

httpbis
Internet-Draft
Intended status: Informational
Expires: April 21, 2019

L. Pardue
October 18, 2018

HTTP-initiated Network Tunnelling (HiNT)
draft-pardue-httpbis-http-network-tunnelling-01

Abstract

The HTTP CONNECT method allows an HTTP client to initiate, via a proxy, a TCP-based tunnel to a single destination origin. This memo explores options for expanding HTTP-initiated Network Tunnelling (HiNT) to cater for diverse UDP and IP associations.

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 April 21, 2019.

Copyright Notice

Copyright (c) 2018 IETFTrust and the persons identified as the document authors. All rights reserved.

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

Internet-Draft HTTP-initiated Network Tunnelling (HiNT) October 2018

Table of Contents

1.	Introduction	3
2.	Notational Conventions	6
2.1.	Definitions	6
3.	Design Consideration Aspects	7
3.1.	HTTP Version	7
3.2.	HTTP Forward Proxying	7
3.3.	Message Destination Agility	7
3.4.	Path MTU Discovery	7
3.5.	Blind forwarding vs. in-the-loop Processing	8
3.6.	Head-of-line Blocking	8
4.	Candidate Solutions	9
4.1.	CONNECT Method Augmentation	9
4.2.	UDPASSOCIATE with HINT Frames for HTTP/2 and HTTP/QUIC	9
4.3.	HELIUM over WebSockets for all HTTP Versions	9
4.4.	HELIUM over WebSockets for HTTP/1.1, Native Framing for HTTP/2 or HTTP/QUIC	9
5.	Technical Specification for HiNT Requests	10
5.1.	The UDPASSOCIATE Method for HTTP/1.1x	10
5.2.	The UDPASSOCIATE Method for HTTP/2 and HTTP/QUIC	11
5.3.	The IPASSOCIATE Method	12
6.	Technical Specification for HiNT Message Transfer	12
6.1.	HiNT Message Framing	12
6.1.1.	The HINT HTTP/2 Frame	13
6.1.2.	The HINT HTTP/QUIC Frame	14
6.2.	Light HIP HTTP/2 Framing	14
6.3.	Full HIP HTTP/2 Framing	15
6.3.1.	The OHIP HTTP/2 Frame	16
6.3.2.	The IHIP HTTP/2 Frame	17
6.3.3.	The MHIP HTTP/2 Frame	18
7.	Security Considerations	20
8.	IANA Considerations	20
8.1.	UDPASSOCIATE Method Registration	20
8.2.	IPASSOCIATE Method Registration	21
8.3.	The HINT HTTP/2 Frame Type	21
8.4.	The HINT HTTP/QUIC Frame Type	21
8.5.	The HIP HTTP/2 Frame Type	22
8.6.	The OHIP HTTP/2 Frame Type	22
8.7.	The IHIP HTTP/2 Frame Type	22
8.8.	The MHIP HTTP/2 Frame Type	22
9.	References	23
9.1.	Normative References	23

9.2.	Informative References	23
Appendix A.	Acknowledgments	24
Appendix B.	HiNT Request Options	25
Appendix C.	HiNT Message Transfer Options	26
Appendix D.	Changelog	28

Internet-Draft HTTP-initiated Network Tunnelling (HiNT) October 2018

D.1.	Since draft-pardue-httpbis-http-network-tunnelling-00 . .	29
Author's Address		29

[1.](#) Introduction

A wide range of network tunnelling solutions already exist (e.g. SOCKS [[RFC1928](#)], TURN [[RFC5766](#)] etc.), with various applicability. So why consider creating another one? Several tunnelling specifications reserve well known TCP or UDP ports that are easy to block. Even if port usage is more agile, plain text communications allow potential attackers to easily analyse traffic and cause interference.

This document we consider options for HTTP-initiated Network Tunnelling (HiNT) as a solution. The use case is a client behind a forward proxy but other uses may be supported. Using HTTP as a substrate for other protocols follows a trend seen elsewhere (DNS Queries over HTTPS [[DOH](#)]). Shifting to an HTTP port, makes port blocking less effective. However, the real advantage comes from securing HTTP (TLS [[RFC5246](#)], QUIC [[QUIC-TRANSPORT](#)]) to provide confidentiality, integrity and authenticity, which makes analysis and interference harder. This also enables secure communication to a remote proxy on the Internet (in contrast to SOCKS etc.).

A HiNT session is initiated by some HTTP mechanism. This could be a HTTP request or some binary frame format (HTTP/2 and HTTP/QUIC only).

