Stream: Internet Engineering Task Force (IETF)
RFC: 9484
Updates: 9298
Category: Standards Track
Published: October 2023
ISSN: 2070-1721
Authors: T. Pauly, Ed. *Apple Inc.* D. Schinazi *Google LLC* A. Chernyakhovsky *Google LLC* M. Kühlewind *Ericsson* M. Westerlund *Ericsson*

RFC 9484
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 an IP proxy. This document updates RFC 9298.

Status of This Memo

Copyright Notice

Table of Contents

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 through an HTTP server acting as an IP-specific proxy over HTTP. This can be used for various use cases, such as remote access VPN, site-to-site 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].

This document updates [CONNECT-UDP] to change the "masque" well-known URI; see Section 12.3.

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. The term "client" is used in the HTTP sense; the client constructs the IP proxying request. If there are HTTP intermediaries (as defined in Section 3.7 of [HTTP]) between the client and the IP proxy, those are referred to as "intermediaries" in this document. The term "IP proxying endpoints" refers to both the client and the IP proxy.

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 in 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 a 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 the variable is either self-identifying or possible to exclude 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.

IP proxying over HTTP **MUST** be operated over TLS or QUIC encryption, or another equivalent encryption protocol, to provide confidentiality, integrity, and authentication.

4.1. IP Proxy Handling

Upon receiving an IP proxying request:

- If the recipient is configured to use another HTTP server, it will act as an intermediary by forwarding the request to the other 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. The IP proxy can choose to reject the IP proxying request. Otherwise, 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 (to obtain the corresponding IPv4 and/or IPv6 addresses via A and/or AAAA records) 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 IP proxying 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 IP proxying response indicates that the request has failed; thus, the client **MUST** abort the request.

Along with a successful IP proxying 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 server indicates a successful IP proxying 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 close the connection.

For example, the server 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 server indicates a successful IP proxying 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. As an example, any status code in the 3xx range will be treated as a failure and cause the client to abort the request.

For example, the server 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 [IPv6-ZONE-ID] 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 Internet Protocol Number; see the defined list in the "Assigned Internet Protocol Numbers" IANA registry [IANA-PN]. 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. The IP protocol indicated in the "ipproto" variable represents an allowable next header value carried in IP headers that are directly sent in HTTP Datagrams (the outermost IP headers). ICMP traffic is always allowed, regardless of the value of this field.

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.
- If "target" contains an IP prefix and the prefix length is strictly less than the length of the IP address in bits, the lower bits of the IP address that are not covered by the prefix length **MUST** all be set to 0.
- "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, i.e., if the client is not authorized to access any of the destinations included in the scope, then the IP proxy can immediately reject 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 (capsule type 0x01) allows an endpoint to assign its peer a list of IP addresses or prefixes. 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) = 0x01,
  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. It **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 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 the prefix length is strictly less than the length of the IP address in bits, the lower bits of the IP Address field that are not covered by the prefix length **MUST** all be set to 0.

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, it 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 VPN 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 (capsule type 0x02) 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) = 0x02,
  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. It **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. It **MUST** be less than or equal to the length of the IP Address field in bits. If the prefix length is strictly less than the length of the IP address in bits, the lower bits of the IP Address field that are not covered by the prefix length **MUST** all be set to 0.

