httpbis
Internet-Draft
Intended status: Standards Track
Expires: December 27, 2018

B. Schwartz
Google

June 25, 2018

Hybrid Encapsulation Layer for IP and UDP Messages (HELIUM)
draft-schwartz-httpbis-helium-00

Abstract

HELIUM is a protocol that can be used to implement a UDP proxy, a VPN, or a hybrid of these. It is intended to run over a reliable, secure substrate transport. It can serve a variety of use cases, but its initial purpose is to enable HTTP proxies to forward non-TCP flows.

Status of This Memo

Copyright Notice

Table of Contents

1. Overview

This proposal describes a network tunnel that is intended as a natural extension or complement to existing HTTP proxies. It has two components

o A flexible packet-oriented tunneling protocol that can act as either a VPN or a UDP proxy (Section 2)

o A substrate for this protocol that allows it to run as part of an HTTPS server (Section 3)

This design combines the benefits of several existing protocols, such as [OpenConnect] and [TURN]. Like OpenConnect, this protocol gains the privacy, authentication, and management benefits of HTTPS. Like

TURN, this protocol can be used as a UDP proxy for realtime and P2P applications.

2. HELIUM Inner Protocol (HIP)

The protocol is designed to span two different use cases

o a UDP tunnel (proxy)

o an IP tunnel (VPN)

These two use cases are normally handled by entirely separate protocols, like [TURN] and [L2TP]. However, UDP is fundamentally very similar to IP (differing mostly by the addition of a 2-byte "port number"), so it seems plausible that a single protocol may serve both purposes. Additionally, a UDP proxy can be enriched by partial support for ICMP (enabling [PMTUD], traceroute, etc.), so there may be configurations that benefit from blending these uses.

The protocol is intended to run between a client and a proxy, on a substrate that provides confidentiality, integrity, flow control, congestion control, and reliability (at least optionally). It should take advantage of substrates that support out-of-order delivery, but still function acceptably on strictly ordered transports.

2.1. Terminology

o Proxy: the server implementing this protocol, acting as a UDP proxy or IP tunnel endpoint

o Client: the endpoint that is implementing this protocol on the client side

o Destination: a service that the client is trying to reach through the proxy

o Context: the identity of the transport session used to transfer messages between a client and the proxy (e.g. one WebSocket)

o Substrate: the transport protocol used to transfer these messages (e.g. WebSocket)

o Flow: a sequence of related packets between the client and a single destination

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.

2.2. Requirements

- o It shall be possible for a proxy to operate in an environment without elevated privileges.

  - * Such a proxy might only support operating as a UDP tunnel.

- o It shall be possible for a proxy with elevated privileges to operate without any parsing of IP payloads.

  - * Such a proxy would operate as an IP tunnel.

- o A client can direct the proxy to send multiple packets from the same IP (and UDP port).

- o A client can tell what IP address and port the proxy is using to communicate on its behalf.

  - * A client can bind an address (or address:port) and learn it before emitting any packets.

- o A client can tell if the proxy doesn't support a feature it's trying to use.

- o New connections can be established without waiting for a roundtrip between client and proxy.

- o The protocol enables good performance when tunneling streams that use delay-based congestion control (e.g. TCP Vegas, [BBR], [RMCAT-GCC]).

- o The client has an option to let the proxy resolve DNS names itself, with a latency benefit.

- o The proxy can be implemented with tightly bounded memory usage.

2.3. Abstract Structure

Each HIP message consists of a type, optional metadata, and at most one packet (or prefix of a packet). The packet (or prefix) is a standard [IPv4] or [IPv6] packet, starting with the IP header.

There are three message types defined: "outbound", "inbound", and "meta".

A message from the client to the proxy is always of type "outbound". It always includes a complete packet for the proxy to send to the destination (potentially after header modifications). The possible metadata fields that this message may contain are as follows:

- id (integer): An ID number identifying this message. If present, the client is implicitly requesting a "meta" message from the proxy. A client MUST NOT reuse an ID until a "meta" reply message is received.

- domain (UTF-8 string): A DNS name to override the destination IP. The proxy will perform an A or AAAA lookup, depending on the IP version of the included packet. The proxy will buffer the packet until name lookup completes. The proxy SHOULD avoid creating duplicate outstanding DNS queries, and SHOULD cache the result to provide a consistent mapping.

- dns (integer): The presence of this option indicates that the client wishes to direct the packet to one of the proxy's preferred DNS servers. Its value is an index into the proxy's list of preferred recursive resolvers for this IP version, modulo the length of the list. This option overrides the destination IP, and MUST NOT appear in a message with the "domain" option.

