HTTPBIS
Internet-Draft
Intended status: Standards Track
Expires: 8 January 2026

D. Schinazi
Google LLC
L. Pardue
Cloudflare

7 July 2025

The HTTP Wrap Up Capsule
draft-ietf-httpbis-wrap-up-01

Abstract

HTTP intermediaries sometimes need to terminate long-lived request streams in order to facilitate load balancing or impose data limits. However, Web browsers commonly cannot retry failed proxied requests when they cannot ascertain whether an in-progress request was acted on. To avoid user-visible failures, it is best for the intermediary to inform the client of upcoming request stream terminations in advance of the actual termination so that the client can wrap up existing operations related to that stream and start sending new work to a different stream or connection. This document specifies a new "WRAP_UP" capsule that allows a proxy to instruct a client that it should not start new requests on a tunneled connection, while still allowing it to finish existing requests.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://httpwg.org/http-extensions/draft-ietf-httpbis-wrap-up.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-httpbis-wrap-up/.

Discussion of this document takes place on the HTTP Working Group mailing list (mailto:ietf-http-wg@w3.org), which is archived at https://lists.w3.org/Archives/Public/ietf-http-wg/. Working Group information can be found at https://httpwg.org/.

Source for this draft and an issue tracker can be found at https://github.com/httpwg/http-extensions/labels/wrap-up.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

This Internet-Draft will expire on 8 January 2026.

Copyright Notice

Table of Contents

1. Introduction 3
  1.1. The Need for a Request Termination Intent Signal 3
  1.2. The WRAP_UP Capsule 5
  1.3. Conventions and Definitions 6
2. Mechanism 6
  2.1. Proxy Behavior 6
  2.2. Client Handling 6
  2.3. Minutiae 7
3. Security Considerations 7
4. IANA Considerations 7
5. References 7
  5.1. Normative References 7
  5.2. Informative References 8
Acknowledgments 8
Authors' Addresses 9

1. Introduction

[HTTP/1.1], [HTTP/2] and [HTTP/3] all have the notion of persistent connections, where a single connection can carry multiple request and response messages. While it is expected that the connection persists, there are situations where a client or server may wish to terminate the connection gracefully.

An HTTP/1.1 connection can be terminated by using a Connection header field with the close option; see Section 9.6 of [HTTP/1.1]. When a connection has short-lived requests/responses, this mechanism allows timely and non-disruptive connection termination. However, when requests/responses are longer lived, the opportunity to use headers happens less frequently (or not at all). There is no way for client or server to signal a future intent to terminate the connection. Instead, an abrupt termination, realized via a transport-layer close or reset, is required, which is potentially disruptive and can lead to truncated content.

HTTP/2 and HTTP/3 support request multiplexing, making header-based connection lifecycle control impractical. Connection headers are prohibited entirely. Instead, a shutdown process using the GOAWAY frame is defined (see Section 6.8 of [HTTP/2] and Section 5.2 of [HTTP/3]). GOAWAY signals a future intent to terminate the connection, supporting cases such as scheduled maintenance. Active requests/responses can continue to run, while new requests need to be sent on a new HTTP connection. Endpoints that use GOAWAY typically have a grace period in which requests/responses can run naturally to completion. If they run longer than the grace period, they are abruptly terminated when the transport layer is closed or reset, which is potentially disruptive and can lead to truncated content.

1.1. The Need for a Request Termination Intent Signal

Intermediaries (see Section 3.7 of [HTTP]) can provide a variety of benefits to HTTP systems, such as load balancing, caching, and privacy improvements. Deployments of intermediaries also need to be maintained, which can sometimes require taking intermediaries temporarily offline. For example, if a gateway has a client HTTP/2 connection and needs to go down for maintenance, it can send a GOAWAY to stop the client issuing requests that would be forwarded to the origin.

