Workgroup: HTTP
Internet-Draft: draft-ietf-httpbis-priority-07
Published: 25 October 2021
Intended Status: Standards Track
Expires: 28 April 2022
Authors: K. Oku L. Pardue
Fastly Cloudflare

Extensible Prioritization Scheme for HTTP

Abstract

This document describes a scheme that allows an HTTP client to communicate its preferences for how the upstream server prioritizes responses to its requests, and also allows a server to hint to a downstream intermediary how its responses should be prioritized when they are forwarded. 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 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/.

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

It is common for representations of an HTTP [HTTP] resource to have relationships to one or more other resources. Clients will often discover these relationships while processing a retrieved representation, which may lead to further retrieval requests. Meanwhile, the nature of the relationship determines whether the client is blocked from continuing to process locally available resources. An example of this is visual rendering of an HTML document, which 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.

HTTP/2 [HTTP2] and HTTP/3 [HTTP3] support multiplexing of requests and responses in a single connection. An important feature of any implementation of a protocol that provides multiplexing is the ability to prioritize the sending of information. For example, to provide meaningful presentation of an HTML 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 to a client.

A server that operates in ignorance of how clients issue requests and consume responses can cause suboptimal client application performance. Priority signals allow clients to communicate their view of request priority. Servers have their own needs that are independent from client needs, so they often combine priority signals with other available information in order to inform scheduling of response data.

RFC 7540 [RFC7540] stream priority allowed a client to send a series of priority signals that communicate to the server a "priority tree"; the structure of this tree represents the client's preferred relative ordering and weighted distribution of the bandwidth among HTTP responses. Servers could use these priority signals as input into prioritization decision making.

The design and implementation of RFC 7540 stream priority was observed to have shortcomings, explained in Section 2. HTTP/2 [HTTP2] has consequently deprecated the use of these stream priority signals.

This document describes an extensible scheme for prioritizing HTTP responses that uses absolute values. Section 4 defines priority parameters, which are a standardized and extensible format of priority information. Section 5 defines the Priority HTTP header field, a protocol-version-independent and end-to-end priority signal. Clients can use this header to signal priority to servers in order to specify the precedence of HTTP responses. Similarly, servers behind an intermediary can use it to signal priority to the intermediary. Section 7.1 and Section 7.2 define version-specific frames that carry parameters, which clients can use for reprioritization.

Header field and frame priority signals are input to a server's response prioritization process. They are only a suggestion and do not guarantee any particular processing or transmission order for one response relative to any other response. Section 10 and Section 12 provide consideration and guidance about how servers might act upon signals.

The prioritization scheme and priority signals defined herein can act as a substitute for RFC 7540 stream priority.

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 Dictionary, sf-boolean, sf-dictionary, and sf-integer are imported from [STRUCTURED-FIELDS].

Example HTTP requests and responses use the HTTP/2-style formatting from [HTTP2].

This document uses the variable-length integer encoding from [QUIC].

The term control stream is used to describe the HTTP/2 stream with identifier 0x0, and HTTP/3 control stream; see Section 6.2.1 of [HTTP3].

The term HTTP/2 priority signal is used to describe the priority information sent from clients to servers in HTTP/2 frames; see Section 5.3.2 of [HTTP2].

2. Motivation for Replacing RFC 7540 Priorities

RFC 7540 stream priority (see Section 5.3 of [RFC7540]) is a complex system where clients signal stream dependencies and weights to describe an unbalanced tree. It suffered from limited deployment and interoperability and was deprecated in a revision of HTTP/2 [HTTP2]. However, in order to maintain wire compatibility, HTTP/2 priority signals are still mandatory to handle (see Section 5.3.2 of [HTTP2]).

Clients can build RFC 7540 trees with rich flexibility but experience has shown this is rarely exercised. Instead they tend to choose a single model optimized for a single 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 RFC 7540 server implementations do not act on HTTP/2 priority signals. Some instead favor custom server-driven schemes based on heuristics or other hints, such as resource content type or request generation order. For example, a server, with knowledge of an HTML 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.

