Internet Engineering Task Force (IETF)
Request for Comments: 9298
Category: Standards Track
ISSN: 2070-1721

D. Schinazi
Google LLC

August 2022

Proxying UDP in HTTP

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 an HTTP client to create a tunnel for UDP communications through an HTTP server that acts as a proxy.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9298.

Table of Contents

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 lacked a method for doing so for UDP [UDP] traffic prior to this specification.

This document describes a protocol for tunneling 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 the HTTP CONNECT method, 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 existing versions of HTTP by using HTTP Datagrams [HTTP-DGRAM]. When using HTTP/2 [HTTP/2] or HTTP/3 [HTTP/3], it uses HTTP Extended CONNECT as described in [EXT-CONNECT2] and [EXT-CONNECT3]. When using HTTP/1.x [HTTP/1.1], it uses HTTP Upgrade as defined in Section 7.8 of [HTTP].

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 tunneling request to open a UDP socket to a target server and that 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://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; see Section 2.1 of [URI]).

* The URI Template MUST NOT use Reserved Expansion ("+" operator), Fragment Expansion ("#" operator), Label Expansion with Dot-Prefix, Path Segment Expansion with Slash-Prefix, nor Path-Style Parameter Expansion with Semicolon-Prefix.

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

The original HTTP CONNECT method allowed for the conveyance of the target host and port, but not the scheme, proxy authority, path, or query. Thus, clients with proxy configuration interfaces that only allow the user to configure the proxy host and the proxy port exist. 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 Datagrams in the format defined in Section 5.

To initiate a UDP tunnel associated with a single HTTP stream, a client issues 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.

"target_host" supports using DNS names, IPv6 literals and IPv4 literals. Note that IPv6 scoped addressing zone identifiers are not supported. Using the terms IPv6address, IPv4address, reg-name, and port from [URI], the "target_host" and "target_port" variables MUST adhere to the format in Figure 2, using notation from [ABNF]. Additionally:

* both the "target_host" and "target_port" variables MUST NOT be empty.

* if "target_host" contains an IPv6 literal, the colons (":") MUST be percent-encoded. For example, if the target host is "2001:db8::42", it will be encoded in the URI as "2001%3Adb8%3A%3A42".

* "target_port" MUST represent an integer between 1 and 65535 inclusive.

```
target_host = IPv6address / IPv4address / reg-name
target_port = port
```

Figure 2: URI Template Variable Format

When sending its UDP proxying request, the client SHALL perform URI Template expansion to determine the path and query of its request.

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.

By virtue of the definition of the Capsule Protocol (see Section 3.2 of [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 by forwarding the request to another HTTP server. Note that such intermediaries may need to re-encode the

request if they forward it using a version of HTTP that is different from the one used to receive it, as the request encoding differs by version (see below).

* otherwise, the recipient will act as a UDP proxy. It extracts the "target_host" and "target_port" variables from the URI it has reconstructed from the request headers, decodes their percent-encoding, and establishes a tunnel by directly opening a UDP socket to the requested target.

Unlike TCP, UDP is connectionless. 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 reject the request and SHOULD send details using an appropriate Proxy-Status header field [PROXY-STATUS]. For example, if DNS resolution returns an error, the proxy can use the dns_error Proxy Error Type from Section 2.3.2 of [PROXY-STATUS].

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, it MUST close the request stream. For example, this can happen when an ICMP Destination Unreachable message is received; see Section 3.1 of [ICMP6]. 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 Sections 3.3 and 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; thus, the client MUST abort the request.

UDP proxies MUST NOT introduce fragmentation at the IP layer when forwarding HTTP Datagrams onto a UDP socket; overly large datagrams are silently dropped. 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.

Implementers of UDP proxies will benefit from reading the guidance in [UDP-USAGE].

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".

A UDP proxying request that does not conform to these restrictions is malformed. The recipient of such a malformed request MUST respond with an error and SHOULD use the 400 (Bad Request) status code.

For example, if the client is configured with URI Template "https://example.org/.well-known/masque/udp/{target_host}/{target_port}/" and wishes to open a UDP proxying tunnel to target 192.0.2.6:443, it could send the following request:

```
GET https://example.org/.well-known/masque/udp/192.0.2.6/443/ HTTP/1.1
Host: example.org
Connection: Upgrade
Upgrade: connect-udp
Capsule-Protocol: ?1
```

Figure 3: 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 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 meet the requirements of HTTP responses that start the Capsule Protocol; see Section 3.2 of [HTTP-DGRAM].

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

For example, the UDP proxy could respond with:

```
HTTP/1.1 101 Switching Protocols
Connection: Upgrade
Upgrade: connect-udp
Capsule-Protocol: ?1
```

Figure 4: 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 HTTP Extended CONNECT. This requires that servers send an HTTP Setting as specified in [EXT-CONNECT2] and [EXT-CONNECT3] and that requests use HTTP pseudo-header fields with the following requirements:

* The :method pseudo-header field SHALL be "CONNECT".

