Workgroup: MASQUE
Internet-Draft:
draft-ietf-masque-connect-ip-03
Published: 28 September 2022
Intended Status: Standards Track
Expires: 1 April 2023
Authors: T. Pauly, Ed. D. Schinazi A. Chernyakhovsky
Apple Inc. Google LLC Google LLC
M. Kuehlewind M. Westerlund
Ericsson Ericsson

IP Proxying Support for HTTP

Abstract

This document describes a method of proxying IP packets over HTTP. This protocol is similar to CONNECT-UDP, but allows transmitting arbitrary IP packets, without being limited to just TCP like CONNECT or UDP like CONNECT-UDP.

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 1 April 2023.

Copyright Notice

Table of Contents

1. Introduction

This document describes a method of proxying IP packets over HTTP. When using HTTP/2 or HTTP/3, IP proxying uses HTTP Extended CONNECT as described in [EXT-CONNECT2] and [EXT-CONNECT3]. When using HTTP/1.x, IP proxying uses HTTP Upgrade as defined in Section 7.8 of [SEMANTICS]. This protocol is similar to CONNECT-UDP [CONNECT-UDP], but allows transmitting arbitrary IP packets, without being limited to just TCP like CONNECT [SEMANTICS] or UDP like CONNECT-UDP.

The HTTP Upgrade Token defined for this mechanism is "connect-ip", which is also referred to as CONNECT-IP in this document.

The CONNECT-IP protocol allows endpoints to set up a tunnel for proxying IP packets using an HTTP proxy. This can be used for various solutions that include general-purpose packet tunnelling, such as for a point-to-point or point-to-network VPN, or for limited forwarding of packets to specific hosts.

Forwarded IP packets can be sent efficiently via the proxy using HTTP Datagram support [HTTP-DGRAM].

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

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" (Section 4.1). The optionality of the variables needs to be considered when defining the template so that either the variable is self-identifying or it works 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 4.1.
- 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 CONNECT-UDP, some client configurations for CONNECT-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. The CONNECT-IP Protocol

This document defines the "connect-ip" HTTP Upgrade Token. "connect-ip" uses the Capsule Protocol as defined in [HTTP-DGRAM].

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.

When using HTTP/2 or HTTP/3, the following requirements apply to requests:

- The ":method" pseudo-header field **SHALL** be set to "CONNECT".
- The ":protocol" pseudo-header field **SHALL** be set to "connect-ip".
- The ":authority" pseudo-header field **SHALL** contain the host and port of the proxy, not an individual endpoint with which a connection is desired.
- The contents of the ":path" pseudo-header **SHALL** be determined by the URI template expansion, 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 4.1.

The client **SHOULD** also include the "Capsule-Protocol" header with a value of "?1" to negotiate support for sending and receiving HTTP capsules ([HTTP-DGRAM]).

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

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

The lifetime of the forwarding tunnel is tied to the CONNECT stream. Closing the stream (in HTTP/3 via the FIN bit on a QUIC STREAM frame, or a QUIC RESET_STREAM frame) closes the associated forwarding tunnel.

Along with a successful response, the proxy can send capsules to assign addresses and advertise routes to the client (Section 4.2).

The client can also assign addresses and advertise routes to the proxy for network-to-network routing.

4.1. Limiting Request Scope

Unlike CONNECT-UDP requests, which require specifying a target host, CONNECT-IP requests can allow endpoints to send arbitrary IP packets to any host. The client can choose to restrict a given request to a specific prefix or IP protocol by adding parameters to its request. When the server 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 server can assign the same public IP address to two CONNECT-IP requests that are scoped to different prefixes and/or different protocols.

The scope of the request is indicated by the client to the 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 literals and IPv4 literals. 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 server is expected to perform DNS resolution to determine which route(s) to advertise to the client. The server **SHOULD** send a ROUTE_ADVERTISEMENT capsule that includes routes for all addresses that were resolved for the requested hostname, that are accessible to the server, and belong to an address family for which the server also sends an Assigned Address.

ipproto: The variable "ipproto" contains an IP protocol number, as defined in the "Assigned Internet Protocol Numbers" IANA registry. 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 2, using notation from [ABNF]. Additionally:

- if "target" contains an IPv6 literal, 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 an integer between 0 and the length of the IP address in bits, inclusive.

- "ipproto" **MUST** represent an 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 2: URI Template Variable Format

4.2. 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.

4.2.1. ADDRESS_ASSIGN Capsule

The ADDRESS_ASSIGN capsule (see Section 11.3 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 3: 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 4: Assigned Address Format

Request ID: If this address assignment is in response to an Address Request (see Section 4.2.2), 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. **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. 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.

Note that an ADDRESS_ASSIGN capsule can also indicate that a previously assigned address is no longer assigned. An ADDRESS_ASSIGN capsule can also be empty.

In some deployments of CONNECT-IP, 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.

