MASQUE
Internet-Draft
Intended status: Standards Track
Expires: 19 July 2026

D. Schinazi
A. Singh
Google LLC

15 January 2026

Proxying Bound UDP in HTTP
draft-ietf-masque-connect-udp-listen-11

Abstract

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

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

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

Source for this draft and an issue tracker can be found at https://github.com/ietf-wg-masque/draft-ietf-masque-connect-udp-listen.

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/.

Copyright Notice

Table of Contents

1. Introduction 3
  1.1. Conventions and Definitions 3
2. Bound UDP Proxying Mechanism 3
3. Context Identifiers 4
  3.1. The COMPRESSION_ASSIGN capsule 4
  3.2. The COMPRESSION_ACK capsule 6
  3.3. The COMPRESSION_CLOSE capsule 7
4. Uncompressed Operation 7
5. Compressed Operation 8
6. The Connect-UDP-Bind Header Field 9
7. The Proxy-Public-Address Response Header Field 9
8. Proxy Behavior 10
  8.1. Restricting IPs 10
9. Security Considerations 10
10. Operational Considerations 11
11. IANA Considerations 11
  11.1. HTTP Fields 11
  11.2. Capsules 11
12. References 12
  12.1. Normative References 12
  12.2. Informative References 13
Appendix A. Example 14
Appendix B. Comparison with CONNECT-IP 16
Acknowledgments 17
Authors' Addresses 17

1. Introduction

The mechanism to proxy UDP in HTTP [CONNECT-UDP] allows creating tunnels for communicating UDP payloads [UDP] to a fixed host and port; this enables proxying of HTTP/3 connections, since they run over UDP. Similarly, the HTTP CONNECT method (see Section 9.3.6 of [HTTP]) allows proxying HTTP/1.x and HTTP/2, which run over TCP. Combining both allows proxying the majority of a Web Browser's HTTP traffic. However WebRTC [WebRTC] relies on ICE [ICE] to provide connectivity between two Web browsers, and ICE relies on the ability to send and receive UDP packets to multiple hosts. While in theory it might be possible to accomplish this using multiple UDP proxying HTTP requests, HTTP semantics [HTTP] do not guarantee that distinct requests will be handled by the same server. This can lead to the UDP packets being sent from distinct IP addresses, thereby preventing ICE from operating correctly. Consequently, UDP proxying requests cannot enable WebRTC connectivity between peers.

This document describes an extension to UDP Proxying in HTTP that allows sending and receiving UDP payloads to multiple hosts within the scope of a single UDP proxying HTTP request.

1.1. Conventions and Definitions

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

This document uses terminology from [CONNECT-UDP] and notational conventions from [QUIC]. This document uses the terms Boolean, List, and String from Section 3 of [STRUCTURED-FIELDS] to specify syntax and parsing. This document uses Augmented Backus-Naur Form and parsing/serialization behaviors from [ABNF].

2. Bound UDP Proxying Mechanism

In unextended UDP proxying requests, the target host is encoded in the HTTP request path or query. For bound UDP proxying, the target is either conveyed in each HTTP Datagram (see Figure 4), or registered via capsules and then compressed (see Figure 1).

When performing URI Template Expansion of the UDP proxying template (see Section 3 of [CONNECT-UDP]), the client follows the same template as unextended UDP proxying and sets the "target_host" and the "target_port" variables to one of its targets. It adds the Connect-UDP-Bind header field as specified in Section 6 to request bind. If the proxy supports bound UDP proxying, it returns the Connect-UDP-Bind response header field value set to true.

When "target_host" and "target_port" are set to a valid target, the client is requesting bound UDP proxying but would accept fallback to unextended UDP proxying to that target. If the client doesn't have a specific target, or if it wants bound UDP proxying without fallback, it sets both the "target_host" and the "target_port" variables to the '*' character (ASCII character 0x2A). Note that the '*' character MUST be percent-encoded before sending, per Section 3.2.2 of [TEMPLATE].

3. Context Identifiers

As with unextended UDP proxying, the semantics of HTTP Datagrams are conveyed by Context IDs (see Section 4 of [CONNECT-UDP]). Endpoints first allocate a new Context ID (per [CONNECT-UDP], clients allocate even Context IDs while proxies allocate odd ones), and then use the COMPRESSION_ASSIGN capsule (see Section 3.1) to convey the semantics of the new Context ID to their peer. This process is known as registering the Context ID.