RFC 7540 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. Additionally, RFC 7540 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.

RFC 7540 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 RFC 7540 implementations. One attack, [CVE-2019-9513] aka

"Resource Loop", is based on using priority signals to manipulate the server's stored prioritization state.

HTTP/2 priority associated with an HTTP request is signalled as a value relative to those of other requests sharing the same HTTP/2 connection. Therefore, in order to prioritize requests, endpoints are compelled to have the knowledge of the underlying HTTP version and how the requests are coalesced. This has been a burden to HTTP endpoints that generate or forward requests in a version-agnostic manner.

HTTP/2 priority signals are required to be delivered and processed in the order they are sent so that the receiver handling is deterministic. Porting HTTP/2 priority signals to protocols that do not provide ordering guarantees presents challenges. For example, HTTP/3 [HTTP3] lacks global ordering across streams that would carry priority signals. Early attempts to port HTTP/2 priority signals to HTTP/3 required adding additional information to the signals, leading to more complicated processing. Problems found with this approach could not be resolved and definition of a HTTP/3 priority signalling feature was removed before publication.

Considering the deployment problems and the design restrictions of RFC 7540 stream priority, as well as the difficulties in adapting it to HTTP/3, continuing to base prioritization on this mechanism risks increasing the complexity of systems. Multiple experiments from independent research have shown that simpler schemes can reach at least equivalent performance characteristics compared to the more complex RFC 7540 setups seen in practice, at least for the web use case.

2.1. Disabling RFC 7540 Priorities

The problems and insights set out above provided the motivation for deprecating RFC 7540 stream priority (see Section 5.3 of [RFC7540]).

The SETTINGS_NO_RFC7540_PRIORITIES HTTP/2 setting is defined by this document in order to allow endpoints to explicitly opt out of using HTTP/2 priority signals (see Section 5.3.2 of [HTTP2]). Endpoints are encouraged to use alternative priority signals (for example, Section 5 or Section 7.1) but there is no requirement to use a specific signal type.

The value of SETTINGS_NO_RFC7540_PRIORITIES MUST be 0 or 1. Any value other than 0 or 1 MUST be treated as a connection error (see Section 5.4.1 of [HTTP2]) of type PROTOCOL_ERROR. The initial value is 0.

Endpoints MUST send this SETTINGS parameter as part of the first SETTINGS frame. A sender MUST NOT change the

SETTINGS_NO_RFC7540_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.

The SETTINGS frame precedes any HTTP/2 priority signal sent from a client, so a server can determine if it needs to allocate any resource to signal handling before they arrive. A server that receives SETTINGS_NO_RFC7540_PRIORITIES with value of 1 MUST ignore HTTP/2 priority signals.

2.1.1. Advice when Using Extensible Priorities as the Alternative

Until the client receives the SETTINGS frame from the server, the client SHOULD send both the HTTP/2 priority signals and the signals of this prioritization scheme (see Section 5 and Section 7.1). When the client receives the first SETTINGS frame that contains the SETTINGS_NO_RFC7540_PRIORITIES parameter with value of 1, it SHOULD stop sending the HTTP/2 priority signals. If the value was 0 or if the settings parameter was absent, it SHOULD stop sending PRIORITY_UPDATE frames (Section 7.1), but MAY continue sending the Priority header field (Section 5), as it is an end-to-end signal that might be useful to nodes behind the server that the client is directly connected to.

3. Applicability of the Extensible Priority Scheme

The priority scheme defined by this document considers only the prioritization of HTTP messages and tunnels, see Section 9, Section 10, and Section 11.

Where HTTP extensions change stream behavior or define new data carriage mechanisms, they can also define how this priority scheme can be applied.

4. 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 5) 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 PRIORITY_UPDATE frames (Section 7.1 and Section 7.2) 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 Fields Dictionary (see Section 3.2 of [STRUCTURED-FIELDS]).