4.2.2. ADDRESS_REQUEST Capsule

The ADDRESS_REQUEST capsule (see Section 11.3 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 5: ADDRESS_REQUEST Capsule Format

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

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

Figure 6: Requested Address Format

Request ID: 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. **MUST** be either 4 or 6.

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

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

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.

Upon receiving the ADDRESS_REQUEST capsule, an endpoint **SHOULD** assign an IP address to its peer, and then respond with an ADDRESS_ASSIGN capsule to inform the peer of the assignment. Note that the receiver of the ADDRESS_REQUEST capsule is not required to assign the requested address, and that it can also assign some requested addresses but not others.

4.2.3. ROUTE_ADVERTISEMENT Capsule

The ROUTE_ADVERTISEMENT capsule (see Section 11.3 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 7: 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 8: IP Address Range Format

IP Version: IP Version of this range. **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 lesser or equal to the End IP Address.

IP Protocol: The Internet Protocol Number for traffic that can be sent to this range. If the value is 0, all protocols are allowed.

ICMP traffic is always allowed, regardless of the value of this field.

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 lesser or equal than IP Version of B
- If the IP Version of A and B are equal, the IP Protocol of A **MUST** be lesser 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 received a ROUTE_ADVERTISEMENT capsule that does not meet these requirements, it **MUST** abort the stream.

5. Context Identifiers

This protocol allows future extensions to exchange HTTP Datagrams which carry different semantics from IP packets. For example, an extension could define a way to send compressed IP header fields. In order to allow for this extensibility, all HTTP Datagrams associated with IP proxying request streams start with a context ID, 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 packets, while non-zero values are dynamically allocated: non-zero even-numbered context IDs are client-allocated, and odd-numbered context IDs are server-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 assigned different semantics in distinct requests, potentially with different semantics. Context IDs **MUST NOT** be re-

allocated within a given HTTP namespace but **MAY** be allocated in any order. Once allocated, any context ID can be used by both client and server - only allocation carries separate namespaces to avoid requiring synchronization.

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. Depending on the method being used, it is possible for datagrams to be received with Context IDs which have not yet been registered, for instance due to reordering of the datagram and the registration packets 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 9. 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 9: IP Proxying HTTP Datagram Format

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 proxy if it responds to the request with a failure, or if the datagrams are received by the 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 proxy if it

chooses to provide different addressing or routing information than what the client assumed.

When a CONNECT-IP endpoint receives an HTTP Datagram containing an IP packet, it will parse the packet's IP header, perform any local policy checks (e.g., source address validation), check their routing table to pick an outbound interface, and then send the IP packet on that interface.

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

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

IPv6 requires that every link have an MTU of at least 1280 bytes [IPv6]. Since CONNECT-IP 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 a CONNECT-IP link can be limited by the MTU of the QUIC connection that CONNECT-IP is operating over. This can lead to situations where the IPv6 minimum link MTU is violated. CONNECT-IP endpoints that support IPv6 **MUST** ensure that the CONNECT-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 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 CONNECT-IP 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.)

* CONNECT-IP 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 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 CONNECT-IP stream.

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

7. Error Signalling

Since CONNECT-IP 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 doesn't 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, CONNECT-IP 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 CONNECT-IP 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 sends ROUTE_ADVERTISEMENT capsules, its routes **SHOULD** include an allowance for receiving ICMP messages. If an endpoint does not send ROUTE_ADVERTISEMENT capsules, such as a client opening an IP flow through a 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 CONNECT-IP peer.

8. Examples

CONNECT-IP 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 CONNECT-IP 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 server through the proxy. Such VPN setups can be either full-tunnel or split-tunnel.

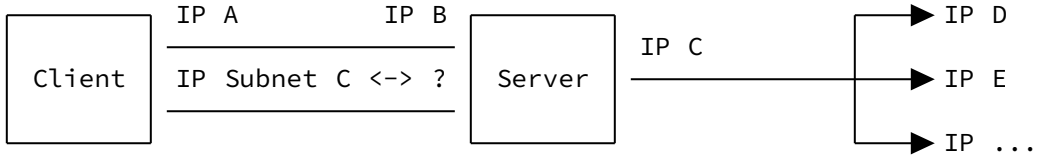

Figure 10: VPN Tunnel Setup

In this case, the client does not specify any scope in its request. The server 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 Server ]]

SETTINGS
H3_DATAGRAM = 1

                              SETTINGS
                              SETTINGS_ENABLE_CONNECT_PROTOCOL = 1
                              H3_DATAGRAM = 1

STREAM(44): HEADERS
:method = CONNECT
:protocol = connect-ip
:scheme = https
:path = /vpn
:authority = server.example.com
capsule-protocol = ?1

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

                              STREAM(44): CAPSULE
                              Capsule Type = ADDRESS_ASSIGN
                              IP Version = 4
                              IP Address = 192.0.2.11
                              IP Prefix Length = 32

                              STREAM(44): CAPSULE
                              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 11: 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 Server ]]

                               STREAM(44): CAPSULE
                               Capsule Type = ADDRESS_ASSIGN
                               IP Version = 4
                               IP Address = 192.0.2.42
                               IP Prefix Length = 32

                               STREAM(44): CAPSULE
                               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 12: 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.

