

MASQUE
Internet-Draft
Intended status: Standards Track
Expires: 28 April 2022

T. Pauly, Ed.
Apple Inc.
D. Schinazi
A. Chernyakhovsky
Google LLC
M. Kuehlewind
M. Westerlund
Ericsson
25 October 2021

IP Proxying Support for HTTP
draft-age-masque-connect-ip-01

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 28 April 2022.

Copyright Notice

Copyright (c) 2021 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 Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified 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	5
4.2. Capsules	5
4.2.1. ADDRESS_ASSIGN Capsule	6
4.2.2. ADDRESS_REQUEST Capsule	7
4.2.3. ROUTE_ADVERTISEMENT Capsule	7
5. Transmitting IP Packets using HTTP Datagrams	9
6. Examples	10
6.1. Remote Access VPN	10
6.2. IP Flow Forwarding	12
6.3. Proxied Connection Racing	15
7. Security Considerations	17
8. IANA Considerations	17
8.1. CONNECT-IP HTTP Upgrade Token	17
8.2. Datagram Format Type	17
8.3. Capsule Type Registrations	17
9. References	18
9.1. Normative References	18
9.2. Informative References	19
Acknowledgments	20
Authors' Addresses	20

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.

Along with a request, the client can send a REGISTER_DATAGRAM_CONTEXT capsule [HTTP-DGRAM] to negotiate support for sending IP packets in HTTP Datagrams (Section 5).

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 8.3 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 8.3 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 8.3 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 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. Transmitting IP Packets using HTTP Datagrams

IP packets are encoded using HTTP Datagrams [HTTP-DGRAM] with the IP_PACKET HTTP Datagram Format Type (see value in Section 8.2). When using the IP_PACKET HTTP Datagram Format Type, full IP packets (from the IP Version field until the last byte of the IP Payload) are sent unmodified in the "HTTP Datagram Payload" field of an HTTP Datagram.

In order to use HTTP Datagrams, the client will first decide whether or not it will attempt to use HTTP Datagram Contexts and then register its context ID (or lack thereof) using the corresponding registration capsule, see [HTTP-DGRAM].

When sending a registration capsule using the "Datagram Format Type" set to IP_PACKET, the "Datagram Format Additional Data" field SHALL be empty. Servers MUST NOT register contexts using the IP_PACKET HTTP Datagram Format Type. Clients MUST NOT register more than one context using the IP_PACKET HTTP Datagram Format Type. Endpoints MUST NOT close contexts using the IP_PACKET HTTP Datagram Format Type. If an endpoint detects a violation of any of these requirements, it MUST abort the stream.

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.

Extensions to this mechanism MAY define new HTTP Datagram Format Types in order to use different semantics or encodings for IP payloads. For example, an extension could define a new HTTP Datagram Format Type which enables compression of IP header fields.

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.

6. 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.

6.1. 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.

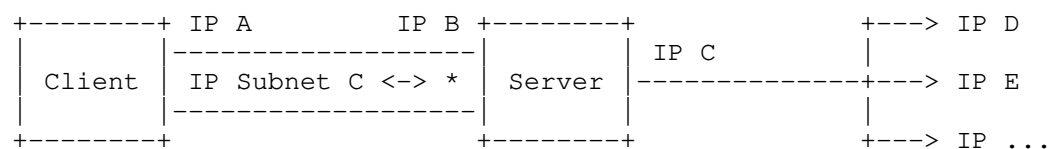


Figure 6: VPN Tunnel Setup

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  
  
STREAM(44): HEADERS  
:method = CONNECT  
:protocol = connect-ip  
:scheme = https  
:path = /vpn  
:authority = server.example.com  
  
STREAM(44): CAPSULE  
Capsule Type = REGISTER_DATAGRAM_CONTEXT  
Context ID = 0  
Context Extension = {}  
  
STREAM(44): HEADERS  
:status = 200  
  
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 7: 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 ]]

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 8: VPN Split-Tunnel Capsule Example

6.2. 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.

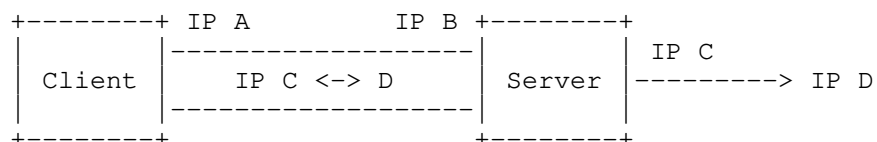


Figure 9: Proxied Flow Setup

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  
  
STREAM(52): HEADERS  
:method = CONNECT  
:protocol = connect-ip  
:scheme = https  
:path = /proxy?target=target.example.com&ipproto=132  
:authority = server.example.com  
  
STREAM(52): CAPSULE  
Capsule Type = REGISTER_DATAGRAM_CONTEXT  
Context ID = 0  
Context Extension = {}  
  
STREAM(52): HEADERS  
:status = 200  
  
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 = 11  
Context ID = 0  
Payload = Encapsulated SCTP/IP Packet  
  
DATAGRAM  
Quarter Stream ID = 11  
Context ID = 0  
Payload = Encapsulated SCTP/IP Packet
```

Figure 10: Proxied SCTP Flow Example

6.3. 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.

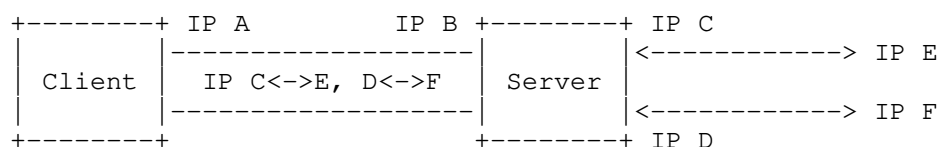


Figure 11: Proxied Connection Racing Setup

As with proxied flows, the client specifies both a target hostname and an IP protocol number in the scope of its request. When the 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 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

STREAM(44): CAPSULE