Each Context ID can have either compressed or uncompressed semantics. The uncompressed variant encodes the target IP and port into each HTTP Datagram. Conversely, the compressed variant exchanges the target IP and port once in the capsule during registration, and then relies on shared state to map from the Context ID to the IP and port.

Context ID 0 was reserved by unextended UDP proxying to represent UDP payloads sent to and from the "target_host" and "target_port" from the URI template. When the mechanism from this document is in use:

* if the "target_host" and "target_port" variables are set to '*', then context ID 0 MUST NOT be used in HTTP Datagrams.

* otherwise, HTTP Datagrams with context ID 0 have the same semantics as in unextended UDP proxying.

3.1. The COMPRESSION_ASSIGN capsule

The Compression Assign capsule (capsule type 0x11) is used to register the semantics of a Context ID. It has the following format:

```
COMPRESSION_ASSIGN Capsule {
  Type (i) = 0x11,
  Length (i),
  Context ID (i),
  IP Version (8),
  [IP Address (32..128)],
  [UDP Port (16)],
}
```

Figure 1: Compression Assign Capsule Format

It contains the following fields:

IP Version: The IP Version of the following IP Address field. MUST be 0, 4 or 6. Setting this to zero indicates that this capsule registers an uncompressed context. Otherwise, the capsule registers a compressed context for the IP address and UDP port it carries.

IP Address: The IP Address of this context. This field is omitted if the IP Version field is set to 0. Otherwise, it has a length of 32 bits when the corresponding IP Version field value is 4, and 128 when the IP Version is 6.

UDP Port: The UDP Port of this context, in network byte order. This field is omitted if the IP Version field is set to 0.

When an endpoint receives a COMPRESSION_ASSIGN capsule, it MUST either accept or reject the corresponding registration:

* if it accepts the registration, first the receiver MUST save the mapping from Context ID to address and port (or save the fact that this context ID is uncompressed). Second, the receiver MUST return a COMPRESSION_ACK capsule with the Context ID set to the one from the received COMPRESSION_ASSIGN capsule back to its peer, indicating it has accepted the registration.

* if it rejects the registration, the receiver MUST respond by sending a COMPRESSION_CLOSE capsule with the Context ID set to the one from the received COMPRESSION_ASSIGN capsule.

As mandated in Section 4 of [CONNECT-UDP], clients can only allocate even Context IDs, while proxies can only allocate odd ones. Since the value 0 was reserved by unextended UDP proxying, the Context ID value of COMPRESSION_ASSIGN can never be zero.

Endpoints MUST NOT send two COMPRESSION_ASSIGN capsules with the same Context ID. If a recipient detects a repeated Context ID, it MUST treat the capsule as malformed. Receipt of a malformed capsule MUST be treated as an error processing the Capsule Protocol, as defined in Section 3.3 of [HTTP-DGRAM].

If the uncompressed context is closed, the proxy MUST NOT open new compressed contexts. In such a case, the proxy opening contexts results in tuples not desired by the client reaching it thereby nullifying the IP restriction property of uncompressed compression close as described in Section 8.1.

Only one Context ID can be used per IP-port tuple. If an endpoint detects that both it and its peer have opened a Context ID for the same tuple, the endpoint MUST close the Context ID that was opened by the proxy. If an endpoint receives a COMPRESSION_ASSIGN capsule whose tuple matches another open Context ID, it MUST treat the capsule as malformed.

Endpoints MAY pre-emptively use Context IDs not yet acknowledged by the peer via COMPRESSION_ACK, knowing that those HTTP Datagrams can be dropped if they arrive before the corresponding COMPRESSION_ASSIGN capsule, or if the peer rejects the registration.

3.2. The COMPRESSION_ACK capsule

The Compression Acknowledgment capsule (capsule type 0x12) serves to confirm registration of a context ID that was received via a COMPRESSION_ASSIGN capsule.

```
COMPRESSION_ACK Capsule {
  Type (i) = 0x12,
  Length (i),
  Context ID (i),
}
```

Figure 2: Compression Acknowledgment Capsule Format

An endpoint can only send a COMPRESSION_ACK capsule if it received a COMPRESSION_ASSIGN capsule with the same Context ID. If an endpoint receives COMPRESSION_ACK capsule for a context ID it did not attempt to register via COMPRESSION_ASSIGN, that capsule is considered malformed.