A message from the proxy to the client may be of type "inbound" or "meta". An "inbound" message contains a packet that the proxy received from the destination, unmodified, including the IP header. It contains one metadata field:

- timestamp (integer): A timestamp in microseconds modulo 2^32, indicating when the proxy received this packet from the destination. The absolute base time is unspecified, as this is only used for computing time differences. If the proxy reassembled the packet from fragments, this timestamp is the time when reassembly completed.

A "meta" message is only sent by the proxy to a client after it receives an "outbound" message with an ID from the client. If the proxy modified the outbound packet in any way, the "meta" message MUST contain a prefix of the outbound packet as sent, including any parts that were modified. Changes might include the source IP, destination IP, TTL, DSCP priority, UDP source port, etc. If there was an error, the proxy MAY include a modified prefix that would not have encountered the error (e.g. by changing the protocol ID from an unsupported protocol (e.g. TCP) to a supported protocol (e.g. UDP)). The message contains the following metadata:

- id (integer): This is the ID number of the "outbound" message to which this is a reply.

- error (Array of integer): If present, these error codes indicate why the proxy could not send the packet contained in the "outbound" message to the destination.

- timestamp (integer): The time when the outbound packet was sent from the proxy to the destination, in the same format used for "inbound" messages. If there was an error, this is the time that the error was detected.

If the proxy receives a message from the client of an unrecognized type, and the message has an "id" field, the server SHOULD reply with a "meta" message matching that ID and indicating an "Unsupported message type" error.

If the proxy receives a message from the client with unknown metadata fields, it SHOULD ignore the unknown fields and process the message as normal.

If the proxy receives an "outbound" message with an all-zero destination address and no address-overriding metadata, the proxy SHOULD rewrite the packet for transmission and establish any required address or port mappings, but not attempt to send the packet. If an ID number is present, the proxy SHOULD reply with a "meta" message indicating success unless a non-address-related error occurred.

All messages can also include padding. Padding can be represented as a metadata field named "padding" whose value is discarded by the recipient.

All integer values defined in this section are non-negative. All metadata keys defined here MUST NOT appear more than once. Recipients SHOULD treat negative numbers and repeated keys as metadata parsing errors.

2.3.1. Error codes

These are the numeric error codes that the proxy may include in a "meta" message

Code	Error
1	Unsupported message type
2	Metadata parsing error
3	Unsupported IP version
4	Invalid IP header
5	Can't send fragment
6	Packet too large
7	Unsupported IP option
8	Unsupported protocol
9	No route to host
10	Network unreachable
11	Destination IP not allowed
12	Destination DNS name not allowed
13	DNS name has no address (NXDOMAIN)
14	DNS name resolution failed
15	General server failure
16	Usage limit exceeded

Additional error codes may be defined in the future.

2.4. CBOR-based Encoding (HIP-CBOR)

To encode abstract HIP messages into concrete form, we use a [CBOR]-based encoding. Other equivalent but incompatible encodings might be defined in the future.

In this encoding, each message is formed by concatenating a one-byte type field, the metadata encoded in CBOR, and the packet or packet-prefix.

Byte	Type
0x01	outbound
0x02	inbound
0x03	meta

Metadata is encoded in CBOR as a Map. For compactness, keys are integer-valued, with the following significance:

Key	Field
0	padding
1	id
2	domain
3	dns
4	timestamp
5	error

Additional message types and metadata fields may be defined in the future.

When sending a message, endpoints SHOULD use the most compact available encoding of each metadata value. When receiving a message, recipients are NOT REQUIRED to accept extremely inefficient or obscure encodings that are allowed by CBOR (e.g. Bignums, Decimal Fractions).

2.5. **Addressing**

There are two major modes of operation that a proxy might use: IP tunnel and UDP tunnel. Both operation modes require the proxy to inspect and possibly modify the IP header of the packet contained in an "outbound" message before sending the packet to the destination. The UDP tunnel mode in addition requires the proxy to inspect and possibly modify the UDP header in the IP payload.

2.5.1. IP Header

Initially, the client does not know the IP address that the proxy will use as the source IP for packets it sends to the destination. The protocol does not require the client to correctly populate the source IP in its outbound packets to the proxy. Rather, the client chooses any IP address, and the proxy will rewrite this address into one of its own outbound IP addresses. Within a single context, the proxy MUST maintain a stable address mapping with a reasonable lifetime, similar to Network Address Translation [NAT].

In IP tunnel mode, the proxy MUST NOT map multiple contexts to the same outbound IP address at the same time, as it would then be impossible to determine unambiguously where to direct packets received from the destination. These outbound IP addresses MAY be publicly routable, or they MAY be in a reserved range (e.g. [RFC1918], [RFC4193]), using [NAT] to reach the public internet.