```

```
Capsule Type = REGISTER_DATAGRAM_CONTEXT
Context ID = 0
Context Extension = {}
```

```
STREAM(44): HEADERS
:status = 200
```

```
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
```

```
STREAM(44): CAPSULE
Capsule Type = ROUTE_ADVERTISEMENT
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 12: Proxied Connection Racing Example

7. 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 sent by their peer, they SHOULD follow the guidance in [BCP38] to help prevent denial of service attacks.

8. IANA Considerations

8.1. 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

8.2. Datagram Format Type

This document will request IANA to register IP_PACKET in the "HTTP Datagram Format Types" registry established by [HTTP-DGRAM].

Type	Value	Specification
IP_PACKET	0xff8b00	This Document

Table 1: Registered Datagram Format Type

8.3. 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
0xffff100	ADDRESS_ASSIGN	Address Assignment	This Document
0xffff101	ADDRESS_REQUEST	Address Request	This Document
0xffff102	ROUTE_ADVERTISEMENT	Route Advertisement	This Document

Table 2: New Capsules

9. References

9.1. 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-00, 9 September 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-h3-websockets-00>>.
- [HTTP-DGRAM] Schinazi, D. and L. Pardue, "Using Datagrams with HTTP", Work in Progress, Internet-Draft, draft-ietf-masque-h3-datagram-04, 6 October 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-masque-h3-datagram-04>>.
- [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>>.

9.2. 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-05, 6 October 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-masque-connect-udp-05>>.

[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

MASQUE
Internet-Draft
Intended status: Standards Track
Expires: 4 November 2022

D. Schinazi
Google LLC
3 May 2022

Proxying UDP in HTTP
draft-ietf-masque-connect-udp-12

Abstract

This document describes how to proxy UDP in HTTP, similar to how the HTTP CONNECT method allows proxying TCP in HTTP. More specifically, this document defines a protocol that allows HTTP clients to create a tunnel for UDP communications through an HTTP server that acts as a proxy.

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

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-udp>.

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 4 November 2022.

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	2
1.1. Conventions and Definitions	3
2. Client Configuration	3
3. Tunnelling UDP over HTTP	4
3.1. UDP Proxy Handling	5
3.2. HTTP/1.1 Request	6
3.3. HTTP/1.1 Response	7
3.4. HTTP/2 and HTTP/3 Requests	8
3.5. HTTP/2 and HTTP/3 Responses	8
3.6. Note About Draft Versions	9
4. Context Identifiers	9
5. HTTP Datagram Payload Format	10
6. Performance Considerations	11
6.1. MTU Considerations	11
6.2. Tunneling of ECN Marks	12
7. Security Considerations	12
8. IANA Considerations	13
8.1. HTTP Upgrade Token	13
8.2. Well-Known URI	13
9. References	13
9.1. Normative References	13
9.2. Informative References	15
Acknowledgments	16
Author's Address	16

1. Introduction

While HTTP provides the CONNECT method (see Section 9.3.6 of [HTTP]) for creating a TCP [TCP] tunnel to a proxy, it lacks a method for doing so for UDP [UDP] traffic.

This document describes a protocol for tunnelling UDP to a server acting as a UDP-specific proxy over HTTP. UDP tunnels are commonly used to create an end-to-end virtual connection, which can then be secured using QUIC [QUIC] or another protocol running over UDP. Unlike CONNECT, the UDP proxy itself is identified with an absolute URL containing the traffic's destination. Clients generate those URLs using a URI Template [TEMPLATE], as described in Section 2.

This protocol supports all versions of HTTP by using HTTP Datagrams [HTTP-DGRAM]. When using HTTP/2 [HTTP/2] or HTTP/3 [HTTP/3], it uses HTTP Extended CONNECT as described in [EXT-CONNECT2] and [EXT-CONNECT3]. When using HTTP/1.x [HTTP/1.1], it uses HTTP Upgrade as defined in Section 7.8 of [HTTP].

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.

In this document, we use the term "UDP proxy" to refer to the HTTP server that acts upon the client's UDP tunnelling request to open a UDP socket to a target server, and generates the response to this request. If there are HTTP intermediaries (as defined in Section 3.7 of [HTTP]) between the client and the UDP proxy, those are referred to as "intermediaries" in this document.

Note that, when the HTTP version in use does not support multiplexing streams (such as HTTP/1.1), any reference to "stream" in this document represents the entire connection.

2. Client Configuration

HTTP clients are configured to use a UDP proxy with a URI Template [TEMPLATE] that has the variables "target_host" and "target_port". Examples are shown below:

```
https://masque.example.org/.well-known/masque/udp/{target_host}/{target_port}/
https://proxy.example.org:4443/masque?h={target_host}&p={target_port}
https://proxy.example.org:4443/masque{?target_host,target_port}
```

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 MUST contain the two variables `"target_host"` and `"target_port"` 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).
- * 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.

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

Since the original HTTP CONNECT method allowed conveying the target host and port but not the scheme, proxy authority, path, nor query, there exist proxy configuration interfaces that only allow the user to configure the proxy host and the proxy port. Client implementations of this specification that are constrained by such limitations MAY attempt to access UDP proxying capabilities using the default template, which is defined as:

`"https://{PROXY_HOST}:{PROXY_PORT}/.well-known/masque/udp/{target_host}/{target_port}"` where `{PROXY_HOST}` and `{PROXY_PORT}` are the configured host and port of the UDP proxy respectively. UDP proxy deployments SHOULD offer service at this location if they need to interoperate with such clients.

3. Tunneling UDP over HTTP

To allow negotiation of a tunnel for UDP over HTTP, this document defines the `"connect-udp"` HTTP Upgrade Token. The resulting UDP tunnels use the Capsule Protocol (see Section 3.2 of [HTTP-DGRAM]) with HTTP Datagram in the format defined in Section 5.

To initiate a UDP tunnel associated with a single HTTP stream, clients issue a request containing the "connect-udp" upgrade token. The target of the tunnel is indicated by the client to the UDP proxy via the "target_host" and "target_port" variables of the URI Template, see Section 2. If the request is successful, the UDP proxy commits to converting received HTTP Datagrams into UDP packets and vice versa until the tunnel is closed.

When sending its UDP proxying request, the client SHALL perform URI Template expansion to determine the path and query of its request. target_host supports using DNS names, IPv6 literals and IPv4 literals. Note that this URI Template expansion requires using pct-encoding, so for example if the target_host is "2001:db8::42", it will be encoded in the URI as "2001%3Adb8%3A%3A42".

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

3.1. UDP Proxy Handling

Upon receiving a UDP proxying request:

- * if the recipient is configured to use another HTTP proxy, it will act as an intermediary: it forwards the request to another HTTP server. Note that such intermediaries may need to reencode the request if they forward it using a version of HTTP that is different from the one used to receive it, as the request encoding differs by version (see below).
- * otherwise, the recipient will act as a UDP proxy: it extracts the "target_host" and "target_port" variables from the URI it has reconstructed from the request headers, and establishes a tunnel by directly opening a UDP socket to the requested target.

Unlike TCP, UDP is connection-less. The UDP proxy that opens the UDP socket has no way of knowing whether the destination is reachable. Therefore it needs to respond to the request without waiting for a packet from the target. However, if the target_host is a DNS name, the UDP proxy MUST perform DNS resolution before replying to the HTTP request. If errors occur during this process, the UDP proxy MUST fail the request and SHOULD send details using an appropriate "Proxy-Status" header field [PROXY-STATUS] (for example, if DNS resolution returns an error, the proxy can use the dns_error Proxy Error Type from Section 2.3.2 of [PROXY-STATUS]).

UDP proxies can use connected UDP sockets if their operating system supports them, as that allows the UDP proxy to rely on the kernel to only send it UDP packets that match the correct 5-tuple. If the UDP proxy uses a non-connected socket, it MUST validate the IP source address and UDP source port on received packets to ensure they match the client's request. Packets that do not match MUST be discarded by the UDP proxy.

The lifetime of the socket is tied to the request stream. The UDP proxy MUST keep the socket open while the request stream is open. If a UDP proxy is notified by its operating system that its socket is no longer usable (for example, this can happen when an ICMP "Destination Unreachable" message is received, see Section 3.1 of [ICMP6]), it MUST close the request stream. UDP proxies MAY choose to close sockets due to a period of inactivity, but they MUST close the request stream when closing the socket. UDP proxies that close sockets after a period of inactivity SHOULD NOT use a period lower than two minutes, see Section 4.3 of [BEHAVE].

A successful response (as defined in Section 3.3 and Section 3.5) indicates that the UDP proxy has opened a socket to the requested target and is willing to proxy UDP payloads. Any response other than a successful response indicates that the request has failed, and the client MUST therefore abort the request.

UDP proxies MUST NOT introduce fragmentation at the IP layer when forwarding HTTP Datagrams onto a UDP socket. In IPv4, the Don't Fragment (DF) bit MUST be set if possible, to prevent fragmentation on the path. Future extensions MAY remove these requirements.

3.2. HTTP/1.1 Request

When using HTTP/1.1 [HTTP/1.1], a UDP proxying request will meet the following requirements:

- * the method SHALL be "GET".
- * the request SHALL include a single "Host" header field containing the origin of the UDP proxy.
- * the request SHALL include a "Connection" header field with value "Upgrade" (note that this requirement is case-insensitive as per Section 7.6.1 of [HTTP]).
- * the request SHALL include an "Upgrade" header field with value "connect-udp".

For example, if the client is configured with URI Template "https://proxy.example.org/.well-known/masque/udp/{target_host}/{target_port}/" and wishes to open a UDP proxying tunnel to target 192.0.2.42:443, it could send the following request:

```
GET https://proxy.example.org/.well-known/masque/udp/192.0.2.42/443/ HTTP/1.1
Host: proxy.example.org
Connection: Upgrade
Upgrade: connect-udp
```

Figure 2: Example HTTP/1.1 Request

In HTTP/1.1, this protocol uses the GET method to mimic the design of the WebSocket Protocol [WEBSOCKET].

3.3. HTTP/1.1 Response

The UDP proxy SHALL indicate a successful response by replying with the following requirements:

- * the HTTP status code on the response SHALL be 101 (Switching Protocols).
- * the response SHALL include a single "Connection" header field with value "Upgrade" (note that this requirement is case-insensitive as per Section 7.6.1 of [HTTP]).
- * the response SHALL include a single "Upgrade" header field with value "connect-udp".
- * the response SHALL NOT include any "Transfer-Encoding" or "Content-Length" header fields.

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

For example, the UDP proxy could respond with:

```
HTTP/1.1 101 Switching Protocols
Connection: Upgrade
Upgrade: connect-udp
```

Figure 3: Example HTTP/1.1 Response

3.4. HTTP/2 and HTTP/3 Requests

When using HTTP/2 [HTTP/2] or HTTP/3 [HTTP/3], UDP proxying requests use Extended CONNECT. This requires that servers send an HTTP Setting as specified in [EXT-CONNECT2] and [EXT-CONNECT3], and that requests use HTTP pseudo-header fields with the following requirements:

- * The ":method" pseudo-header field SHALL be "CONNECT".
- * The ":protocol" pseudo-header field SHALL be "connect-udp".
- * The ":authority" pseudo-header field SHALL contain the authority of the UDP proxy.
- * The ":path" and ":scheme" pseudo-header fields SHALL NOT be empty. Their values SHALL contain the scheme and path from the URI Template after the URI template expansion process has been completed.

A UDP proxying request that does not conform to these restrictions is malformed (see Section 8.1.1 of [HTTP/2]).

For example, if the client is configured with URI Template "https://proxy.example.org/{target_host}/{target_port}/" and wishes to open a UDP proxying tunnel to target 192.0.2.42:443, it could send the following request:

```
HEADERS
:method = CONNECT
:protocol = connect-udp
:scheme = https
:path = /.well-known/masque/udp/192.0.2.42/443/
:authority = proxy.example.org
```

Figure 4: Example HTTP/2 Request

3.5. HTTP/2 and HTTP/3 Responses

The UDP proxy SHALL indicate a successful response by replying with any 2xx (Successful) HTTP status code, without any "Transfer-Encoding" or "Content-Length" header fields.

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

For example, the UDP proxy could respond with:

```
HEADERS
:status = 200
```

Figure 5: Example HTTP/2 Response

3.6. Note About Draft Versions

[[RFC editor: please remove this section before publication.]]

In order to allow implementations to support multiple draft versions of this specification during its development, we introduce the "connect-udp-version" header field. When sent by the client, it contains a list of draft numbers supported by the client (e.g., "connect-udp-version: 0, 2"). When sent by the UDP proxy, it contains a single draft number selected by the UDP proxy from the list provided by the client (e.g., "connect-udp-version: 2"). Sending this header field is RECOMMENDED but not required. The "connect-udp-version" header field is a List Structured Field, see Section 3.1 of [STRUCT-FIELD]. Each list member MUST be an Integer.

4. Context Identifiers

This protocol allows future extensions to exchange HTTP Datagrams which carry different semantics from UDP payloads. Some of these extensions can augment UDP payloads with additional data, while others can exchange data that is completely separate from UDP payloads. In order to accomplish this, all HTTP Datagrams associated with UDP Proxying request streams start with a context ID, see Section 5.

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 UDP payloads, while non-zero values are dynamically allocated: non-zero even-numbered context IDs are client-allocated, and odd-numbered context IDs are proxy-allocated. The context ID namespace is tied to a given HTTP request: it is possible for a context ID with the same numeric value to be simultaneously allocated in distinct requests, potentially with different semantics. Context IDs MUST NOT be re-allocated within a given HTTP namespace but MAY be allocated in any order. The context ID allocation restrictions to the use of even-numbered and odd-numbered context IDs exist in order to avoid the need for synchronisation between endpoints. However, once a context ID has been allocated, those restrictions do not apply to the use of the context ID: it can be used by any client or UDP proxy, independent of which endpoint initially allocated it.

Registration is the action by which an endpoint informs its peer of the semantics and format of a given context ID. This document does not define how registration occurs. Future extensions MAY use HTTP header fields or capsules to register contexts. 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 packet containing the datagram and the packet containing the registration message during transmission.

5. HTTP Datagram Payload Format

When HTTP Datagrams (see [HTTP-DGRAM]) are associated with UDP proxying request streams, the HTTP Datagram Payload field 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:

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

Figure 6: UDP Proxying HTTP Datagram Format

Context ID: A variable-length integer (see Section 16 of [QUIC]) 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.

UDP 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 the unmodified payload of a UDP packet (referred to as "data octets" in [UDP]).

Clients MAY optimistically start sending UDP packets in HTTP Datagrams before receiving the response to its UDP proxying request. However, implementors should note that such proxied packets may not be processed by the UDP proxy if it responds to the request with a failure, or if the proxied packets are received by the UDP proxy before the request.

By virtue of the definition of the UDP header [UDP], it is not possible to encode UDP payloads longer than 65527 bytes. Therefore, endpoints MUST NOT send HTTP Datagrams with a Payload field longer than 65527 using Context ID zero. An endpoint that receives a DATAGRAM capsule using Context ID zero whose Payload field is longer than 65527 MUST abort the stream. If a UDP proxy knows it can only send out UDP packets of a certain length due to its underlying link MTU, it SHOULD discard incoming DATAGRAM capsules using Context ID zero whose Payload field is longer than that limit without buffering the capsule contents.

6. Performance Considerations

UDP proxies SHOULD strive to avoid increasing burstiness of UDP traffic: they SHOULD NOT queue packets in order to increase batching.

When the protocol running over UDP that is being proxied uses congestion control (e.g., [QUIC]), the proxied traffic will incur at least two nested congestion controllers. This can reduce performance but the underlying HTTP connection MUST NOT disable congestion control unless it has an out-of-band way of knowing with absolute certainty that the inner traffic is congestion-controlled.

If a client or UDP proxy with a connection containing a UDP proxying request stream disables congestion control, it MUST NOT signal ECN support on that connection. That is, it MUST mark all IP headers with the Not-ECT codepoint. It MAY continue to report ECN feedback via ACK_ECN frames, as the peer may not have disabled congestion control.

When the protocol running over UDP that is being proxied uses loss recovery (e.g., [QUIC]), and the underlying HTTP connection runs over TCP, the proxied traffic will incur at least two nested loss recovery mechanisms. This can reduce performance as both can sometimes independently retransmit the same data. To avoid this, UDP proxying SHOULD be performed over HTTP/3 to allow leveraging the QUIC DATAGRAM frame.

6.1. MTU Considerations

When using HTTP/3 with the QUIC Datagram extension [DGRAM], UDP payloads are transmitted in QUIC DATAGRAM frames. Since those cannot be fragmented, they can only carry payloads up to a given length determined by the QUIC connection configuration and the path MTU. If a UDP proxy is using QUIC DATAGRAM frames and it receives a UDP payload from the target that will not fit inside a QUIC DATAGRAM frame, the UDP proxy SHOULD NOT send the UDP payload in a DATAGRAM capsule, as that defeats the end-to-end unreliability characteristic

that methods such as Datagram Packetization Layer Path MTU Discovery (DPLPMTUD) depend on [DPLPMTUD]. In this scenario, the UDP proxy SHOULD drop the UDP payload and send an ICMP "Packet Too Big" message to the target, see Section 3.2 of [ICMP6].

6.2. Tunneling of ECN Marks

UDP proxying does not create an IP-in-IP tunnel, so the guidance in [ECN-TUNNEL] about transferring ECN marks between inner and outer IP headers does not apply. There is no inner IP header in UDP proxying tunnels.

Note that UDP proxying clients do not have the ability in this specification to control the ECN codepoints on UDP packets the UDP proxy sends to the target, nor can UDP proxies communicate the markings of each UDP packet from target to UDP proxy.

A UDP proxy MUST ignore ECN bits in the IP header of UDP packets received from the target, and MUST set the ECN bits to Not-ECT on UDP packets it sends to the target. These do not relate to the ECN markings of packets sent between client and UDP proxy in any way.

7. Security Considerations

There are significant risks in allowing arbitrary clients to establish a tunnel to arbitrary targets, as that could allow bad actors to send traffic and have it attributed to the UDP proxy. HTTP servers that support UDP proxying ought to restrict its use to authenticated users.

Because the CONNECT method creates a TCP connection to the target, the target has to indicate its willingness to accept TCP connections by responding with a TCP SYN-ACK before the CONNECT proxy can send it application data. UDP doesn't have this property, so a UDP proxy could send more data to an unwilling target than a CONNECT proxy. However, in practice denial of service attacks target open TCP ports so the TCP SYN-ACK does not offer much protection in real scenarios. While a UDP proxy could potentially limit the number of UDP packets it is willing to forward until it has observed a response from the target, that is unlikely to provide any protection against denial of service attacks because such attacks target open UDP ports where the protocol running over UDP would respond, and that would be interpreted as willingness to accept UDP by the UDP proxy.

UDP sockets for UDP proxying have a different lifetime than TCP sockets for CONNECT, therefore implementors would be well served to follow the advice in Section 3.1 if they base their UDP proxying implementation on a preexisting implementation of CONNECT.

The security considerations described in [HTTP-DGRAM] also apply here.

8. IANA Considerations

8.1. HTTP Upgrade Token

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

Value: connect-udp
Description: Proxying of UDP Payloads
Expected Version Tokens: None
Reference: This document

8.2. Well-Known URI

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

URI Suffix: masque/udp
Change Controller: IETF
Reference: This document
Status: permanent (if this document is approved)
Related Information: Includes all resources identified with the path prefix `"/.well-known/masque/udp/"`

9. References

9.1. Normative References

[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", 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]

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>>.

[HTTP-DGRAM]

Schinazi, D. and L. Pardue, "HTTP Datagrams and the Capsule Protocol", Work in Progress, Internet-Draft, draft-ietf-masque-h3-datagram-09, 11 April 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-masque-h3-datagram-09>>.

[HTTP/1.1]

Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP/1.1", Work in Progress, Internet-Draft, draft-ietf-httpbis-messaging-19, 12 September 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-messaging-19>>.

[HTTP/2]

Thomson, M. and C. Benfield, "HTTP/2", Work in Progress, Internet-Draft, draft-ietf-httpbis-http2bis-07, 24 January 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-http2bis-07>>.

[HTTP/3]

Bishop, M., "Hypertext Transfer Protocol Version 3 (HTTP/3)", Work in Progress, Internet-Draft, draft-ietf-quic-http-34, 2 February 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-34>>.

[PROXY-STATUS]

Nottingham, M. and P. Sikora, "The Proxy-Status HTTP Response Header Field", Work in Progress, Internet-Draft, draft-ietf-httpbis-proxy-status-08, 13 October 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-proxy-status-08>>.

[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>>.
- [STRUCT-FIELD] Nottingham, M. and P-H. Kamp, "Structured Field Values for HTTP", RFC 8941, DOI 10.17487/RFC8941, February 2021, <<https://www.rfc-editor.org/rfc/rfc8941>>.
- [TCP] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<https://www.rfc-editor.org/rfc/rfc793>>.
- [TEMPLATE] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <<https://www.rfc-editor.org/rfc/rfc6570>>.
- [UDP] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <<https://www.rfc-editor.org/rfc/rfc768>>.

9.2. Informative References

- [BEHAVE] Audet, F., Ed. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, DOI 10.17487/RFC4787, January 2007, <<https://www.rfc-editor.org/rfc/rfc4787>>.
- [DPLPMTUD] Fairhurst, G., Jones, T., Tüxen, M., Rüngeler, I., and T. Völker, "Packetization Layer Path MTU Discovery for Datagram Transports", RFC 8899, DOI 10.17487/RFC8899, September 2020, <<https://www.rfc-editor.org/rfc/rfc8899>>.
- [ECN-TUNNEL] Briscoe, B., "Tunnelling of Explicit Congestion Notification", RFC 6040, DOI 10.17487/RFC6040, November 2010, <<https://www.rfc-editor.org/rfc/rfc6040>>.

[ICMP6] 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>>.

[WEBSOCKET] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <<https://www.rfc-editor.org/rfc/rfc6455>>.

Acknowledgments

This document is a product of the MASQUE Working Group, and the author thanks all MASQUE enthusiasts for their contributions. This proposal was inspired directly or indirectly by prior work from many people. In particular, the author would like to thank Eric Rescorla for suggesting to use an HTTP method to proxy UDP. The author is indebted to Mark Nottingham and Lucas Pardue for the many improvements they contributed to this document. The extensibility design in this document came out of the HTTP Datagrams Design Team, whose members were Alan Frindell, Alex Chernyakhovsky, Ben Schwartz, Eric Rescorla, Lucas Pardue, Marcus Ihlar, Martin Thomson, Mike Bishop, Tommy Pauly, Victor Vasiliev, and the author of this document.

Author's Address

David Schinazi
Google LLC
1600 Amphitheatre Parkway
Mountain View, CA 94043
United States of America
Email: dschinazi.ietf@gmail.com

MASQUE
Internet-Draft
Intended status: Standards Track
Expires: 13 October 2022

D. Schinazi
Google LLC
L. Pardue
Cloudflare
11 April 2022

HTTP Datagrams and the Capsule Protocol
draft-ietf-masque-h3-datagram-09

Abstract

This document describes HTTP Datagrams, a convention for conveying multiplexed, potentially unreliable datagrams inside an HTTP connection.

In HTTP/3, HTTP Datagrams can be conveyed natively using the QUIC DATAGRAM extension. When the QUIC DATAGRAM frame is unavailable or undesirable, they can be sent using the Capsule Protocol, a more general convention for conveying data in HTTP connections.

HTTP Datagrams and the Capsule Protocol are intended for use by HTTP extensions, not applications.

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-h3-datagram/draft-ietf-masque-h3-datagram.html>. Status information for this document may be found at <https://datatracker.ietf.org/doc/draft-ietf-masque-h3-datagram/>.

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-h3-datagram>.

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 13 October 2022.

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
1.1. Conventions and Definitions	3
2. HTTP Datagrams	3
2.1. HTTP/3 Datagrams	4
2.1.1. The SETTINGS_H3_DATAGRAM HTTP/3 Setting	5
2.2. HTTP Datagrams using Capsules	6
3. Capsules	7
3.1. HTTP Data Streams	7
3.2. The Capsule Protocol	8
3.3. Error Handling	9
3.4. The Capsule-Protocol Header Field	9
3.5. The DATAGRAM Capsule	10
4. Security Considerations	12
5. IANA Considerations	12
5.1. HTTP/3 Setting	12
5.2. HTTP/3 Error Code	12
5.3. HTTP Header Field Name	12
5.4. Capsule Types	13
6. References	13
6.1. Normative References	13

6.2. Informative References	15
Acknowledgments	15
Authors' Addresses	15

1. Introduction

HTTP extensions sometimes need to access underlying transport protocol features such as unreliable delivery (as offered by [DGRAM]) to enable desirable features. For example, this could allow introducing an unreliable version of the CONNECT method, or adding unreliable delivery to WebSockets [RFC6455].

In Section 2, this document describes HTTP Datagrams, a convention that supports the bidirectional and optionally multiplexed exchange of data inside an HTTP connection. While HTTP datagrams are associated with HTTP requests, they are not part of message content; instead, they are intended for use by HTTP extensions (such as the CONNECT method), and are compatible with all versions of HTTP.

When HTTP is running over a transport protocol that supports unreliable delivery (such as when the QUIC DATAGRAM extension is available to HTTP/3), HTTP Datagrams can use that capability.

This document also describes the HTTP Capsule Protocol in Section 3, to allow conveyance of HTTP Datagrams using reliable delivery. This addresses HTTP/3 cases where use of the QUIC DATAGRAM frame is unavailable or undesirable, or where the transport protocol only provides reliable delivery, such as with HTTP/1 or HTTP/2 over TCP.

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.

2. HTTP Datagrams

HTTP Datagrams are a convention for conveying bidirectional and potentially unreliable datagrams inside an HTTP connection, with multiplexing when possible. All HTTP Datagrams are associated with an HTTP request.

When HTTP Datagrams are conveyed on an HTTP/3 connection, the QUIC DATAGRAM frame can be used to achieve these goals, including unreliable delivery; see Section 2.1. Negotiating the use of QUIC DATAGRAM frames for HTTP Datagrams is achieved via the exchange of HTTP/3 settings; see Section 2.1.1.

When running over HTTP/2, demultiplexing is provided by the HTTP/2 framing layer, but unreliable delivery is unavailable. HTTP Datagrams are negotiated and conveyed using the Capsule Protocol; see Section 3.5.

When running over HTTP/1, requests are strictly serialized in the connection, and therefore demultiplexing is not available. Unreliable delivery is likewise not available. HTTP Datagrams are negotiated and conveyed using the Capsule Protocol; see Section 3.5.

HTTP Datagrams MUST only be sent with an association to an HTTP request that explicitly supports them. For example, existing HTTP methods GET and POST do not define semantics for associated HTTP Datagrams; therefore, HTTP Datagrams cannot be sent associated with GET or POST request streams.

If an HTTP Datagram is received and it is associated with a request that has no known semantics for HTTP Datagrams, the receiver MUST terminate the request; if HTTP/3 is in use, the request stream MUST be aborted with H3_DATAGRAM_ERROR (0x33). HTTP extensions can override these requirements by defining a negotiation mechanism and semantics for HTTP Datagrams.

2.1. HTTP/3 Datagrams

When used with HTTP/3, the Datagram Data field of QUIC DATAGRAM frames uses the following format (using the notation from the "Notational Conventions" section of [QUIC]):

