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

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 9 June 2024.

Copyright Notice

Copyright (c) 2023 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
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 [CONNECT-UDP], and more generally IP datagrams.
[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

HTTP clients can be configured to use proxies by selecting a proxy hostname, a port, and whether to use a security protocol. However, CONNECT requests using the proxy do not carry this configuration information. Instead, they only indicate the hostname and port of the target. This prevents any HTTP server from hosting multiple distinct proxy services, as the server cannot distinguish them by path (as with distinct resources) or by origin (as in "virtual hosting").

The absence of an explicit origin for the proxy also rules out the usual defenses against server port misdirection attacks (see Section 7.4 of [RFC9110]) and creates ambiguity about the use of origin-scoped response header fields (e.g., "Alt-Svc" [RFC7838], "Strict-Transport-Security" [RFC6797]).

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 [CONNECT-UDP] and [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", and the client **MAY** reuse this connection for additional HTTP requests.

After a success response is returned, 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

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
...
3.3. Use of Relevant Headers

3.3.1. Origin-scoped Headers

Ordinary HTTP headers apply only to the single resource identified in the request or response. An origin-scoped HTTP header is a special response header that is intended to change the client's behavior for subsequent requests to any resource on this origin.

Unlike classic HTTP CONNECT proxies, a templated TCP proxy has an unambiguous origin of its own. Origin-scoped headers apply to this origin when they are associated with a templated TCP proxy response. Here are some origin-scoped headers that could potentially be sent by a templated TCP proxy:

- "Alt-Svc" [RFC7838]
- "Strict-Transport-Security" [RFC6797]
- "Public-Key-Pins" [RFC7469]
- "Accept-CH" [RFC8942]
- "Set-Cookie" [RFC6265], which has configurable scope.
- "Clear-Site-Data" [CLEAR-SITE-DATA]

3.3.2. Authentication Headers

Authentication to a templated TCP proxy normally uses ordinary HTTP authentication via the "401 (Unauthorized)" response code, the "WWW-Authenticate" response header field, and the "Authorization" request header field ([RFC9110], Section 11.6). A templated TCP proxy does not use the "407 (Proxy Authentication Required)" response code and related header fields ([RFC9110], Section 11.7) because they do not traverse HTTP gateways (see Section 7).

Clients SHOULD assume that all proxy resources generated by a single template share a protection space (i.e., a realm) ([RFC9110], Section 11.5). For many authentication schemes, this will allow the client to avoid waiting for a "401 (Unauthorized)" response before each new connection through the proxy.

4. Additional Connection Setup Behaviors

This section discusses some behaviors that are permitted or recommended in order to enhance the performance or functionality of connection setup.
4.1. Latency optimizations

When using this specification in HTTP/2 or HTTP/3, clients MAY start sending TCP stream content optimistically, subject to flow control limits (Section 5.2 of [RFC9113] or Section 4.1 of [RFC9000]). Proxies MUST buffer this "optimistic" content until the TCP stream becomes writable, and discard it if the TCP connection fails. (This "optimistic" behavior is not permitted in HTTP/1.1 because it would interfere with reuse of the connection after an error response such as "401 (Unauthorized)").

Servers that host a proxy under this specification MAY offer support for TLS early data in accordance with [RFC8470]. Clients MAY send "connect-tcp" requests in early data, and MAY include "optimistic" TCP content in early data (in HTTP/2 and HTTP/3). At the TLS layer, proxies MAY ignore, reject, or accept the early_data extension ([RFC8446], Section 4.2.10). At the HTTP layer, proxies MAY process the request immediately, return a "425 (Too Early)" response ([RFC8470], Section 5.2), or delay some or all processing of the request until the handshake completes. For example, a proxy with limited anti-replay defenses might choose to perform DNS resolution of the target_host when a request arrives in early data, but delay the TCP connection until the TLS handshake completes.

4.2. Conveying metadata

This specification supports the "Expect: 100-continue" request header ([RFC9110], Section 10.1.1) in any HTTP version. The "100 (Continue)" status code confirms receipt of a request at the proxy without waiting for the proxy-destination TCP handshake to succeed or fail. This might be particularly helpful when the destination host is not responding, as TCP handshakes can hang for several minutes before failing. Implementation of "100 (Continue)" support is OPTIONAL for clients and REQUIRED for proxies.

Proxies implementing this specification SHOULD include a "Proxy-Status" response header [RFC9209] in any success or failure response (i.e., status codes 101, 2XX, 4XX, or 5XX) to support advanced client behaviors and diagnostics. In HTTP/2 or HTTP/3, proxies MAY additionally send a "Proxy-Status" trailer in the event of an unclean shutdown.

5. Applicability

5.1. 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.

5.2. 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.

In some cases, it is valuable to allow "connect-tcp" clients to reach "connect-tcp"-only proxies when using a legacy configuration method that cannot convey a URI template. To support this arrangement, clients SHOULD treat certain errors during classic HTTP CONNECT as indications that the proxy might only support "connect-tcp":

* In HTTP/1.1: the response status code is "426 (Upgrade Required)", with an "Upgrade: connect-tcp" response header.

* In any HTTP version: the response status code is "501 (Not Implemented)".

- Requires SETTINGS_ENABLE_CONNECT_PROTOCOL to have been negotiated in HTTP/2 or HTTP/3.

If the client infers that classic HTTP CONNECT is not supported, it SHOULD retry the request using the registered default template for "connect-tcp":

```
https://$PROXY_HOST:$PROXY_PORT/.well-known/masque/tcp/{target_host}/{tcp_port}/
```

Figure 3: Registered default template

If this request succeeds, the client SHOULD record a preference for "connect-tcp" to avoid further retry delays.

5.3. 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 4: Example multipurpose template for a combined TCP, UDP, and IP proxy and DoH server

6. Security Considerations

Template-driven TCP proxying is largely subject to the same security risks as classic HTTP CONNECT. For example, any restrictions on authorized use of the proxy (see [RFC9110], Section 9.3.6) apply equally to both.

A small additional risk is posed by the use of a URI Template parser on the client side. The template input string could be crafted to exploit any vulnerabilities in the parser implementation. Client implementers should apply their usual precautions for code that processes untrusted inputs.

7. 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 (if acting as an HTTP/2 or HTTP/3 server).

* support HTTP/1.1 Upgrade to "connect-tcp" (if acting as an HTTP/1.1 server)

  - only after forwarding the upgrade request to the origin and observing a success response.

* forward the "connect-tcp" protocol to the origin.

* convert "connect-tcp" requests between all supported HTTP server and client versions.

* allow any "Proxy-Status" headers to traverse the gateway.
8. IANA Considerations

8.1. New Upgrade Token

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)

8.2. New MASQUE Default Template

IF APPROVED, IANA is requested to add the following entry to the "MASQUE URI Suffixes" registry:

<table>
<thead>
<tr>
<th>Path Segment</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcp</td>
<td>TCP Proxying</td>
<td>(This document)</td>
</tr>
</tbody>
</table>

Table 1

9. References

9.1. Normative References


9.2. Informative References


Acknowledgments

Thanks to Amos Jeffries, Tommy Pauly, Kyle Nekritz, David Schinazi, and Kazuho Oku for close review and suggested changes.

Author's Address

Benjamin M. Schwartz
Meta Platforms, Inc.

Email: ietf@bemasc.net