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.


Discussion of this document takes place on the MASQUE Working Group mailing list (mailto:masque@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/masque/.

Source for this draft and an issue tracker can be found at https://github.com/ietf-wg-masque/draft-ietf-masque-connect-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-Draft                HTTP IP Proxy                    July 2022

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 12 January 2023.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction ................................................. 3
2. Conventions and Definitions .................................... 3
3. Configuration of Clients ....................................... 3
4. The CONNECT-IP Protocol ...................................... 4
   4.1. Limiting Request Scope .................................. 6
   4.2. Capsules ................................................ 6
       4.2.1. ADDRESS_ASSIGN Capsule .............................. 7
       4.2.2. ADDRESS_REQUEST Capsule ........................... 8
       4.2.3. ROUTE_ADVERTISEMENT Capsule ......................... 8
5. Context Identifiers .......................................... 10
6. HTTP Datagram Payload Format .................................. 10
7. Error Signalling ............................................. 12
8. Examples .................................................... 12
   8.1. Remote Access VPN ....................................... 12
   8.2. IP Flow Forwarding ..................................... 15
   8.3. Proxied Connection Racing ................................ 17
9. Security Considerations ...................................... 19
10. IANA Considerations ......................................... 19
   10.1. CONNECT-IP HTTP Upgrade Token ........................... 19
10.2. Capsule Type Registrations ................................. 19
11. References .................................................. 20
   11.1. Normative References ................................... 20
   11.2. Informative References ................................ 21
Acknowledgments .................................................. 22
Authors’ Addresses ............................................... 22
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://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.

* 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.

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 field 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.

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 DNS hostname (reg-name) or IP prefix (IPv6address / IPv4address ["%2F" 1*3DIGIT]) ([URI] syntax elements within parentheses) 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. 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 ADDRESS_ASSIGN capsule. Note that IPv6 scoped addressing zone identifiers are not supported.

**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.

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 10.2 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.

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, endpoints need to send
ADDRESS_ASSIGN capsules to allow their peers to send traffic.
4.2.2. ADDRESS_REQUEST Capsule

The ADDRESS_REQUEST capsule (see Section 10.2 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 10.2 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 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 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].

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 HTTP intermediaries are not in use, both CONNECT-IP endpoints can pad the QUIC INITIAL packets of the underlying QUIC connection that CONNECT-IP is running over.

* if HTTP intermediaries are in use, CONNECT-IP endpoints can enter in an out of band agreement with the intermediaries to ensure that endpoints and intermediaries pad QUIC INITIAL packets.

* 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 one of the two previous techniques is sufficient, they MUST use this method.

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] to signal the forwarding error to its 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.
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.
SETTINGS
H3_DATAGRAM = 1

STREAM(44): HEADERS
:method = CONNECT
:protocol = connect-ip
:scheme = https
:path = /vpn
:authority = server.example.com
:status = 200

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

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.

```
+--------+ IP A         IP B +--------+
|        |-------------------|        | IP C
| Client |    IP C <-> D     | Server |---------> IP D
+--------+                   +--------+

Figure 10: Proxied Flow Setup
```

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

```
+--------+ IP A   IP B +--------+ IP C
    |        |-------------------|        |<------------> IP E
    | Client |  IP C<->E, D<->F| Server |<------------> IP F
    +--------+-------------------+--------+ IP D
```

