Workgroup: httpbis
Internet-Draft:
draft-schwartz-httpbis-connect-tcp-latest
Published: 9 March 2023
Intended Status: Standards Track
Expires: 10 September 2023
Authors: B. M. Schwartz
Google LLC

Template-Driven HTTP CONNECT Proxying for TCP

Abstract

TCP proxying using HTTP CONNECT has long been part of the core HTTP specification. However, this proxying functionality has several important deficiencies in modern HTTP environments. This specification defines an alternative HTTP proxy service configuration for TCP connections. This configuration is described by a URI Template, similar to the CONNECT-UDP and CONNECT-IP protocols.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

1.1. History

HTTP has used the CONNECT method for proxying TCP connections since HTTP/1.1. When using CONNECT, the request target specifies a host and port number, and the proxy forwards TCP payloads between the client and this destination ([RFC9110], Section 9.3.6). To date, this is the only mechanism defined for proxying TCP over HTTP. In this specification, this is referred to as a "classic HTTP CONNECT proxy".

HTTP/3 uses a UDP transport, so it cannot be forwarded using the pre-existing CONNECT mechanism. To enable forward proxying of HTTP/3, the MASQUE effort has defined proxy mechanisms that are capable of proxying UDP datagrams [RFC9298], and more generally IP datagrams [I-D.ietf-masque-connect-ip]. The destination host and port number (if applicable) are encoded into the HTTP resource path, and end-to-end datagrams are wrapped into HTTP Datagrams [RFC9297] on the client-proxy path.

1.2. Problems

Classic HTTP CONNECT proxies are identified by an origin. The proxy does not have a path of its own. This prevents any origin from hosting multiple distinct proxy services.

Ordinarily, HTTP allows multiple origin hostnames to share a single server IP address and port number (i.e., virtual-hosting), by specifying the applicable hostname in the "Host" or ":authority" header field. However, classic HTTP CONNECT proxies use these fields to indicate the CONNECT request's destination ([RFC9112], Section 3.2.3), leaving no way to determine the proxy's origin from the request. As a result, classic HTTP CONNECT proxies cannot be deployed using virtual-hosting, nor can they apply the usual defenses against server port misdirection attacks (see Section 7.4 of [RFC9110]).

Classic HTTP CONNECT proxies can be used to reach a target host that is specified as a domain name or an IP address. However, because only a single target host can be specified, proxy-driven Happy Eyeballs and cross-IP fallback can only be used when the host is a domain name. For IP-targeted requests to succeed, the client must know which address families are supported by the proxy via some out-of-band mechanism, or open multiple independent CONNECT requests and abandon any that prove unnecessary.

1.3. Overview

This specification describes an alternative mechanism for proxying TCP in HTTP. Like [RFC9298] and [I-D.ietf-masque-connect-ip], the proxy service is identified by a URI Template. Proxy interactions reuse standard HTTP components and semantics, avoiding changes to the core HTTP protocol.

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.

3. Specification

A template-driven TCP transport proxy for HTTP is identified by a URI Template [RFC6570] containing variables named "target_host" and "tcp_port". The client substitutes the destination host and port number into these variables to produce the request URI.

The "target_host" variable **MUST** be a domain name, an IP address literal, or a list of IP addresses. The "tcp_port" variable **MUST** be a single integer. If "target_host" is a list (as in Section 2.4.2 of [RFC6570]), the server **SHOULD** perform the same connection procedure as if these addresses had been returned in response to A and AAAA queries for a domain name.

3.1. In HTTP/1.1

In HTTP/1.1, the client uses the proxy by issuing a request as follows:

* The method **SHALL** be "GET".
* The request **SHALL** include a single Host header field containing the origin of the proxy.
* The request **SHALL** include a Connection header field with the value "Upgrade". (Note that this requirement is case-insensitive as per Section 7.6.1 of [RFC9110].)
* The request **SHALL** include an "Upgrade" header field with the value "connect-tcp".
* The request's target **SHALL** correspond to the URI derived from expansion of the proxy's URI Template.

If the request is well-formed and permissible, the proxy **MUST** attempt the TCP connection before returning its response header. If the TCP connection is successful, the response **SHALL** be as follows:

* The HTTP status code **SHALL** be 101 (Switching Protocols).
* The response **SHALL** include a Connection header field with the value "Upgrade".
* The response **SHALL** include a single Upgrade header field with the value "connect-tcp".

If the request is malformed or impermissible, the proxy **MUST** return a 4XX error code. If a TCP connection was not established, the proxy **MUST NOT** switch protocols to "connect-tcp".