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 (capsule type 0x03) 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 that 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) = 0x03,
  Length (i),
  IP Address Range (..) ...,
}
```

Figure 11: ROUTE_ADVERTISEMENT Capsule Format

The ROUTE_ADVERTISEMENT capsule contains a sequence of zero or more 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. It **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. If the value is not 0, it represents an allowable next header value carried in IP headers that are sent directly in HTTP Datagrams (the outermost IP headers). 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** update its local state regarding what its peer is willing to route (subject to local policy), such as by installing entries in a routing table.

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:

- The IP Version of A **MUST** be less than or equal to the 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 to the 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 its IP protocol set to zero and the other has it 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.

4.8. IPv6 Extension Headers

Both request scoping (see Section 4.6) and the ROUTE_ADVERTISEMENT capsule (see Section 4.7.3) use Internet Protocol Numbers. These numbers represent both upper layers (as defined in Section 2 of [IPv6], with examples that include TCP and UDP) and IPv6 extension headers (as defined in Section 4 of [IPv6], with examples that include Fragment and Options headers). IP proxies **MAY** reject requests to scope to protocol numbers that are used for extension headers. Upon receiving packets, implementations that support scoping or routing by Internet Protocol Number **MUST** walk the chain of extensions to find the outermost non-extension Internet Protocol Number to match against the scoping rule. Note that the ROUTE_ADVERTISEMENT capsule uses Internet Protocol Number 0 to indicate that all protocols are allowed; it does not restrict the route to the IPv6 Hop-by-Hop Options header (Section 4.3 of [IPv6]).

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 that 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 that 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).

7. IP Packet Handling

This document defines a tunneling mechanism that is conceptually an IP link. However, because links are attached to IP routers, implementations might need to handle some of the responsibilities of IP routers if they do not delegate them to another implementation, such as a kernel.

7.1. Link Operation

The IP forwarding tunnels described in this document are not fully featured "interfaces" in the IPv6 addressing architecture sense [IPv6-ADDR]. In particular, they do not necessarily have IPv6 link-local addresses. Additionally, IPv6 stateless autoconfiguration or router advertisement messages are not used in such interfaces, and neither is neighbor discovery.

When using HTTP/2 or HTTP/3, a client **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.

Note that it is possible for multiple proxied IP packets to be encapsulated in the same outer packet, for example, because a QUIC packet can carry more than one QUIC DATAGRAM frame. It is also possible for a proxied IP packet to span multiple outer packets, because a DATAGRAM capsule can be split across multiple QUIC or TCP packets.

7.2. Routing Operation

The requirements in this section are a repetition of requirements that apply to IP routers in general and might not apply to implementations of IP proxying that rely on external software for routing.

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. The endpoint can also choose to drop any received packets instead of forwarding them. If a received IP packet fails any correctness or policy checks, that is a forwarding error, not a protocol violation, as far as IP proxying is concerned; see Section 7.2.1. IP proxying endpoints **MAY** implement additional filtering policies on the IP packets they forward.

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.

When IP proxying endpoints forward IP packets between different links, they will decrement the IP Hop Count (or TTL) upon encapsulation but not upon 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]. This does not apply to IP packets generated by the IP proxying endpoint itself.

Implementers need to ensure that they do not forward any link-local traffic beyond the IP proxying interface that it was received on. IP proxying endpoints also need to properly reply to packets destined to link-local multicast addresses.

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 that 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 operate as routers and support IPv6 **MUST** ensure that the IP tunnel link MTU is at least 1280 bytes (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 outer QUIC connection that IP proxying is running over. (Assuming QUIC version 1 is in use, the overhead is 1 byte for the type, 20 bytes for the maximal connection ID length, 4 bytes for the maximal packet number length, 1 byte for the DATAGRAM frame type, 8 bytes for the maximal Quarter Stream ID, 1 byte for the zero Context ID, and 16 bytes for the Authenticated Encryption with Associated Data (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 are 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.

7.2.1. 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, 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 by generating ICMP packets and sending them using HTTP Datagrams.

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

- 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 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 and also from outside the target if scoping is in use (see Section 4.6).

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.

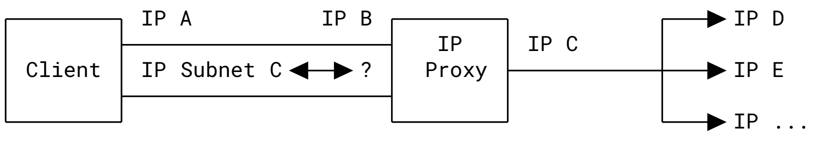

Figure 14: VPN Tunnel Setup

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 its assigned address as its source address.

