httpbis
Internet-Draft
Intended status: Standards Track
Expires: 8 June 2026

B. M. Schwartz
Meta Platforms, Inc.

5 December 2025

Template-Driven HTTP CONNECT Proxying for TCP
draft-ietf-httpbis-connect-tcp-10

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 8 June 2026.

Copyright Notice

Copyright (c) 2025 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 Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

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 [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, Classic HTTP 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 requests cannot carry in-stream metadata. For example, the WRAP_UP capsule [I-D.schinazi-httpbis-wrap-up] cannot be used with Classic HTTP CONNECT.

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 "target_port". This URI Template and its variable values MUST meet all the same requirements as for UDP proxying ([RFC9298], Section 2), and are subject to the same validation rules. The client MUST substitute the destination host and port number into this template to produce the request URI. The derived URI serves as the destination of a Capsule Protocol connection using the Upgrade Token "connect-tcp" (see registration in Section 8.1).

When using "connect-tcp", TCP payload data is sent in the payload of new Capsule Types named DATA and FINAL_DATA (see registrations in Section 8.3). The ordered concatenation of these capsule payloads represents the TCP payload data. A FINAL_DATA capsule additionally indicates that the sender has closed this stream, semantically equivalent to TCP FIN. After sending a FINAL_DATA capsule, an endpoint MUST NOT send any more DATA or FINAL_DATA capsules on this data stream. (See Section 3.4 for related requirements.)

An intermediary MAY merge and split successive DATA and FINAL_DATA capsules, subject to the following requirements:

* There are no intervening capsules of other types.

* The order of payload content is preserved.

* The final emitted capsule uses the same capsule type (DATA or FINAL_DATA) as the final input capsule, and all others use the DATA capsule type.

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’s target SHALL correspond to the URI derived from expansion of the proxy’s URI Template.

* 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 SHOULD include a "Capsule-Protocol: ?1" header (as recommended in [RFC9297], Section 3.4).

If the request is well-formed and permissible, the proxy MUST attempt to establish the TCP connection before sending any response status code other than "100 (Continue)" (see Section 4.2). 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".

* The response SHOULD include a "Capsule-Protocol: ?1" header (as above).

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.

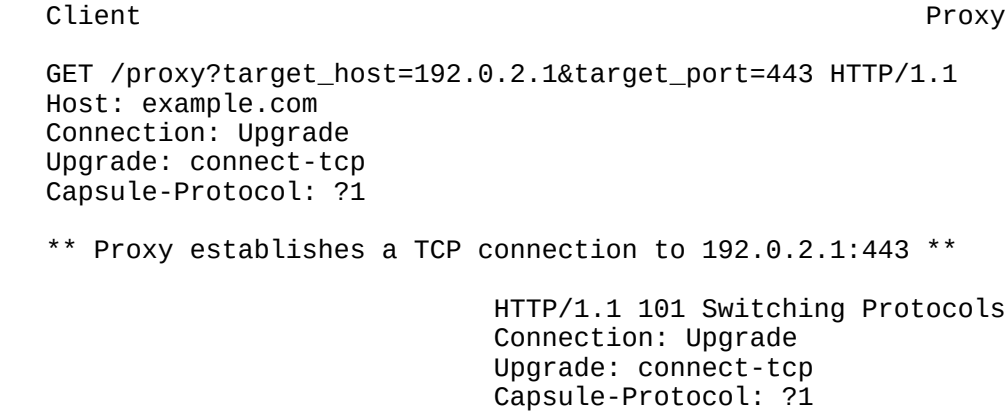

Figure 1: Templated TCP proxy example in HTTP/1.1

3.2. In HTTP/2 and HTTP/3

In HTTP/2 and HTTP/3, the proxy MUST include SETTINGS_ENABLE_CONNECT_PROTOCOL in its SETTINGS frame [RFC8441][RFC9220]. 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.

A templated TCP proxying request that does not conform to all of these requirements represents a client error (see [RFC9110], Section 15.5) and may be malformed (see Section 8.1.1 of [RFC9113] and Section 4.1.2 of [RFC9114]).