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 8.

Receivers parse the Dictionary as defined in Section 4.2 of [STRUCTURED-FIELDS]. Where the Dictionary is successfully parsed, this document places the additional requirement that unknown priority parameters, parameters with out-of-range values, or values of unexpected types MUST be ignored.

4.1. Urgency

The urgency parameter (u) takes an integer between 0 and 7, in descending order of priority.

The value is encoded as an sf-integer. The default value is 3.

Endpoints use this parameter to communicate their view of the precedence of HTTP responses. The chosen value of urgency can be based on the expectation that servers might use this information to transmit HTTP responses in the order of their urgency. 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 8).

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.

4.2. Incremental

The incremental parameter (i) takes an sf-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).

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

If a client makes concurrent requests with the incremental parameter set to true, serving requests with the same urgency concurrently might be beneficial. Doing this distributes the connection bandwidth, meaning that responses take longer to complete. Incremental delivery is most useful where multiple partial responses might provide some value to clients ahead of a complete response being available.

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
```

4.3. Defining New Parameters

When attempting to define new parameters, care must be taken so that they do not adversely interfere with prioritization performed by existing endpoints or intermediaries that do not understand the newly defined parameter. Since unknown parameters are ignored, new parameters should not change the interpretation of, or modify, the urgency (see Section 4.1) or incremental (see Section 4.2) parameters in a way that is not backwards compatible or fallback safe.

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.

Generic parameters are preferred over vendor-specific, application-specific or deployment-specific values. If a generic value cannot be agreed upon in the community, the parameter's name should be correspondingly specific (e.g., with a prefix that identifies the vendor, application or deployment).

4.3.1. Registration

New Priority parameters can be defined by registering them in the HTTP Priority Parameters Registry. The registry governs the keys (short textual strings) used in Structured Fields Dictionary (see Section 3.2 of [STRUCTURED-FIELDS]). Since each HTTP request can have associated priority signals, there is value in having short key lengths, especially single-character strings. In order to encourage extension while avoiding unintended conflict among attractive key values, the HTTP Priority Parameters Registry operates two registration policies depending on key length.

* Registration requests for parameters with a key length of one use the Specification Required policy, as per Section 4.6 of [RFC8126].

* Registration requests for parameters with a key length greater than one use the Expert Review policy, as per Section 4.5 of [RFC8126]. A specification document is appreciated, but not required.

When reviewing registration requests, the designated expert(s) can consider the additional guidance provided in Section 4.3 but cannot use it as a basis for rejection.

Registration requests should use the following template:

Name: [a name for the Priority Parameter that matches key]

Description: [a description of the parameter semantics and value]

Reference: [to a specification defining this parameter]

See the registry at https://iana.org/assignments/http-priority for details on where to send registration requests.

5. The Priority HTTP Header Field

The Priority HTTP header field carries priority parameters Section 4. It can appear in requests and responses. It is an end-to-end signal of the request priority from the client or the response priority from the server. Section 8 describes how intermediaries can combine the priority information from client requests and server responses to correct or amend the precedence. Clients cannot interpret the appearance or omission of a Priority response header as acknowledgement that any prioritization has occurred. Guidance for how endpoints can act on Priority header values is given in Section 10 and Section 9.

Priority is a Dictionary (Section 3.2 of [STRUCTURED-FIELDS]):