```
HTTP/3 Datagram {  
    Quarter Stream ID (i),  
    HTTP Datagram Payload (...),  
}
```

Figure 1: HTTP/3 Datagram Format

Quarter Stream ID: A variable-length integer that contains the value of the client-initiated bidirectional stream that this datagram is associated with, divided by four (the division by four stems from the fact that HTTP requests are sent on client-initiated bidirectional streams, and those have stream IDs that are divisible by four). The largest legal QUIC stream ID value is

$2^{62}-1$, so the largest legal value of Quarter Stream ID is $2^{60}-1$. Receipt of an HTTP/3 Datagram that includes a larger value MUST be treated as an HTTP/3 connection error of type `H3_DATAGRAM_ERROR` (0x33).

HTTP Datagram Payload: The payload of the datagram, whose semantics are defined by the extension that is using HTTP Datagrams. Note that this field can be empty.

Receipt of a QUIC DATAGRAM frame whose payload is too short to allow parsing the Quarter Stream ID field MUST be treated as an HTTP/3 connection error of type `H3_DATAGRAM_ERROR` (0x33).

HTTP/3 Datagrams MUST NOT be sent unless the corresponding stream's send side is open. If a datagram is received after the corresponding stream's receive side is closed, the received datagrams MUST be silently dropped.

If an HTTP/3 datagram is received and its Quarter Stream ID maps to a stream that has not yet been created, the receiver SHALL either drop that datagram silently or buffer it temporarily (on the order of a round trip) while awaiting the creation of the corresponding stream.

If an HTTP/3 datagram is received and its Quarter Stream ID maps to a stream that cannot be created due to client-initiated bidirectional stream limits, it SHOULD be treated as an HTTP/3 connection error of type `H3_ID_ERROR`. Generating an error is not mandatory in this case because HTTP/3 implementations might have practical barriers to determining the active stream concurrency limit that is applied by the QUIC layer.

Prioritization of HTTP/3 datagrams is not defined in this document. Future extensions MAY define how to prioritize datagrams, and MAY define signaling to allow communicating prioritization preferences.

2.1.1. The `SETTINGS_H3_DATAGRAM` HTTP/3 Setting

Endpoints can indicate to their peer that they are willing to receive HTTP/3 Datagrams by sending the `SETTINGS_H3_DATAGRAM` (0x33) setting with a value of 1.

The value of the `SETTINGS_H3_DATAGRAM` setting MUST be either 0 or 1. A value of 0 indicates that the implementation is not willing to receive HTTP Datagrams. If the `SETTINGS_H3_DATAGRAM` setting is received with a value that is neither 0 or 1, the receiver MUST terminate the connection with error `H3_SETTINGS_ERROR`.

QUIC DATAGRAM frames MUST NOT be sent until the SETTINGS_H3_DATAGRAM setting has been both sent and received with a value of 1.

When clients use 0-RTT, they MAY store the value of the server's SETTINGS_H3_DATAGRAM setting. Doing so allows the client to send QUIC DATAGRAM frames in 0-RTT packets. When servers decide to accept 0-RTT data, they MUST send a SETTINGS_H3_DATAGRAM setting greater than or equal to the value they sent to the client in the connection where they sent them the NewSessionTicket message. If a client stores the value of the SETTINGS_H3_DATAGRAM setting with their 0-RTT state, they MUST validate that the new value of the SETTINGS_H3_DATAGRAM setting sent by the server in the handshake is greater than or equal to the stored value; if not, the client MUST terminate the connection with error H3_SETTINGS_ERROR. In all cases, the maximum permitted value of the SETTINGS_H3_DATAGRAM setting parameter is 1.

It is RECOMMENDED that implementations that support receiving HTTP/3 Datagrams always send the SETTINGS_H3_DATAGRAM setting with a value of 1, even if the application does not intend to use HTTP/3 Datagrams. This helps to avoid "sticking out"; see Section 4.

2.1.1.1. Note About Draft Versions

[[RFC editor: please remove this section before publication.]]

Some revisions of this draft specification use a different value (the Identifier field of a Setting in the HTTP/3 SETTINGS frame) for the SETTINGS_H3_DATAGRAM setting. This allows new draft revisions to make incompatible changes. Multiple draft versions MAY be supported by sending multiple values for SETTINGS_H3_DATAGRAM. Once SETTINGS have been sent and received, an implementation that supports multiple drafts MUST compute the intersection of the values it has sent and received, and then it MUST select and use the most recent draft version from the intersection set. This ensures that both peers negotiate the same draft version.

2.2. HTTP Datagrams using Capsules

When HTTP/3 Datagrams are unavailable or undesirable, HTTP Datagrams can be sent using the Capsule Protocol, see Section 3.5.

3. Capsules

One mechanism to extend HTTP is to introduce new HTTP Upgrade Tokens (see Section 16.7 of [HTTP]). In HTTP/1.x, these tokens are used via the Upgrade mechanism (see Section 7.8 of [HTTP]). In HTTP/2 and HTTP/3, these tokens are used via the Extended CONNECT mechanism (see [EXT-CONNECT2] and [EXT-CONNECT3]).

This specification introduces the Capsule Protocol. The Capsule Protocol is a sequence of type-length-value tuples that definitions of new HTTP Upgrade Tokens can choose to use. It allows endpoints to reliably communicate request-related information end-to-end on HTTP request streams, even in the presence of HTTP intermediaries. The Capsule Protocol can be used to exchange HTTP Datagrams, which is necessary when HTTP is running over a transport that does not support the QUIC DATAGRAM frame. The Capsule Protocol can also be used to communicate reliable and bidirectional control messages associated with a datagram-based protocol even when HTTP/3 Datagrams are in use.

3.1. HTTP Data Streams

This specification defines the "data stream" of an HTTP request as the bidirectional stream of bytes that follows the header section of the request message and the final, successful (i.e., 2xx) response message.

In HTTP/1.x, the data stream consists of all bytes on the connection that follow the blank line that concludes either the request header section, or the response header section. As a result, only a single HTTP request starting the capsule protocol can be sent on HTTP/1.x connections.

In HTTP/2 and HTTP/3, the data stream of a given HTTP request consists of all bytes sent in DATA frames with the corresponding stream ID.

The concept of a data stream is particularly relevant for methods such as CONNECT where there is no HTTP message content after the headers.

Data streams can be prioritized using any means suited to stream or request prioritization. For example, see Section 11 of [PRIORITY].

3.2. The Capsule Protocol

Definitions of new HTTP Upgrade Tokens can state that their associated request's data stream uses the Capsule Protocol. If they do so, that means that the contents of the associated request's data stream uses the following format (using the notation from the "Notational Conventions" section of [QUIC]):