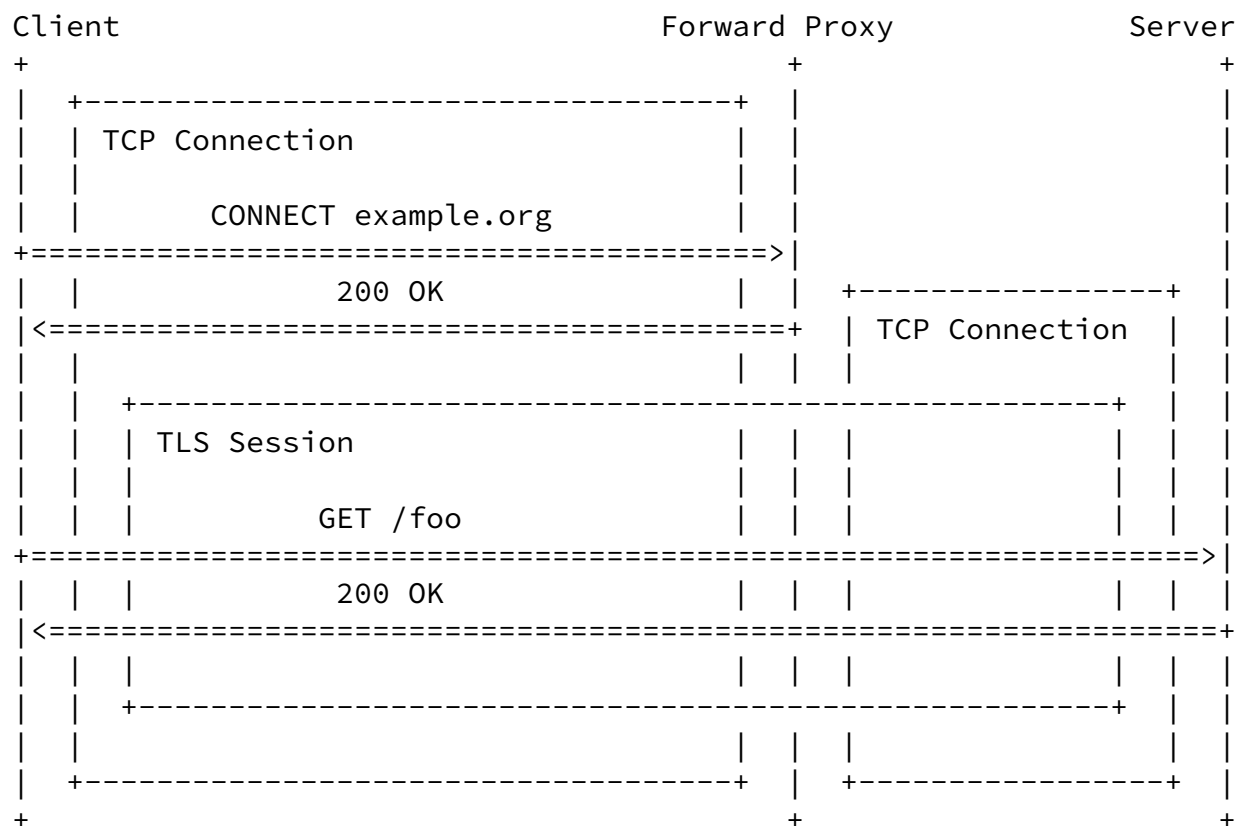
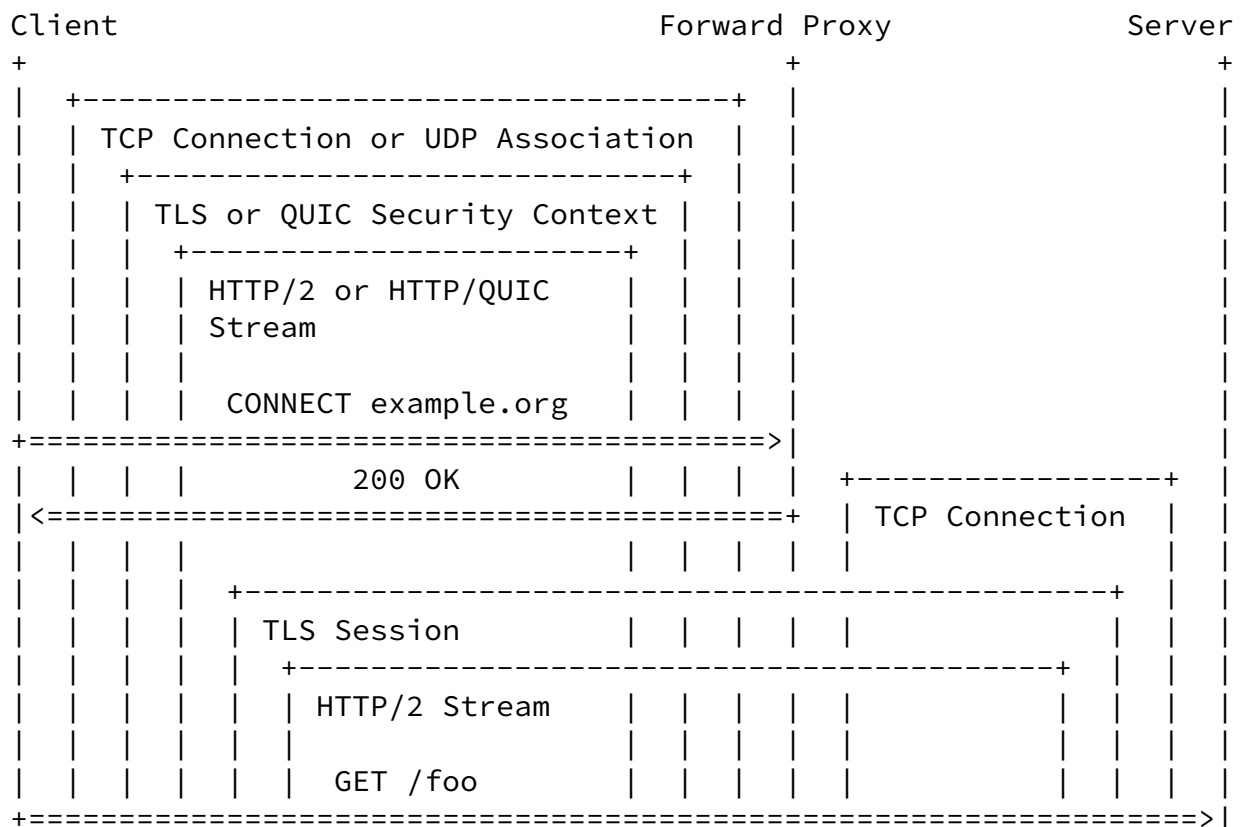
Internet-Draft HTTP-initiated Network Tunneling (HiNT) October 2018

Figure 1: HTTP/1.1 CONNECT-based TLS tunnel

The CONNECT request method (see [Section 4.3.6 of \[RFC7231\]](#)) is commonly used to establish a tunnelled TLS session with an origin identified by a request-target. In HTTP/1.1, the entire client-to-proxy HTTP connection is converted into a tunnel (Figure 1). In

HTTP/2 (see [Section 8.3 of \[RFC7540\]](#)) and HTTP/QUIC (see Section 3.1.2 of [\[QUIC-HTTP\]](#)), a single stream gets dedicated to a tunnel (Figure 2).



