

Multiplexed Application Substrate over QUIC Encryption  
Internet-Draft  
Intended status: Standards Track  
Expires: 14 October 2026

A. R. Sedo  
Google LLC  
12 April 2026

Proxying Ethernet in HTTP  
draft-ietf-masque-connect-ethernet-09

## Abstract

This document describes how to proxy Ethernet frames in HTTP. This protocol is similar to IP proxying in HTTP, but for Layer 2 instead of Layer 3. More specifically, this document defines a protocol that allows an HTTP client to create a tunnel to exchange Layer 2 Ethernet frames through an HTTP server with an attached physical or virtual Ethernet segment.

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

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-ethernet>.

## 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 14 October 2026.

## Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

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

## Table of Contents

1. Introduction . . . . .	3
2. Conventions and Definitions . . . . .	3
3. Configuration of Clients . . . . .	4
4. Tunnelling Ethernet over HTTP . . . . .	5
4.1. Ethernet Proxy Handling . . . . .	5
4.2. HTTP/1.1 Request . . . . .	6
4.3. HTTP/1.1 Response . . . . .	7
4.4. HTTP/2 and HTTP/3 Requests . . . . .	7
4.5. HTTP/2 and HTTP/3 Responses . . . . .	8
5. Context Identifiers . . . . .	9
6. HTTP Datagram Payload Format . . . . .	9
7. Ethernet Frame Handling . . . . .	10
8. Examples . . . . .	11
8.1. Remote Access L2VPN . . . . .	11
8.2. Site-to-Site L2VPN . . . . .	12
9. Performance Considerations . . . . .	13
9.1. MTU and Frame Ordering Considerations . . . . .	14
9.2. IEEE 802.1Q tagging . . . . .	14
10. Security Considerations . . . . .	15
11. IANA Considerations . . . . .	15
11.1. HTTP Upgrade Token . . . . .	16
11.2. Updates to the MASQUE URI Suffixes Registry . . . . .	16
12. References . . . . .	16
12.1. Normative References . . . . .	16
12.2. Informative References . . . . .	18

Acknowledgments . . . . .	19
Author's Address . . . . .	19

## 1. Introduction

HTTP provides the CONNECT method (see Section 9.3.6 of [HTTP]) for creating a TCP [TCP] tunnel to a destination, a similar mechanism for UDP [CONNECT-UDP], and an additional mechanism for IP [CONNECT-IP]. However, these mechanisms can't carry Layer 2 frames without further encapsulation inside of IP, for instance with EtherIP [ETHERIP] or L2TP [L2TP] [L2TPv3], which consume additional header bytes, reducing the available MTU.

This document describes a protocol for exchanging IEEE 802.3 [IEEE802.3] Ethernet frames with an HTTP server. Either participant in the HTTP connection can then relay Ethernet frames to and from a local or virtual interface. This can be used by a node to support remote bridging of two Ethernet broadcast domains to establish a Layer 2 VPN. This can simplify connectivity to network-connected appliances that are configured to only interact with peers connected to the same Ethernet broadcast domain.

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].

This protocol necessarily incurs additional encapsulation overhead. When possible, users should use higher-level proxying protocols, such as connect-ip or connect-udp.

## 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 "Ethernet proxy" to refer to the HTTP server that responds to the Ethernet proxying request. The term "client" is used in the HTTP sense; the client constructs the Ethernet proxying request. If there are HTTP intermediaries (as defined in Section 3.7 of [HTTP]) between the client and the Ethernet proxy, those are referred to as "intermediaries" in this document. The term "Ethernet proxying endpoints" refers to both the client and the Ethernet proxy.

This document uses terminology from [QUIC]. Where this document defines protocol types, the definition format uses the notation from Section 1.3 of [QUIC]. This specification uses the variable-length integer encoding from Section 16 of [QUIC]. Variable-length integer values do not need to be encoded in the minimum number of bytes necessary.

