HTTP K. Oku
Internet-Draft Fastly
Intended status: Standards Track L. Pardue
Expires: September 6, 2020 Cloudflare
March 05, 2020

Extensible Prioritization Scheme for HTTP
draft-ietf-httpbis-priority-00

Abstract

This document describes a scheme for prioritizing HTTP responses. This scheme expresses the priority of each HTTP response using absolute values, rather than as a relative relationship between a group of HTTP responses.

This document defines the Priority header field for communicating the initial priority in an HTTP version-independent manner, as well as HTTP/2 and HTTP/3 frames for reprioritizing the responses. These share a common format structure that is designed to provide future extensibility.

Note to Readers

RFC EDITOR: please remove this section before publication

Discussion of this draft takes place on the HTTP working group mailing list (ietf-http-wg@w3.org), which is archived at https://lists.w3.org/Archives/Public/ietf-http-wg/ [1].

Working Group information can be found at https://httpwg.org/ [2]; source code and issues list for this draft can be found at https://github.com/httpwg/http-extensions/labels/priorities [3].

Table of Contents

1. Introduction

It is common for an HTTP ([RFC7230]) resource representation to have relationships to one or more other resources. Clients will often discover these relationships while processing a retrieved representation, leading to further retrieval requests. Meanwhile, the nature of the relationship determines whether the client is blocked from continuing to process locally available resources. For example, visual rendering of an HTML document could be blocked by the retrieval of a CSS file that the document refers to. In contrast, inline images do not block rendering and get drawn incrementally as the chunks of the images arrive.

To provide meaningful presentation of a document at the earliest moment, it is important for an HTTP server to prioritize the HTTP responses, or the chunks of those HTTP responses, that it sends.

HTTP/2 ([RFC7540]) provides such a prioritization scheme. A client sends a series of PRIORITY frames to communicate to the server a "priority tree"; this represents the client's preferred ordering and weighted distribution of the bandwidth among the HTTP responses. However, the design and implementation of this scheme has been observed to have shortcomings, explained in Section 2.

This document defines the Priority HTTP header field that can be used by both client and server to specify the precedence of HTTP responses in a standardized, extensible, protocol-version-independent, end-to-end format. Along with the protocol-version-specific frame for reprioritization, this prioritization scheme acts as a substitute for the original prioritization scheme of HTTP/2.

1.1. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The terms sh-token and sh-boolean are imported from [STRUCTURED-HEADERS].

Example HTTP requests and responses use the HTTP/2-style formatting from [RFC7540].

This document uses the variable-length integer encoding from [I-D.ietf-quic-transport].

2. Motivation for Replacing HTTP/2 Priorities

An important feature of any implementation of a protocol that provides multiplexing is the ability to prioritize the sending of information. This was an important realization in the design of HTTP/2. Prioritization is a difficult problem, so it will always be suboptimal, particularly if one endpoint operates in ignorance of the needs of its peer.

HTTP/2 introduced a complex prioritization signaling scheme that used a combination of dependencies and weights, formed into an unbalanced tree. This scheme has suffered from poor deployment and interoperability.

The rich flexibility of client-driven HTTP/2 prioritization tree building is rarely exercised. Experience has shown that clients tend to choose a single model optimized for a web use case and experiment within the model constraints, or do nothing at all. Furthermore, many clients build their prioritization tree in a unique way, which makes it difficult for servers to understand their intent and act or intervene accordingly.

Many HTTP/2 server implementations do not include support for the priority scheme, some favoring instead bespoke server-driven schemes based on heuristics and other hints, like the content type of resources and the request generation order. For example, a server, with knowledge of the document structure, might want to prioritize the delivery of images that are critical to user experience above other images, but below the CSS files. Since client trees vary, it is impossible for the server to determine how such images should be prioritized against other responses.

The HTTP/2 scheme allows intermediaries to coalesce multiple client trees into a single tree that is used for a single upstream HTTP/2 connection. However, most intermediaries do not support this. The scheme does not define a method that can be used by a server to express the priority of a response. Without such a method, intermediaries cannot coordinate client-driven and server-driven priorities.

HTTP/2 describes denial-of-service considerations for implementations. On 2019-08-13 Netflix issued an advisory notice

about the discovery of several resource exhaustion vectors affecting multiple HTTP/2 implementations. One attack, [CVE-2019-9513] aka "Resource Loop", is based on manipulation of the priority tree.