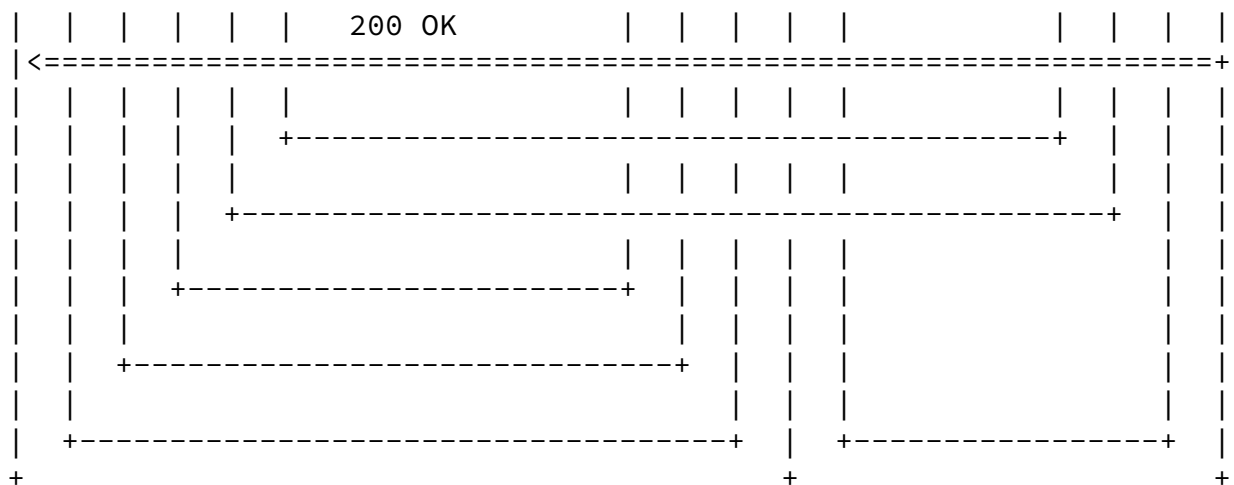


Figure 2: HTTP/2 and HTTP/QUIC CONNECT-based TLS tunnel

A proxy that supports CONNECT blindly forwards packets, in both directions, using TCP for both client-to-proxy and proxy-to-origin hops. The use of TCP for the latter hop is a limiting factor: other application or transport protocols are unsupported. This document specifically concerns itself with finding a solution that permits a UDP-based HTTP/QUIC client behind an HTTP proxy to establish an HTTP/QUIC session with the origin. Without such a capability, there continues to be a dependency on origins to support TCP-based HTTP (for a small subset of the client population).

The document is arranged in the following order:

- o Design aspects are considered in [Section 3](#).
- o Tunnel initiation options are surveyed in [Appendix B](#).
- o Messaging (post-handshake data transfer) options are surveyed in [Appendix C](#).
- o Four candidate solutions are presented in [Section 4](#), based on the above options.

Candidate solutions have the purpose of stimulating discussion in the community in order to drive toward a single solution.

[2.](#) Notational Conventions

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

[2.1.](#) Definitions

Definitions of terms that are used in this document:

- o HiNT request: a message that requests the establishment of a network tunnel to a HiNT destination.
- o HiNT response: a message that confirms the establishment of a network tunnel.
- o HiNT message: a message that allows data transfer between client, proxy and/or destination during a HiNT session.
- o HiNT client: an HTTP endpoint that sends a HiNT request to a HiNT proxy. Also referred to as a client.
- o HiNT proxy: an HTTP endpoint that services HiNT requests. It returns a HiNT response that indicates the outcome of network tunnel creation. Also referred to as a proxy.
- o HiNT destination: the service that the HiNT client is trying to reach via a HiNT proxy. Also referred to as a destination.
- o HiNT session: a specific instance of a network tunnel.

- o Network Tunnel: describes any forms of association between client and destination (end-to-end). A tunnel ceases to exist when both ends of the association are closed (implicitly or explicitly).

[3.](#) Design Consideration Aspects

[3.1.](#) HTTP Version

The design should consider if all HTTP Versions need be supported. Differences in version syntax (in particular binary framing and streams) may provide certain design advantages.

[3.2.](#) HTTP Forward Proxying

The design considers the "forward proxying" intermediary (see [Section 2.3 of \[RFC7230\]](#)) model, which is widely deployed.

HTTP clients may use a range of methods to discover the presence of an HTTP proxy (WPAD, DHCP, manual configuration). Client application-layer communications remain unaware of such configuration. (In other words, handshake and data transfer interactions with the HTTP proxy are invisible to the application layer.)

Intermediaries may themselves have an HTTP proxy configured. A client attempting to initiate a tunnel to a remote host may end up traversing a proxy chain. This is a useful design characteristic and should be considered when selecting a preferred option.

[3.3.](#) Message Destination Agility

The CONNECT method currently expresses a request-target. This is a "fixed destination mode" where all messages travel on the same fixed TCP path to the same destination (ignoring lower level network elements).

The design should consider if more agile approach i.e. a "per-message destination mode" would support new network interaction models. This may add per-message overhead but optimisation may be possible.

[3.4.](#) Path MTU Discovery

The design should consider that endpoints may want/be required to avoid IP fragmentation. Support for reasonable attempts at path MTU discovery (PMTUD) should be included. Traditional PMTUD methods (such as those described in [\[RFC1191\]](#) and [\[RFC8201\]](#) are intended for TCP and rely on ICMP and ICMPv6 messages. [\[RFC2293\]](#) catalogs some of the problems with PMTUD. Packetization Layer PMTUD (PLPMTUD)

operate at the transport layer. Datagram PLPMTUD [[DPLPMTUD](#)] is a proposed further extension that describes approaches for various UDP-based transports.

[3.5.](#) Blind forwarding vs. in-the-loop Processing

[RFC7230] describes a tunnel as "a blind relay between two connections without changing messages". This approach may be overly restrictive for new interaction modes.

In the case of CONNECT for TCP-based tunnelling, the HiNT message sent by a client (TCP/IP packet payload) is decapsulated at the proxy and recapsulated in a new TCP/IP packet created and sent by the proxy. The proxy performs no processing of the HiNT message.

[HELIUM] proposes an alternative model, where the proxy does (and is expected to) modify HiNT messages.

[3.6.](#) Head-of-line Blocking

The current design of CONNECT-based tunnelling reserves either a whole TCP connection (HTTP/1.1) or an ordered byte stream (HTTP/2 and HTTP/QUIC) for the client-to-proxy hop. These are subject to head-of-line (HoL) blocking. For example, where there is an end-to-end tunnelled HTTP/2 connection, all of its streams are subject to the blocking on the single reserved stream. It is unknown to the author if this is perceived to be a high impact problem.

This document defines HTTP/2 and HTTP/QUIC frames ([Section 6](#)) that are sent on HTTP/2 or QUIC streams respectively.

For UDP or IP-based tunnels, HoL blocking may be problematic. It is unlikely that the application expects blocking to occur, leading to potential issues. (QUIC is specifically designed to avoid HoL blocking and is designed to operate on unreliable UDP, a reliable bearer may adversely affect performance.)

Future versions of QUIC may offer partial reliability. If it were used for the client-to-proxy hop, it could help mitigate HoL blocking

The design should consider the tension between the benefits of tunnelling, impact of HoL, and HTTP version [Section 3.1](#).

[4.](#) Candidate Solutions

Strawman candidate solutions are presented in order of increasing perceived complexity. It is hoped that wider input will help shape the solution.

[4.1.](#) CONNECT Method Augmentation

Enhance or augment the current definitions of the CONNECT method in HTTP/1.x, HTTP/2 and HTTP/QUIC. Data exchanges between a client and a single destination will be conveyed over existing byte streams with no additional framing. Client and proxy are required to assign meaning to groups of bytes delivered on the stream, which may be impractical.