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.

### 3. Configuration of Clients

Clients are configured to use Ethernet proxying over HTTP via a URI Template [TEMPLATE]. The URI Templates used by this protocol do not require any variables; implementations or extensions MAY specify their own. URI Templates specified for this protocol MAY use the well-known location [WELL-KNOWN] registered by this document.

Examples are shown below:

```
https://example.org/.well-known/masque/ethernet/  
https://proxy.example.org:4443/masque/ethernet/  
https://masque.example.org/?user=bob
```

An implementation that supports connecting to multiple Ethernet segments might add a "vlan-identifier" variable to specify which segment to connect to. The optionality of variables needs to be considered when defining the template so that variables are either self-identifying or possible to exclude in the syntax. How valid values for such variables are communicated to the client is not a part of this protocol.

Hypothetical examples are shown below:

```
https://proxy.example.org:4443/masque/ethernet?vlan={vlan-identifier}  
https://etherproxy.example.org/{vlan-identifier}
```

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 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 Ethernet proxy.

#### 4. Tunnelling Ethernet over HTTP

To allow negotiation of a tunnel for Ethernet over HTTP, this document defines the "connect-ethernet" HTTP upgrade token. The resulting Ethernet tunnels use the Capsule Protocol (see Section 3.2 of [HTTP-DGRAM]) with HTTP Datagrams in the format defined in Section 6.

To initiate an Ethernet tunnel associated with a single HTTP stream, a client issues a request containing the "connect-ethernet" upgrade token.

By virtue of the definition of the Capsule Protocol (see Section 3.2 of [HTTP-DGRAM]), Ethernet proxying requests do not carry any message content. Similarly, successful Ethernet proxying responses also do not carry any message content.

Ethernet proxying over HTTP MUST be operated over TLS or QUIC encryption, or another equivalent encryption protocol, to provide confidentiality, integrity, and authentication.

##### 4.1. Ethernet Proxy Handling

Upon receiving an Ethernet 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 an Ethernet proxy. The Ethernet proxy can choose to reject the Ethernet proxying request or establish an Ethernet tunnel.

The lifetime of the Ethernet tunnel is tied to the Ethernet proxying request stream.

A successful response (as defined in Sections 4.3 and 4.5) indicates that the Ethernet proxy has established an Ethernet tunnel and is willing to proxy Ethernet frames. Any response other than a successful response indicates that the request has failed; thus, the client MUST abort the request.

#### 4.2. HTTP/1.1 Request

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

- \* The method SHALL be "GET".
- \* The request SHALL include a single Host header field containing the host and optional port of the Ethernet 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-ethernet".

An Ethernet 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 the URI Template "https://example.org/.well-known/masque/ethernet/" and wishes to open an Ethernet tunnel, it could send the following request.

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

Figure 1: Example HTTP/1.1 Request

#### 4.3. HTTP/1.1 Response

The server indicates success by replying with a response that conforms to 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-ethernet".
- \* 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 close the connection.

For example, the server could respond with:

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

Figure 2: Example HTTP/1.1 Response

#### 4.4. HTTP/2 and HTTP/3 Requests

When using HTTP/2 [HTTP/2] or HTTP/3 [HTTP/3], Ethernet 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-ethernet".
- \* The :authority pseudo-header field SHALL contain the authority of the IP proxy.

- \* The :path and :scheme pseudo-header fields SHALL NOT be empty. Their values SHALL contain the scheme and path from the configured URI Template; see Section 3.

An Ethernet 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 the URI Template "https://example.org/.well-known/masque/ethernet/" and wishes to open an Ethernet tunnel, it could send the following request.

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

Figure 3: Example HTTP/2 or HTTP/3 Request

#### 4.5. HTTP/2 and HTTP/3 Responses

The server indicates success by replying with a response that conforms to 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. As an example, any status code in the 3xx range will be treated as a failure and cause the client to abort the request.