2.5.2. UDP Header

In UDP tunnel mode, the proxy MAY also rewrite the UDP source port of a packet before sending it to the destination. The client has no way to initially know what source port the proxy will use in this mode, so the protocol does not require the client to correctly populate the source port in its outbound packets to the proxy. In UDP tunnel mode, the proxy MAY map the same outbound IP address to multiple contexts with overlapping lifetimes, but the proxy SHOULD ensure that each UDP port is only mapped to a single context (i.e. an endpoint-independent mapping policy as described in [RFC4787]). A proxy MAY violate this condition only if it serves a limited use case in which the correct context for an inbound packet will never be ambiguous.

2.6. Example Configurations

2.6.1. Single IP tunnel

The client sends outbound IP packets to the server with empty metadata, and with various destinations and protocols (e.g. ICMP, TCP, UDP). The proxy rewrites the source address of all packets to match the reserved IP address for this client, and forwards all incoming packets to the client.

2.6.2. Multiple source IPs in one context

A client sends IP packets to the proxy with various source addresses, and includes an ID number in each one. For each ID number, the server's "meta" reply reveals the proxy source IP that was mapped to the client's chosen source IP. Once the client has learned the

mapping, the client stops including an ID number in subsequent messages.

2.6.3. Domain-based proxy

The client sends its initial flight of packets with an ID number and a domain in the metadata, and all zeroes in the destination IP address. The "meta" replies indicate the rewritten destination IP address, which is the resolved location of the destination. The client then emits subsequent packets with this destination IP address, and omits all metadata.

If the proxy does not know the exact IP header used (e.g. because it is using the network through a UDP socket API), it will synthesize an approximate IP header for the "meta" replies.

2.6.4. UDP proxy with PMTUD and traceroute

The client sends "outbound" UDP packets with the ID set and varying size or TTL. The proxy MUST NOT fragment unless the packet is IPv4 and the DONT-FRAGMENT bit is unset.

If the proxy could not send the packet because it was too large, it MUST reply with an error (Packet too large) and SHOULD include a rewritten header indicating the maximum size.

If the proxy fragmented the packet, it will reply with success and a prefix including the size of the first fragment.

If the proxy modified the outbound TTL, it will indicate this in the reply prefix.

If the proxy receives an ICMP response (e.g. Time Exceeded, Fragmentation Needed), it MAY forward it to the client. To support this use case, it MUST do so.

A proxy with this behavior can be implemented without elevated permissions on most common operating systems (see [I-D.martinsen-tram-stuntrace]).

2.6.5. Advanced DNS queries

The client sends an "outbound" UDP packet to port 53 with an ID number set, and a "dns" metadata value of 0. This packet is a DNS query, perhaps for a DNSKEY, TLSA, or TXT record.

The proxy overwrites the destination IP address with the IP of its first DNS server and sends the outbound packet. It also sends a

"meta" message to the client, containing the IP header with this destination address, as well as the modified source address and port.

The client is now waiting for an "inbound" message containing a reply from this DNS server to the modified source address and port. If no reply is received within some timeout, the client retries. This time, it sets a "dns" value of 1, indicating that the retry should use the proxy's second DNS server, if one exists.

2.6.6. UDP Server Application

The client sends an "outbound" UDP packet with an ID number set and all zeros in the destination IP. The "meta" reply includes the rewritten source IP and port, which is bound to this context. The client can now inform third parties to send data to this IP and port.

2.6.7. High-Performance Delay-based Congestion Control

The client is sending and receiving a flow that uses delay-based congestion control. Between client and proxy, this flow is transmitted according to the congestion control behaviors of the HELIUM substrate. From the proxy to the destination, congestion control is the responsibility of the client and destination.

To monitor delay on the proxy-destination path, the client can include an ID number in every outbound message. This will cause the proxy to reply with a "meta" message, including the send timestamp. By comparing these send timestamps with the receive timestamps in inbound messages, the client can accurately monitor the round-trip time between proxy and destination.

If the proxy-destination roundtrip time is gradually increasing, the client can reduce its send rate below the limit imposed by the HELIUM substrate.

2.7. Optimizations

Proxies are NOT REQUIRED to perform reassembly of inbound IP fragments. Proxies MAY reassemble IP fragments, or they MAY forward each fragment independently to the client. This helps to limit proxy memory usage.

When the client sends an "outbound" message with the "domain" metadata, the proxy has to buffer the corresponding packet until the domain name is resolved. To limit memory usage, the proxy can "peek" at the query without removing it from the transport's receive buffer. The transport's flow control will then limit the amount of memory that the client can consume.

3. WebSocket as a HELIUM Substrate (HELIUM-WebSocket)

The HELIUM Inner Protocol (Section 2) requires a substrate transport to deliver messages between client and proxy. The WebSocket protocol is a suitable substrate. Each HIP-CBOR message (Section 2.4) can be sent as a WebSocket message of type "binary".

If a browser is configured to act as a HELIUM client, communicating with the proxy over a WebSocket, the WebSocket is controlled and terminated by the browser itself, not associated with any particular origin or webpage.