[4.2.](#) UDPASSOCIATE with HINT Frames for HTTP/2 and HTTP/QUIC

Define a new method, UDPASSOCIATE ([Section 5.1](#)), that reserves a stream for the carriage of newly defined HINT frames ([Section 6.1](#)). Data exchanges between a client and a single destination will be conveyed using these frames. This requires HTTP/2 or HTTP/QUIC proxies, and precludes HTTP/1.x (because there is no means for framing HiNT messages).

[4.3.](#) HELIUM over WebSockets for all HTTP Versions

Tunnelling of UDP or IP using HELIUM ([\[HELIUM\]](#)) over WebSockets. Data exchanges between a client and destination(s) will be conveyed using CBOR-encoded HIP messages. WebSockets connections between client and proxy are established by existing means. This option would work for all HTTP versions that support WebSockets.

[4.4.](#) HELIUM over WebSockets for HTTP/1.1, Native Framing for HTTP/2 or HTTP/QUIC

Tunnelling of UDP or IP using HELIUM ([\[HELIUM\]](#)). Data exchanges between a client and destination(s) will be conveyed using HIP messages appropriate for the HTTP version.

For HTTP/1.x, WebSockets with CBOR-encoded HIP messages would be used.

For HTTP/2 and HTTP/QUIC, HIP messages would be framed and exchanged on a stream reserved by the new method, IPASSOCIATE ([Section 5.3](#)).

There are two framing options presented: light framing ([Section 6.2](#))

that uses the CBOR-encoded format, which would allow direct reuse of code to that used for the above WebSocket substrate; full framing

Internet-Draft HTTP-initiated Network Tunnelling (HiNT) October 2018

([Section 6.3](#)) that uses the native features of the application layer substrate, which may have advantages.

[5.](#) Technical Specification for HiNT Requests

This section outlines the technical specifications required to support the candidate solutions. Discussion of respective merits and drawbacks is captured in [Appendix B](#).

[5.1.](#) The UDPASSOCIATE Method for HTTP/1.1x

In HTTP/1.x, the UDPASSOCIATE method requests that the recipient establish a UDP-based tunnel to the destination origin server identified by the request-target and, if successful, thereafter restrict its behavior to blind forwarding of UDP datagram payloads, in both directions, until the tunnel is closed.

UDPASSOCIATE is intended only for use in requests to a proxy. An origin server that receives a UDPASSOCIATE request for itself MAY respond with a 2xx (Successful) status code to indicate that a connection is established. TODO: explicitly ban this?

A client sending a UDPASSOCIATE request MUST send the authority form of request-target ([Section 5.3 of \[RFC7230\]](#)); i.e., the request-target consists of only the host name and port number of the tunnel destination, separated by a colon. The port number is for UDP only.

```
UDPASSOCIATE hq.example.com:50781 HTTP/1.1
Host: hq.example.com:50781
```

The recipient proxy can establish a tunnel either by directly connecting to the request-target or, if configured to use another proxy, by forwarding the UDPASSOCIATE request to the next inbound proxy. Any 2xx (Successful) response indicates that the sender (and all inbound proxies) will switch to tunnel mode immediately after the blank line that concludes the successful response's header section; data received after that blank line is from the server identified by the request-target. Any response other than a successful response indicates that the tunnel has not yet been formed and that the

connection remains governed by HTTP.

TODO: how do connectionless UDP associations affirm that connection to the remote host succeeded? Perhaps a 2xx should be formed when the proxy believes it has sufficient capability to send or receive packets.

A tunnel is closed when an intermediary detects that either side has closed its connection (explicitly or implicitly). The intermediary

MUST attempt to send any outstanding data that came from the closed side to the other side, close both connections, and then discard any remaining data left undelivered.

A server MUST NOT send any Transfer-Encoding or Content-Length header fields in a 2xx (Successful) response to UDPASSOCIATE. A client MUST ignore any Content-Length or Transfer-Encoding header fields received in a successful response to UDPASSOCIATE.

A payload within a UDPASSOCIATE request message has no defined semantics.

[5.2.](#) The UDPASSOCIATE Method for HTTP/2 and HTTP/QUIC

In HTTP/2 and HTTP/QUIC, the UDPASSOCIATE method requests the establishment of a tunnel to a single remote host over a single stream. This mechanism has a few differences from the header field mapping described in [\[RFC7540\]](#), [Section 8.1.2.3](#):

- o The ":method" pseudo-header field is set to "UDPASSOCIATE"
- o The ":scheme" and ":path" pseudo-header fields MUST be omitted
- o The ":authority" pseudo-header field contains the host and port to connect to (equivalent to the authority-form of the request-target of CONNECT requests (see [\[RFC7230\]](#), [Section 5.3](#))).

A UDPASSOCIATE method that does not conform to these restrictions is malformed ([\[RFC7540\]](#), [Section 8.1.2.6](#)).

A proxy that supports UDPASSOCIATE can establish a tunnel to the server identified in the ":authority" pseudo-header field. Once this

is completed (see earlier TODO), the proxy sends a HEADERS frame containing a 2xx series status code to the client.

A successful UDPASSOCIATE request reserves the request stream for tunnelling. After the initial HEADERS frame sent by each peer, all subsequent frames exchanged on this stream correspond to data sent on the UDP association. [Section 6.1](#), [Section 6.2](#) and [Section 6.3](#) explore options for application-level framing and the mapping to UDP. Some frame types MUST NOT be sent on the reserved stream (e.g. RST_STREAM and more TBD). An endpoint that receives any of these MUST respond with a connection error.

The UDP association can be closed (explicitly or implicitly) by either peer. It is RECOMMENDED that peers close the association explicitly using tunnelled application-level means (if possible). Once this has happened, the client SHOULD close the reserved stream

on the client-to-proxy hop. Closing the reserved stream before an explicit close is likely to trigger an application-level implicit close (i.e. idle timeout).

[5.3](#). The IPASSOCIATE Method

The IPASSOCIATE method can be used by a client to request that the recipient establish an IP-based tunnel to the destination origin server identified by the request-target and, if successful, thereafter restrict its behaviour to blind forwarding of IP payloads, in both direction, until the tunnel is closed.

The IPASSOCIATE method would look and behave much like the UDPASSOCIATE method.

TODO: expand this definition if this method is preferred or required. Additional parameters may be required to accommodate the extra capabilities of IP-based tunnels.

[6](#). Technical Specification for HiNT Message Transfer

This section outlines the technical specifications required to support the candidate solutions. Discussion of respective merits and drawbacks is captured in [Appendix C](#).