For example, the server could respond with:

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

Figure 4: Example HTTP/2 or HTTP/3 Response



## 5. Context Identifiers

The mechanism for proxying Ethernet in HTTP defined in this document allows future extensions to exchange HTTP Datagrams that have different semantics, similar to the extension mechanisms specified in Section 5 of [CONNECT-IP]. Some of these extensions could augment Ethernet payloads with additional data or compress Ethernet frame header fields. To provide this extension point, all HTTP Datagrams associated with Ethernet proxying request streams start with a Context ID field; see Section 6.

Context IDs are 62-bit integers ( $0-2^{62}-1$ ). Context IDs are encoded as variable-length integers; see Section 16 of [QUIC]. The Context ID value of 0 is reserved for Ethernet 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 request 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 either the client or the Ethernet 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.

## 6. HTTP Datagram Payload Format

When associated with Ethernet proxying request streams, the HTTP Datagram Payload field of HTTP Datagrams (see [HTTP-DGRAM]) has the format defined in Figure 5. 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.

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

Figure 5: Ethernet Proxying HTTP Datagram Format

The Ethernet Proxying HTTP Datagram Payload contains the following fields:

**Context ID:** A variable-length integer that contains the value of the Context ID. If an HTTP/3 datagram which carries an unknown Context ID is received, the receiver SHALL either drop that datagram silently or buffer it temporarily (on the order of a round trip) while awaiting the registration of the corresponding Context ID.

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

Ethernet frames are encoded using HTTP Datagrams with the Context ID set to zero. When the Context ID is set to zero, the Payload field contains a full Layer 2 Ethernet Frame (from the MAC destination field until the last byte of the Frame check sequence field), as defined by IEEE 802.3 [IEEE802.3]. A complete frame could include include an IEEE 802.1Q [IEEE802.1Q] tag (see Section 9.2).

If an Ethernet 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.

## 7. Ethernet Frame Handling

This document defines a tunnelling mechanism that is conceptually an Ethernet link. An Ethernet proxying connection established between two Ethernet proxying endpoints emulates a single Ethernet link between those two endpoints. This provides an Ethernet MAC service that will deliver each Ethernet frame that is received at the ingress to the egress at the other end of the tunnel.

Endpoints implementing this mechanism might need to handle some of the responsibilities of an Ethernet switch or bridge if they do not delegate them to another component of the endpoint such as a kernel. Those responsibilities are beyond the scope of this document, and include, but are not limited to, the handling of broadcast packets and multicast groups, or the local termination of PAUSE frames.

If an Ethernet proxying endpoint fails to deliver a frame to an underlying Ethernet segment, the endpoint **MUST** drop the frame.

## 8. Examples

Ethernet proxying in HTTP enables the bridging of Ethernet broadcast domains. These examples are provided to help illustrate some of the ways in which Ethernet proxying can be used.

### 8.1. Remote Access L2VPN

The following example shows a point to point VPN setup where a client appears to be connected to a remote Layer 2 network.

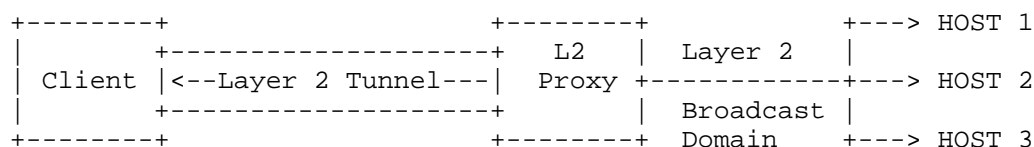


Figure 6: L2VPN Tunnel Setup

In this case, the client connects to the Ethernet proxy and immediately can start sending Ethernet frames to the attached broadcast domain.

```

[[ From Client ]]                [[ From Ethernet Proxy ]]

SETTINGS
  H3_DATAGRAM = 1

                                SETTINGS
                                ENABLE_CONNECT_PROTOCOL = 1
                                H3_DATAGRAM = 1

STREAM(44): HEADERS
:method = CONNECT
:protocol = connect-ethernet
:scheme = https
:path = /.well-known/masque/ethernet/
:authority = proxy.example.com
capsule-protocol = ?1

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

DATAGRAM
Quarter Stream ID = 11
Context ID = 0
Payload = Encapsulated Ethernet Frame

                                DATAGRAM
                                Quarter Stream ID = 11
                                Context ID = 0
                                Payload = Encapsulated Ethernet Frame