```
Capsule Protocol {  
    Capsule (..) ...,  
}
```

Figure 2: Capsule Protocol Stream Format

```
Capsule {  
    Capsule Type (i),  
    Capsule Length (i),  
    Capsule Value (..),  
}
```

Figure 3: Capsule Format

Capsule Type: A variable-length integer indicating the Type of the capsule.

Capsule Length: The length of the Capsule Value field following this field, encoded as a variable-length integer. Note that this field can have a value of zero.

Capsule Value: The payload of this capsule. Its semantics are determined by the value of the Capsule Type field.

An intermediary can identify the use of the capsule protocol either through the presence of the Capsule-Protocol header field (Section 3.4) or by understanding the chosen HTTP Upgrade token.

Because new protocols or extensions might define new capsule types, intermediaries that wish to allow for future extensibility SHOULD forward capsules without modification, unless the definition of the Capsule Type in use specifies additional intermediary processing. One such Capsule Type is the DATAGRAM capsule; see Section 3.5. In particular, intermediaries SHOULD forward Capsules with an unknown Capsule Type without modification.

Endpoints which receive a Capsule with an unknown Capsule Type MUST silently drop that Capsule and skip over it to parse the next Capsule.

By virtue of the definition of the data stream:

- * The Capsule Protocol is not in use unless the response includes a 2xx (Successful) status code.
- * When the Capsule Protocol is in use, the associated HTTP request and response do not carry HTTP content. A future extension MAY define a new capsule type to carry HTTP content.

The Capsule Protocol MUST NOT be used with messages that contain Content-Length, Content-Type, or Transfer-Encoding header fields. Additionally, HTTP status codes 204 (No Content), 205 (Reset Content), and 206 (Partial Content) MUST NOT be sent on responses that use the Capsule Protocol. A receiver that observes a violation of these requirements MUST treat the HTTP message as malformed.

3.3. Error Handling

When an error occurs in processing the Capsule Protocol, the receiver MUST treat the message as malformed or incomplete, according to the underlying transport protocol. For HTTP/3, the handling of malformed messages is described in Section 4.1.3 of [HTTP/3]. For HTTP/2, the handling of malformed messages is described in Section 8.1.1 of [HTTP/2]. For HTTP/1.1, the handling of incomplete messages is described in Section 8 of [HTTP/1.1].

Each capsule's payload MUST contain exactly the fields identified in its description. A capsule payload that contains additional bytes after the identified fields or a capsule payload that terminates before the end of the identified fields MUST be treated as a malformed or incomplete message. In particular, redundant length encodings MUST be verified to be self-consistent.

If the receive side of a stream carrying capsules is terminated cleanly (for example, in HTTP/3 this is defined as receiving a QUIC STREAM frame with the FIN bit set) and the last capsule on the stream was truncated, this MUST be treated as a malformed or incomplete message.

3.4. The Capsule-Protocol Header Field

The "Capsule-Protocol" header field is an Item Structured Field, see Section 3.3 of [STRUCT-FIELD]; its value MUST be a Boolean; 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 Capsule-Protocol header field value, but future documents might define parameters.

Receivers MUST ignore unknown parameters.

Endpoints indicate that the Capsule Protocol is in use on a data stream by sending a Capsule-Protocol header field with a true value. A Capsule-Protocol header field with a false value has the same semantics as when the header is not present.

Intermediaries MAY use this header field to allow processing of HTTP Datagrams for unknown HTTP Upgrade Tokens; note that this is only possible for HTTP Upgrade or Extended CONNECT.

The Capsule-Protocol header field MUST NOT be used on HTTP responses with a status code outside the 2xx range.

When using the Capsule Protocol, HTTP endpoints SHOULD send the Capsule-Protocol header field to simplify intermediary processing. Definitions of new HTTP Upgrade Tokens that use the Capsule Protocol MAY alter this recommendation.

3.5. The DATAGRAM Capsule

This document defines the DATAGRAM (0x00) capsule type. This capsule allows HTTP Datagrams to be sent on a stream using the Capsule Protocol. This is particularly useful when HTTP is running over a transport that does not support the QUIC DATAGRAM frame.