```
Priority   = sf-dictionary
```

As is the ordinary case for HTTP caching [CACHING], 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).

6. 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 signal with the priority field value set to u=0. The PRIORITY_UPDATE frame (Section 7) can be used for such reprioritization.

7. The PRIORITY_UPDATE Frame

This document specifies a new PRIORITY_UPDATE frame for HTTP/2 [HTTP2] and HTTP/3 [HTTP3]. It carries priority parameters and references the target of the prioritization based on a version-specific identifier. In HTTP/2, this identifier is the Stream ID; in HTTP/3, the identifier is either the Stream ID or Push ID. Unlike the Priority header field, the PRIORITY_UPDATE frame is a hop-by-hop signal.

PRIORITY_UPDATE frames are sent by clients on the control stream, allowing them to be sent independent from the stream that carries the response. This means they can be used to reprioritize a response or a push stream; or signal the initial priority of a response instead of the Priority header field.

A PRIORITY_UPDATE frame communicates a complete set of all parameters in the Priority Field Value field. Omitting a parameter is a signal to use the parameter's default value. Failure to parse the Priority Field Value MUST be treated as a connection error. In HTTP/2 the error is of type PROTOCOL_ERROR; in HTTP/3 the error is of type H3_FRAME_ERROR.

A client MAY send a PRIORITY_UPDATE frame before the stream that it references is open (except for HTTP/2 push streams; see Section 7.1). Furthermore, HTTP/3 offers no guaranteed ordering across streams, which could cause the frame to be received earlier than intended. Either case leads to a race condition where a server receives a PRIORITY_UPDATE frame that references a request stream that is yet to be opened. To solve this condition, for the purposes of scheduling, the most recently received PRIORITY_UPDATE frame can be considered as the most up-to-date information that overrides any other signal. Servers SHOULD buffer the most recently received PRIORITY_UPDATE frame and apply it once the referenced stream is opened. Holding PRIORITY_UPDATE frames for each stream requires server resources, which can can be bound by local implementation policy. Although there is no limit to the number of PRIORITY_UPDATES that can be sent, storing only the most recently received frame limits resource commitment.

7.1. HTTP/2 PRIORITY_UPDATE Frame

The HTTP/2 PRIORITY_UPDATE frame (type=0x10) is used by clients to signal the initial priority of a response, or to reprioritize a response or push stream. It carries the stream ID of the response and the priority in ASCII text, using the same representation as the Priority header field value.

The Stream Identifier field (see Section 5.1.1 of [HTTP2]) in the PRIORITY_UPDATE frame header MUST be zero (0x0). Receiving a PRIORITY_UPDATE frame with a field of any other value MUST be treated as a connection error of type PROTOCOL_ERROR.

```
HTTP/2 PRIORITY_UPDATE Frame {
  Length (24),
  Type (i) = 10,

  Unused Flags (8).

  Reserved (1),
  Stream Identifier (31),

  Reserved (1),
  Prioritized Stream ID (31),
  Priority Field Value (..),
}
```

Figure 1: HTTP/2 PRIORITY_UPDATE Frame Payload

The Length, Type, Unused Flag(s), Reserved, and Stream Identifier fields are described in Section 4 of [HTTP2]. The frame payload of PRIORITY_UPDATE frame payload contains the following additional fields:

Reserved: 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.

Prioritized 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 Fields. This is the same representation as the Priority header field value.

When the PRIORITY_UPDATE frame applies to a request stream, clients SHOULD provide a Prioritized Stream ID that refers to a stream in the "open", "half-closed (local)", or "idle" state. Servers can discard frames where the Prioritized Stream ID refers to a stream in the "half-closed (local)" or "closed" state. The number of streams which have been prioritized but remain in the "idle" state plus the number of active streams (those in the "open" or either "half-closed" state; see Section 5.1.2 of [HTTP2]) MUST NOT exceed the value of the SETTINGS_MAX_CONCURRENT_STREAMS parameter. Servers that receive such a PRIORITY_UPDATE MUST respond with a connection error of type PROTOCOL_ERROR.

When the PRIORITY_UPDATE frame applies to a push stream, clients SHOULD provide a Prioritized Stream ID that refers to a stream in the "reserved (remote)" or "half-closed (local)" state. Servers can discard frames where the Prioritized Stream ID refers to a stream in the "closed" state. Clients MUST NOT provide a Prioritized Stream ID that refers to a push stream in the "idle" state. Servers that receive a PRIORITY_UPDATE for a push stream in the "idle" state MUST respond with a connection error of type PROTOCOL_ERROR.

If a PRIORITY_UPDATE frame is received with a Prioritized Stream ID of 0x0, the recipient MUST respond with a connection error of type PROTOCOL_ERROR.

If a client receives a PRIORITY_UPDATE frame, it MUST respond with a connection error of type PROTOCOL_ERROR.

7.2. HTTP/3 PRIORITY_UPDATE Frame

The HTTP/3 PRIORITY_UPDATE frame (type=0xF0700 or 0xF0701) is used by clients to signal the initial priority of a response, or to reprioritize a response or push stream. It carries the identifier of the element that is being prioritized, and the updated priority in ASCII text, using the same representation as that of the Priority header field value. PRIORITY_UPDATE with a frame type of 0xF0700 is used for request streams, while PRIORITY_UPDATE with a frame type of 0xF0701 is used for push streams.

The PRIORITY_UPDATE frame MUST be sent on the client control stream (see Section 6.2.1 of [HTTP3]). Receiving a PRIORITY_UPDATE frame on a stream other than the client control stream MUST be treated as a connection error of type H3_FRAME_UNEXPECTED.

