Workgroup: httpbis
Internet-Draft:
draft-schwartz-modern-http-proxies-00
Published: 14 October 2022
Intended Status: Standards Track
Expires: 17 April 2023
Authors: B. M. Schwartz
Google LLC

Modernizing HTTP Forward Proxy Functionality

Abstract

HTTP proxying features have long been part of the core HTTP specification. However, the core proxying functionality has several important deficiencies in modern HTTP environments. This specification defines alternative proxy service configurations for HTTP requests and TCP connections. These services are identified by URI Templates and designed for parallelism with DoH, MASQUE, and Oblivious HTTP.

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 17 April 2023.

Table of Contents

1. Introduction

1.1. History

An HTTP forward proxy (or just "proxy" in the HTTP standards) is an HTTP service that acts on behalf of the client as an intermediary for some or all HTTP requests. HTTP/1.0 defines the initial HTTP proxying mechanism: the client formats its request target in "absolute form" (i.e. with a full URI in the Request-Line) and delivers it to the proxy, which reissues it to the origin specified in the URI ([RFC1945], Section 5.1.2). In this specification, we call this behavior an "HTTP request proxy".

With the introduction of "https" URIs, a new proxying mechanism was needed to enable TLS connections to traverse the proxy. To enable this, HTTP/1.1 introduced the CONNECT method. In this method, 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). In this specification, we call this behavior a "TCP transport proxy".

These two methods sufficed until the introduction of HTTP/3, which uses a UDP transport. The MASQUE effort has filled the gap by defining 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 request proxies and TCP transport proxies are identified by an origin, not a URI. The proxy service does not have a path of its own. This prevents any origin from hosting multiple distinct proxy services and makes it difficult to manage a proxy service in a fashion similar to other HTTP services.

In some circumstances, it may be possible to work around this limitation by hosting many origins on a single server (virtual-hosting). In HTTP/1.1, the "Host" header was introduced to support such virtual-hosting by distinguishing the hostname of the proxy (in the Host header) from the hostname of the destination (in the absolute-form request URI). However, in HTTP/2 and HTTP/3, this distinction no longer exists. As a result, classic HTTP request proxies are not compatible with virtual-hosting in HTTP/2 or HTTP/3.

Classic TCP transport proxies can be used with a host that is specified as a domain name or an IP address. However, because only a single IP address can be specified, Happy Eyeballs and cross-IP fallback can only be used when the host is a domain name. For requests to succeed, the client must know which address families are supported by the proxy.

1.3. Overview

This specification describes alternative protocols for HTTP request proxies and TCP transport proxies in HTTP. Like other modern HTTP access services such as DoH, CONNECT-UDP, and CONNECT-IP, the proxy 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. Modern HTTP Request Proxies

A modern HTTP request proxy is identified by a URI Template containing a variable named "target_uri". To convert an HTTP request into a proxied request, the client **MUST** substitute the request's URI into this variable, expand the template, and use the result as the new request URI.

HTTP headers work the same as in classic HTTP request proxies.

A modern HTTP request proxy is also suitable for use as an Oblivious HTTP relay, if it provides the required privacy guarantees.

3.1. Example

Consider a proxy identified as "https://example.com/proxy{?target_uri}". Requests would then be transformed as follows:

```
Original request:
PATCH /resource HTTP/1.1
Host: api.example
Content-Type: application/example
...

Transformed request:
PATCH /proxy?target_uri=https%3A%2F%2Fapi.example%2Fresource HTTP/1.1
Host: example.com
Content-Type: application/example
Proxy-Authorization: ...
...
```

Notes on this example:

* The HTTP method is not altered.

* The request-related headers such as Content-Type are preserved, but the Host header (or :authority in HTTP/2 and HTTP/3) is altered.

* Certain characers in the target URI are percent-encoded during URI Template expansion.

* The scheme, which is implicit in the original request, is explicit in the transformed request. The scheme in this example is "https", indicating that the client is asking the proxy to establish a secure connection to the target.

* The client can add Proxy-* headers to communicate with the proxy.

4. Modern TCP transport proxies

A modern 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.

4.1. 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".
- The request **SHALL** include an "Upgrade" header field with the value "connect-tcp".
- The request's target **SHALL** be 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 the TCP connection failed, the proxy **MUST NOT** return a 101 or 2XX status code.

If the proxy observes an unclean shutdown from the client (e.g. a TCP RST or TLS error), it **SHOULD** send a TCP RST to the target. If the proxy receives a TCP RST 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.

4.1.1. Example

Consider a proxy identified as "https://example.com/proxy{?target_host,tcp_port}". To establish a TCP connection to 192.0.2.1:443, the following exchange would occur:

```
Client                                                  Proxy

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

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

Figure 1: Modern TCP transport proxy in HTTP/1.1

4.2. 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 streams **SHALL** conform to all the usual requirements for non-extended CONNECT in this HTTP version.

5. Additional Examples

5.1. Template expansion

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:

```
Combined HTTP request and TCP transport proxy:
https://example.com/proxy{?target_uri,target_host,tcp_port}

Combined HTTP, TCP, UDP, and IP proxy with DoH server:
https://proxy.example/{?target_uri,target_host,tcp_port,port,target,ippr
```

Figure 2: Multipurpose templates

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.

5.2. Sample exchanges

A modern HTTP request proxy can be used as an Oblivious HTTP Relay. For example, suppose the relay is identified as "https://proxy.example.org/relay{?target_uri}", and the Oblivious HTTP Gateway is "https://example.com/gateway". The client would send requests to the proxy as follows:

```
POST /relay?target_uri=https%3A%2F%2Fexample.com%2Fgateway HTTP/1.1
Host: proxy.example.org
Proxy-Authorization: ...
Content-Type: message/ohttp-req
...
```

Figure 3: Use of an HTTP request proxy as an Oblivious relay

If a modern HTTP request proxy supports HTTP/2 and Extended CONNECT, it is even possible to reach a modern TCP transport proxy through it:

```
CONNECT HTTP/2.0
:authority = request-proxy.example
:scheme = https
:path = /proxy?target_uri=https%3A%2F%2Ftransport-proxy.example%2Fproxy
    %3Ftarget_host%3Ddestination.example%26tcp_port%3D443
:protocol = connect-tcp
...
```

Figure 4: Use of a TCP transport proxy through an HTTP request proxy

Modern TCP transport proxies support requests that offer multiple IP addresses:

```
CONNECT HTTP/2.0
:authority = request-proxy.example
:scheme = https
:path = /proxy?target_host=192.0.2.1,2001:db8::1&port=443
:protocol = connect-tcp
...
```

Figure 5: TCP transport proxy request with multiple IP addresses

6. Security Considerations

None

7. Operational considerations

Modern HTTP 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 these services. A compatible gateway must:

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

8. 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)

9. References

9.1. 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>.

9.2. Informative References

[I-D.ietf-masque-connect-ip] Pauly, T., Schinazi, D., Chernyakhovsky, A., Kühlewind, M., and M. Westerlund, "IP Proxying Support for HTTP", Work in Progress, Internet-Draft, draft-ietf-masque-connect-ip-03, 27 September 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-masque-connect-ip-03>.

[RFC1945] Berners-Lee, T., Fielding, R., and H. Frystyk, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, DOI 10.17487/RFC1945, May 1996, <https://www.rfc-editor.org/rfc/rfc1945>.

[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>.

[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

TODO acknowledge.

Author's Address

Benjamin M. Schwartz
Google LLC

Email: bemasc@google.com