The HTTP/2 scheme depends on in-order delivery of signals, leading to challenges in porting the scheme to protocols that do not provide global ordering. For example, the scheme cannot be used in HTTP/3 [I-D.ietf-quic-http] without changing the signal and its processing.

Considering the problems with deployment and adaptability to HTTP/3, retaining the HTTP/2 priority scheme increases the complexity of the entire system without any evidence that the value it provides offsets that complexity. In fact, multiple experiments from independent research have shown that simpler schemes can reach at least equivalent performance characteristics compared to the more complex HTTP/2 setups seen in practice, at least for the web use case.

2.1. Disabling HTTP/2 Priorities

The problems and insights set out above are motivation for allowing endpoints to opt out of using the HTTP/2 priority scheme, in favor of using an alternative such as the scheme defined in this specification. The SETTINGS_DEPRECATE_HTTP2_PRIORITIES setting described below enables endpoints to understand their peer's intention. The value of the parameter MUST be 0 or 1. Any value other than 0 or 1 MUST be treated as a connection error (see [RFC7540]; Section 5.4.1) of type PROTOCOL_ERROR.

Endpoints MUST send this SETTINGS parameter as part of the first SETTINGS frame. When the peer receives the first SETTINGS frame, it learns the sender has deprecated the HTTP/2 priority scheme if it receives the SETTINGS_DEPRECATE_HTTP2_PRIORITIES parameter with the value of 1.

A sender MUST NOT change the SETTINGS_DEPRECATE_HTTP2_PRIORITIES parameter value after the first SETTINGS frame. Detection of a change by a receiver MUST be treated as a connection error of type PROTOCOL_ERROR.

Until the client receives the SETTINGS frame from the server, the client SHOULD send both the priority signal defined in the HTTP/2 priority scheme and also that of this prioritization scheme. Once the client learns that the HTTP/2 priority scheme is deprecated, it SHOULD stop sending the HTTP/2 priority signals. If the client learns that the HTTP/2 priority scheme is not deprecated, it SHOULD stop sending PRIORITY_UPDATE frames (Section 5.1), but MAY continue sending the Priority header field (Section 4), as it is an end-to-end

signal that might be useful to nodes behind the server that the client is directly connected to.

The SETTINGS frame precedes any priority signal sent from a client in HTTP/2, so a server can determine if it should respect the HTTP/2 scheme before building state.

3. Priority Parameters

The priority information is a sequence of key-value pairs, providing room for future extensions. Each key-value pair represents a priority parameter.

The Priority HTTP header field (Section 4) is an end-to-end way to transmit this set of parameters when a request or a response is issued. In order to reprioritize a request, HTTP-version-specific frames (Section 5.1 and Section 5.2) are used by clients to transmit the same information on a single hop. If intermediaries want to specify prioritization on a multiplexed HTTP connection, they SHOULD use a PRIORITY_UPDATE frame and SHOULD NOT change the Priority header field.

In both cases, the set of priority parameters is encoded as a Structured Headers Dictionary ([STRUCTURED-HEADERS]).

This document defines the urgency("u") and incremental("i") parameters. When receiving an HTTP request that does not carry these priority parameters, a server SHOULD act as if their default values were specified. Note that handling of omitted parameters is different when processing an HTTP response; see Section 6.

Unknown parameters, parameters with out-of-range values or values of unexpected types MUST be ignored.

3.1. Urgency

The urgency parameter ("u") takes an integer between 0 and 7, in descending order of priority. This range provides sufficient granularity for prioritizing responses for ordinary web browsing, at minimal complexity.

The value is encoded as an sh-integer. The default value is 1.

This parameter indicates the sender's recommendation, based on the expectation that the server would transmit HTTP responses in the order of their urgency values if possible. The smaller the value, the higher the precedence.

The following example shows a request for a CSS file with the urgency set to "0":

```
:method = GET
:scheme = https
:authority = example.net
:path = /style.css
priority = u=0
```

A client that fetches a document that likely consists of multiple HTTP resources (e.g., HTML) SHOULD assign the default urgency level to the main resource. This convention allows servers to refine the urgency using knowledge specific to the web-site (see Section 6).

The lowest urgency level (7) is reserved for background tasks such as delivery of software updates. This urgency level SHOULD NOT be used for fetching responses that have impact on user interaction.

3.2. Incremental

The incremental parameter ("i") takes an sh-boolean as the value that indicates if an HTTP response can be processed incrementally, i.e. provide some meaningful output as chunks of the response arrive.