As with proxied flows, the client specifies 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 receive UDP IP packets to the either of the server addresses to enable Happy Eyeballs through the proxy.

```
[[ From Client ]]

SETTINGS
H3_DATAGRAM = 1

[[ From Server ]]

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)

...
9. 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.

10. IANA Considerations

10.1. 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

10.2. Capsule Type Registrations

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Type</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xfff100</td>
<td>ADDRESS_ASSIGN</td>
<td>Address Assignment</td>
<td>This Document</td>
</tr>
<tr>
<td>0xfff101</td>
<td>ADDRESS_REQUEST</td>
<td>Address Request</td>
<td>This Document</td>
</tr>
<tr>
<td>0xfff102</td>
<td>ROUTE_ADVERTISEMENT</td>
<td>Route Advertisement</td>
<td>This Document</td>
</tr>
</tbody>
</table>

Table 1: New Capsules

11. References

11.1. Normative References


11.2. Informative References

[CONNECT-UDP]

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

Abstract

The mechanism to proxy UDP in HTTP only allows each proxying request to transmit to a specific host and port. This is well suited for UDP client-server protocols such as HTTP/3, but is not sufficient for some UDP peer-to-peer protocols like WebRTC. This document proposes an extension to UDP Proxying in HTTP that enables those use-cases.
1. Introduction

The mechanism to proxy UDP in HTTP [CONNECT-UDP] allows proxying UDP payloads [UDP] to a fixed host and port. Combined with the HTTP CONNECT method (see Section 9.3.6 of [HTTP]), it allows proxying the majority of a Web Browser's HTTP traffic. However WebRTC [WebRTC] relies on ICE [ICE] to provide connectivity between two Web browsers, and that in turn relies on the ability to send and receive UDP packets to multiple hosts. While it would be possible in theory to accomplish this by using multiple UDP proxying HTTP requests, HTTP semantics [HTTP] do not guarantee that those distinct requests will be handled by the same server, which can lead to the UDP packets being sent from distinct IP addresses, which in turn prevents ICE from operating correctly. Because of this, UDP Proxying requests cannot enable WebRTC connectivity between peers.
This document describes an extension to UDP Proxying in HTTP that allows sending and receiving UDP payloads to multiple hosts within the scope of a single UDP proxying HTTP request.

1.1. Conventions and Definitions

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

This document uses terminology from [CONNECT-UDP] and notational conventions from [QUIC].

2. Proxied UDP Listener Mechanism

In unextended UDP Proxying requests, the target host is encoded in the HTTP request path or query. For listener UDP proxying, it is instead conveyed in each HTTP Datagram, see Section 2.1.

When performing URI Template Expansion of the UDP proxying template (see Section 3 of [CONNECT-UDP]), the client sets both the target_host and the target_port variables to the '*' character (ASCII character 0x2A).

Before sending its UDP Proxying request to the proxy, the client allocates an even-numbered context ID, see Section 4 of [CONNECT-UDP]. The client then adds the "connect-udp-listen" header field to its proxying request, with the value equal to the context ID it has allocated, see Section 2.2.

2.1. HTTP Datagram Payload Format

When HTTP Datagrams [HTTP-DGRAM] associated with this listener UDP proxying request use the context ID sent with the connect-udp-listen header field, their Payload field (as defined in Section 5 of [CONNECT-UDP]) has the format defined in Figure 1:

Listener UDP Proxying HTTP Datagram Payload {
    IP Version (8),
    IP Address (32..128),
    UDP Port (16),
    UDP Payload (...),
}

Figure 1: Listener UDP Proxying HTTP Datagram Format
IP Version: The IP Version of the following IP Address field. MUST be 4 or 6.

IP Address: The IP Address of this proxied UDP packet. When sent from client to proxy, this is target host that the proxy will send this UDP payload to. When sent from proxy to client, this represents the source IP address of the UDP packet received by the proxy. This field has length 32 bits when the previous IP Version field value is 4, and 128 when the IP Version is 6.

UDP Port: The UDP Port of this proxied UDP packet. When sent from client to proxy, this is target port that the proxy will send this UDP payload to. When sent from proxy to client, this represents the source UDP port of the UDP packet received by the proxy.

UDP Payload: The unmodified UDP Payload of this proxied UDP packet (referred to as "data octets" in [UDP]).

2.2. The connect-udp-listen Header Field

The "connect-udp-listen" header field is an Item Structured Field, see Section 3.3 of [STRUCT-FIELD]; its value MUST be an Integer; any other value type MUST be handled as if the field were not present by recipients (for example, if this field is included multiple times, its type will become a List and the field will therefore be ignored). This document does not define any parameters for the connect-udp-listen header field value, but future documents might define parameters. Receivers MUST ignore unknown parameters.

3. Security Considerations

The security considerations described in Section 7 of [CONNECT-UDP] also apply here.

4. IANA Considerations

This document will request IANA to register the following entry in the "HTTP Field Name" registry maintained at <https://www.iana.org/assignments/http-fields>:

Field Name: connect-udp-listen
Template: None
Status: provisional (permanent if this document is approved)
Reference: This document
Comments: None

5. References
5.1. Normative References

[CONNECT-UDP]

[HTTP]

[HTTP-DGRAM]

[QUIC]


5.2. Informative References

[CONNECT-IP]
Appendix A.  Example