3.3. The COMPRESSION_CLOSE capsule

The Compression Close capsule (capsule type 0x13) serves two purposes. It can be sent as a direct response to a received COMPRESSION_ASSIGN capsule, to indicate that the registration was rejected. It can also be sent later to indicate the closure of a previously assigned registration.

```
COMPRESSION_CLOSE Capsule {
  Type (i) = 0x13,
  Length (i),
  Context ID (i),
}
```

Figure 3: Compression Close Capsule Format

Once an endpoint has either sent or received a COMPRESSION_CLOSE for a given Context ID, it MUST NOT send any further datagrams with that Context ID. Since the value 0 was reserved by unextended UDP proxying, a COMPRESSION_CLOSE capsule with Context ID set to zero is malformed.

Endpoints MAY close any context regardless of which endpoint registered it. This is useful for example, when a mapping is unused for a long time. Another potential use is restricting some targets (see Section 8.1).

Once a registration is closed, endpoints can instead use an uncompressed Context ID to exchange UDP payloads for the given target, if such a context has been registered (see Section 4).

4. Uncompressed Operation

If the client wishes to send or receive uncompressed datagrams, it MUST first send a COMPRESSION_ASSIGN capsule (see Figure 1) to the proxy with the IP Version set to zero. This registers the Context ID as being uncompressed semantics: all HTTP Datagrams with this Context ID have the following format:

```
Uncompressed Bound UDP Proxying Payload {
  IP Version (8),
  IP Address (32..128),
  UDP Port (16),
  UDP Payload (..),
}
```

Figure 4: Uncompressed Bound UDP Proxying HTTP Datagram Format

It contains the following fields:

IP Version: The IP Version of the following IP Address field. MUST be 4 or 6.

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

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

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

A client MUST NOT open an uncompressed Context ID if one is already open. If a server receives a request to open an uncompressed Context ID and it already has one open, then the server MUST treat the second capsule as malformed. Note that it’s possible for the client to close the uncompressed context and reopen it later with a different Context ID, as long as there aren’t two uncompressed contexts open at the same time. Only the client can request uncompressed contexts. If a client receives a COMPRESSION_ASSIGN capsule with the IP Version set to 0, it MUST treat it as malformed.

5. Compressed Operation

Endpoints MAY choose to compress the IP and port information per datagram for a given target using Context IDs. This is accomplished by registering a compressed Context ID using the COMPRESSION_ASSIGN capsule (see Figure 1).

If the Context ID in an HTTP Datagram matches one previously registered for compressed operation, the rest of the HTTP Datagram represents the UDP payload:

```
Compressed Bound UDP Proxying Payload {
  UDP Payload (..),
}
```

Figure 5: Compressed Bound UDP Proxying HTTP Datagram Format

It contains the following field:

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

6. The Connect-UDP-Bind Header Field

The "Connect-UDP-Bind" header field's value is a Boolean Structured Field set to true. Clients and proxy both indicate support for this extension by sending the Connect-UDP-Bind header field with a value of ?1. Once an endpoint has both sent and received the Connect-UDP-Bind header field set to true, this extension is enabled. Any other value type MUST be handled as if the field were not present by the recipients (for example, if this field is defined multiple times, its type becomes a List and therefore is to be ignored). This document does not define any parameters for the Connect-UDP-Bind header field value, but future documents might define parameters. Receivers MUST ignore unknown parameters.

7. The Proxy-Public-Address Response Header Field

Upon accepting the request, the proxy MUST select at least one public IP address to bind. The proxy MAY assign more addresses. For each selected address, it MUST select an open port to bind to this request. From then and until the tunnel is closed, the proxy SHALL send packets received on these IP-port tuples to the client. The proxy MUST communicate the selected addresses and ports to the client using the "Proxy-Public-Address" header field. The header field is a List. Each member of the List is a String, comprised of the ip-port tuple. The format of the String is defined using IP-literal, IPv4address, and port from Section 3.2 of [URI].

```
ip-port-tuple = DQUOTE ( IP-literal / IPv4address ) ":" port DQUOTE
```

Figure 6: Proxy Address Format

When a single IP-Port tuple is provided in the Proxy-Public-Address field, the proxy MUST use the same public IP and Port for the remainder of the connection. When multiple tuples are provided, maintaining address stability per address family is RECOMMENDED.

Note that since the addresses are conveyed in HTTP response headers, a subsequent change of addresses on the proxy cannot be conveyed to the client.