The default value of the incremental parameter is false ("0").

A server might distribute the bandwidth of a connection between incremental responses that share the same urgency, hoping that providing those responses in parallel would be more helpful to the client than delivering the responses one by one.

If a client makes concurrent requests with the incremental parameter set to false, there is no benefit serving responses in parallel because the client is not going to process those responses incrementally. Serving non-incremental responses one by one, in the order in which those requests were generated is considered to be the best strategy.

The following example shows a request for a JPEG file with the urgency parameter set to "5" and the incremental parameter set to "true".

```
:method = GET
:scheme = https
:authority = example.net
:path = /image.jpg
priority = u=5, i
```

3.3. Defining New Parameters

When attempting to extend priorities, care must be taken to ensure any use of existing parameters are either unchanged or modified in a way that is backwards compatible for peers that are unaware of the extended meaning.

For example, if there is a need to provide more granularity than eight urgency levels, it would be possible to subdivide the range using an additional parameter. Implementations that do not recognize the parameter can safely continue to use the less granular eight levels.

Alternatively, the urgency can be augmented. For example, a graphical user agent could send a "visible" parameter to indicate if the resource being requested is within the viewport.

4. The Priority HTTP Header Field

The Priority HTTP header field can appear in requests and responses. A client uses it to specify the priority of the response. A server uses it to inform the client that the priority was overwritten. An intermediary can use the Priority information from client requests and server responses to correct or amend the precedence to suit it (see Section 6).

The Priority header field is an end-to-end signal of the request priority from the client or the response priority from the server.

As is the ordinary case for HTTP caching ([RFC7234]), a response with a Priority header field might be cached and re-used for subsequent requests. When an origin server generates the Priority response header field based on properties of an HTTP request it receives, the server is expected to control the cacheability or the applicability of the cached response, by using header fields that control the caching behavior (e.g., Cache-Control, Vary).

5. Reprioritization

After a client sends a request, it may be beneficial to change the priority of the response. As an example, a web browser might issue a prefetch request for a JavaScript file with the urgency parameter of the Priority request header field set to "u=7" (background). Then, when the user navigates to a page which references the new JavaScript file, while the prefetch is in progress, the browser would send a reprioritization frame with the priority field value set to "u=0".

In HTTP/2 and HTTP/3, after a request message is sent on a stream, the stream transitions to a state that prevents the client from sending additional frames on the stream. Therefore, a client cannot reprioritize a response by using the Priority header field. Modifying this behavior would require a semantic change to the protocol, but this is avoided by restricting the stream on which a PRIORITY_UPDATE frame can be sent. In HTTP/2 the frame is on stream zero and in HTTP/3 it is sent on the control stream ([I-D.ietf-quic-http], Section 6.2.1).

This document specifies a new PRIORITY_UPDATE frame type for HTTP/2 ([RFC7540]) and HTTP/3 ([I-D.ietf-quic-http]) which enables reprioritization. It carries updated priority parameters and references the target of the reprioritization based on a version-specific identifier; in HTTP/2 this is the Stream ID, in HTTP/3 this is either the Stream ID or Push ID.

Unlike the header field, the reprioritization frame is a hop-by-hop signal.

5.1. HTTP/2 PRIORITY_UPDATE Frame

The HTTP/2 PRIORITY_UPDATE frame (type=0xF) carries the stream ID of the response that is being reprioritized, and the updated priority in ASCII text, using the same representation as that of the Priority header field value.

The Stream Identifier field ([RFC7540], Section 4.1) in the PRIORITY_UPDATE frame header MUST be zero (0x0).