6.1. HiNT Message Framing

The HINT frame carries HiNT messages between client and proxy. Is intended to be used with versions of HTTP that support binary framing. Definitions are provided for HTTP/2 and HTTP/QUIC, differing only in their use of padding. (The QUIC transport ([\[QUIC-TRANSPORT\]](#)) provides padding itself.) Frames are non-critical extensions to their respective protocols. Endpoints that do not support these frames will ignore them.

The payload of each HINT frame corresponds to a UDP datagram (or IP Packet?) sent or received by a HiNT proxy. A separate HiNT request is REQUIRED in order to initiate the tunnel with which these frames relate.

HINT frames are subject to flow control. The size of HINT frames should take into consideration the path MTU. Methods for path MTU discovery are discussed in [Section 3.4](#).

Frames MUST be associated with a non-control stream. If a frame is received on a control stream, the recipient MUST respond with a connection error. For HTTP/2 this is `PROTOCOL_ERROR`, for HTTP/QUIC this is TBD.

6.1.1.1. The HINT HTTP/2 Frame

The HINT HTTP/2 frame (type=0xTBD) defines the following flags (based on HTTP/2 flags):

END_STREAM (0x1): When set, bit 0 indicates that this frame is the last that the endpoint will send for the identified stream. Setting this flag causes the stream to enter one of the "half-closed" states or the "closed" state ([\[RFC7540, Section 5.1\]](#)).

PADDED (0x8): When set, bit 3 indicates that the Pad Length field and any padding that it describes are present.

[illegible]

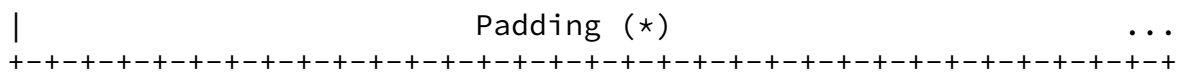


Figure 3: HINT HTTP/2 frame payload

The HINT HTTP/2 frame payload has the following fields:

Pad Length: An OPTIONAL 8-bit field containing the length of the frame padding in units of octets. This field is only present if the PADDED flag is set.

Payload: Arbitrary octets that correspond to messages sent to/from a HiNT proxy.

Padding: Padding octets that contain no application semantic value. Padding octets MUST be set to zero when sending. A receiver is not obligated to verify padding but MAY treat non-zero padding as a connection error ([\[RFC7540\], Section 5.4.1](#)) of type `PROTOCOL_ERROR`.

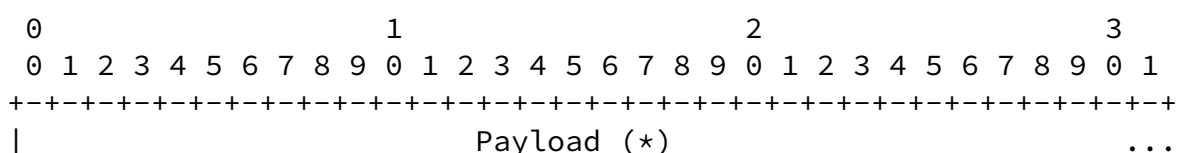
HINT HTTP/2 frames are subject to flow control ([\[RFC7540\], Section 5.2](#)) and can only be sent when a stream is in the "open" or "half-closed (remote)" state. If an HINT HTTP/2 frame is received whose stream is not in "open" or "half-closed (local)" state, the recipient MUST respond with a stream error ([\[RFC7540\] Section 5.4.2](#)) of type `STREAM_CLOSED`.

The HINT HTTP/2 frame is processed hop-by-hop. An intermediary MUST NOT forward HINT HTTP/2 frames, though it can use the information

contained in HINT HTTP/2 frames in forming new HINT HTTP/2 frames to send to its own proxy.

[6.1.2.](#) The HINT HTTP/QUIC Frame

The HINT HTTP/QUIC frame (type=0xTBD) defines no flags.



and any padding that it describes are present.

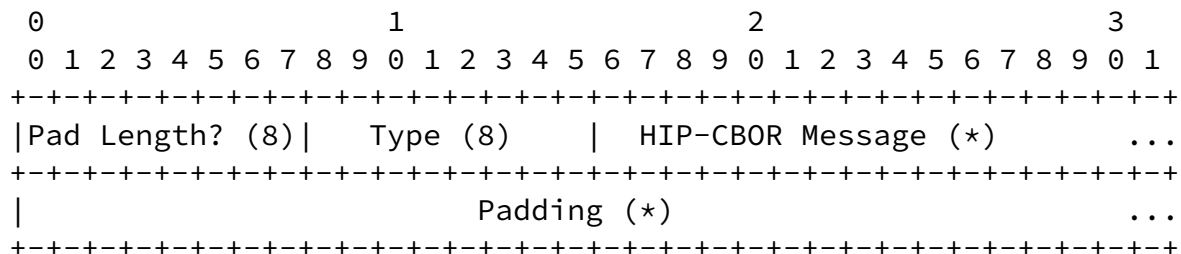


Figure 5: HIP HTTP/2 frame payload

The HIP HTTP/2 frame payload has the following fields:

Pad Length: An OPTIONAL 8-bit field containing the length of the frame padding in units of octets. This field is only present if the PADDED flag is set.

Type: An 8-bit field that identifies the HIP message type as defined in [\[HELIUM\]](#).

HIP-CBOR Message: A HIP message expressed in CBOR encoding including type, metadata (including padding), and packet or packet-prefix.

Padding: Padding octets that contain no application semantic value. Padding octets MUST be set to zero when sending. A receiver is not obligated to verify padding but MAY treat non-zero padding as a connection error ([\[RFC7540\]](#), [Section 5.4.1](#)) of type `PROTOCOL_ERROR`.

6.3. Full HIP HTTP/2 Framing

The OHIP, IHIP and MHIP frames (collectively xHIP) encode all HIP message data directly in the HTTP/2 frame structure.

These frames are non-critical extensions, endpoints that do not support them will ignore them.

The size of these frames should take into consideration the path MTU. Methods for path MTU discovery are discussed in [Section 3.4.2](#).

Frames MUST be associated with a non-control stream. If a frame is received on a control stream, the recipient MUST respond with a connection error. For HTTP/2 this is `PROTOCOL_ERROR`.

Each xHIP frame type contains zero or more instances of the Metadata-entry field. Fields are processed by the HIP application layer.

[illegible]

A Metadata-entry field is a tuple consisting of a Key and a length-delimited Value:

[illegible]

Specifically:

Key: An unsigned, 16-bit integer representing the HIP metadata key.