```
Datagram Capsule {  
  Type (i) = 0x00,  
  Length (i),  
  HTTP Datagram Payload (..),  
}
```

Figure 4: DATAGRAM Capsule Format

HTTP Datagram Payload: The payload of the datagram, whose semantics are defined by the extension that is using HTTP Datagrams. Note that this field can be empty.

HTTP Datagrams sent using the DATAGRAM capsule have the same semantics as those sent in QUIC DATAGRAM frames. In particular, the restrictions on when it is allowed to send an HTTP Datagram and how to process them from Section 2.1 also apply to HTTP Datagrams sent and received using the DATAGRAM capsule.

An intermediary can reencode HTTP Datagrams as it forwards them. In other words, an intermediary MAY send a DATAGRAM capsule to forward an HTTP Datagram that was received in a QUIC DATAGRAM frame, and vice versa. Intermediaries MUST NOT perform this reencoding unless they have identified the use of the Capsule Protocol on the corresponding request stream; see Section 3.2.

Note that while DATAGRAM capsules that are sent on a stream are reliably delivered in order, intermediaries can reencode DATAGRAM capsules into QUIC DATAGRAM frames when forwarding messages, which could result in loss or reordering.

If an intermediary receives an HTTP Datagram in a QUIC DATAGRAM frame and is forwarding it on a connection that supports QUIC DATAGRAM frames, the intermediary SHOULD NOT convert that HTTP Datagram to a DATAGRAM capsule. If the HTTP Datagram is too large to fit in a DATAGRAM frame (for example because the path MTU of that QUIC connection is too low or if the maximum UDP payload size advertised on that connection is too low), the intermediary SHOULD drop the HTTP Datagram instead of converting it to a DATAGRAM capsule. This preserves the end-to-end unreliability characteristic that methods such as Datagram Packetization Layer Path MTU Discovery (DPLPMTUD) depend on [DPLPMTUD]. An intermediary that converts QUIC DATAGRAM frames to DATAGRAM capsules allows HTTP Datagrams to be arbitrarily large without suffering any loss; this can misrepresent the true path properties, defeating methods such as DPLPMTUD.

While DATAGRAM capsules can theoretically carry a payload of length $2^{62}-1$, most HTTP extensions that use HTTP Datagrams will have their own limits on what datagram payload sizes are practical. Implementations SHOULD take those limits into account when parsing DATAGRAM capsules: if an incoming DATAGRAM capsule has a length that is known to be so large as to not be usable, the implementation SHOULD discard the capsule without buffering its contents into memory.

Note that it is possible for an HTTP extension to use HTTP Datagrams without using the Capsule Protocol. For example, if an HTTP extension that uses HTTP Datagrams is only defined over transports that support QUIC DATAGRAM frames, it might not need a stream encoding. Additionally, HTTP extensions can use HTTP Datagrams with their own data stream protocol. However, new HTTP extensions that wish to use HTTP Datagrams SHOULD use the Capsule Protocol as failing to do so will make it harder for the HTTP extension to support versions of HTTP other than HTTP/3 and will prevent interoperability with intermediaries that only support the Capsule Protocol.

4. Security Considerations

Since transmitting HTTP Datagrams using QUIC DATAGRAM frames requires sending the HTTP/3 SETTINGS_H3_DATAGRAM setting, it "sticks out". In other words, probing clients can learn whether a server supports HTTP Datagrams over QUIC DATAGRAM frames. As some servers might wish to obfuscate the fact that they offer application services that use HTTP datagrams, it's best for all implementations that support this feature to always send this setting, see Section 2.1.1.

Since use of the Capsule Protocol is restricted to new HTTP Upgrade Tokens, it is not accessible from Web Platform APIs (such as those commonly accessed via JavaScript in web browsers).

5. IANA Considerations

5.1. HTTP/3 Setting

This document will request IANA to register the following entry in the "HTTP/3 Settings" registry:

Value: 0x33
Setting Name: SETTINGS_H3_DATAGRAM
Default: 0
Status: provisional (permanent if this document is approved)
Specification: This Document
Change Controller: IETF
Contact: HTTP_WG; HTTP working group; ietf-http-wg@w3.org

5.2. HTTP/3 Error Code

This document will request IANA to register the following entry in the "HTTP/3 Error Codes" registry:

Value: 0x33
Name: H3_DATAGRAM_ERROR
Description: Datagram or capsule protocol parse error
Status: provisional (permanent if this document is approved)
Specification: This Document
Change Controller: IETF
Contact: HTTP_WG; HTTP working group; ietf-http-wg@w3.org

5.3. HTTP Header Field Name

This document will request IANA to register the following entry in the "HTTP Field Name" registry:

Field Name: Capsule-Protocol

Template: None
Status: provisional (permanent if this document is approved)
Reference: This document
Comments: None

5.4. Capsule Types

This document establishes a registry for HTTP capsule type codes. The "HTTP Capsule Types" registry governs a 62-bit space, and operates under the QUIC registration policy documented in Section 22.1 of [QUIC]. This new registry includes the common set of fields listed in Section 22.1.1 of [QUIC]. In addition to those common fields, all registrations in this registry MUST include a "Capsule Type" field which contains a short name or label for the capsule type.

Permanent registrations in this registry are assigned using the Specification Required policy (Section 4.6 of [IANA-POLICY]), except for values between 0x00 and 0x3f (in hexadecimal; inclusive), which are assigned using Standards Action or IESG Approval as defined in Sections 4.9 and 4.10 of [IANA-POLICY].

Capsule types with a value of the form $0x29 * N + 0x17$ for integer values of N are reserved to exercise the requirement that unknown capsule types be ignored. These capsules have no semantics and can carry arbitrary values. These values MUST NOT be assigned by IANA and MUST NOT appear in the listing of assigned values.

This registry initially contains the following entry:

Value: 0x00
Capsule Type: DATAGRAM
Status: permanent
Specification: This document
Change Controller: IETF
Contact: MASQUE Working Group masque@ietf.org
(mailto:masque@ietf.org)
Notes: None

6. References

6.1. Normative References

[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>>.

- [HTTP] 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>>.
- [HTTP/1.1] Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP/1.1", Work in Progress, Internet-Draft, draft-ietf-httpbis-messaging-19, 12 September 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-messaging-19>>.
- [HTTP/2] Thomson, M. and C. Benfield, "HTTP/2", Work in Progress, Internet-Draft, draft-ietf-httpbis-http2bis-07, 24 January 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-http2bis-07>>.
- [HTTP/3] Bishop, M., "Hypertext Transfer Protocol Version 3 (HTTP/3)", Work in Progress, Internet-Draft, draft-ietf-quic-http-34, 2 February 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-34>>.
- [IANA-POLICY] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/rfc/rfc8126>>.
- [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>>.
- [STRUCT-FIELD] Nottingham, M. and P-H. Kamp, "Structured Field Values for HTTP", RFC 8941, DOI 10.17487/RFC8941, February 2021, <<https://www.rfc-editor.org/rfc/rfc8941>>.

6.2. Informative References

- [DPLPMTUD] Fairhurst, G., Jones, T., Tüxen, M., Rüngeler, I., and T. Völker, "Packetization Layer Path MTU Discovery for Datagram Transports", RFC 8899, DOI 10.17487/RFC8899, September 2020, <<https://www.rfc-editor.org/rfc/rfc8899>>.
- [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>>.
- [PRIORITY] Oku, K. and L. Pardue, "Extensible Prioritization Scheme for HTTP", Work in Progress, Internet-Draft, draft-ietf-httpbis-priority-12, 17 January 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-priority-12>>.
- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <<https://www.rfc-editor.org/rfc/rfc6455>>.

Acknowledgments

Portions of this document were previously part of the QUIC DATAGRAM frame definition itself, the authors would like to acknowledge the authors of that document and the members of the IETF MASQUE working group for their suggestions. Additionally, the authors would like to thank Martin Thomson for suggesting the use of an HTTP/3 setting. Furthermore, the authors would like to thank Ben Schwartz for writing the first proposal that used two layers of indirection. The final design in this document came out of the HTTP Datagrams Design Team, whose members were Alan Frindell, Alex Chernyakhovsky, Ben Schwartz, Eric Rescorla, Marcus Ihlar, Martin Thomson, Mike Bishop, Tommy Pauly, Victor Vasiliev, and the authors of this document. The authors thank Mark Nottingham and Philipp Tiesel for their helpful comments.

Authors' Addresses

David Schinazi
Google LLC
1600 Amphitheatre Parkway
Mountain View, CA 94043
United States of America
Email: dschinazi.ietf@gmail.com

Lucas Pardue
Cloudflare
Email: lucaspardue.24.7@gmail.com

MASQUE
Internet-Draft
Intended status: Experimental
Expires: 27 January 2022

L. Pardue
Cloudflare
26 July 2021

HTTP Datagram Prioritization
draft-pardue-masque-dgram-priority-01

Abstract

Application protocols using the QUIC transport protocol rely on streams, and optionally the DATAGRAM extension, to carry application data. Streams and datagrams can be multiplexed but QUIC provides no interoperable prioritization scheme or signaling mechanism itself. The HTTP Extensible Prioritization scheme describes how to prioritize streams in HTTP/2 and HTTP/3. This document adopts the scheme to support HTTP datagrams.

Note to Readers

RFC EDITOR: please remove this section before publication

Source code and issues list for this draft can be found at
<https://github.com/LPardue/draft-pardue-masque-dgram-priority>
(<https://github.com/LPardue/draft-pardue-masque-dgram-priority>).

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 January 2022.

Copyright Notice

Copyright (c) 2021 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 Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Notational Conventions	3
2. Signalling Datagram Priority	3
2.1. Datagram Urgency	4
2.2. Prioritization of Contexts	4
2.3. Reprioritization	4
3. Client Scheduling	4
4. Server Scheduling	5
5. Retransmission Scheduling	5
6. Security Considerations	5
7. IANA Considerations	5
8. References	5
8.1. Normative References	5
8.2. Informative References	6
Appendix A. Acknowledgements	6
Author's Address	7

1. Introduction

Application protocols using the QUIC transport protocol [QUIC] rely on streams, and optionally the DATAGRAM extension [QUIC-DATAGRAM], to carry application data. Streams and datagrams can be multiplexed but QUIC provides no interoperable prioritization scheme or signaling mechanism itself. The HTTP Extensible Prioritization scheme [I-D.ietf-httpbis-priority] describes how to prioritize streams in HTTP/2 and HTTP/3. This document adopts the scheme to support HTTP datagrams [HTTP-DATAGRAM].

The Extensible Priorities scheme for HTTP describes how clients can send priority signals related to requests in order to suggest how a server allocates resources to serving responses. When the protocol is HTTP/2, responses are carried on streams. When the protocol is HTTP/3, responses are carried on QUIC streams.

While QUIC streams support multiplexing natively via use of a stream identifier, the QUIC DATAGRAM extension does not provide any such identifier. HTTP datagrams [HTTP-DATAGRAM] supports multiplexing

using a set of application-level identifiers that can be controlled and accessed by HTTP/3. One identifier relates to a request stream, the second, optional, identifier relates to an abstract context. [HTTP-DATAGRAM] does not, however, define any means for multiplexed datagram prioritization.

When the application protocol is HTTP/3, HTTP Datagrams can map directly to QUIC datagrams or they can be carried on streams using a DATAGRAM Capsule; see Section 4.4 of [HTTP-DATAGRAM].

This document describes how the Extensible Priorities scheme applies to HTTP datagrams. Priority signals sent by clients, related to requests, can also be considered input to server scheduling decisions for HTTP datagrams mapped to QUIC datagrams.

1.1. Notational Conventions

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.

The term sf-integer is imported from [STRUCTURED-FIELDS].

2. Signalling Datagram Priority

The Extensible Prioritization scheme [I-D.ietf-httpbis-priority] provides a framework for communicating and acting upon priority parameters, using [STRUCTURED-FIELDS] formats. It defines the urgency and incremental parameters and provides guidance to implementers about how to act on these parameters, in combination with other inputs, to make resource allocation and scheduling choices. Urgency communicates the client-view of request importance, and incremental communicates how the client intends to process response data as it arrives. Parameters are communicated in HTTP headers or version-specific frames. A client omitting the urgency or incremental parameters can be interpreted by the server as a signal to apply default priorities. The core scheme is extensible, new parameters can be defined to augment the base ones.

This specification defines the datagram-urgency ("du") extension parameter that operates in addition to the base urgency. There is no extension to the base incremental behavior; individual datagrams, even if belonging to the same identifier, are messages that are expected to be processed individually as they arrive.

2.1. Datagram Urgency

The datagram-urgency parameter ("du") takes an integer between 0 and 7, in descending order of priority. This range matches the base urgency ("u") parameter range; see Section 4.1 of [I-D.ietf-httpbis-priority].

The value is encoded as an sf-integer. There is no default value.

This parameter indicates the sender's recommendation, based on the expectation that the server would transmit HTTP datagrams in the order of their datagram-urgency values if possible. The smaller the value, the higher the precedence. Omitting the datagram-urgency parameter is a signal to apply the value of the urgency parameter.

The following example shows a request for a CSS file with the urgency set to "0", any associated datagrams have the lower urgency of "2":