If the proxy only shares IP addresses from a single address family, that indicates that the proxy only supports that family. The client SHOULD NOT attempt to register compressed contexts or send

uncompressed datagrams intended for targets whose IP address families were not indicated via the IP addresses listed in the Proxy-Public-Address header field, as the proxy will drop those datagrams and reject those registrations.

8. Proxy Behavior

After accepting the bound UDP proxying request, the proxy uses an assigned IP address and port to transmit UDP payloads received from the client to the target IP Address and UDP Port specified in each HTTP Datagram received from the client. The proxy uses the same ports to listen for UDP packets from any authorized target and forwards them to the client by encapsulating them in HTTP Datagrams, using the corresponding Context ID.

If the proxy receives UDP payloads that don’t correspond to any registration (i.e., no compression for the given target was ever established and there is no uncompressed registration), the proxy will either drop the datagram or temporarily buffer it (see Section 5 of [CONNECT-UDP]).

8.1. Restricting IPs

If a client does not wish to receive datagrams from unknown senders, it can close the uncompressed registration (or not open it in the first place). In that scenario, the proxy effectively acts as a firewall against unwanted or unknown IPs.

9. Security Considerations

The security considerations described in Section 7 of [CONNECT-UDP] also apply here. Since TURN can be run over this mechanism, implementors will benefit from reviewing the security considerations in Section 21 of [TURN].

Since unextended UDP proxying requests carry the target as part of the request, the proxy can protect unauthorized targets by rejecting requests before creating the tunnel, and communicate the rejection reason in response header fields. The uncompressed context allows transporting datagrams to and from any target. Clients that keep the uncompressed context open need to be able to receive from all targets. If the UDP proxy would reject unextended UDP proxying requests to some targets (as recommended in Section 7 of [CONNECT-UDP]), then for bound UDP proxying requests where the uncompressed context is open, the UDP proxy needs to perform checks on the target of each uncompressed context datagram it receives.

Note that if the compression response (COMPRESSION_ACK OR COMPRESSION_CLOSE) cannot be immediately sent due to flow or congestion control, an upper limit on how many compression responses the endpoint is willing to buffer MUST be set to prevent memory exhaustion. The proxy MUST abort the request stream if this limit is reached.

10. Operational Considerations

When moving traffic between uncompressed and compressed contexts, the effective MTU will change. This can hinder Datagram Packetization Layer PMTU Discovery (DPLPMTUD) between the client and the target [DPLPMTUD]. To avoid that, if an endpoint intends to use compression, it SHOULD request it as early as possible.

11. IANA Considerations

11.1. HTTP Fields

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

Field Name	Structured Type
Connect-UDP-Bind	Item
Proxy-Public-Address	List

Table 1: New Fields

All of these new entries use the following values for these fields:

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

11.2. Capsules

This document will request IANA to register the following new items to the "HTTP Capsule Types" registry maintained at <https://www.iana.org/assignments/masque>:

Value	Capsule Type
0x11	COMPRESSION_ASSIGN
0x12	COMPRESSION_ACK
0x13	COMPRESSION_CLOSE

Table 2: New Capsules

All of these new entries use the following values for these fields:

Status: provisional (permanent if this document is approved)
Reference: This document
Change Controller: IETF
Contact: MASQUE Working Group masque@ietf.org
(mailto:masque@ietf.org)
Notes: None

12. References

12.1. Normative References

[ABNF] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/rfc/rfc5234>.

[CONNECT-UDP] Schinazi, D., "Proxying UDP in HTTP", RFC 9298, DOI 10.17487/RFC9298, August 2022, <https://www.rfc-editor.org/rfc/rfc9298>.

[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/rfc/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/rfc/rfc9297>.

[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>.

[STRUCTURED-FIELDS] Nottingham, M. and P. Kamp, "Structured Field Values for HTTP", RFC 9651, DOI 10.17487/RFC9651, September 2024, <https://www.rfc-editor.org/rfc/rfc9651>.

[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>.

[URI] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/rfc/rfc3986>.

12.2. Informative References

[CONNECT-IP] Pauly, T., Ed., Schinazi, D., Chernyakhovsky, A., Khlewind, M., and M. Westerlund, "Proxying IP in HTTP", RFC 9484, DOI 10.17487/RFC9484, October 2023, <https://www.rfc-editor.org/rfc/rfc9484>.

