Workgroup: MASQUE
Internet-Draft:
draft-ietf-masque-connect-ip-01
Published: 5 March 2022
Intended Status: Standards Track
Expires: 6 September 2022
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.

Discussion Venues

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

Discussion of this document takes place on the Multiplexed Application Substrate over QUIC Encryption Working Group mailing list (masque@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/masque/.

Source for this draft and an issue tracker can be found at https://github.com/tfpauly/draft-age-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 6 September 2022.

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 "ip_proto". Examples are shown below:

```
https://masque.example.org/{target}/{ip_proto}/
https://proxy.example.org:4443/masque?t={target}&p={ip_proto}
https://proxy.example.org:4443/masque{?target,ip_proto}
https://masque.example.org/?user=bob
```

Figure 1: URI Template Examples

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 host or IP protocol by adding parameters to its request. When the server knows that a request is scoped to a target host 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 hosts and/or different protocols.

CONNECT-IP uses URI template variables (Section 3) to determine the scope of the request for packet proxying. All variables defined here are optional, and have default values if not included.

The defined variables are:

target: The variable "target" contains a hostname or IP address of a specific host to which the client wants to proxy packets. If the "target" variable is not specified, the client is requesting to communicate with any allowable host. If the target is an IP address, 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 usable resolved addresses for the requested hostname.

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 0, or the variable is not included, the client is requesting to use any IP protocol.

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 9.2 for the value of the capsule type) allows an endpoint to inform its peer that it has assigned an IP address or prefix to it. The ADDRESS_ASSIGN capsule allows assigning a prefix which can contain multiple addresses. 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),
  IP Version (8),
  IP Address (32..128),
  IP Prefix Length (8),
}
```

Figure 2: ADDRESS_ASSIGN Capsule Format

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

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

IP Prefix Length: 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.

If an endpoint receives multiple ADDRESS_ASSIGN capsules, all of the assigned addresses or prefixes can be used. For example, multiple ADDRESS_ASSIGN capsules are necessary to assign both IPv4 and IPv6 addresses.

4.2.2. ADDRESS_REQUEST Capsule

The ADDRESS_REQUEST capsule (see Section 9.2 for the value of the capsule type) allows an endpoint to request assignment of an IP address from its peer. This capsule is not required for simple client/proxy communication where the client only expects to receive one address from the proxy. 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),
  IP Version (8),
  IP Address (32..128),
  IP Prefix Length (8),
}
```

Figure 3: ADDRESS_REQUEST Capsule Format

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

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

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

Upon receiving the ADDRESS_REQUEST capsule, an endpoint SHOULD assign an IP address to its peer, and then respond with an ADDRESS_ASSIGN capsule to inform the peer of the assignment.

4.2.3. ROUTE_ADVERTISEMENT Capsule

The ROUTE_ADVERTISEMENT capsule (see Section 9.2 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 4: 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 5: 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.

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 lesser 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 6. Note that when HTTP Datagrams are encoded using QUIC DATAGRAM frames, the Context ID field defined below directly follows the Quarter Stream ID field which is at the start of the QUIC DATAGRAM frame payload:

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

Figure 6: 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.

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].

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

7. 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.

7.1. 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.

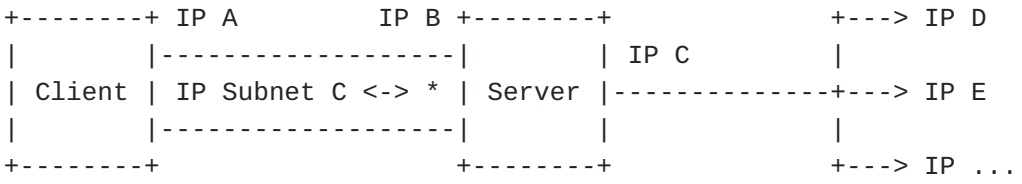

Figure 7: VPN Tunnel Setup

In this case, the client does not specify any scope in its request. The server assigns the client an IPv4 address to the client (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 8: 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 9: VPN Split-Tunnel Capsule Example

7.2. 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.

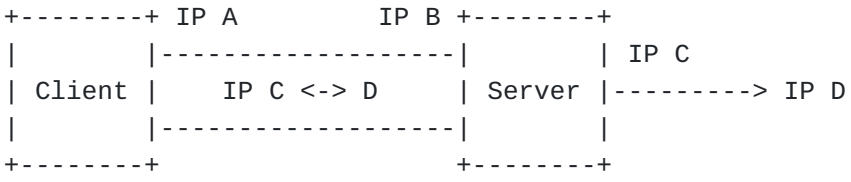

Figure 10: Proxied Flow Setup

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 11: Proxied SCTP Flow Example

7.3. 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.

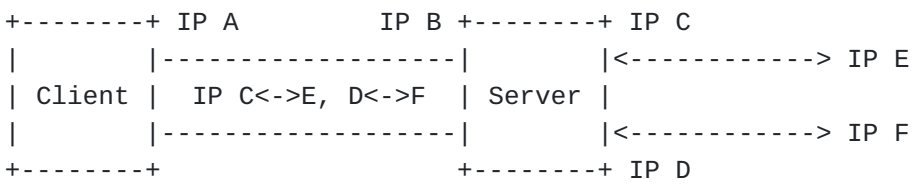

Figure 12: Proxied Connection Racing Setup

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

                              STREAM(44): CAPSULE
                              Capsule Type = ADDRESS_ASSIGN
                              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 13: Proxied Connection Racing Example

8. Security Considerations

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

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

9. IANA Considerations

9.1. 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

9.2. Capsule Type Registrations

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

Value	Type	Description	Reference
0xfff100	ADDRESS_ASSIGN	Address Assignment	This Document
0xfff101	ADDRESS_REQUEST	Address Request	This Document
0xfff102	ROUTE_ADVERTISEMENT	Route Advertisement	This Document

Table 1: New Capsules

10. References

10.1. Normative References

[BCP38]

Ferguson, P. and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing", BCP 38, RFC 2827, DOI 10.17487/RFC2827, May 2000, <https://www.rfc-editor.org/rfc/rfc2827>.

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

[EXT-CONNECT3] Hamilton, R., "Bootstrapping WebSockets with HTTP/3", Work in Progress, Internet-Draft, draft-ietf-httpbis-h3-websockets-04, 8 February 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-h3-websockets-04>.

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

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

[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>.

10.2. Informative References

[AUTH]

Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, <https://www.rfc-editor.org/rfc/rfc7235>.

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

[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.

Authors' Addresses

Tommy Pauly (editor)
Apple Inc.

Email: tpauly@apple.com

David Schinazi
Google LLC

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