In the example below, the client is configured with URI Template "https://example.org/.well-known/masque/udp/{target_host}/{target_port}/" and wishes to use WebRTC to another browser over a listener UDP proxying tunnel. It then contacts a STUN server at 192.0.2.42. The STUN server then sends the proxy’s IP address to the other browser at 203.0.113.33 leading that other browser to send a UDP packet to the proxy, and that packets gets proxied over HTTP back to the client.
Client                                             Server

STREAM(44): HEADERS                        -------->
   :method = CONNECT
   :protocol = connect-udp
   :scheme = https
   :path = /.well-known/masque/udp/*/*/
   :authority = proxy.example.org
   connect-udp-listen = 2
   capsule-protocol = ?1

DATAGRAM                        -------->
   Quarter Stream ID = 11
   Context ID = 2
   IP Version = 4
   IP Address = 192.0.2.42
   UDP Port = 1234
   UDP Payload = Encapsulated UDP Payload

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

   /* Wait for STUN server to respond to UDP packet. */

<-------- DATAGRAM
   Quarter Stream ID = 11
   Context ID = 2
   IP Version = 4
   IP Address = 192.0.2.42
   UDP Port = 1234
   UDP Payload = Encapsulated UDP Payload

   /* Wait for the STUN server to send the proxy’s IP and */
   /* port to the other browser and for the other browser */
   /* to send a UDP packet to the proxy. */

<-------- DATAGRAM
   Quarter Stream ID = 11
   Context ID = 2
   IP Version = 4
   IP Address = 203.0.113.33
   UDP Port = 4321
   UDP Payload = Encapsulated UDP Payload
Appendix B. Comparison with CONNECT-IP

While the use-cases described in Section 1 could be solved using IP Proxying in HTTP [CONNECT-IP], that would require that every HTTP Datagram carry a complete IP header. This would not only cause inefficiencies in the wire encoding, it would additionally reduce the available Maximum Transmission Unit (MTU). Furthermore, it would require that Web browsers implement IPv4 and IPv6 header generation and parsing, alongside with validation and error handling.

Acknowledgments

This proposal is the result of many conversations with MASQUE working group participants.

Author’s Address

David Schinazi
Google LLC
1600 Amphitheatre Parkway
Mountain View, CA 94043
United States of America
Email: dschinazi.ietf@gmail.com
HTTP Access Service Description Objects
draft-schwartz-masque-access-descriptions-02

Abstract

HTTP proxies can operate several different kinds of access services. This specification provides a format for identifying a collection of such services.

About This Document

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

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-schwartz-masque-access-descriptions/.

Source for this draft and an issue tracker can be found at https://github.com/bemasc/access-services.

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 2 January 2023.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.
1. Introduction

In HTTP/1.1, forward proxy service was originally defined in two ways: absolute-uri request form (encrypted at most hop-by-hop), and HTTP CONNECT (potentially encrypted end-to-end). Both of these services were effectively origin-scoped: the access service was a property of the origin, not associated with any particular path.

Recently, a variety of new standardized proxy-like services have emerged for HTTP. These new services are defined by a URI template or path, allowing distinct instances of the same service type to be served by a single origin. These services include:

* DNS over HTTPS [RFC8484]
* CONNECT-UDP [I-D.draft-ietf-masque-connect-udp]
* CONNECT-IP [I-D.draft-ietf-masque-connect-ip]
* Oblivious HTTP [I-D.draft-ietf-ohai-ohttp]

This specification provides a unified format for describing a collection of such access services, and a mechanism for reaching such services when the initial information contains only an HTTP origin.
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.

3. Format

An access service collection is defined by a JSON dictionary containing keys specified in the corresponding registry (Section 6). Inclusion of each key is OPTIONAL. The corresponding media type is application/access-services+json.

The "dns", "udp", and "ip" keys are each defined to hold a JSON dictionary containing the key "template" with a value that is a URI template suitable for configuring DNS over HTTPS, CONNECT-UDP, or CONNECT-IP, respectively.

The "ohttp" key contains a dictionary with either or both of these keys:

* "relay", containing a dictionary with a "template" key indicating the Oblivious Relay’s resource mapping. The template MUST contain a "gateway_uri" variable indicating the Oblivious Gateway Resource.

* "gateway", containing a dictionary with a "uri" key indicating the Oblivious Gateway Resource and a "key" key conveying its KeyConfig in base64.

If the Access Description is for a general-purpose proxy, all Oblivious Gateways and proxy targets (respectively) are presumed to be supported; otherwise the supported Gateways and targets must be understood from context (but see Section 4).

3.1. Examples
In cases where the HTTP access service is identified only by an origin (e.g. when configured as a Secure Web Proxy), operators can publish an associated access service collection at the path "/.well-known/access-services", with the Content-Type "application/access-services+json".

When the "ohttp.gateway" URI appears in an Access Description at this location, all URIs on this origin (except the Oblivious Gateway URI) are presumed to be reachable as Oblivious Targets.
Clients MAY fetch this Access Description and use the indicated services (in addition to any origin-scoped services) automatically. Clients SHOULD use the description only while it is fresh according to its HTTP cache lifetime, refreshing it as needed.

5. Security Considerations

TODO Security