```
HTTP/3 PRIORITY_UPDATE Frame {
  Type (i) = 0xF0700..0xF0701,
  Length (i),
  Prioritized Element ID (i),
  Priority Field Value (..),
}
```

Figure 2: HTTP/3 PRIORITY_UPDATE Frame

The PRIORITY_UPDATE frame payload has the following fields:

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 Fields. This is the same representation as the Priority header field value.

The request-stream variant of PRIORITY_UPDATE (type=0xF0700) MUST reference a request stream. If a server receives a PRIORITY_UPDATE (type=0xF0700) for a Stream ID that is not a request stream, this MUST be treated as a connection error of type H3_ID_ERROR. The Stream ID MUST be within the client-initiated bidirectional stream limit. If a server receives a PRIORITY_UPDATE (type=0xF0700) with a Stream ID that is beyond the stream limits, this SHOULD be treated as a connection error of type H3_ID_ERROR. Generating an error is not mandatory because HTTP/3 implementations might have practical barriers to determining the active stream concurrency limit that is applied by the QUIC layer.

The push-stream variant PRIORITY_UPDATE (type=0xF0701) MUST reference a promised push stream. If a server receives a PRIORITY_UPDATE (type=0xF0701) with a Push ID that is greater than the maximum Push ID or which has not yet been promised, this MUST be treated as a connection error of type H3_ID_ERROR.

PRIORITY_UPDATE frames of either type are only sent by clients. If a client receives a PRIORITY_UPDATE frame, this MUST be treated as a connection error of type H3_FRAME_UNEXPECTED.

8. 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 4).

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.

9. Client Scheduling

A client MAY use priority values to make local processing or scheduling choices about the requests it initiates.

10. Server Scheduling

Priority signals are input to a prioritization process. They do not guarantee any particular processing or transmission order for one response relative to any other response. An endpoint cannot force a peer to process concurrent request in a particular order using priority. Expressing priority is therefore only a suggestion.

A server can use priority signals along with other inputs to make scheduling decisions. No guidance is provided about how this can or should be done. Factors such as implementation choices or deployment environment also play a role. Any given connection is likely to have many dynamic permutations. For these reasons, there is no unilateral perfect scheduler and this document only provides some basic recommendations for implementations.

Clients cannot depend on particular treatment based on priority signals. Servers can use other information to prioritize responses.

It is RECOMMENDED that, when possible, servers respect the urgency parameter (Section 4.1), sending higher urgency responses before lower urgency responses.

The incremental parameter indicates how a client processes response bytes as they arrive. It is RECOMMENDED that, when possible, servers respect the incremental parameter (Section 4.2). Non-incremental resources can only be used when all of the response payload has been received. Therefore, non-incremental responses of the same urgency SHOULD be served in their entirety, one-by-one, based on the stream ID, which corresponds to the order in which clients make requests. Doing so ensures that clients can use request ordering to influence response order.

Incremental responses of the same urgency SHOULD be served by sharing bandwidth amongst them. Incremental resources are used as parts, or chunks, of the response payload are received. A client might benefit more from receiving a portion of all these resources rather than the entirety of a single resource. How large a portion of the resource is needed to be useful in improving performance varies. Some resource types place critical elements early, others can use information progressively. This scheme provides no explicit mandate about how a server should use size, type or any other input to decide how to prioritize.

There can be scenarios where a server will need to schedule multiple incremental and non-incremental responses at the same urgency level. Strictly abiding the scheduling guidance based on urgency and request generation order might lead to sub-optimal results at the client, as early non-incremental responses might prevent serving of incremental responses issued later. The following are examples of such challenges.