```
[[ 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 = 192.0.2.41
                               IP Protocol = 0) // Any
                              (IP Version = 4
                               Start IP Address = 192.0.2.43
                               End IP Address = 192.0.2.255
                               IP Protocol = 0) // Any
```

Figure 16: VPN Split-Tunnel Example

8.2. Site-to-Site VPN

The following example shows how to connect a branch office network to a corporate network such that all machines on those networks can communicate. In this example, the IP proxying client is attached to the branch office network 192.0.2.0/24, and the IP proxy is attached to the corporate network 203.0.113.0/24. There are legacy clients on the branch office network that only allow maintenance requests from machines on their subnet, so the IP proxy is provisioned with an IP address from that subnet.

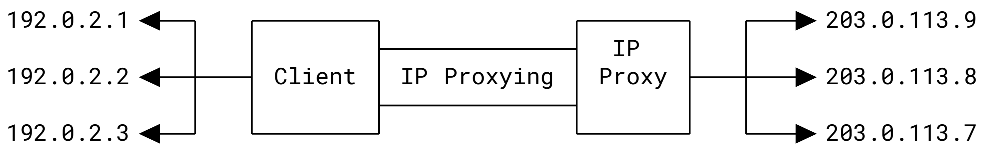

Figure 17: Site-to-Site VPN Example

In this case, the client does not specify any scope in its request. The IP proxy assigns the client an IPv4 address (203.0.113.100) and a split-tunnel route to the corporate network (203.0.113.0/24). The client assigns the IP proxy an IPv4 address (192.0.2.200) and a split-tunnel route to the branch office network (192.0.2.0/24). This allows hosts on both networks to communicate with each other and allows the IP proxy to perform maintenance on legacy hosts in the branch office. Note that IP proxying endpoints will decrement the IP Hop Count (or TTL) when encapsulating forwarded packets, so protocols that require that field be set to 255 will not function.

```
[[ 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 = /corp
:authority = proxy.example.com
capsule-protocol = ?1

                               STREAM(44): HEADERS
                               :status = 200
                               capsule-protocol = ?1

STREAM(44): DATA
Capsule Type = ADDRESS_ASSIGN
(Request ID = 0
IP Version = 4
IP Address = 192.0.2.200
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

                               STREAM(44): DATA
                               Capsule Type = ADDRESS_ASSIGN
                               (Request ID = 0
                                IP Version = 4
                                IP Address = 203.0.113.100
                                IP Prefix Length = 32)

                               STREAM(44): DATA
                               Capsule Type = ROUTE_ADVERTISEMENT
                               (IP Version = 4
                                Start IP Address = 203.0.113.0
                                End IP Address = 203.0.113.255
                                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 18: Site-to-Site VPN Capsule Example

8.3. 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 the Stream Control Transmission Protocol (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, Encapsulating Security Payload (ESP), etc.

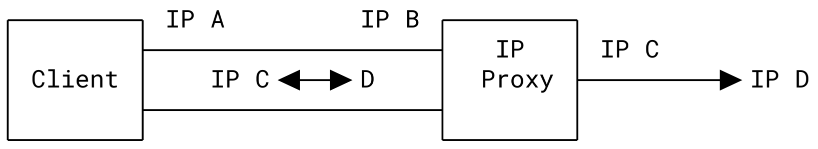

Figure 19: Proxied Flow Setup

In this case, the client specifies both a target hostname and an Internet 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 regular 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.

```
[[ 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 = /proxy?target=target.example.com&ipproto=132
:authority = proxy.example.com
capsule-protocol = ?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

                               DATAGRAM
                               Quarter Stream ID = 11
                               Context ID = 0
                               Payload = Encapsulated SCTP/IP Packet
```

Figure 20: Proxied SCTP Flow Example

8.4. 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.