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------------------------------------------------------+
|R|                        Stream ID (31)                       |
+---------------------------------------------------------------+
|                   Priority Field Value (*)                  ...
+---------------------------------------------------------------+
```

Figure 1: HTTP/2 PRIORITY_UPDATE Frame Payload

The PRIORITY_UPDATE frame payload has the following fields:

R: A reserved 1-bit field. The semantics of this bit are undefined, and the bit MUST remain unset (0x0) when sending and MUST be ignored when receiving.

Stream ID: A 31-bit stream identifier for the stream that is the target of the priority update.

Priority Field Value: The priority update value in ASCII text, encoded using Structured Headers.

The HTTP/2 PRIORITY_UPDATE frame MUST NOT be sent prior to opening the stream. If a PRIORITY_UPDATE is received prior to the stream being opened, it MAY be treated as a connection error of type PROTOCOL_ERROR.

TODO: add more description of how to handle things like receiving PRIORITY_UPDATE on wrong stream, a PRIORITY_UPDATE with an invalid ID, etc.

5.2. HTTP/3 PRIORITY_UPDATE Frame

The HTTP/3 PRIORITY_UPDATE frame (type=0xF) carries the identifier of the element that is being reprioritized, and the updated priority in ASCII text, using the same representation as that of the Priority header field value.

The PRIORITY_UPDATE frame MUST be sent on the control stream ([I-D.ietf-quic-http], Section 6.2.1).

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|T|    Empty    |   Prioritized Element ID (i)                ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Priority Field Value (*)                  ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2: HTTP/3 PRIORITY_UPDATE Frame Payload

The PRIORITY_UPDATE frame payload has the following fields:

T (Prioritized Element Type): A one-bit field indicating the type of element being prioritized. A value of 0 indicates a reprioritization for a Request Stream, so the Prioritized Element ID is interpreted as a Stream ID. A value of 1 indicates a reprioritization for a Push stream, so the Prioritized Element ID is interpreted as a Push ID.

Empty: A seven-bit field that has no semantic value.

Prioritized Element ID: The stream ID or push ID that is the target of the priority update.

Priority Field Value: The priority update value in ASCII text, encoded using Structured Headers.

The HTTP/3 PRIORITY_UPDATE frame MUST NOT be sent with an invalid identifier, including before the request stream has been opened or before a promised request has been received. If a server receives a PRIORITY_UPDATE specifying a push ID that has not been promised, it SHOULD be treated as a connection error of type H3_ID_ERROR.

Because the HTTP/3 PRIORITY_UPDATE frame is sent on the control stream and there are no ordering guarantees between streams, a client that reprioritizes a request before receiving the response data might cause the server to receive a PRIORITY_UPDATE for an unknown request. If the request stream ID is within bidirectional stream limits, the PRIORITY_UPDATE frame SHOULD be buffered until the stream is opened and applied immediately after the request message has been processed. Holding PRIORITY_UPDATES consumes extra state on the peer, although the size of the state is bounded by bidirectional stream limits. There is no bound on the number of PRIORITY_UPDATES that can be sent, so an endpoint SHOULD store only the most recently received frame.

TODO: add more description of how to handle things like receiving PRIORITY_UPDATE on wrong stream, a PRIORITY_UPDATE with an invalid ID, etc.

6. Merging Client- and Server-Driven Parameters

It is not always the case that the client has the best understanding of how the HTTP responses deserve to be prioritized. The server might have additional information that can be combined with the client's indicated priority in order to improve the prioritization of the response. For example, use of an HTML document might depend heavily on one of the inline images; existence of such dependencies is typically best known to the server. Or, a server that receives requests for a font [RFC8081] and images with the same urgency might give higher precedence to the font, so that a visual client can render textual information at an early moment.

An origin can use the Priority response header field to indicate its view on how an HTTP response should be prioritized. An intermediary that forwards an HTTP response can use the parameters found in the Priority response header field, in combination with the client Priority request header field, as input to its prioritization process. No guidance is provided for merging priorities, this is left as an implementation decision.

Absence of a priority parameter in an HTTP response indicates the server's disinterest in changing the client-provided value. This is different from the logic being defined for the request header field, in which omission of a priority parameter implies the use of their default values (see Section 3).

As a non-normative example, when the client sends an HTTP request with the urgency parameter set to "5" and the incremental parameter set to "true"

```
:method = GET
:scheme = https
:authority = example.net
:path = /menu.png
priority = u=5, i
```

and the origin responds with