1. At the same urgency level, a non-incremental request for a large resource followed by an incremental request for a small resource.

2. At the same urgency level, an incremental request of indeterminate length followed by a non-incremental large resource.

It is RECOMMENDED that servers avoid such starvation where possible. The method to do so is an implementation decision. For example, a server might pre-emptively send responses of a particular incremental type based on other information such as content size.

Optimal scheduling of server push is difficult, especially when pushed resources contend with active concurrent requests. Servers can consider many factors when scheduling, such as the type or size of resource being pushed, the priority of the request that triggered the push, the count of active concurrent responses, the priority of other active concurrent responses, etc. There is no general guidance on the best way to apply these. A server that is too simple could easily push at too high a priority and block client requests, or push at too low a priority and delay the response, negating intended goals of server push.

Priority signals are a factor for server push scheduling. The concept of parameter value defaults applies slightly differently because there is no explicit client-signalled initial priority. A server can apply priority signals provided in an origin response; see the merging guidance given in Section 8. In the absence of origin signals, applying default parameter values could be suboptimal. By whatever means a server decides to schedule a pushed response, it can signal the intended priority to the client by including the Priority field in a PUSH_PROMISE or HEADERS frame.

10.1. Intermediaries with Multiple Backend Connections

An intermediary serving an HTTP connection might split requests over multiple backend connections. When it applies prioritization rules strictly, low priority requests cannot make progress while requests with higher priorities are inflight. This blocking can propagate to backend connections, which the peer might interpret as a connection stall. Endpoints often implement protections against stalls, such as abruptly closing connections after a certain time period. To reduce the possibility of this occurring, intermediaries can avoid strictly following prioritization and instead allocate small amounts of bandwidth for all the requests that they are forwarding, so that every request can make some progress over time.

Similarly, servers SHOULD allocate some amount of bandwidths to streams acting as tunnels.

11. Scheduling and the CONNECT Method

When a request stream carries the CONNECT method, the scheduling guidance in this document applies to the frames on the stream. A client that issues multiple CONNECT requests can set the incremental parameter to true, servers that implement the recommendation in Section 10 will schedule these fairly.

12. Retransmission Scheduling

Transport protocols such as TCP and QUIC provide reliability by detecting packet losses and retransmitting lost information. While this document specifies HTTP-layer prioritization, its effectiveness can be further enhanced if the transport layer factors priority into scheduling both new data and retransmission data. The remainder of this section discusses considerations when using QUIC.

Section 13.3 of [QUIC] states "Endpoints SHOULD prioritize retransmission of data over sending new data, unless priorities specified by the application indicate otherwise". When an HTTP/3 application uses the priority scheme defined in this document and the QUIC transport implementation supports application indicated stream priority, a transport that considers the relative priority of streams when scheduling both new data and retransmission data might better match the expectations of the application. However, there are no requirements on how a transport chooses to schedule based on this information because the decision depends on several factors and trade-offs. It could prioritize new data for a higher urgency stream over retransmission data for a lower priority stream, or it could prioritize retransmission data over new data irrespective of urgencies.