```
:method = GET
:scheme = https
:authority = example.net
:path = /style.css
priority = u=0, du=2
```

Endpoints MUST NOT treat reception of the datagram-urgency parameter, even if HTTP datagram support is not enabled.

The datagram-urgency parameter applies only to HTTP datagrams mapped to QUIC datagrams. Datagram capsules are sent on streams, so the base urgency parameter applies to them.

2.2. Prioritization of Contexts

The datagram-urgency parameter applies to all HTTP datagram contexts related to a request stream. Prioritization of individual contexts is not supported.

2.3. Reprioritization

Reprioritization is supported using the existing mechanisms defined in Section 6 of [I-D.ietf-httpbis-priority].

3. Client Scheduling

Clients MAY use datagram-urgency to make local processing or scheduling choices about HTTP datagrams related to the requests it initiates.

4. Server Scheduling

Priority signals are input to a prioritization process. Expressing priority is only a suggestion. The datagram-urgency parameter introduces new scheduling considerations on top of those presented in Section 10 of [I-D.ietf-httpbis-priority].

It is RECOMMENDED that, when possible, servers send higher urgency HTTP datagrams before lower urgency datagrams.

Where streams and datagrams have equal urgency and datagram-urgency, it is RECOMMENDED that servers alternate emitting HTTP datagrams and stream bytes. Where servers implement the recommendations in Section 10 of [I-D.ietf-httpbis-priority], alternating between datagram and stream data will result in fair scheduling. This recommendation holds whether stream are incremental or not.

It is RECOMMENDED that servers schedule DATAGRAM capsules the same as response data.

5. Retransmission Scheduling

Section 12 of [I-D.ietf-httpbis-priority] provides guidance about scheduling of retransmission data vs. new data. Since QUIC datagrams are not retransmitted, endpoints that prioritize QUIC stream retransmission data could delay datagrams. Furthermore, since DATAGRAM capsules are sent as stream data, they *are* subject to retransmission and could also delay native QUIC datagrams.

6. Security Considerations

There are believed to be no additional considerations to those presented in [I-D.ietf-httpbis-priority].

7. IANA Considerations

This specification registers the following entry in the HTTP Priority Parameters Registry

Name: datagram-urgency

Description: Priority of HTTP datagrams

Reference: This document

8. References

8.1. Normative References

[HTTP-DATAGRAM]

Schinazi, D. and L. Pardue, "Using Datagrams with HTTP", Work in Progress, Internet-Draft, draft-ietf-masque-h3-datagram-03, 12 July 2021, <<https://www.ietf.org/archive/id/draft-ietf-masque-h3-datagram-03.txt>>.

[I-D.ietf-httpbis-priority]

Oku, K. and L. Pardue, "Extensible Prioritization Scheme for HTTP", Work in Progress, Internet-Draft, draft-ietf-httpbis-priority-04, 11 July 2021, <<https://www.ietf.org/archive/id/draft-ietf-httpbis-priority-04.txt>>.

[QUIC-DATAGRAM]

Pauly, T., Kinnear, E., and D. Schinazi, "An Unreliable Datagram Extension to QUIC", Work in Progress, Internet-Draft, draft-ietf-quic-datagram-03, 12 July 2021, <<https://www.ietf.org/archive/id/draft-ietf-quic-datagram-03.txt>>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

[STRUCTURED-FIELDS]

Nottingham, M. and P-H. Kamp, "Structured Field Values for HTTP", RFC 8941, DOI 10.17487/RFC8941, February 2021, <<https://www.rfc-editor.org/info/rfc8941>>.

8.2. Informative References

[QUIC] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <<https://www.rfc-editor.org/info/rfc9000>>.

Appendix A. Acknowledgements

This document is inspired by discussion by many people across HTTP, QUIC and MASQUE WGs.

Author's Address

Lucas Pardue
Cloudflare

Email: lucaspardue.24.7@gmail.com

masque
Internet-Draft
Intended status: Standards Track
Expires: 7 April 2022

B. Schwartz
Google LLC
4 October 2021

HTTP Datagram PING
draft-schwartz-masque-h3-datagram-ping-01

Abstract

This draft defines an HTTP Datagram Format Type for measuring the functionality of a Datagram path.

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 7 April 2022.

Copyright Notice

Copyright (c) 2021 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 Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Conventions and Definitions	2
2. PING Datagram Format Type	2
2.1. Format	2
2.2. Use	3
3. Use cases	3
4. IANA considerations	3
5. References	3
5.1. Normative References	3
5.2. Informative References	4
Acknowledgments	4
Author's Address	4

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 Datagram Format Type

PING is an HTTP Datagram Format Type [I-D.draft-ietf-masque-h3-datagram]. It has no Additional Data.

2.1. Format

PING Datagrams have the following format:

```
PING {
    Sequence Number (i),
    Opaque Data (...),
}
```

Figure 1: PING Datagram Format

All Sequence Number and Opaque Data values are potentially valid.

2.2. 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. Use cases

PING Datagrams can be used to characterize 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 Datagram path, in order to avoid sending HTTP Datagrams 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]).

4. IANA considerations

IANA is directed to add the following entry to the "HTTP Datagram Format Types" registry:

- * Type: PING
- * Value: TBD
- * Reference: (This document)

5. References

5.1. Normative References

[I-D.draft-ietf-masque-h3-datagram]
Schinazi, D. and L. Pardue, "Using Datagrams with HTTP",
Work in Progress, Internet-Draft, draft-ietf-masque-h3-
datagram-03, 12 July 2021,
<<https://datatracker.ietf.org/doc/html/draft-ietf-masque-h3-datagram-03>>.

- [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>>.

5.2. Informative References

- [I-D.draft-ietf-httpbis-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>>.
- [RFC8899] Fairhurst, G., Jones, T., Tüxen, M., Rüngeler, I., and T. Völker, "Packetization Layer Path MTU Discovery for Datagram Transports", RFC 8899, DOI 10.17487/RFC8899, September 2020, <<https://www.rfc-editor.org/rfc/rfc8899>>.

Acknowledgments

Thanks to Alex Chernyakhovsky for constructive input.

Author's Address

Benjamin Schwartz
Google LLC

Email: bemasc@google.com