```
:status = 200
content-type = image/png
priority = u=1
```

the intermediary might alter its understanding of the urgency from "5" to "1", because it prefers the server-provided value over the client's. The incremental value continues to be "true", the value specified by the client, as the server did not specify the incremental("i") parameter.

7. Security Considerations

7.1. Fairness

As a general guideline, a server SHOULD NOT use priority information for making schedule decisions across multiple connections, unless it knows that those connections originate from the same client. Due to this, priority information conveyed over a non-coalesced HTTP connection (e.g., HTTP/1.1) might go unused.

The remainder of this section discusses scenarios where unfairness is problematic and presents possible mitigations, or where unfairness is desirable.

TODO: Discuss if we should add a signal that mitigates this issue. For example, we might add a SETTINGS parameter that indicates the next hop that the connection is NOT coalesced (see https://github.com/kazuho/draft-kazuho-httpbis-priority/issues/99).

7.1.1. Coalescing Intermediaries

When an intermediary coalesces HTTP requests coming from multiple clients into one HTTP/2 or HTTP/3 connection going to the backend server, requests that originate from one client might have higher precedence than those coming from others.

It is sometimes beneficial for the server running behind an intermediary to obey to the value of the Priority header field. As an example, a resource-constrained server might defer the transmission of software update files that would have the background urgency being associated. However, in the worst case, the asymmetry between the precedence declared by multiple clients might cause responses going to one end client to be delayed totally after those going to another.

In order to mitigate this fairness problem, when a server responds to a request that is known to have come through an intermediary, the server SHOULD prioritize the response as if it was assigned the priority of "u=1, i" (i.e. round-robin) regardless of the value of the Priority header field being transmitted, unless the server knows the intermediary is not coalescing requests from multiple clients.

A server can determine if a request came from an intermediary through configuration, or by consulting if that request contains one of the following header fields:

- o Forwarded, X-Forwarded-For ([RFC7239])

- o Via ([RFC7230], Section 5.7.1)

Responding to requests coming through an intermediary in a round-robin manner works well when the network bottleneck exists between the intermediary and the end client, as the intermediary would be buffering the responses and then be forwarding the chunks of those buffered responses based on the prioritization scheme it implements. A sophisticated server MAY use a weighted round-robin reflecting the urgencies expressed in the requests, so that less urgent responses would receive less bandwidth in case the bottleneck exists between the server and the intermediary.

7.1.2. HTTP/1.x Back Ends

It is common for CDN infrastructure to support different HTTP versions on the front end and back end. For instance, the client-facing edge might support HTTP/2 and HTTP/3 while communication to back end servers is done using HTTP/1.1. Unlike with connection coalescing, the CDN will "de-mux" requests into discrete connections to the back end. As HTTP/1.1 and older do not provide a way to concurrently transmit multiple responses, there is no immediate fairness issue in protocol. However, back end servers MAY still use client headers for request scheduling. Back end servers SHOULD only schedule based on client priority information where that information can be scoped to individual end clients. Authentication and other session information might provide this linkability.

7.1.3. Intentional Introduction of Unfairness

It is sometimes beneficial to deprioritize the transmission of one connection over others, knowing that doing so introduces a certain amount of unfairness between the connections and therefore between the requests served on those connections.

For example, a server might use a scavenging congestion controller on connections that only convey background priority responses such as software update images. Doing so improves responsiveness of other connections at the cost of delaying the delivery of updates.

Also, a client MAY use the priority values for making local scheduling choices for the requests it initiates.

8. Considerations

8.1. Why use an End-to-End Header Field?

Contrary to the prioritization scheme of HTTP/2 that uses a hop-by-hop frame, the Priority header field is defined as end-to-end.

The rationale is that the Priority header field transmits how each response affects the client's processing of those responses, rather than how relatively urgent each response is to others. The way a client processes a response is a property associated to that client generating that request. Not that of an intermediary. Therefore, it is an end-to-end property. How these end-to-end properties carried by the Priority header field affect the prioritization between the responses that share a connection is a hop-by-hop issue.