Value Length: An unsigned, 16-bit integer indicating the length, in octets of the Value field.

Value: An OPTIONAL sequence of octets containing an application-specific value.

6.3.1. The OHIP HTTP/2 Frame

The OHIP HTTP/2 frame (type=0xTBD) carries an "outbound" HIP message.

The OHIP HTTP/2 frame defines the following flags:

END_STREAM (0x1): When set, bit 0 indicates that this frame is the last that the endpoint will send for the identified stream. Setting this flag causes the stream to enter one of the "half-closed" states or the "closed" state ([\[RFC7540, Section 5.1\]](#)).

METADATA (0x2): When set, bit 1 indicates that the Metadata Entries field and metadata that describes are present

PADDED (0x8): When set, bit 3 indicates that the Pad Length field and any padding that it describes are present.

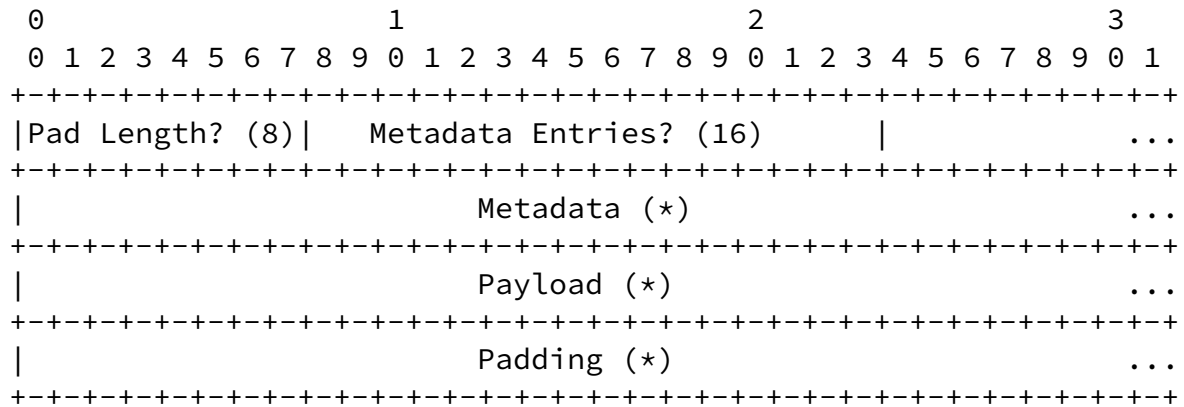
Internet-Draft HTTP-initiated Network Tunnelling (HiNT) October 2018


Figure 6: OHIP HTTP/2 frame payload

The OHIP HTTP/2 frame payload has the following fields:

Pad Length: An OPTIONAL 8-bit field containing the length of the frame padding in units of octets. This field is only present if the PADDED flag is set.

Metadata Entries: An OPTIONAL 16-bit field that indicates the number of Metadata-entries held in the Metadata field. This field is only present if the METADATA flag is set.

Metadata: Zero or more instances of the Metadata-entry field.

Payload: At most one packet (or prefix of a packet), in essence, a standard IP packet starting with an IP header.

Padding: Padding octets that contain no application semantic value. Padding octets MUST be set to zero when sending. A receiver is not obligated to verify padding but MAY treat non-zero padding as a connection error ([\[RFC7540\]](#), [Section 5.4.1](#)) of type `PROTOCOL_ERROR`.

[6.3.2](#). The IHIP HTTP/2 Frame

The IHIP HTTP/2 frame (type=0xTBD) carries an "inbound" HIP message.

The IHIP HTTP/2 frame defines the following flags:

END_STREAM (0x1): When set, bit 0 indicates that this frame is the last that the endpoint will send for the identified stream. Setting this flag causes the stream to enter one of the "half-closed" states or the "closed" state ([\[RFC7540\], Section 5.1](#)).

METADATA (0x2): When set, bit 1 indicates that the Metadata Entries field and metadata that it describes are present

PADDED (0x8): When set, bit 3 indicates that the Pad Length field and any padding that it describes are present.

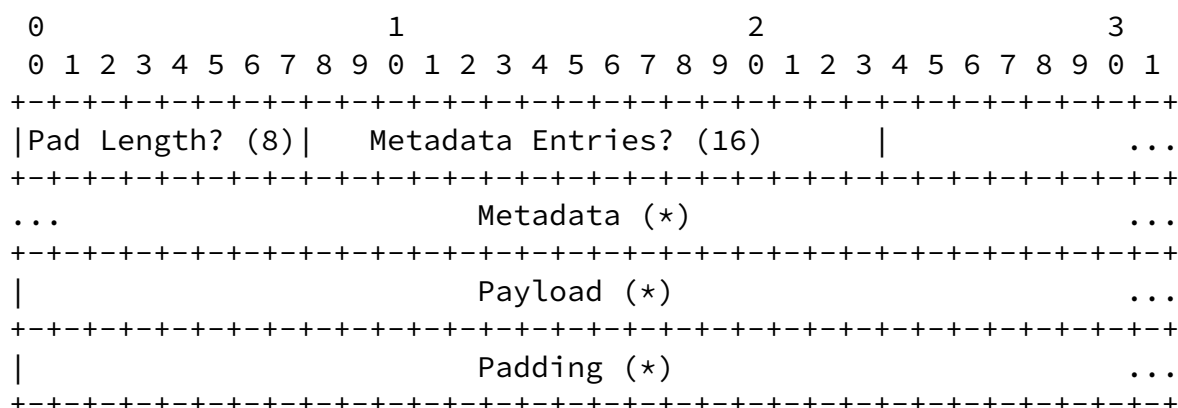


Figure 7: IHIP HTTP/2 frame payload

The IHIP HTTP/2 frame payload has the following fields:

Pad Length: An OPTIONAL 8-bit field containing the length of the frame padding in units of octets. This field is only present if the PADDED flag is set.

Metadata Entries: An OPTIONAL 16-bit field that indicates the number of Metadata-entries held in the Metadata field. This field is only present if the METADATA flag is set.

Metadata: Zero or more instances of the Metadata-entry field.

Payload: A packet, in essence, a standard IP packet starting with an IP header, as received by the proxy.

Padding: Padding octets that contain no application semantic value.

Figure 8: MHIP HTTP/2 frame payload

The MHIP HTTP/2 frame payload has the following fields:

Pad Length: An OPTIONAL 8-bit field containing the length of the frame padding in units of octets. This field is only present if the PADDED flag is set.

Metadata Entries: An OPTIONAL 16-bit field that indicates the number of Metadata-entries held in the Metadata field. This field is only present if the METADATA flag is set.