```

Figure 7: VPN Full-Tunnel Example

## 8.2. Site-to-Site L2VPN

The following example shows a site-to-site VPN setup where a client joins a locally attached broadcast domain to a remote broadcast domain through the Proxy.

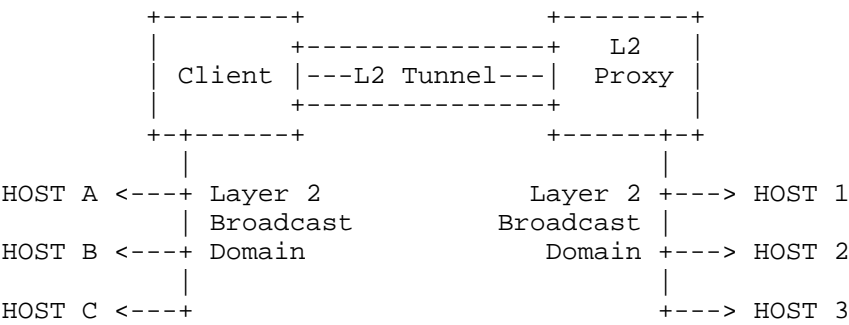


Figure 8: Site-to-site L2VPN Example

In this case, the client connects to the Ethernet proxy and immediately can start relaying Ethernet frames from its attached broadcast domain to the proxy. The difference between this example and Section 8.1 is limited to what the Client is doing with the the tunnel; the exchange between the Client and the Proxy is the same as in Figure 7 above.

9. Performance Considerations

When the protocol running inside the tunnel uses congestion control (e.g., [TCP] or [QUIC]), the proxied traffic will incur at least two nested congestion controllers. Implementers will benefit from reading the guidance in Section 3.1.11 of [UDP-USAGE]. By default the tunneling of Ethernet frames MUST NOT assume that the carried Ethernet frames contain congestion controlled traffic. Optimizations for traffic flows carried within the Ethernet Frames MAY be done in cases where the content of the Ethernet Frames have been identified to be congestion controlled traffic.

Some implementations might find it beneficial to maintain a small buffer of frames to be sent through the tunnel to smooth out short term variations and bursts in tunnel capacity. As such a buffer is limited, Ethernet frames can get dropped when the buffer limit is exceeded.

When the protocol running inside the tunnel uses loss recovery (e.g., [TCP] or [QUIC]) and the outer 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, Ethernet proxying SHOULD be performed over HTTP/3 to allow leveraging the QUIC DATAGRAM frame.

### 9.1. MTU and Frame Ordering Considerations

When using HTTP/3 with the QUIC Datagram extension [QUIC-DGRAM], Ethernet frames can be transmitted in QUIC DATAGRAM frames. Since DATAGRAM frames cannot be fragmented, they can only carry Ethernet frames up to a given length determined by the QUIC connection configuration and the Path MTU (PMTU). Implementations MAY rely on [QUIC]'s use of [DPLPMTUD] to probe and discover the PMTU over the connection's lifetime, and adjust any associated interface MTU as needed. Furthermore, the UDP packets carrying these frames could be reordered by the network.

When using HTTP/1.1 or HTTP/2, and when using HTTP/3 without the QUIC Datagram extension [QUIC-DGRAM], Ethernet frames are transmitted in DATAGRAM capsules as defined in [HTTP-DGRAM]. DATAGRAM capsules are transmitted reliably over an underlying stream, maintaining frame order, though they could be split across multiple QUIC or TCP packets.