Having the Priority header field defined as end-to-end is important for caching intermediaries. Such intermediaries can cache the value of the Priority header field along with the response, and utilize the value of the cached header field when serving the cached response, only because the header field is defined as end-to-end rather than hop-by-hop.

It should also be noted that the use of a header field carrying a textual value makes the prioritization scheme extensible; see the discussion below.

9. IANA Considerations

This specification registers the following entry in the Permanent Message Header Field Names registry established by [RFC3864]:

Header field name: Priority

Applicable protocol: http

Status: standard

Author/change controller: IETF

Specification document(s): This document

Related information: n/a

This specification registers the following entry in the HTTP/2 Settings registry established by [RFC7540]:

Name: SETTINGS_DEPRECATE_HTTP2_PRIORITIES

Code: 0x9

Initial value: 0

Specification: This document

This specification registers the following entry in the HTTP/2 Frame Type registry established by [RFC7540]:

Frame Type: PRIORITY_UPDATE

Code: 0xF

Specification: This document

This specification registers the following entries in the HTTP/3 Frame Type registry established by [I-D.ietf-quic-http]:

Frame Type: PRIORITY_UPDATE

Code: 0xF

Specification: This document

10. References

10.1. Normative References

[I-D.ietf-quic-http]
Bishop, M., "Hypertext Transfer Protocol Version 3 (HTTP/3)", draft-ietf-quic-http-27 (work in progress), February 2020.

[I-D.ietf-quic-transport]
Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-27 (work in progress), February 2020.

[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/info/rfc2119>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/info/rfc7540>.

[STRUCTURED-HEADERS]
Nottingham, M. and P. Kamp, "Structured Headers for HTTP", draft-ietf-httpbis-header-structure-15 (work in progress), January 2020.

10.2. Informative References

[CVE-2019-9513]
Common Vulnerabilities and Exposures, "CVE-2019-9513", March 2019, <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-9513>.

[I-D.lassey-priority-setting]
Lassey, B. and L. Pardue, "Declaring Support for HTTP/2 Priorities", draft-lassey-priority-setting-00 (work in progress), July 2019.

[RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, DOI 10.17487/RFC3864, September 2004, <https://www.rfc-editor.org/info/rfc3864>.

[RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <https://www.rfc-editor.org/info/rfc7234>.

[RFC7239] Petersson, A. and M. Nilsson, "Forwarded HTTP Extension", RFC 7239, DOI 10.17487/RFC7239, June 2014, <https://www.rfc-editor.org/info/rfc7239>.

[RFC8081] Lilley, C., "The "font" Top-Level Media Type", RFC 8081, DOI 10.17487/RFC8081, February 2017, <https://www.rfc-editor.org/info/rfc8081>.

10.3. URIs

[1] https://lists.w3.org/Archives/Public/ietf-http-wg/

[2] https://httpwg.org/

[3] https://github.com/httpwg/http-extensions/labels/priorities

[4] http://tools.ietf.org/agenda/83/slides/slides-83-httpbis-5.pdf

[5] https://github.com/pmeenan/http3-prioritization-proposal

Appendix A. Acknowledgements

Roy Fielding presented the idea of using a header field for representing priorities in http://tools.ietf.org/agenda/83/slides/slides-83-httpbis-5.pdf [4]. In https://github.com/pmeenan/http3-prioritization-proposal [5], Patrick Meenan advocates for representing the priorities using a tuple of urgency and concurrency. The ability to deprecate HTTP/2 prioritization is based on [I-D.lassey-priority-setting], authored by Brad Lassey and Lucas Pardue, with modifications based on feedback that was not incorporated into an update to that document.

The motivation for defining an alternative to HTTP/2 priorities is drawn from discussion within the broad HTTP community. Special thanks to Roberto Peon, Martin Thomson and Netflix for text that was incorporated explicitly in this document.

In addition to the people above, this document owes a lot to the extensive discussion in the HTTP priority design team, consisting of Alan Frindell, Andrew Galloni, Craig Taylor, Ian Swett, Kazuho Oku, Lucas Pardue, Matthew Cox, Mike Bishop, Roberto Peon, Robin Marx, Roy Fielding.

Appendix B. Change Log

B.1. Since draft-kazuho-httpbis-priority-04

- o Minimize semantics of Urgency levels (#1023, #1026)
- o Reduce guidance about how intermediary implements merging priority signals (#1026)
- o Remove mention of CDN-Loop (#1062)
- o Editorial changes
- o Make changes due to WG adoption
- o Removed outdated Consideration (#118)

B.2. Since draft-kazuho-httpbis-priority-03

- o Changed numbering from "[-1,6]" to "[0,7]" (#78)
- o Replaced priority scheme negotiation with HTTP/2 priority deprecation (#100)
- o Shorten parameter names (#108)
- o Expand on considerations (#105, #107, #109, #110, #111, #113)

B.3. Since draft-kazuho-httpbis-priority-02

- o Consolidation of the problem statement (#61, #73)
- o Define SETTINGS_PRIORITIES for negotiation (#58, #69)
- o Define PRIORITY_UPDATE frame for HTTP/2 and HTTP/3 (#51)
- o Explain fairness issue and mitigations (#56)

B.4. Since draft-kazuho-httpbis-priority-01

- o Explain how reprioritization might be supported.

B.5. Since draft-kazuho-httpbis-priority-00

- o Expand urgency levels from 3 to 8.

Authors' Addresses

Kazuho Oku
Fastly

Email: kazuhooku@gmail.com

Lucas Pardue
Cloudflare

Email: lucaspardue.24.7@gmail.com