Section 6.2.4 of [QUIC-RECOVERY], also highlights consideration of application priorities when sending probe packets after Probe Timeout timer expiration. A QUIC implementation supporting application-indicated priorities might use the relative priority of streams when choosing probe data.

13. Fairness

As a general guideline, a server SHOULD NOT use priority information for making scheduling 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.

13.1. 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 user agent to be delayed totally after those going to another.

In order to mitigate this fairness problem, a server could use knowledge about the intermediary as another signal in its prioritization decisions. For instance, if a server knows the intermediary is coalescing requests, then it could avoid serving the responses in their entirety and instead distribute bandwidth (for example, in a round-robin manner). This can work if the constrained resource is network capacity between the intermediary and the user agent, as the intermediary buffers responses and forwards the chunks based on the prioritization scheme it implements.

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:

* Forwarded [FORWARDED], X-Forwarded-For

* Via (see Section 7.6.3 of [HTTP])

13.2. 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. HTTP/1.1 and older do not support response multiplexing in a single connection, so there is not a fairness problem. 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.

13.3. 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.

14. 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.

15. Security Considerations

[RFC7540] stream prioritization relies on dependencies. Considerations are presented to implementations, describing how limiting state or work commitments can avoid some types of problems. In addition, [CVE-2019-9513] aka "Resource Loop", is an example of a DoS attack that abuses stream dependencies. Extensible priorities does not use dependencies, which avoids these issues.

Section 7 describes considerations for server buffering of PRIORITY_UPDATE frames.

Section 10 presents examples where servers that prioritize responses in a certain way might be starved of the ability to transmit payload.

The security considerations from [STRUCTURED-FIELDS] apply to processing of priority parameters defined in Section 4.

16. IANA Considerations

This specification registers the following entry in the the Hypertext Transfer Protocol (HTTP) Field Name Registry established by [HTTP]:

Field name: Priority

Status: permanent

Specification document(s): This document

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

Name:
SETTINGS_NO_RFC7540_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: 0x10

Specification: This document

This specification registers the following entries in the HTTP/3 Frame Type registry established by [HTTP3]:

Frame Type: PRIORITY_UPDATE

Code: 0xF0700 and 0xF0701

Specification: This document

Upon publication, please create the HTTP Priority Parameters registry at https://iana.org/assignments/http-priority and populate it with the types defined in Section 4; see Section 4.3.1 for its associated procedures.

17. References

17.1. Normative References

[HTTP] Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP Semantics", Work in Progress, Internet-Draft, draft-ietf-httpbis-semantics-19, 12 September 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-semantics-19>.

[HTTP2] Thomson, M. and C. Benfield, "Hypertext Transfer Protocol Version 2 (HTTP/2)", Work in Progress, Internet-Draft, draft-ietf-httpbis-http2bis-05, 26 September 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-http2bis-05>.

[HTTP3] Bishop, M., "Hypertext Transfer Protocol Version 3 (HTTP/3)", Work in Progress, Internet-Draft, draft-ietf-quic-http-34, 2 February 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-34>.

[QUIC] 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>.

[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>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/rfc/rfc8126>.

[STRUCTURED-FIELDS] Nottingham, M. and P-H. Kamp, "Structured Field Values for HTTP", RFC 8941, DOI 10.17487/RFC8941, February 2021, <https://www.rfc-editor.org/rfc/rfc8941>.