3.1. Direct Configuration

The location of a WebSocket HELIUM proxy is defined by a WebSocket URL, e.g. "wss://proxy.example/example-path". If the client knows the address of a WebSocket HELIUM proxy, then the client may simply connect to the proxy by establishing a WebSocket connection. The client's WebSocket handshake request MUST contain the "Sec-WebSocket-Protocol" header with value "helium-cbor" as well as an authorization header (e.g. Proxy-Authorization) if needed.

3.2. Implicit Configuration from an HTTP proxy

Operators that run both an HTTP proxy, defined by some http or https URL, as well as a WebSocket HELIUM proxy, SHOULD return a response containing a new header, "Helium-Proxy-URL", when a client sends a proxy-specific request (e.g. HTTP CONNECT) to the operator's HTTP proxy. This new header, containing the WebSocket address of the HELIUM proxy, allows clients to discover the existence and location of a HELIUM proxy when they already know about an associated HTTP proxy. Clients can then connect to the discovered HELIUM proxy as described above.

In cases where user-facing proxy configuration options are limited (e.g. a web browser's settings menu), a user may not be able to directly configure a HELIUM proxy even if they know its address. If an option for configuring a HTTP(S) proxy is available, however, the Helium-Proxy-URL header will allow a user to implicitly configure a WebSocket HELIUM proxy by entering an associated HTTP(S) proxy address.

A client with access to both an HTTP(S) proxy and a HELIUM proxy SHOULD use the HTTP(S) proxy for all connections that it can support, and use the HELIUM proxy for all other network activity.

3.3. Optimizations

After initiating the WebSocket connection, a client MAY send its initial HIP messages without waiting for the server's reply. This saves 1 RTT, similar to TLS False Start [FALSESTART].

Clients and proxies MAY negotiate WebSocket DEFLATE compression with context takeover (see Section 7 of [RFC7692]). This will replace consistent headers with back-references to the previous matching packet. On typical streams, this removes most of the IP and HIP-CBOR overhead, and can even compress the payload if it contains patterns that appear in each packet. However, implementers should use caution when combining compression and padding, as compression can render some padding schemes ineffective.

4. IANA Considerations

The names and numbers of the HIP message types, metadata fields, and error codes will each require a new IANA registry. Additionally, HELIUM-WebSocket will require registration of a new WebSocket Protocol ("helium-cbor") and a new HTTP header ("Helium-Proxy-URL").

5. Acknowledgements

Many thanks to Katharine Daly and Lucas Pardue for their early and extensive review of this proposal.

6. References

6.1. Normative References

[CBOR] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

[IPv4] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <https://www.rfc-editor.org/info/rfc791>.

[IPv6] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/info/rfc8200>.

[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>.

[RFC7692] Yoshino, T., "Compression Extensions for WebSocket", RFC 7692, DOI 10.17487/RFC7692, December 2015, <https://www.rfc-editor.org/info/rfc7692>.

[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>.

6.2. Informative References

[BBR] Cardwell, N., Cheng, Y., Yeganeh, S., and V. Jacobson, "BBR Congestion Control", draft-cardwell-iccrg-bbr-congestion-control-00 (work in progress), July 2017.

[FALSESTART] Langley, A., Modadugu, N., and B. Moeller, "Transport Layer Security (TLS) False Start", RFC 7918, DOI 10.17487/RFC7918, August 2016, <https://www.rfc-editor.org/info/rfc7918>.

[I-D.martinsen-tram-stuntrace] Martinsen, P. and D. Wing, "STUN Traceroute", draft-martinsen-tram-stuntrace-01 (work in progress), June 2015.

[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/info/rfc2661>.

[NAT] Srisuresh, P. and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)", RFC 3022, DOI 10.17487/RFC3022, January 2001, <https://www.rfc-editor.org/info/rfc3022>.

[OpenConnect] Mavrogiannopoulos, N., "The OpenConnect VPN Protocol Version 1.0", draft-mavrogiannopoulos-openconnect-00 (work in progress), September 2016.

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

[RFC1918] Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, DOI 10.17487/RFC1918, February 1996, <https://www.rfc-editor.org/info/rfc1918>.

[RFC4193] Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast Addresses", RFC 4193, DOI 10.17487/RFC4193, October 2005, <https://www.rfc-editor.org/info/rfc4193>.

[RFC4787] Audet, F., Ed. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, DOI 10.17487/RFC4787, January 2007, <https://www.rfc-editor.org/info/rfc4787>.

[RMCAT-GCC] Holmer, S., Lundin, H., Carlucci, G., Cicco, L., and S. Mascolo, "A Google Congestion Control Algorithm for Real-Time Communication", draft-ietf-rmcat-gcc-02 (work in progress), July 2016.

[TURN] 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>.

Author's Address

Ben Schwartz
Google

Email: bemasc@google.com