsule and that they do not take effect until the ADDRESS_ASSIGN capsule is parsed. This allows modifications to address assignment to operate atomically. Similarly, extensions that modify routing **SHOULD** behave similarly with regards to the ROUTE_ADVERTISEMENT capsule.

10. Security Considerations

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

Falsifying IP source addresses in sent traffic has been common for denial of service attacks. Implementations of this mechanism need to ensure that they do not facilitate such attacks. In particular, there are scenarios where an endpoint knows that its peer is only allowed to send IP packets from a given prefix. For example, that can happen through out of band configuration information, or when allowed prefixes are shared via ADDRESS_ASSIGN capsules. In such scenarios, endpoints **MUST** follow the recommendations from [[BCP38](#)] to prevent source address spoofing.

Limiting request scope (see [Section 4.6](#)) allows two clients to share one of the proxy's external IP addresses if their requests are scoped to different IP protocol numbers. If the proxy receives an ICMP packet destined for that external IP address, it has the option to forward it back to the clients. However, some of these ICMP packets carry part of the original IP packet that triggered the ICMP response. Forwarding such packets can accidentally divulge information about one client's traffic to another client. To avoid this, proxies that forward ICMP on shared external IP addresses **MUST** inspect the invoking packet included in the ICMP packet and only forward the ICMP packet to the client whose scoping matches the invoking packet.

11. IANA Considerations

11.1. HTTP Upgrade Token

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

Value: connect-ip

Description: Proxying of IP Payloads

Expected Version Tokens: None

References: This document

11.2. Creation of the MASQUE URI Suffixes Registry

This document requests that IANA create a new "MASQUE URI Suffixes" registry maintained at IANA_URL_TBD. This new registry governs the path segment that immediately follows "masque" in paths that start with "/.well-known/masque/", see <<https://www.iana.org/assignments/well-known-uris>> for the registration of "masque" in the "Well-Known URIs" registry. This new registry contains three columns:

Path Segment: An ASCII string containing only characters allowed in tokens; see [Section 5.6.2](#) of [HTTP]. Entries in this registry **MUST** all have distinct entries in this column.

Description: A description of the entry.

Reference: An optional reference defining the use of the entry.

The registration policy for this registry is Expert Review; see [Section 4.5](#) of [IANA-POLICY].

There are initially two entries in this registry:

Path Segment	Description	Reference
udp	UDP Proxying	RFC 9298
ip	IP Proxying	This Document

Table 1: New MASQUE URI Suffixes

11.3. Updates to masque Well-Known URI

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

IANA is requested to update the "Reference" field to include this document in addition to previous values from that field.

IANA is requested to replace the "Related Information" field with "For sub-suffix allocations, see registry at IANA_URL_TBD." where IANA_URL_TBD is the URL of the new registry described in [Section 11.2](#).

11.4. Capsule Type Registrations

This document will request IANA to add the following values to the "HTTP Capsule Types" registry maintained at <https://www.iana.org/assignments/http-capsule-protocol/http-capsule-protocol.xhtml>.

Value	Type	Description	Reference
0x1ECA6A00	ADDRESS_ASSIGN	Address Assignment	This Document
0x1ECA6A01	ADDRESS_REQUEST	Address Request	This Document
0x1ECA6A02	ROUTE_ADVERTISEMENT	Route Advertisement	This Document

Table 2: New Capsules

12. References

12.1. Normative References

- [ABNF] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, DOI 10.17487/RFC2234, November 1997, <https://www.rfc-editor.org/rfc/rfc2234>.
- [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>.
- [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", RFC 9220, DOI 10.17487/RFC9220, June 2022, <https://www.rfc-editor.org/rfc/rfc9220>.
- [HTTP] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/rfc/rfc9110>.

[HTTP-DGRAM]

Schinazi, D. and L. Pardue, "HTTP Datagrams and the Capsule Protocol", RFC 9297, DOI 10.17487/RFC9297, August 2022, <<https://www.rfc-editor.org/rfc/rfc9297>>.

[HTTP/1.1] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1", STD 99, RFC 9112, DOI 10.17487/RFC9112, June 2022, <<https://www.rfc-editor.org/rfc/rfc9112>>.

[HTTP/2] Thomson, M., Ed. and C. Benfield, Ed., "HTTP/2", RFC 9113, DOI 10.17487/RFC9113, June 2022, <<https://www.rfc-editor.org/rfc/rfc9113>>.

[HTTP/3] Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <<https://www.rfc-editor.org/rfc/rfc9114>>.

[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>>.

[ICMP] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, DOI 10.17487/RFC0792, September 1981, <<https://www.rfc-editor.org/rfc/rfc792>>.

[ICMPv6] 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>>.

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

[PROXY-STATUS] Nottingham, M. and P. Sikora, "The Proxy-Status HTTP Response Header Field", RFC 9209, DOI 10.17487/RFC9209, June 2022, <<https://www.rfc-editor.org/rfc/rfc9209>>.

[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>>.

[TCP]

Postel, J., "Transmission Control Protocol", 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>>.

[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>>.

12.2. Informative References

[CONNECT-UDP]

Schinazi, D., "Proxying UDP in HTTP", RFC 9298, DOI 10.17487/RFC9298, August 2022, <<https://www.rfc-editor.org/rfc/rfc9298>>.

[HEV2]

Schinazi, D. and T. Pauly, "Happy Eyeballs Version 2: Better Connectivity Using Concurrency", RFC 8305, DOI 10.17487/RFC8305, December 2017, <<https://www.rfc-editor.org/rfc/rfc8305>>.

[IANA-PN]

IANA, "Protocol Numbers", <<https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml>>.

[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>>.

[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-03, 27 August 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-masque-ip-proxy-reqs-03>>.

Acknowledgments

The design of this method was inspired by discussions in the MASQUE working group around [[PROXY-REQS](#)]. The authors would like to thank participants in those discussions for their feedback. Additionally,

Mike Bishop, Lucas Pardue, and Alejandro Sedeño provided valuable feedback on the document.

Most of the text on client configuration is based on the corresponding text in [[CONNECT-UDP](#)].

Authors' Addresses

Tommy Pauly (editor)
Apple Inc.

Email: tpauly@apple.com

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

Email: dschinazi.ietf@gmail.com

Alex Chernyakhovsky
Google LLC

Email: achernya@google.com

Mirja Kuehlewind
Ericsson

Email: mirja.kuehlewind@ericsson.com

Magnus Westerlund
Ericsson

Email: magnus.westerlund@ericsson.com