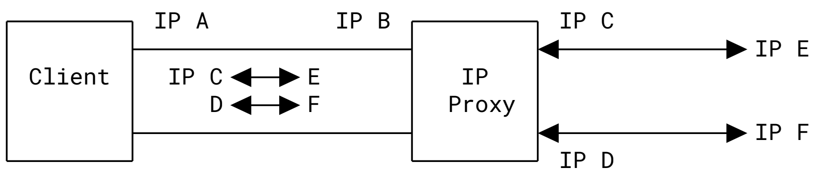

Figure 21: Proxied Connection Racing Setup

As with proxied flows, the client specifies both a target hostname and an Internet 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 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 ]]             [[ 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 = /proxy?target=target.example.com&ipproto=17
:authority = proxy.example.com
capsule-protocol = ?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 22: 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 regard to the ROUTE_ADVERTISEMENT capsule.

10. Performance Considerations

Bursty traffic can often lead to temporally correlated packet losses; in turn, this can lead to suboptimal responses from congestion controllers in protocols running inside the tunnel. To avoid this, IP proxying endpoints **SHOULD** strive to avoid increasing burstiness of IP traffic; they **SHOULD NOT** queue packets in order to increase batching beyond the minimal amount required to take advantage of hardware offloads.

When the protocol running inside the tunnel uses congestion control (e.g., [TCP] or [QUIC]), the proxied traffic will incur at least two nested congestion controllers. When tunneled packets are sent using QUIC DATAGRAM frames, the outer HTTP connection **MAY** disable congestion control for those packets that contain only QUIC DATAGRAM frames encapsulating IP packets. Implementers will benefit from reading the guidance in Section 3.1.11 of [UDP-USAGE].

When the protocol running inside the tunnel uses loss recovery (e.g., [TCP] or [QUIC]) and the outer 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, IP proxying **SHOULD** be performed over HTTP/3 to allow leveraging the QUIC DATAGRAM frame.

10.1. MTU Considerations

When using HTTP/3 with the QUIC Datagram extension [DGRAM], IP packets are transmitted in QUIC DATAGRAM frames. Since these frames cannot be fragmented, they can only carry packets up to a given length determined by the QUIC connection configuration and the Path MTU (PMTU). If an endpoint is using QUIC DATAGRAM frames and it attempts to route an IP packet through the tunnel that will not fit inside a QUIC DATAGRAM frame, the IP proxy **SHOULD NOT** send the IP packet in a DATAGRAM capsule, as that defeats the end-to-end unreliability characteristic that methods such as Datagram Packetization Layer PMTU Discovery (DPLPMTUD) depend on [DPLPMTUD]. In this scenario, the endpoint **SHOULD** drop the IP packet and send an ICMP Packet Too Big message to the sender of the dropped packet; see Section 3.2 of [ICMPv6].

10.2. ECN Considerations

If an IP proxying endpoint with a connection containing an IP proxying request stream disables congestion control, it cannot signal Explicit Congestion Notification (ECN) [ECN] support on that outer connection. That is, the QUIC sender **MUST** mark all IP headers with the Not ECN-Capable Transport (Not-ECT) codepoint for QUIC packets that are outside of congestion control. The endpoint can still report ECN feedback via QUIC ACK_ECN frames or the TCP ECN-Echo (ECE) bit, as the peer might not have disabled congestion control.

Conversely, if congestion control is not disabled on the outer congestion, the guidance in [ECN-TUNNEL] about transferring ECN marks between inner and outer IP headers does not apply because the outer connection will react correctly to congestion notifications if it uses ECN. The inner traffic can also use ECN, independently of whether it is in use on the outer connection.

10.3. Differentiated Services Considerations

Tunneled IP packets can have Differentiated Services Code Points (DSCPs) [DSCP] set in the traffic class IP header field to request a particular per-hop behavior. If an IP proxying endpoint is configured as part of a Differentiated Services domain, it **MAY** implement traffic differentiation based on these markings. However, the use of HTTP can limit the possibilities for differentiated treatment of the tunneled IP packets on the path between the IP proxying endpoints.