From this point on, the connection **SHALL** conform to all the usual requirements for classic CONNECT proxies in HTTP/1.1 ([RFC9110], Section 9.3.6). Additionally, if the proxy observes a connection error from the client (e.g., a TCP RST, TCP timeout, or TLS error), it **SHOULD** send a TCP RST to the target. If the proxy observes a connection error from the target, it **SHOULD** send a TLS "internal_error" alert to the client, or set the TCP RST bit if TLS is not in use.

```
Client                                                       Proxy

GET /proxy?target_host=192.0.2.1&tcp_port=443 HTTP/1.1
Host: example.com
Connection: Upgrade
Upgrade: connect-tcp

** Proxy establishes a TCP connection to 192.0.2.1:443 **

                            HTTP/1.1 101 Switching Protocols
                            Connection: Upgrade
                            Upgrade: connect-tcp
```

Figure 1: Templated TCP proxy example in HTTP/1.1

3.2. In HTTP/2 and HTTP/3

In HTTP/2 and HTTP/3, the client uses the proxy by issuing an "extended CONNECT" request as follows:

* The :method pseudo-header field **SHALL** be "CONNECT".
* The :protocol pseudo-header field **SHALL** be "connect-tcp".
* The :authority pseudo-header field **SHALL** contain the authority of the proxy.
* The :path and :scheme pseudo-header fields **SHALL** contain the path and scheme of the request URI derived from the proxy's URI Template.

From this point on, the request and response **SHALL** conform to all the usual requirements for classic CONNECT proxies in this HTTP version (see Section 8.5 of [RFC9113] and Section 4.4 of [RFC9114]).

```
HEADERS
:method = CONNECT
:scheme = https
:authority = request-proxy.example
:path = /proxy?target_host=192.0.2.1,2001:db8::1&tcp_port=443
:protocol = connect-tcp
...
```

Figure 2: Templated TCP proxy example in HTTP/2

3.3. Use of 100 (Continue)

This protocol is compatible with the use of an "Expect: 100-continue" request header ([RFC9110], Section 10.1.1) in any HTTP version. The "100 Continue" response confirms receipt of a request at the proxy without waiting for the proxy-destination TCP handshake to succeed or fail. This may be particularly helpful when the destination host is not responding, as TCP handshakes can hang for several minutes before failing.

4. Applicability

4.1. Servers

For server operators, template-driven TCP proxies are particularly valuable in situations where virtual-hosting is needed, or where multiple proxies must share an origin. For example, the proxy might benefit from sharing an HTTP gateway that provides DDoS defense, performs request sanitization, or enforces user authorization.

The URI template can also be structured to generate high-entropy Capability URLs [CAPABILITY], so that only authorized users can discover the proxy service.

4.2. Clients

Clients that support both classic HTTP CONNECT proxies and template-driven TCP proxies **MAY** accept both types via a single configuration string. If the configuration string can be parsed as a URI Template containing the required variables, it is a template-driven TCP proxy. Otherwise, it is presumed to represent a classic HTTP CONNECT proxy.

4.3. Multi-purpose proxies

The names of the variables in the URI Template uniquely identify the capabilities of the proxy. Undefined variables are permitted in URI Templates, so a single template can be used for multiple purposes.

Multipurpose templates can be useful when a single client may benefit from access to multiple complementary services (e.g., TCP and UDP), or when the proxy is used by a variety of clients with different needs.

```
https://proxy.example/{?target_host,tcp_port,target_port,
                        target,ipproto,dns}
```

Figure 3: Example multipurpose template for a combined TCP, UDP, and IP proxy and DoH server

5. Security Considerations

TODO

6. Operational Considerations

Templated TCP proxies can make use of standard HTTP gateways and path-routing to ease implementation and allow use of shared infrastructure. However, current gateways might need modifications to support TCP proxy services. To be compatible, a gateway must:

* support Extended CONNECT.
* convert HTTP/1.1 Upgrade requests into Extended CONNECT.
* allow the Extended CONNECT method to pass through to the origin.
* forward Proxy-* request headers to the origin.

7. IANA Considerations

IF APPROVED, IANA is requested to add the following entry to the HTTP Upgrade Token Registry:

* Value: "connect-tcp"
* Description: Proxying of TCP payloads
* Reference: (This document)

8. References

8.1. Normative References

[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>.

[RFC6570] 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>.

[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>.

[RFC9110] 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>.

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

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

8.2. Informative References

[CAPABILITY] "Good Practices for Capability URLs", February 2014, <https://www.w3.org/TR/capability-urls/>.

[I-D.ietf-masque-connect-ip]
Pauly, T., Schinazi, D., Chernyakhovsky, A., Kühlewind, M., and M. Westerlund, "Proxying IP in HTTP", Work in Progress, Internet-Draft, draft-ietf-masque-connect-ip-08, 1 March 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-masque-connect-ip-08>.

[RFC9112] 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>.

[RFC9297] 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>.

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

Acknowledgments

Thanks to Amos Jeffries for close review.

Author's Address

Benjamin M. Schwartz
Google LLC

Email: ietf@bemasc.net