Additionally the "capsule-protocol" header field SHOULD be present with a value of "?1" (as recommended in [RFC9297], Section 3.4).

```
HEADERS
:method = CONNECT
:scheme = https
:authority = request-proxy.example
:path = /proxy?target_host=2001%3Adb8%3A%3A1&target_port=443
:protocol = connect-tcp
capsule-protocol = ?1
...
```

Figure 2: Templated TCP proxy example in HTTP/2

3.3. Use of Other 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.

3.4. Closing Connections

Connection termination is essentially symmetrical for proxies and their clients. In this section, we use the term "endpoint" to describe an implementation of this specification in either role.

When closing connections, endpoints are subject to the following requirements:

* When an endpoint receives a valid TCP FIN, it MUST send a FINAL_DATA capsule.

* When an endpoint receives a valid FINAL_DATA capsule, it MUST send a TCP FIN.

* When a TCP connection reaches the TIME-WAIT or CLOSED state, the associated endpoint MUST close its send stream.

  - If the connection closed gracefully, the endpoint MUST close the send stream gracefully.

  - Otherwise, the endpoint SHOULD close the send stream abruptly, using a mechanism appropriate to the HTTP version:

    o HTTP/3: reset the stream with H3_CONNECT_ERROR; see [RFC9000], Section 19.4 and [RFC9114], Section 8.1

    o HTTP/2: reset the stream with CONNECT_ERROR; see [RFC9113], Sections 6.4 and 7

    o HTTP/1.1 over TLS: TCP shutdown without a TLS closure alert; see [RFC8446], Section 6.1.

    o HTTP/1.1 without TLS: TCP RST

* When the receive stream is closed abruptly or without a FINAL_DATA capsule received, the endpoint SHOULD send a TCP RST if the TCP subsystem permits it.

The mandatory behaviors above enable endpoints to detect any truncation of incoming TCP data. The recommended behaviors propagate any TCP errors through the proxy connection.

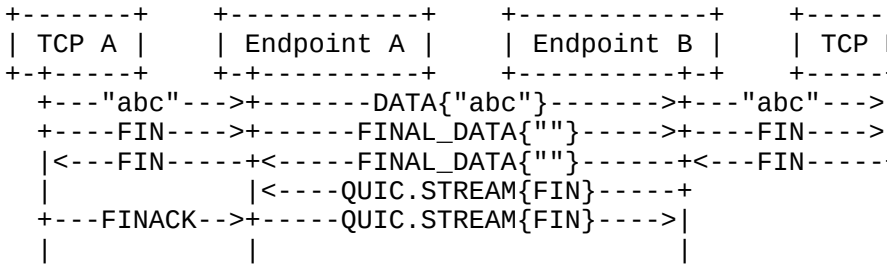

Figure 3: Simple graceful termination example (HTTP/3)

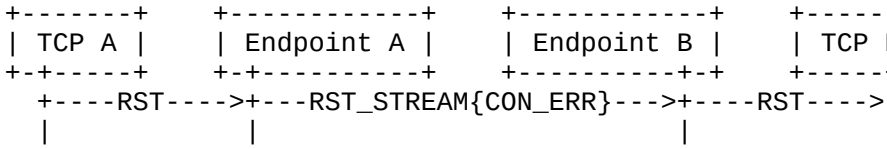

Figure 4: Simple TCP RST termination example (HTTP/2)

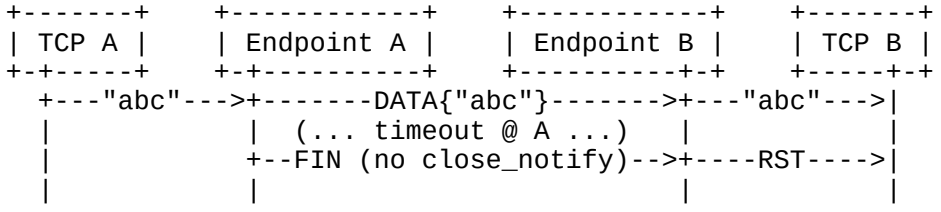

Figure 5: Timeout example (HTTP/1.1)