[DPLPMTUD] Fairhurst, G., Jones, T., Txen, M., Rngeler, I., and T. Vlker, "Packetization Layer Path MTU Discovery for Datagram Transports", RFC 8899, DOI 10.17487/RFC8899, September 2020, <https://www.rfc-editor.org/rfc/rfc8899>.

[ICE] Keranen, A., Holmberg, C., and J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal", RFC 8445, DOI 10.17487/RFC8445, July 2018, <https://www.rfc-editor.org/rfc/rfc8445>.

[TURN] Reddy, T., Ed., Johnston, A., Ed., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 8656, DOI 10.17487/RFC8656, February 2020, <https://www.rfc-editor.org/rfc/rfc8656>.

[WebRTC] "WebRTC", W3C Recommendation, 26 January 2021, <https://www.w3.org/TR/webrtc/>.

Appendix A. Example

In the example below, the client is configured with URI Template "https://example.org/.well-known/masque/udp/{target_host}/{target_port}/" and listens for traffic on the proxy, eventually decides that it no longer wants to listen for connections from new targets, and limits its communication with only 203.0.113.11:4321 and no other UDP target.

```
Client                                                  Server

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

           <--------  STREAM(44): HEADERS
                        :status = 200
                        connect-udp-bind = ?1
                        capsule-protocol = ?1
                        proxy-public-address = "192.0.2.45:54321",  \
                                               "[2001:db8::1234]:54321"

// Register Context ID 2 to be used for uncompressed UDP payloads
// to/from any target.

CAPSULE                        -------->
  Type = COMPRESSION_ASSIGN
  Context ID = 2
```

```
     IP Version = 0

  // Proxy confirms registration.

             <-------- CAPSULE
                         Type = COMPRESSION_ACK
                         Context ID = 2

  // Target talks to Client using the uncompressed context.

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

  // Client responds on the same uncompressed context.

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

  // Another target talks to Client using the uncompressed context.

             <--------  DATAGRAM
                          Quarter Stream ID = 11
                          Context ID = 2
                          IP Version = 4
                          IP Address = 203.0.113.11
                          UDP Port = 4321
                          UDP Payload = Encapsulated UDP Payload

  // Client responds on the same uncompressed context.

   DATAGRAM                         -------->
     Quarter Stream ID = 11
     Context ID = 2
     IP Version = 4
     IP Address = 203.0.113.11
     UDP Port = 4321
     UDP Payload = Encapsulated UDP Payload
```

```
  // Register 203.0.113.11:4321 to compress it in the future.

   CAPSULE                         -------->
     Type = COMPRESSION_ASSIGN
     Context ID = 4
     IP Version = 4
     IP Address = 203.0.113.11
     UDP Port = 4321


  // Proxy confirms registration.

              <-------- CAPSULE
                          Type = COMPRESSION_ACK
                          Context ID = 4

  // Omit IP and Port for future packets intended for
  // 203.0.113.11:4321 hereon.

   DATAGRAM                         -------->
     Context ID = 4
     UDP Payload = Encapsulated UDP Payload

              <--------  DATAGRAM
                          Context ID = 4
                          UDP Payload = Encapsulated UDP Payload

  // Request packets without a corresponding compressed Context
  // to be dropped by closing the uncompressed Context.

   CAPSULE                         -------->
     Type = COMPRESSION_CLOSE
     Context ID = 2

  // Context ID 4 = 203.0.113.11:4321 traffic is accepted,
  // and other traffic is dropped at the proxy.
```

Appendix B. Comparison with CONNECT-IP

While the use-cases described in Section 1 could be supported using IP proxying in HTTP [CONNECT-IP], it would require that every HTTP Datagram carries a complete IP header. This would lead to both inefficiencies in the wire encoding and reduction in available Maximum Transmission Unit (MTU). Furthermore, Web browsers would need to support IPv4 and IPv6 header generation, parsing, validation and error handling.

Acknowledgments

This proposal is the result of many conversations with MASQUE working group participants. In particular, the authors would like to thank Alejandro Sedeo, Ben Schwartz, Lucas Pardue, Magnus Westerlund, Marius Kleidl, Mark Nottingham, Marten Seemann, and Tommy Pauly for their reviews.

Authors' Addresses

David Schinazi
Google LLC
Email: dschinazi.ietf@gmail.com

Abhi Singh
Google LLC
Email: abhisinghietf@gmail.com