Err Length: An OPTIONAL 8-bit field containing the length of the Errors field. This field is only present if the ERROR flag is set.

Metadata: Zero or more instances of the Metadata-entry field.

Errors: An OPTIONAL octet array of length Err Length. Each octet of the array represents a HIP error as described in [[HELIUM](#)].

Payload: An OPTIONAL payload containing a prefix of the outbound packet as sent, including any parts that were modified. This field is only present if the PAYLOAD flag is set.

Padding: Padding octets that contain no application semantic value. Padding octets MUST be set to zero when sending. A receiver is not obligated to verify padding but MAY treat non-zero padding as a connection error ([\[RFC7540\]](#), [Section 5.4.1](#)) of type `PROTOCOL_ERROR`.

[7](#). Security Considerations

This document is partly motivated by the desire to prevent exposure to observers, to make detection and interference more difficult. The effectiveness of this is dependent on the chosen solution. Where HTTP is used only to bootstrap a HiNT session, messages will be carried without additional HTTP traffic to mask them. A more secure option would be to both bootstrap and carry HiNT messages inside an HTTP session. This of course relies on secure HTTP to provide

confidentiality.

It is noted that different HiNT traffic may have different characteristics (e.g. volumes and timing) when compared to the HTTP context that it is operating in. Session level encryption is weak with respect to traffic analysis. HTTP/2 provides further advice about the use of compression ([\[RFC7540\] Section 10.6](#)) and padding ([\[RFC7540\] Section 10.7](#)) to mitigate the ability for an observer to discriminate different forms of traffic. Additional application-layer padding may help.

TODO: Proxy authentication might be used to establish the authority to create a tunnel.

There are significant risks in establishing a tunnel to arbitrary servers. Proxies that support HiNT requests SHOULD restrict a HiNT session to a limited set of known ports or a configurable white list of safe request targets.

This section will address more security considerations once a single solution is chosen.

[8.](#) IANA Considerations

[8.1.](#) UDPASSOCIATE Method Registration

This section registers the "UDPASSOCIATE" method in "HTTP Method Registry" ([\[RFC7230\], Section 8.1](#)).

Pardue

Expires April 21, 2019

[Page 20]

Internet-Draft HTTP-initiated Network Tunnelling (HiNT) October 2018

Method Name: UDPASSOCIATE

Safe: No

Idempotent: No

Cacheable: No

Specification document(s): [Section 5.1](#) of this document

[8.2.](#) IPASSOCIATE Method Registration

This section registers the "IPASSOCIATE" method in "HTTP Method Registry" ([\[RFC7230\]](#), [Section 8.1](#)).

Method Name: IPASSOCIATE

Safe: No

Idempotent: No

Cacheable: No

Specification document(s): [Section 5.3](#) of this document

[8.3](#). The HINT HTTP/2 Frame Type

This section registers the "HINT" frame type in the "HTTP/2 Frame Type" registry ([\[RFC7540\]](#), [Section 11.2](#)).

Frame Type: HINT

Code: 0XTBD

Specification: [Section 6.1.1](#) of this document

[8.4](#). The HINT HTTP/QUIC Frame Type

This section registers the "HINT" frame type in the "HTTP/QUIC Frame Type" registry ([\[QUIC-HTTP\]](#), [Section 9.3](#)).

Frame Type: HINT

Code: 0XTBD

Specification: [Section 6.1.2](#) of this document

[8.5](#). The HIP HTTP/2 Frame Type

This section registers the "HIP" frame type in the "HTTP/2 Frame Type" registry ([\[RFC7540\]](#), [Section 11.2](#)).

Frame Type: HIP

Code: 0XTBD

Specification: [Section 6.2](#) of this document

[8.6.](#) The OHIP HTTP/2 Frame Type

This section registers the "OHIP" frame type in the "HTTP/2 Frame Type" registry ([\[RFC7540\]](#), [Section 11.2](#)).

Frame Type: OHIP

Code: 0XTBD

Specification: [Section 6.3.1](#) of this document

[8.7.](#) The IHIP HTTP/2 Frame Type

This section registers the "IHIP" frame type in the "HTTP/2 Frame Type" registry ([\[RFC7540\]](#), [Section 11.2](#)).

Frame Type: IHIP

Code: 0XTBD

Specification: [Section 6.3.2](#) of this document

[8.8.](#) The MHIP HTTP/2 Frame Type

This section registers the "MHIP" frame type in the "HTTP/2 Frame Type" registry ([\[RFC7540\]](#), [Section 11.2](#)).

Frame Type: MHIP

Code: 0XTBD

Specification: [Section 6.3.3](#) of this document

9. References

9.1. Normative References

- [HELIUM] Schwartz, B., "Hybrid Encapsulation Layer for IP and UDP Messages (HELIUM)", [draft-schwartz-httpbis-helium-00](#) (work in progress).
- [QUIC-HTTP] Bishop, M., Ed., "Hypertext Transfer Protocol (HTTP) over QUIC", [draft-ietf-quic-http-13](#) (work in progress).
- [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>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", [RFC 7230](#), DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", [RFC 7231](#), DOI 10.17487/RFC7231, June 2014, <<https://www.rfc-editor.org/info/rfc7231>>.
- [RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", [RFC 7540](#), DOI 10.17487/RFC7540, May 2015, <<https://www.rfc-editor.org/info/rfc7540>>.
- [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>>.

9.2. Informative References

- [DOH] Hoffman, P. and P. McManus, "DNS Queries over HTTPS", [draft-ietf-doh-dns-over-https-10](#) (work in progress).
- [DPLPMTUD] Ruengeler, I., "Packetization Layer Path MTU Discovery for Datagram Transports", [draft-ietf-tsvwg-datagram-plpmtud-01](#) (work in progress).

Internet-Draft HTTP-initiated Network Tunnelling (HiNT) October 2018

[H2-WEB_SOCKETS]

McManus, P., Ed., "Bootstrapping WebSockets with HTTP/2", [draft-ietf-httpbis-h2-websockets-02](#) (work in progress).

[QUIC-TRANSPORT]

Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", [draft-ietf-quic-transport-13](#) (work in progress).

[RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", [RFC 1191](#), DOI 10.17487/RFC1191, November 1990, <<https://www.rfc-editor.org/info/rfc1191>>.

[RFC1928] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and L. Jones, "SOCKS Protocol Version 5", [RFC 1928](#), DOI 10.17487/RFC1928, March 1996, <<https://www.rfc-editor.org/info/rfc1928>>.