* The :protocol pseudo-header field SHALL be "connect-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] and Section 4.1.2 of [HTTP/3]).

For example, if the client is configured with URI Template "https://example.org/.well-known/masque/udp/{target_host}/{target_port}/" and wishes to open a UDP proxying tunnel to target 192.0.2.6:443, it could send the following request:

```
HEADERS
:method = CONNECT
:protocol = connect-udp
:scheme = https
:path = /.well-known/masque/udp/192.0.2.6/443/
:authority = example.org
capsule-protocol = ?1
```

Figure 5: Example HTTP/2 Request

3.5. HTTP/2 and HTTP/3 Responses

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

* the HTTP status code on the response SHALL be in the 2xx (Successful) range.

* the response SHALL meet the requirements of HTTP responses that start the Capsule Protocol; see Section 3.2 of [HTTP-DGRAM].

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

For example, the UDP proxy could respond with:

```
HEADERS
:status = 200
capsule-protocol = ?1
```

Figure 6: Example HTTP/2 Response

4. Context Identifiers

The mechanism for proxying UDP in HTTP defined in this document allows future extensions to exchange HTTP Datagrams that 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 field; 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 synchronization between endpoints. However, once a Context ID has been allocated, those restrictions do not apply to the use of the Context ID; it can be used by 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 Context IDs. Depending on the method being used, it is possible for datagrams to be received with Context IDs that have not yet been registered. For instance, this can be due to reordering of the packet containing the datagram and the packet containing the registration message during transmission.

5. HTTP Datagram Payload Format

When HTTP Datagrams (see Section 2 of [HTTP-DGRAM]) are associated with UDP Proxying request streams, the HTTP Datagram Payload field has the format defined in Figure 7, using notation from Section 1.3

of [QUIC]. Note that when HTTP Datagrams are encoded using QUIC DATAGRAM frames [QUIC-DGRAM], the Context ID field defined below directly follows the Quarter Stream ID field, which is at the start of the QUIC DATAGRAM frame payload; see Section 2.1 of [HTTP-DGRAM].

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

Figure 7: 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 that carries an unknown Context ID is received, the receiver SHALL either drop that datagram silently or buffer it temporarily (on the order of a round trip) while awaiting the registration of the corresponding Context ID.

UDP Proxying Payload: The payload of the datagram, whose semantics depend on the value of the previous field. Note that this field can be empty.

UDP packets are encoded using HTTP Datagrams with the Context ID field set to zero. When the Context ID field is set to zero, the UDP Proxying Payload field contains the unmodified payload of a UDP packet (referred to as data octets in [UDP]).

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 UDP Proxying Payload field longer than 65527 using Context ID zero. An endpoint that receives an HTTP Datagram using Context ID zero whose UDP Proxying Payload field is longer than 65527 MUST abort the corresponding stream. If a UDP proxy knows it can only send out UDP packets of a certain length due to its underlying link MTU, it has no choice but to discard incoming HTTP Datagrams using Context ID zero whose UDP Proxying Payload field is longer than that limit. If the discarded HTTP Datagram was transported by a DATAGRAM capsule, the receiver SHOULD discard that capsule without buffering the capsule contents.

If a UDP proxy receives an HTTP Datagram before it has received the corresponding request, it SHALL either drop that HTTP Datagram silently or buffer it temporarily (on the order of a round trip) while awaiting the corresponding request.

Note that buffering datagrams (either because the request was not yet received or because the Context ID is not yet known) consumes resources. Receivers that buffer datagrams SHOULD apply buffering limits in order to reduce the risk of resource exhaustion occurring. For example, receivers can limit the total number of buffered datagrams or the cumulative size of buffered datagrams on a per-stream, per-context, or per-connection basis.

A client MAY optimistically start sending UDP packets in HTTP Datagrams before receiving the response to its UDP proxying request. However, implementers 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 and the UDP proxy chooses to not buffer them.

6. Performance Considerations

Bursty traffic can often lead to temporally correlated packet losses; in turn, this can lead to suboptimal responses from congestion controllers in protocols running over UDP. To avoid this, 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. 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 Explicit Congestion Notification (ECN) [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 QUIC ACK_ECN frames or the TCP ECE bit, 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 [QUIC-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 (PMTU). 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 PMTU 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.

In this specification, note that UDP proxying clients do not have the ability 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 it 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.

There exist software and network deployments that perform access control checks based on the source IP address of incoming requests. For example, some software allows unauthenticated configuration changes if they originated from 127.0.0.1. Such software could be running on the same host as the UDP proxy or in the same broadcast domain. Proxied UDP traffic would then be received with a source IP address belonging to the UDP proxy. If this source address is used for access control, UDP proxying clients could use the UDP proxy to escalate their access privileges beyond those they might otherwise have. This could lead to unauthorized access by UDP proxying clients unless the UDP proxy disallows UDP proxying requests to vulnerable targets, such as the UDP proxy's own addresses and localhost, link-local, multicast, and broadcast addresses. UDP proxies can use the destination_ip_prohibited Proxy Error Type from Section 2.3.5 of [PROXY-STATUS] when rejecting such requests.