When an HTTP connection is congestion-controlled, marking packets with different DSCPs can lead to reordering between them, and that can in turn lead the underlying transport connection's congestion controller to perform poorly. If tunneled packets are subject to congestion control by the outer connection, they need to avoid carrying DSCP markings that are not equivalent in forwarding behavior to prevent this situation. In this scenario, the IP proxying endpoint **MUST NOT** copy the DSCP field from the inner IP header to the outer IP header of the packet carrying this packet. Instead, an application would need to use separate connections to the proxy, one for each DSCP. Note that this document does not define a way for requests to scope to particular DSCP values; such support is left to future extensions.

If tunneled packets use QUIC datagrams and are not subject to congestion control by the outer connection, the IP proxying endpoints **MAY** translate the DSCP field value from the tunneled traffic to the outer IP header. IP proxying endpoints **MUST NOT** coalesce multiple inner packets into the same outer packet unless they have the same DSCP marking or an equivalent traffic class. Note that the ability to translate DSCP values is dependent on the tunnel ingress and egress belonging to the same Differentiated Service domain or not.

11. Security Considerations

There are significant risks in allowing arbitrary clients to establish a tunnel that permits sending to arbitrary hosts, regardless of whether tunnels are scoped to specific hosts or not. Bad actors could abuse this capability to send traffic and have it attributed to the IP proxy. HTTP servers that support IP proxying **SHOULD** restrict its use to authenticated users. Depending on the deployment, possible authentication mechanisms include mutual TLS between IP proxying endpoints, HTTP-based authentication via the HTTP Authorization header [HTTP], or even bearer tokens. Proxies can enforce policies for authenticated users to further constrain client behavior or deal with possible abuse. For example, proxies can rate limit individual clients that send an excessively large amount of traffic through the proxy. As another example, proxies can restrict address (prefix) assignment to clients based on certain client attributes, such as geographic location.

Address assignment can have privacy implications for endpoints. For example, if a proxy partitions its address space by the number of authenticated clients and then assigns distinct address ranges to each client, target hosts could use this information to determine when IP packets correspond to the same client. Avoiding such tracking vectors may be important for certain proxy deployments. Proxies **SHOULD** avoid persistent per-client address (prefix) assignment when possible.

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 Internet 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.

Implementers will benefit from reading the guidance in [TUNNEL-SECURITY]. Since there are known risks with some IPv6 extension headers (e.g., [ROUTING-HDR]), implementers need to follow the latest guidance regarding handling of IPv6 extension headers.

Transferring DSCP markings from inner to outer packets (see Section 10.3) exposes end-to-end flow level information to an on-path observer between the IP proxying endpoints. This can potentially expose a single end-to-end flow. Because of this, such use of DSCPs in privacy-sensitive contexts is **NOT RECOMMENDED**.

Opportunistic sending of IP packets (see Section 7.1) is not allowed in HTTP/1.x because a server could reject the HTTP Upgrade and attempt to parse the IP packets as a subsequent HTTP request, allowing request smuggling attacks; see [OPTIMISTIC]. In particular, an intermediary that re-encodes a request from HTTP/2 or 3 to HTTP/1.1 **MUST NOT** forward any received capsules until it has parsed a successful IP proxying response.

12. IANA Considerations

12.1. HTTP Upgrade Token Registration

IANA has registered "connect-ip" in the "HTTP Upgrade Tokens" registry maintained at <https://www.iana.org/assignments/http-upgrade-tokens>.

Value: connect-ip

Description: Proxying of IP Payloads

Expected Version Tokens: None

References: RFC 9484

12.2. MASQUE URI Suffixes Registry Creation

IANA has created the "MASQUE URI Suffixes" registry maintained at <https://www.iana.org/assignments/masque>. The registration policy is Expert Review; see Section 4.5 of [IANA-POLICY]. 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 registry's initial entries are as follows:

Path Segment	Description	Reference
udp	UDP Proxying	RFC 9298
ip	IP Proxying	RFC 9484

Table 1: MASQUE URI Suffixes Registry

Designated experts for this registry are advised that they should approve all requests as long as the expert believes that both (1) the requested Path Segment will not conflict with existing or expected future IETF work and (2) the use case is relevant to proxying.