[RFC2293] Kille, S., "Representing Tables and Subtrees in the X.500 Directory", [RFC 2293](#), DOI 10.17487/RFC2293, March 1998, <<https://www.rfc-editor.org/info/rfc2293>>.

[RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", [RFC 4821](#), DOI 10.17487/RFC4821, March 2007, <<https://www.rfc-editor.org/info/rfc4821>>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", [RFC 5246](#), DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.

[RFC5766] Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", [RFC 5766](#), DOI 10.17487/RFC5766, April 2010, <<https://www.rfc-editor.org/info/rfc5766>>.

[RFC8201] McCann, J., Deering, S., Mogul, J., and R. Hinden, Ed., "Path MTU Discovery for IP version 6", STD 87, [RFC 8201](#), DOI 10.17487/RFC8201, July 2017, <<https://www.rfc-editor.org/info/rfc8201>>.

[Appendix A](#). Acknowledgments

The first draft of this document was written with support from BBC Research & Development while Lucas was employed there.

Pardue

Expires April 21, 2019

[Page 24]

Internet-Draft HTTP-initiated Network Tunnelling (HiNT) October 2018

Many aspects of this document were inspired by the existing outputs of the HTTP Working Group and the wider IETF community. Some aspects were inspired by Mark Nottingham's previous work on HTTP/2 VPN.

The author would like to thank Richard Bradbury, Katharine Daly, Piers O'Hanlon, and Ben Schwartz for design input and review of this document.

[Appendix B](#). HiNT Request Options

The following list presents options for a HiNT request in no particular order:

1. Enhance the CONNECT method (i.e. request/response headers) that permits negotiation of the proxy-to-destination transport protocol.
 - * Pros:
 - + Already widely supported for HTTP proxying use case.
 - + Bootstrapping WebSockets for HTTP/2 [[H2-WEBSOCKETS](#)] has made some headway here.
 - * Cons:
 - + Deployability may be unrealistic. New types of tunnelling behaviour may not meet expectations of extant endpoints.
 - + CONNECT method extension may not be popular. Need to consider if this is suited for all HTTP or specific version.
2. Define a new method (e.g. UDPASSOCIATE [Section 5.1](#)) that is restricted to use UDP for the proxy-to-destination transport

protocol.

* Pros:

- + Clear demarcation between the conventional TCP case.
- + Well suited for HTTP/QUIC use case.

* Cons:

- + Limited applicability (because it is UDP-only?).

Pardue

Expires April 21, 2019

[Page 25]

Internet-Draft HTTP-initiated Network Tunnelling (HiNT) October 2018

3. Define a new method (e.g. IPASSOCIATE) that permits negotiation of the proxy-to-destination transport protocol.

* Pros:

- + Clear demarcation between the conventional TCP case.
- + Well suited for HTTP/QUIC use case.

* Cons:

- + Too complicated for most needs (?).

4. Define a substrate that is already supported by HTTP proxying i.e. WebSocket.

* Pros:

- + Capable of functioning irrespective of HTTP version.

* Cons:

- + Multiple layers requires implementation complexity and adds data transfer overhead.

5. Define HTTP/2 and HTTP/QUIC means of HiNT request, e.g. a new frame or setting that is used to reserve a stream (or streams) for special processing of HiNT messages.

- * Pros:
 - + Avoids coining a new method.
- * Cons:
 - + Excludes HTTP/1.1.

[Appendix C](#). HiNT Message Transfer Options

The following list presents options for framing of messages within a HiNT session in no particular order:

1. Where CONNECT is used by an HTTP/1.1 client, each TCP/IP packet on the client-to-proxy hop maps directly to a packet (TCP/IP or UDP/IP) on the proxy-to-destination hop.

- * Pros:

- + "Simple" option that requires no new TCP framing definition.
- * Cons:
 - + Breaks the layering model
 - + In practice, the endpoints are not likely to be able to do this.
2. Where CONNECT is used by an HTTP/2 or HTTP/QUIC client, each DATA frame on the client-to-proxy hop maps directly to a packet (TCP/IP or UDP/IP) on the proxy-to-destination hop.
- * Pros:
 - + Simple option that requires no additional framing.
 - + Client and proxy already handle DATA frames.
 - * Cons:

- + DATA frames are delivered on streams, which are treated as an ordered byte stream. It may not be possible to treat them individually.
3. Define framing format that uses a WebSocket substrate. For example, the HELIUM Inner Protocol [[HELIUM](#)].
- * Pros:
 - + Would be supported in HTTP/1.1, HTTP/2 and HTTP/QUIC (subject to further work).
 - * Cons:
 - + Framing overhead which could be optimised away in HTTP/2 and HTTP/QUIC.
 - + Requires WebSocket support in endpoints.
 - + Breaks the layering model(?).
4. Define a new simple HTTP/2 and HTTP/QUIC extension frame for carriage of HiNT messages. (This would likely be subject to stream-level flow control). The frame payload would be encapsulated by the proxy. This approach is reliant on a fixed destination tunnel [Section 3.3](#).

- * Pros:
 - + Clear separation between stream-based and message-based tunnels.
 - + Similar to how endpoints already handle CONNECT today.
- * Cons:
 - + New frame may change the semantic of HTTP/2 and HTTP/QUIC. Therefore, it may need to be negotiated by a new SETTINGS parameter.
 - + Excludes HTTP/1.1

- + Dependence on fixed destination tunnel may not support all desired interaction modes.
5. Define a new HTTP/2 and HTTP/QUIC extension frame(s) for carriage of HiNT messages. (This would likely be subject to stream-level flow control). This could express HELIUM Inner Protocol [[HELIUM](#)] messages directly and, by virtue, would support per-message destination.
- * Pros:
 - + Clear separation between stream-based and message-based tunnels.
 - + Reduced overhead compared for HTTP/2 and HTTP/QUIC compared to carriage over WebSocket substrate.
 - * Cons:
 - + New frame may change the semantic of HTTP/2 and HTTP/QUIC. Therefore, it may need to be negotiated by a new SETTINGS parameter.
 - + Some divergence from HTTP/1.1.
 - + Differs from blind forwarding which is implemented in CONNECT proxies today.

[Appendix D.](#) Changelog

RFC Editor's Note: Please remove this section prior to publication of a final version of this document.

Pardue

Expires April 21, 2019

[Page 28]

Internet-Draft HTTP-initiated Network Tunnelling (HiNT)

October 2018

[D.1.](#) Since [draft-pardue-httpbis-http-network-tunnelling-00](#)

- o Author's address.

Author's Address

Lucas Pardue

Email: lucaspardue.24.7@gmail.com