UDP proxies share many similarities with TCP CONNECT proxies when considering them as infrastructure for abuse to enable denial-of-service (DoS) attacks. Both can obfuscate the attacker's source address from the attack target. In the case of a stateless volumetric attack (e.g., a TCP SYN flood or a UDP flood), both types of proxies pass the traffic to the target host. With stateful volumetric attacks (e.g., HTTP flooding) being sent over a TCP CONNECT proxy, the proxy will only send data if the target has indicated its willingness to accept data by responding with a TCP SYN-ACK. Once the path to the target is flooded, the TCP CONNECT proxy will no longer receive replies from the target and will stop sending data. Since UDP does not establish shared state between the UDP proxy and the target, the UDP proxy could continue sending data to the target in such a situation. 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 provides limited protection against DoS attacks when 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. Such a packet limit could also cause issues for valid traffic.

The security considerations described in Section 4 of [HTTP-DGRAM] also apply here. Since it is possible to tunnel IP packets over UDP, the guidance in [TUNNEL-SECURITY] can apply.

8. IANA Considerations

8.1. HTTP Upgrade Token

IANA has registered "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: RFC 9298

8.2. Well-Known URI

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

URI Suffix: masque
Change Controller: IETF
Reference: RFC 9298
Status: permanent
Related Information: Includes all resources identified with the path prefix "/.well-known/masque/udp/"

9. References

9.1. Normative References

[ABNF] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, DOI 10.17487/RFC2234, November 1997, <https://www.rfc-editor.org/info/rfc2234>.

[ECN] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <https://www.rfc-editor.org/info/rfc3168>.

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

[EXT-CONNECT3] Hamilton, R., "Bootstrapping WebSockets with HTTP/3", RFC 9220, DOI 10.17487/RFC9220, June 2022, <https://www.rfc-editor.org/info/rfc9220>.

[HTTP] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

[HTTP-DGRAM]
Schinazi, D. and L. Pardue, "HTTP Datagrams and the Capsule Protocol", RFC 9297, DOI 10.17487/RFC9297, August 2022, <https://www.rfc-editor.org/info/rfc9297>.

[HTTP/1.1] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1", STD 99, RFC 9112, DOI 10.17487/RFC9112, June 2022, <https://www.rfc-editor.org/info/rfc9112>.

[HTTP/2] Thomson, M., Ed. and C. Benfield, Ed., "HTTP/2", RFC 9113, DOI 10.17487/RFC9113, June 2022, <https://www.rfc-editor.org/info/rfc9113>.

[HTTP/3] Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <https://www.rfc-editor.org/info/rfc9114>.

[PROXY-STATUS]
Nottingham, M. and P. Sikora, "The Proxy-Status HTTP Response Header Field", RFC 9209, DOI 10.17487/RFC9209, June 2022, <https://www.rfc-editor.org/info/rfc9209>.

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

[QUIC-DGRAM]
Pauly, T., Kinnear, E., and D. Schinazi, "An Unreliable Datagram Extension to QUIC", RFC 9221, DOI 10.17487/RFC9221, March 2022, <https://www.rfc-editor.org/info/rfc9221>.

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

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

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

[TEMPLATE] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <https://www.rfc-editor.org/info/rfc6570>.

[UDP] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/info/rfc768>.

[URI] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform

Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

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/info/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/info/rfc8899>.

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

[HELIUM] Schwartz, B. M., "Hybrid Encapsulation Layer for IP and UDP Messages (HELIUM)", Work in Progress, Internet-Draft, draft-schwartz-httpbis-helium-00, 25 June 2018, <https://datatracker.ietf.org/doc/html/draft-schwartz-httpbis-helium-00>.

[HiNT] Pardue, L., "HTTP-initiated Network Tunnelling (HiNT)", Work in Progress, Internet-Draft, draft-pardue-httpbis-http-network-tunnelling-00, 2 July 2018, <https://datatracker.ietf.org/doc/html/draft-pardue-httpbis-http-network-tunnelling-00>.

[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/info/rfc4443>.

[MASQUE-ORIGINAL] Schinazi, D., "The MASQUE Protocol", Work in Progress, Internet-Draft, draft-schinazi-masque-00, 28 February 2019, <https://datatracker.ietf.org/doc/html/draft-schinazi-masque-00>.

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

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

[WEBSOCKET]
Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <https://www.rfc-editor.org/info/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 [HELIUM] by Ben Schwartz, [HiNT] by Lucas Pardue, and the original MASQUE Protocol [MASQUE-ORIGINAL] by the author of this document.

The author would like to thank Eric Rescorla for suggesting the use of 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