12.3. Updates to masque Well-Known URI Registration

IANA has updated the entry for the "masque" URI suffix in the "Well-Known URIs" registry maintained at <https://www.iana.org/assignments/well-known-uris>.

IANA has updated the "Reference" field to include this document and has replaced the "Related Information" field with "For sub-suffix allocations, see the registry at <https://www.iana.org/assignments/masque>.".

12.4. HTTP Capsule Types Registrations

IANA has added the following values to the "HTTP Capsule Types" registry maintained at <https://www.iana.org/assignments/masque>.

Value	Capsule Type
0x01	ADDRESS_ASSIGN
0x02	ADDRESS_REQUEST
0x03	ROUTE_ADVERTISEMENT

Table 2: New Capsules

All of these new entries use the following values for these fields:

Status: permanent

Reference: RFC 9484

Change Controller: IETF

Contact: masque@ietf.org

Notes: None

13. References

13.1. Normative References

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

[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/info/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/info/rfc9221>.

[DSCP] 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://www.rfc-editor.org/info/rfc2474>.

[ECN] 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://www.rfc-editor.org/info/rfc3168>.

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

[EXT-CONNECT3] Hamilton, R., "Bootstrapping WebSockets with HTTP/3", RFC 9220, DOI 10.17487/RFC9220, June 2022, <https://www.rfc-editor.org/info/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/info/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/info/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/info/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/info/rfc9113>.

[HTTP/3] Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <https://www.rfc-editor.org/info/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/info/rfc8126>.

[ICMP] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, DOI 10.17487/RFC0792, September 1981, <https://www.rfc-editor.org/info/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/info/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/info/rfc8200>.

[IPv6-ZONE-ID] Carpenter, B., Cheshire, S., and R. Hinden, "Representing IPv6 Zone Identifiers in Address Literals and Uniform Resource Identifiers", RFC 6874, DOI 10.17487/RFC6874, February 2013, <https://www.rfc-editor.org/info/rfc6874>.

[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/info/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/info/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/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>.

[TCP] Eddy, W., Ed., "Transmission Control Protocol (TCP)", STD 7, RFC 9293, DOI 10.17487/RFC9293, August 2022, <https://www.rfc-editor.org/info/rfc9293>.

[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/info/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/info/rfc3986>.

13.2. Informative References

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

[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/info/rfc8899>.

[ECN-TUNNEL] Briscoe, B., "Tunnelling of Explicit Congestion Notification", RFC 6040, DOI 10.17487/RFC6040, November 2010, <https://www.rfc-editor.org/info/rfc6040>.

[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/info/rfc8305>.

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

[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/info/rfc4301>.

[IPv6-ADDR] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <https://www.rfc-editor.org/info/rfc4291>.

[OPTIMISTIC] Schwartz, B. M., "Security Considerations for Optimistic Use of HTTP Upgrade", Work in Progress, Internet-Draft, draft-schwartz-httpbis-optimistic-upgrade-00, 21 August 2023, <https://datatracker.ietf.org/doc/html/draft-schwartz-httpbis-optimistic-upgrade-00>.

[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>.

[ROUTING-HDR] Abley, J., Savola, P., and G. Neville-Neil, "Deprecation of Type 0 Routing Headers in IPv6", RFC 5095, DOI 10.17487/RFC5095, December 2007, <https://www.rfc-editor.org/info/rfc5095>.

[TUNNEL-SECURITY] Krishnan, S., Thaler, D., and J. Hoagland, "Security Concerns with IP Tunneling", RFC 6169, DOI 10.17487/RFC6169, April 2011, <https://www.rfc-editor.org/info/rfc6169>.

[UDP-USAGE] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017, <https://www.rfc-editor.org/info/rfc8085>.

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 Kühlewind
Ericsson
Email: mirja.kuehlewind@ericsson.com

Magnus Westerlund
Ericsson
Email: magnus.westerlund@ericsson.com