The trade-off between supporting a larger MTU and avoiding fragmentation should be considered when deciding what mode(s) to operate in. Implementations SHOULD NOT intentionally reorder Ethernet frames, but are not required to provide guaranteed in-order delivery. If in-order delivery of Ethernet frames is required, DATAGRAM capsules can be used.

### 9.2. IEEE 802.1Q tagging

When the proxy transports Ethernet frames that carry an IEEE 802.1Q [IEEE802.1Q] VLAN tag, these are by default transparently forwarded through the tunnel. When the tunnel ingress and/or egress interprets the tags, there must be agreement (signaled or manually configured) on how to consistently process each tag at the ingress and the egress. The procedure for this signalling/configuration is not defined in this document.

A proxy that is used to access to multiple VLANs MAY map each individual VLAN to a distinct URI, such that each Ethernet proxying request is associated with only one VLAN. This provides flexibility in forwarding, while meeting the requirements for the relative priority and ordering between frames associated with a VLAN. To reduce overhead, the IEEE 802.1Q field could be stripped and, when required, could be reapplied at the egress associating the frame with the appropriate priority and VLAN.

## 10. Security Considerations

There are risks in allowing arbitrary clients to establish a tunnel to a Layer 2 network. Bad actors could abuse this capability to attack hosts on that network that they would otherwise be unable to reach. HTTP servers that support Ethernet proxying SHOULD restrict its use to authenticated users. Depending on the deployment, possible authentication mechanisms include mutual TLS between IP proxying endpoints, HTTP-based authentication via the HTTP Authorization header field [HTTP], or even bearer tokens. Proxies can enforce policies for authenticated users to further constrain client behavior or deal with possible abuse. For example, proxies can rate limit individual clients that send an excessively large amount of traffic through the proxy.

Users of this protocol may send arbitrary Ethernet frames through the tunnel, including frames with arbitrary source MAC addresses. This could allow impersonation of other hosts, poisoning of ARP [RFC826], NDP [RFC4861] and CAM (Content Addressable Memory) tables, and cause a denial of service to other hosts on the network. These are the same attacks available to an arbitrary client with physical access to the network. An implementation that is intended for point-to-site connections might limit clients to a single source MAC address, or Ethernet proxying endpoints might be configured to limit forwarding to pre-configured MAC addresses, or clients could be authenticated by [IEEE802.1X] Port Based Network Access Control, though such filtering is outside the scope of this protocol. Dynamic signalling or negotiation of MAC address filtering is left to future extensions.

This protocol is agnostic to where on the Ethernet segment a gateway for higher-level routing might be located. A client may connect via an Ethernet proxy and discover an existing gateway on the Ethernet segment, supply a new gateway to the Ethernet segment, both, or neither.

Opportunistic sending of Ethernet frames is not allowed in HTTP/1.x because a server could reject the HTTP Upgrade and attempt to parse the Ethernet frames as a subsequent HTTP request, allowing request smuggling attacks; see [OPTIMISTIC]. In particular, an intermediary that re-encodes a request from HTTP/2 or 3 to HTTP/1.1 MUST NOT forward any received capsules until it has parsed a successful Ethernet proxying response.

## 11. IANA Considerations

### 11.1. HTTP Upgrade Token

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

Value: connect-ethernet  
 Description: Proxying of Ethernet Payloads  
 Expected Version Tokens: None  
 References: This document

### 11.2. Updates to the MASQUE URI Suffixes Registry

This document will request IANA to register "ethernet" in the MASQUE URI Suffixes Registry maintained at <https://www.iana.org/assignments/masque>, created by Section 12.2 of [CONNECT-IP].