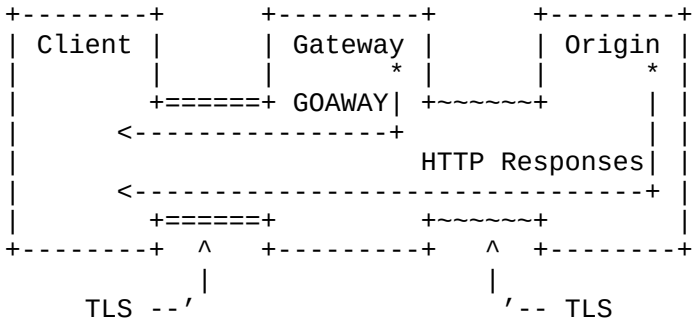

Figure 1: Gateway Sends GOAWAY

The connection close details described above apply to an intermediary's upstream and downstream connections. Since a proxy can do request aggregation or fan out, there is no guarantee of a 1:1 ratio of upstream/downstream. As such, the lifetimes of these connections are not coupled tightly. For example, a gateway can terminate a client HTTP/2 connections and map individual requests to an origin HTTP/1.1 connection pool. If any single origin connection indicates an intent to close, it doesn't make sense for the gateway to issue a GOAWAY to the client, or to respond to a client GOAWAY by closing connections in the pool.

Long-lived requests pose a problem for maintenance, especially for HTTP/2 and HTTP/3, and even more so for intermediaries. Sometimes they need to terminate individual request streams in order to facilitate load balancing or impose data limits, while leaving the connection still active. GOAWAY is not suitable for this task.

Some applications using HTTP have their own control plane running over HTTP, that could be used for a graceful termination. For example, WebSockets has separate control and data frames. The Close frame (Section 5.5.1 of [WEBSOCKET]) is used for the WebSocket close sequence. However, in the maintenance scenario, an intermediary that is not WebSocket aware cannot use the formal sequence. Nor is there any standard for it to signal to the endpoints to initiate that sequence. Some intermediaries are WebSocket aware, and in theory could send Close frames. However, there can be other considerations that prevent this working effectively in real deployments, since the intermediary is a generic proxy that may invalidate endpoint expectations.

Many long-lived HTTP request types do not have control messages that could signal an intent to terminate the request. For example, see CONNECT (Section 9.3.6 of [HTTP]) or connect-udp ({?CONNECT-UDP=RFC9298}}). In these models, the client requests that a proxy

create a tunnel to a target origin. On success, the newly established tunnel is used as the underlying transport to then establish a second HTTP connection directly to the origin. In that situation, the proxy cannot inspect the contents of the tunnel, nor inject any data into it; the proxy only sees a single long-lived request. The proxy is responsible for managing the lifetime of the tunnel, but can only terminate it abortively. Such abrupt termination can lead to truncated content, which the client cannot safely request again. This is especially disruptive if the tunnelled HTTP connection has many active requests. Web browsers, for example, commonly cannot retry failed proxied requests when they cannot ascertain whether an in-progress request was acted on.

To avoid user-visible failures, it is best for the proxy to inform the client of upcoming request stream terminations in advance of the actual termination. This allows the client to wrap up existing operations related to that stream and start sending new work to a different stream or connection.

1.2. The WRAP_UP Capsule

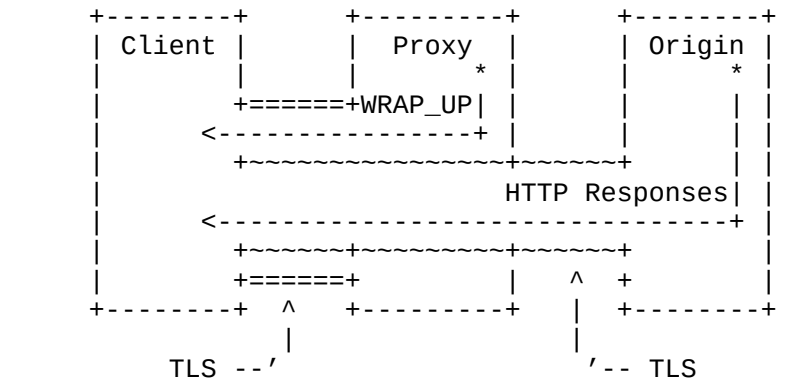

Figure 2: Proxy Sends WRAP_UP

This document specifies a new "WRAP_UP" capsule (see Section 3 of [HTTP-DGRAM]), which a server can send on an HTTP Data Stream, to inform a client that it intends to close the stream.