6. IANA Considerations

IANA is requested to open a Specification Required registry entitled "HTTP Access Service Descriptors", with the following initial contents:

<table>
<thead>
<tr>
<th>Key</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>dns</td>
<td>(This document)</td>
</tr>
<tr>
<td>udp</td>
<td>(This document)</td>
</tr>
<tr>
<td>ip</td>
<td>(This document)</td>
</tr>
<tr>
<td>ohttp</td>
<td>(This document)</td>
</tr>
</tbody>
</table>

Table 1

IANA is requested to add the following entry to the "Well-Known URIs" registry:

<table>
<thead>
<tr>
<th>URI Suffix</th>
<th>Change Controller</th>
<th>Reference</th>
<th>Status</th>
<th>Related Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>access-services</td>
<td>IETF</td>
<td>(This document)</td>
<td>provisional</td>
<td>Sub-registry at (link)</td>
</tr>
</tbody>
</table>

Table 2

IANA is requested to add the following entry to the "application" sub-registry of the "Media Types" registry:
<table>
<thead>
<tr>
<th>Name</th>
<th>Template</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>access-services+json</td>
<td>application/access-services+json</td>
<td>(This document)</td>
</tr>
</tbody>
</table>

Table 3

TODO: Full registration template for this Media Type.

7. Normative References

[I-D.draft-ietf-masque-connect-ip]

[I-D.draft-ietf-masque-connect-udp]

[I-D.draft-ietf-ohai-ohttp]


Acknowledgments

TODO acknowledge.

Author’s Address

Benjamin M. Schwartz
Google LLC
Email: bemasc@google.com
HTTP Datagram PING and TIMESTAMP
draft-schwartz-masque-h3-datagram-ping-02

Abstract

This draft defines new mechanisms for measuring the functionality and performance of an HTTP Datagram path. These mechanisms can be used with CONNECT-UDP, CONNECT-IP, or any other instantiation of the Capsule Protocol.

Discussion Venues

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

Discussion of this document takes place on the 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/bemasc/h3-datagram-ping.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 27 November 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.
1. Conventions and Definitions

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

2. PING

PING Datagrams can be used to characterize and monitor the end-to-end HTTP Datagram path associated with an HTTP request. For example, HTTP endpoints can easily use PING Datagrams to estimate the round-trip time and loss rate of the HTTP Datagram path.

PING Datagrams are also suitable for use as DPLPMTUD Probe Packets [RFC8899]. This enables endpoints to estimate the HTTP Datagram MTU of each request-response pair, in order to avoid sending HTTP Datagram packets that will be dropped.
Note that these path characteristics can differ from those inferred from the underlying transport (e.g. QUIC), if the HTTP request traverses one or more HTTP intermediaries (see Section 3.7 of [I-D.draft-ietf-httpbis-semantics]).

2.1. Registration

Endpoints indicate support for the PING Datagram type using the Item Structured Field "DG-Ping" in the HTTP Request and Response headers. Its value MUST be an integer indicating the Context ID allocated for PING datagrams. (See Section 3.3.1 of [RFC8941] for information about the integer format.)

Endpoints MUST NOT allocate more than one Context ID for PING Datagrams. As a side effect, this means that only the HTTP client can choose the Context ID used for PING Datagrams.

2.2. Format

PING Datagrams have the following format:

PING Datagram {
  Context ID (i),
  Sequence Number (i),
  Opaque Data (..),
}

All Sequence Number and Opaque Data values are potentially valid.

2.3. Use

The sender emits a PING Datagram with any even Sequence Number and any Opaque Data. Upon receiving a PING Datagram with an even Sequence Number, the recipient MUST reply with a PING Datagram whose Sequence Number is one larger, with empty Opaque Data.

Intermediaries MUST forward PING Datagrams without modification, just like any other HTTP Datagram.

3. TIMESTAMP

The TIMESTAMP Datagram extension allows marking any datagram with a timestamp indicating the time that it was sent. Where PING allows measurement of the round-trip time between peers, TIMESTAMP allows peers to observe changes in the one-way latency. Increasing one-way latency can indicate congestion on that path, informing peers’ congestion control decisions.
3.1. Registration

Endpoints indicate support for TIMESTAMP Datagram type by including the boolean-valued Item Structured Field "DG-Timestamp: ?1" in the HTTP Request and Response headers. (See Section 3.3.6 of [RFC8941] for information about the boolean format.)

A TIMESTAMP Datagram context is opened by a REGISTER_TIMESTAMP_CONTEXT Capsule with the following structure:

REGISTER_TIMESTAMP_CONTEXT Capsule {
  Context ID (i)
  Inner Context ID (i)
  Short Format (1)
}

"Inner Context ID" specifies how to interpret the payload after the timestamp. It MUST be smaller than "Context ID", and MUST already be registered (although that registration does not need to have been confirmed yet).

If "Short Format" is 1 (i.e. true), timestamps MUST use the NTP Short Format (Section 6 of [RFC5905]). Otherwise, the full NTP Timestamp Format MUST be used.

Registration is confirmed by an ACK_TIMESTAMP_CONTEXT Capsule:

ACK_TIMESTAMP_CONTEXT Capsule {
  Context ID (i)
  Error Code (i)
}

Error Code 0 means registration succeeded. Error Code 1 means registration failed. All other error code values also mean failure, but they are reserved for future use.

Registrations can be closed by a CLOSE_TIMESTAMP_CONTEXT Capsule:

CLOSE_TIMESTAMP_CONTEXT Capsule {
  Context ID (i)
}

Endpoints SHOULD close any TIMESTAMP context before closing its Inner Context. If the Inner Context is closed first, datagrams subsequently received on the TIMESTAMP context MUST be dropped.
3.2. Format

TIMESTAMP Datagrams have the following format:

```plaintext
TIMESTAMP Datagram {
    Context ID (i),
    Timestamp (32..64),
    Inner Data (..),
}
```

"Timestamp" is an NTP timestamp in the short or full format, as specified at registration. The NTP Short Format occupies 4 bytes and provides a resolution of 15 microseconds; the full NTP Timestamp Format occupies 8 bytes and provides a resolution of 232 picoseconds.

"Inner Data" is a payload to be interpreted in accordance with this context's "Inner Context ID".

4. Examples

This example shows the PING and TIMESTAMP types used in combination. Note that the client is using a "false start" pattern, creating and using two registrations before either is confirmed.

```
Client                              Origin

# Headers
Capsule-Protocol: ?1
DG-Timestamp: ?1
DG-Ping: 42

# Capsules
REGISTER_TIMESTAMP_CONTEXT(Context ID = 6, Inner ID = 42, Short = 1) =>>

# Datagrams
[Context ID(6) + Timestamp(X) + Sequence Number(0) + Opaque Data] --->

# Headers
Capsule-Protocol: ?1
DG-Timestamp: ?1
DG-Ping: 42

# Capsules
<= ACK_TIMESTAMP_CONTEXT(Context ID = 6, Error Code = 0)

# Datagrams
<--- [Context ID(6) + Timestamp(Y) + Sequence Number(1)]
```
Figure 1: TIMESTAMP and PING example

TIMESTAMP can also be applied to other payload types, such as UDP packets. In CONNECT-UDP, these are pre-allocated with Context ID 0. This example similarly shows a "false start" pattern, sending a datagram before its context registration, or support for this format, is confirmed.

Client                                               CONNECT-UDP Server

# Headers
:method = CONNECT
:protocol = connect-udp
Capsule-Protocol: ?1
DG-Timestamp: ?1

# Capsules
REGISTER_TIMESTAMP_CONTEXT(Context ID = 2, Inner ID = 0, Short = 1) ==> 

# Datagrams
[Context ID(2) + Timestamp(X) + UDP Payload] --->

# Headers
Capsule-Protocol: ?1
DG-Timestamp: ?1

# Capsules
<= ACK_TIMESTAMP_CONTEXT(Context ID = 2, Error Code = 0)

# ... server waits for a UDP response packet.
# Datagrams
<= [Context ID(2) + Timestamp(Y) + UDP Payload]

Figure 2: TIMESTAMP and UDP example

5. IANA considerations

5.1. Capsule types

IANA is directed to add the following entries to the "HTTP Capsule Types" registry:
5.2. HTTP headers

IANA is directed to add the following entries to the "Hypertext Transfer Protocol (HTTP) Field Name Registry":

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Template</th>
<th>Status</th>
<th>Reference</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>DG-Ping</td>
<td></td>
<td>permanent</td>
<td>(This document)</td>
<td></td>
</tr>
<tr>
<td>DG-Timestamp</td>
<td></td>
<td>permanent</td>
<td>(This document)</td>
<td></td>
</tr>
</tbody>
</table>

Table 1

6. References

6.1. Normative References


6.2. Informative References

[I-D.draft-ietf-httpbis-semantics]


Acknowledgments

Thanks to Alex Chernyakhovsky for constructive input.

Author’s Address

Benjamin Schwartz
Google LLC
Email: bemasc@google.com