Path Segment	Description	Reference
ethernet	Ethernet Proxying	This Document

Table 1: New MASQUE URI Suffixes

## 12. References

### 12.1. Normative References

#### [CONNECT-IP]

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

#### [CONNECT-UDP]

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

[DPLPMTUD] Fairhurst, G., Jones, T., Tennen, M., Rengeler, I., and T. Vahler, "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", RFC 9220, DOI 10.17487/RFC9220, June 2022, <<https://www.rfc-editor.org/rfc/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/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>>.

## [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/rfc/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/rfc/rfc9113>>.

## [HTTP/3]

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

## [IEEE802.1Q]

"IEEE Standard for Local and Metropolitan Area Networks--Bridges and Bridged Networks", IEEE, DOI 10.1109/ieeestd.2022.10004498, ISBN ["9781504491884"], December 2022, <<https://doi.org/10.1109/ieeestd.2022.10004498>>.

## [IEEE802.3]

"IEEE Standard for Ethernet", IEEE, DOI 10.1109/ieeestd.2022.9844436, ISBN ["9781504487252"], July 2022, <<https://doi.org/10.1109/ieeestd.2022.9844436>>.

## [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>>.

## [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/rfc/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/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>>.

[TCP] Eddy, W., Ed., "Transmission Control Protocol (TCP)", STD 7, RFC 9293, DOI 10.17487/RFC9293, August 2022, <<https://www.rfc-editor.org/rfc/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/rfc/rfc6570>>.

[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>>.

## [WELL-KNOWN]

Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <<https://www.rfc-editor.org/rfc/rfc8615>>.

## 12.2. Informative References

[ETHERIP] Housley, R. and S. Hollenbeck, "EtherIP: Tunneling Ethernet Frames in IP Datagrams", RFC 3378, DOI 10.17487/RFC3378, September 2002, <<https://www.rfc-editor.org/rfc/rfc3378>>.

## [IEEE802.1X]

"IEEE Standard for Local and Metropolitan Area Networks--Port-Based Network Access Control", IEEE, DOI 10.1109/ieeestd.2020.9018454, ISBN ["9781504464406"], February 2020, <<https://doi.org/10.1109/ieeestd.2020.9018454>>.

- [L2TP] Townsley, W., Valencia, A., Rubens, A., Pall, G., Zorn, G., and B. Palter, "Layer Two Tunneling Protocol "L2TP"", RFC 2661, DOI 10.17487/RFC2661, August 1999, <<https://www.rfc-editor.org/rfc/rfc2661>>.
- [L2TPv3] Lau, J., Ed., Townsley, M., Ed., and I. Goyret, Ed., "Layer Two Tunneling Protocol - Version 3 (L2TPv3)", RFC 3931, DOI 10.17487/RFC3931, March 2005, <<https://www.rfc-editor.org/rfc/rfc3931>>.
- [OPTIMISTIC] Schwartz, B. M., "Security Considerations for Optimistic Protocol Transitions in HTTP/1.1", Work in Progress, Internet-Draft, draft-ietf-httpbis-optimistic-upgrade-06, 18 September 2025, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-optimistic-upgrade-06>>.
- [RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, DOI 10.17487/RFC4861, September 2007, <<https://www.rfc-editor.org/rfc/rfc4861>>.
- [RFC826] Plummer, D., "An Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware", STD 37, RFC 826, DOI 10.17487/RFC0826, November 1982, <<https://www.rfc-editor.org/rfc/rfc826>>.
- [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/rfc/rfc8085>>.

## Acknowledgments

Much of the initial version of this draft borrows heavily from [CONNECT-IP].

The author would like to thank Alexander Chernyakhovsky and David Schinazi for their advice while preparing this document, and Etienne Dechamps for useful discussion on the subject material. Additionally, Mirja K端hlewind, Magnus Westerlund, Martin Thompson, and Gorrry Fairhurst provided valuable feedback on the document.

## Author's Address

Alejandro R Sede単o  
Google LLC

Email: [asedeno@google.com](mailto:asedeno@google.com)