17.2. Informative References

[CACHING] Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP Caching", Work in Progress, Internet-Draft, draft-ietf-httpbis-cache-19, 12 September 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-cache-19>.

[CVE-2019-9513] Common Vulnerabilities and Exposures, "CVE-2019-9513", 1 March 2019, <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-9513>.

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

[I-D.lassey-priority-setting] Lassey, B. and L. Pardue, "Declaring Support for HTTP/2 Priorities", Work in Progress, Internet-Draft, draft-lassey-priority-setting-00, 25 July 2019, <https://datatracker.ietf.org/doc/html/draft-lassey-priority-setting-00>.

[QUIC-RECOVERY] Iyengar, J., Ed. and I. Swett, Ed., "QUIC Loss Detection and Congestion Control", RFC 9002, DOI

10.17487/RFC9002, May 2021, <https://www.rfc-editor.org/rfc/rfc9002>.

[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/rfc/rfc7540>.

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

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. In https://github.com/pmeenan/http3-prioritization-proposal, Patrick Meenan advocated for representing the priorities using a tuple of urgency and concurrency. The ability to disable HTTP/2 prioritization is inspired by [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.

Yang Chi contributed the section on retransmission scheduling.

Appendix B. Change Log

B.1. Since draft-ietf-httpbis-priority-06

* Focus on editorial changes

* Clarify rules about Sf-Dictionary handling in headers

* Split policy for parameter IANA registry into two sections based on key length

B.2. Since draft-ietf-httpbis-priority-05

* Renamed SETTINGS_DEPRECATE_RFC7540_PRIORITIES to SETTINGS_NO_RFC7540_PRIORITIES

* Clarify that senders of the HTTP/2 setting can use any alternative (#1679, #1705)

B.3. Since draft-ietf-httpbis-priority-04

* Renamed SETTINGS_DEPRECATE_HTTP2_PRIORITIES to SETTINGS_DEPRECATE_RFC7540_PRIORITIES (#1601)

* Reoriented text towards RFC7540bis (#1561, #1601)

* Clarify intermediary behavior (#1562)

B.4. Since draft-ietf-httpbis-priority-03

* Add statement about what this scheme applies to. Clarify extensions can use it but must define how themselves (#1550, #1559)

* Describe scheduling considerations for the CONNECT method (#1495, #1544)

* Describe scheduling considerations for retransmitted data (#1429, #1504)

* Suggest intermediaries might avoid strict prioritization (#1562)

B.5. Since draft-ietf-httpbis-priority-02

* Describe considerations for server push prioritization (#1056, #1345)

* Define HTTP/2 PRIORITY_UPDATE ID limits in HTTP/2 terms (#1261, #1344)

* Add a Parameters registry (#1371)

B.6. Since draft-ietf-httpbis-priority-01

* PRIORITY_UPDATE frame changes (#1096, #1079, #1167, #1262, #1267, #1271)

* Add section to describe server scheduling considerations (#1215, #1232, #1266)

* Remove specific instructions related to intermediary fairness (#1022, #1264)

B.7. Since draft-ietf-httpbis-priority-00

* Move text around (#1217, #1218)

* Editorial change to the default urgency. The value is 3, which was always the intent of previous changes.

B.8. Since draft-kazuho-httpbis-priority-04

* Minimize semantics of Urgency levels (#1023, #1026)

* Reduce guidance about how intermediary implements merging priority signals (#1026)

* Remove mention of CDN-Loop (#1062)

* Editorial changes

* Make changes due to WG adoption

* Removed outdated Consideration (#118)

B.9. Since draft-kazuho-httpbis-priority-03

* Changed numbering from [-1,6] to [0,7] (#78)

* Replaced priority scheme negotiation with HTTP/2 priority deprecation (#100)

* Shorten parameter names (#108)

* Expand on considerations (#105, #107, #109, #110, #111, #113)

B.10. Since draft-kazuho-httpbis-priority-02

* Consolidation of the problem statement (#61, #73)

* Define SETTINGS_PRIORITIES for negotiation (#58, #69)

* Define PRIORITY_UPDATE frame for HTTP/2 and HTTP/3 (#51)

* Explain fairness issue and mitigations (#56)

B.11. Since draft-kazuho-httpbis-priority-01

* Explain how reprioritization might be supported.

B.12. Since draft-kazuho-httpbis-priority-00

* 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