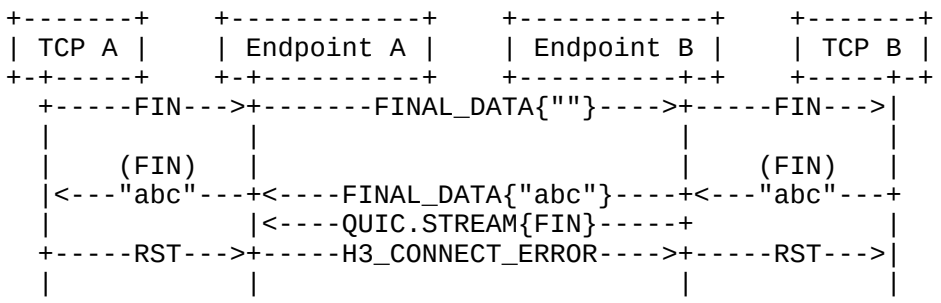

Figure 6: RST after FIN example (HTTP/3)

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. (Clients MUST NOT use "optimistic" behavior in HTTP/1.1, as this 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. Clients MAY send "Expect: 100-continue", and proxies MUST respect it by returning "100 (Continue)" if the request is not immediately rejected.

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 abrupt stream closure.

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}/{target_port}/
```

Figure 7: Registered default template

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

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.

6.1. Resource Exhaustion attacks

A malicious client can achieve cause highly asymmetric resource usage at the proxy by colluding with a destination server and violating the ordinary rules of TCP or HTTP. Some example attacks, and mitigations that proxies can apply:

* *Connection Pileup*: A malicious client can attempt to open a large number of connections to exhaust the proxy's memory, port, or file descriptor limits. When using HTTP/2 or HTTP/3, each incremental TCP connection imposes a much higher cost on the proxy than on the attacker.

  - Mitigation: Limit the number of concurrent connections per client.

* *Window Bloat*: An attacker can grow the receive window size by simulating a "long, fat network" [RFC7323], then fill the window (from the sender) and stop acknowledging it (at the receiver). This leaves the proxy buffering up to 1 GiB of TCP data until some timeout, while the attacker does not have to retain a large buffer.

  - Mitigation: Limit the maximum receive window for TCP and HTTP connections, and the size of userspace buffers used for proxying. Alternatively, monitor the connections' send queues and limit the total buffered data per client.

* *WAIT Abuse*: An attacker can force the proxy into a TIME-WAIT, CLOSE-WAIT, or FIN-WAIT state until the timer expires, tying up a proxy-to-destination 4-tuple for up to four minutes after the client's connection is closed.

  - Mitigation: Limit the number of connections for each client to each destination, even if those connections are in a waiting state and the corresponding CONNECT stream is closed. Alternatively, allocate a large range of IP addresses for TCP connections (especially in IPv6).

7. Operational Considerations

7.1. Avoiding HTTP/1.1

While this specification is fully functional under HTTP/1.1, performance-sensitive deployments SHOULD use HTTP/2 or HTTP/3 instead. When using HTTP/1.1:

* Each CONNECT request requires a new TCP and TLS connection, imposing a higher cost in setup latency, congestion control convergence, CPU time, and data transfer.

* The graceful and abrupt closure signals (Section 3.4) are more likely to be missing or corrupted:

  - Some implementations may be unable to emit the recommended abrupt closure signals, due to limitations in their TCP and TLS subsystems.

  - Faulty implementations may fail to send a TLS closure alert during graceful shutdown, or fail to report an error when the expected closure alert is not received. These misbehaviors are not compliant with [RFC8446], but they are common nonetheless among HTTP/1.1 implementations today.

* The number of active connections through each client may be limited by the number of available TCP client ports, especially if:

  - The client only has one IP address that can be used to reach the proxy.

  - The client is shared between many parties, such as when acting as a gateway or concentrator.