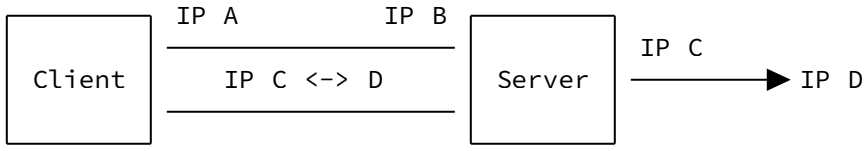

Figure 13: Proxied Flow Setup

In this case, the client specfies 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 proxy server 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 server assigns a single IPv6 address to the client (2001:db8::1234:1234) and a route to a single IPv6 host

(2001:db8::3456), scoped to SCTP. The client can send and recieve SCTP IP packets to the remote host.

```
[[ From Client ]]             [[ From Server ]]

SETTINGS
H3_DATAGRAM = 1
                              SETTINGS
                              SETTINGS_ENABLE_CONNECT_PROTOCOL = 1
                              H3_DATAGRAM = 1

STREAM(52): HEADERS
:method = CONNECT
:protocol = connect-ip
:scheme = https
:path = /proxy?target=target.example.com&ipproto=132
:authority = server.example.com
capsule-protocol = ?1

                              STREAM(52): HEADERS
                              :status = 200
                              capsule-protocol = ?1

                              STREAM(52): CAPSULE
                              Capsule Type = ADDRESS_ASSIGN
                              IP Version = 6
                              IP Address = 2001:db8::1234:1234
                              IP Prefix Length = 128

                              STREAM(52): CAPSULE
                              Capsule Type = ROUTE_ADVERTISEMENT
                              (IP Version = 6
                               Start IP Address = 2001:db8::3456
                               End IP Address = 2001:db8::3456
                               IP Protocol = 132)

DATAGRAM
Quarter Stream ID = 13
Context ID = 0
Payload = Encapsulated SCTP/IP Packet

                              DATAGRAM
                              Quarter Stream ID = 13
                              Context ID = 0
                              Payload = Encapsulated SCTP/IP Packet
```

Figure 14: Proxied SCTP Flow Example

8.3. Proxied Connection Racing

The following example shows a setup where a client is proxying UDP packets through a CONNECT-IP proxy in order to control connection establishement racing through a 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.

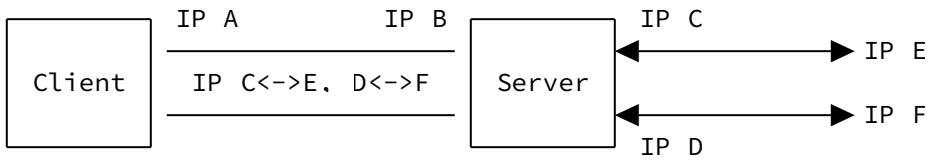

Figure 15: Proxied Connection Racing Setup

As with proxied flows, the client specfies both a target hostname and an IP protocol number in the scope of its request. When the proxy server 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 server assigns the client both an IPv4 address (192.0.2.3) and an IPv6 address (2001:db8::1234:1234) to the client, as well as an IPv4 route (198.51.100.2) and an IPv6 route (2001:db8::3456), which represent the resolved addresses of the target hostname, scoped to UDP. The client can send and recieve UDP IP packets to the either of the server addresses to enable Happy Eyeballs through the proxy.

```
[[ From Client ]]             [[ From Server ]]

SETTINGS
H3_DATAGRAM = 1

                              SETTINGS
                              SETTINGS_ENABLE_CONNECT_PROTOCOL = 1
                              H3_DATAGRAM = 1

STREAM(44): HEADERS
:method = CONNECT
:protocol = connect-ip
:scheme = https
:path = /proxy?ipproto=17
:authority = server.example.com
capsule-protocol = ?1

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

                              STREAM(44): CAPSULE
                              Capsule Type = ADDRESS_ASSIGN
                              IP Version = 4
                              IP Address = 192.0.2.3
                              IP Prefix Length = 32
                              IP Version = 6
                              IP Address = 2001:db8::1234:1234
                              IP Prefix Length = 128

                              STREAM(44): CAPSULE
                              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
                               End IP Address = 2001:db8::3456
                               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 16: Proxied Connection Racing Example

9. Extensibility Considerations

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

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.

11. IANA Considerations

11.1. CONNECT-IP 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: The CONNECT-IP Protocol
Expected Version Tokens: None
References: This document

11.2. 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 add the following sentence to the "Related Information"

field: Includes all resources identified with the path prefix "/.well-known/masque/ip/".

11.3. Capsule Type Registrations

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

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

Table 1: 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-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>.

[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>.

[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>.

[SEMANTICS] 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>.

[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>.

[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.

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