An HTTP proxy can send a WRAP_UP capsule to instruct a client that it should not start new requests on a tunneled connection, while still allowing it to finish existing requests.

1.3. 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.

This document uses the terms "connection", "client", and "server" from Section 3.3 of [HTTP] and the terms "intermediary", "proxy", and "gateway" from Section 3.7 of [HTTP].

2. Mechanism

This document defines the "WRAP_UP" capsule (see Section 4 for the value). The WRAP_UP capsule allows a proxy to indicate to a client that it wishes to close the request stream that the capsule was sent on. The WRAP_UP capsule has no Capsule Value.

2.1. Proxy Behavior

Proxies often know in advance that they will close a request stream. This can be due to maintenance of the proxy itself, load balancing, or applying data usage limits on a particular stream. When a proxy has advance notice that it will soon close a request stream, it MAY send a WRAP_UP capsule to share that information with the client. This can also be used when the proxy wishes to release resources associated with a request stream, such as HTTP Datagrams (see Section 2 of [HTTP-DGRAM]) or WebTransport streams (see [WEBTRANSPORT]).

2.2. Client Handling

When a client receives a WRAP_UP capsule on a request stream, it SHOULD try to wrap up its use of that stream. For example, if the stream carried a connect-udp request and is used as the underlying transport for an HTTP/3 connection, then after receiving a WRAP_UP capsule the client SHOULD NOT send any new requests on the proxied HTTP/3 connection - but existing in-progress proxied requests are not affected by the WRAP_UP capsule.

2.3. Minutiae

Clients MUST NOT send the WRAP_UP capsule. If a server receives a WRAP_UP capsule, it MUST abort the corresponding request stream. Endpoints MUST NOT send the WRAP_UP capsule with a non-zero Capsule Length. If an endpoint receives a WRAP_UP capsule with a non-zero Capsule Length, it MUST abort the corresponding request stream. Proxies MUST NOT send more than one WRAP_UP capsule on a given stream. If a client receives a second WRAP_UP capsule on a given request stream, it MUST abort the stream. Endpoints that abort the request stream due to an unexpected or malformed WRAP_UP capsule MUST follow the error-handling procedure defined in Section 3.3 of [HTTP-DGRAM].

3. Security Considerations

While it might be tempting for clients to implement the WRAP_UP capsule by treating it as if they had received a GOAWAY inside the encryption of the end-to-end connection, doing so has security implications. GOAWAY carries semantics around which requests have been acted on, and those can have security implications. Since WRAP_UP is sent by a proxy outside of the end-to-end encryption, it cannot be trusted to ascertain whether any requests were handled by the origin. Because of this, client implementations can only use receipt of WRAP_UP as a hint and MUST NOT use it to make determinations on whether any requests were handled by the origin or not.

4. IANA Considerations

This document (if approved) will request IANA to add the following entry to the "HTTP Capsule Types" registry maintained at <https://www.iana.org/assignments/masque>.

Value: 0x272DDA5E (will be changed to a lower value if this document is approved)
Capsule Type: WRAP_UP
Status: provisional (will be permanent if this document is approved)
Reference: This document
Change Controller: IETF
Contact: ietf-http-wg@w3.org
Notes: None

5. References

5.1. Normative References

[HTTP] 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>.

[HTTP-DGRAM] 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>.

[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>.

[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>.

5.2. Informative References

[HTTP/1.1] 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>.

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

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

[WEBSOCKET] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <https://www.rfc-editor.org/rfc/rfc6455>.

[WEBTRANSPORT] Frindell, A., Kinnear, E., and V. Vasiliev, "WebTransport over HTTP/3", Work in Progress, Internet-Draft, draft-ietf-webtrans-http3-12, 3 March 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-webtrans-http3-12>.

Acknowledgments

This mechanism was inspired by the GOAWAY frame from HTTP/2.

Authors' Addresses

David Schinazi
Google LLC
1600 Amphitheatre Parkway
Mountain View, CA 94043
United States of America
Email: dschinazi.ietf@gmail.com

Lucas Pardue
Cloudflare
Email: lucas@lucaspardue.com