  - The proxied connections are often closed by the destination. This causes the client to initiate closure of the client-to-proxy connection, leaving the client in a TIME-WAIT state for up to four minutes.

7.2. Gateway Compatibility

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

Table 1

For interoperability testing of this draft version, implementations SHALL use the value "connect-tcp-07".

8.2. New MASQUE Default Template

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

Path Segment	Description	Reference
tcp	TCP Proxying	(This document)

Table 2

8.3. New Capsule Type

IF APPROVED, IANA is requested to add the following entry to the "HTTP Capsule Types" registry:

Value	Capsule Type	Status	Reference	Change Controller	Contact
(TBD)	DATA	permanent	(This document), Section 3	IETF	HTTPBIS
(TBD)	FINAL_DATA	permanent	(This document), Section 3	IETF	HTTPBIS

Table 3

For this draft version of the protocol, the Capsule Type values 0x2028d7f0 and 0x2028d7f1 shall be used provisionally for testing, under the names "DATA-08" and "FINAL_DATA-08".

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>.

[RFC8441] McManus, P., "Bootstrapping WebSockets with HTTP/2", RFC 8441, DOI 10.17487/RFC8441, September 2018, <https://www.rfc-editor.org/rfc/rfc8441>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/rfc/rfc8446>.

[RFC8470] Thomson, M., Nottingham, M., and W. Tarreau, "Using Early Data in HTTP", RFC 8470, DOI 10.17487/RFC8470, September 2018, <https://www.rfc-editor.org/rfc/rfc8470>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>.

[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>.

[RFC9209] Nottingham, M. and P. Sikora, "The Proxy-Status HTTP Response Header Field", RFC 9209, DOI 10.17487/RFC9209, June 2022, <https://www.rfc-editor.org/rfc/rfc9209>.

[RFC9220] Hamilton, R., "Bootstrapping WebSockets with HTTP/3", RFC 9220, DOI 10.17487/RFC9220, June 2022, <https://www.rfc-editor.org/rfc/rfc9220>.

[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>.

9.2. Informative References

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

[CLEAR-SITE-DATA] "Clear Site Data", November 2017, <https://www.w3.org/TR/clear-site-data/>.

[CONNECT-IP] Pauly, T., Ed., Schinazi, D., Chernyakhovsky, A., K端hlewind, M., and M. Westerlund, "Proxying IP in HTTP", RFC 9484, DOI 10.17487/RFC9484, October 2023, <https://www.rfc-editor.org/rfc/rfc9484>.

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

[I-D.schinazi-httpbis-wrap-up]
Schinazi, D. and L. Pardue, "The HTTP Wrap Up Capsule", Work in Progress, Internet-Draft, draft-schinazi-httpbis-wrap-up-01, 16 October 2024, <https://datatracker.ietf.org/doc/html/draft-schinazi-httpbis-wrap-up-01>.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <https://www.rfc-editor.org/rfc/rfc6265>.

[RFC6797] Hodges, J., Jackson, C., and A. Barth, "HTTP Strict Transport Security (HSTS)", RFC 6797, DOI 10.17487/RFC6797, November 2012, <https://www.rfc-editor.org/rfc/rfc6797>.

[RFC7323] Borman, D., Braden, B., Jacobson, V., and R. Scheffenegger, Ed., "TCP Extensions for High Performance", RFC 7323, DOI 10.17487/RFC7323, September 2014, <https://www.rfc-editor.org/rfc/rfc7323>.

[RFC7469] Evans, C., Palmer, C., and R. Sleevi, "Public Key Pinning Extension for HTTP", RFC 7469, DOI 10.17487/RFC7469, April 2015, <https://www.rfc-editor.org/rfc/rfc7469>.

[RFC7838] Nottingham, M., McManus, P., and J. Reschke, "HTTP Alternative Services", RFC 7838, DOI 10.17487/RFC7838, April 2016, <https://www.rfc-editor.org/rfc/rfc7838>.

[RFC8942] Grigorik, I. and Y. Weiss, "HTTP Client Hints", RFC 8942, DOI 10.17487/RFC8942, February 2021, <https://www.rfc-editor.org/rfc/rfc8942>.

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