Internet Engineering Task Force (IETF)
Request for Comments: 9218
Category: Standards Track
ISSN: 2070-1721

#. ##
Fastly
L. Pardue
Cloudflare
June 2022

Extensible Prioritization Scheme for HTTP

draft-ietf-httpbis-priority-12

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.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841[1].

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9218[2].

Copyright Notice

[1] https://www.rfc-editor.org/rfc/rfc7841.html#section-2
[2] https://www.rfc-editor.org/info/rfc9218
[3] https://trustee.ietf.org/license-info

etf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.]]], 2nd string ends in: [[[ection 2 of RFC 78 4111 https://www.rfc-editor.org/rfc/rfc7841.html#section-2.Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc921822 https://www.rfc-editor.org/info/rfc9218.Copyright NoticeCopyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info33 https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.]]] (at line 6)

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 relationships determines whether a client is blocked from continuing to process locally available resources. An example of this is the visual rendering of an HTML document, which could be blocked by the retrieval of a Cascading Style Sheets (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 [HTTP/2] and HTTP/3 [HTTP/3] 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.

HTTP/2 and HTTP/3 servers can schedule transmission of concurrent response data by any means they choose. Servers can ignore client priority signals and still successfully serve HTTP responses. However, servers that operate 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 of 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 decisions.

The design and implementation of RFC 7540 stream priority were observed to have shortcomings, as explained in Section 2. HTTP/2 [HTTP/2] has consequently deprecated the use of these stream priority signals. The prioritization scheme and priority signals defined herein can act as a substitute for RFC 7540 stream priority.

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, which is an end-to-end priority signal that is independent of protocol version. Clients can send this header field to signal their view of how responses should be prioritized. Similarly, servers behind an intermediary can use it to signal priority to the intermediary. After sending a request, a client can change their view of response priority (see Section 6) by sending HTTP-version-specific frames as defined in 7.1 and 7.2.

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. 10 and 12 provide considerations and guidance about how servers might act upon signals.

1.1. Notational Conventions

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 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document uses the following terminology from Section 3 of [STRUCTURED-FIELDS] to specify syntax and parsing: "Boolean", "Dictionary", and "Integer".

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

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

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

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 [HTTP/2].

2. Motivation for Replacing RFC 7540 Stream 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 has been deprecated in a revision of HTTP/2 [HTTP/2]. HTTP/2 retains these protocol elements in order to maintain wire compatibility (see Section 5.3.2 of [HTTP/2]), which means that they might still be used even in the presence of alternative signaling, such as the scheme this document describes.

Many RFC 7540 server implementations do not act on HTTP/2 priority signals.

Prioritization can use information that servers have about resources or the order in which requests are generated. 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. With RFC 7540, it is difficult for servers to interpret signals from clients for prioritization, as the same conditions could result in very different signaling from different clients. This document describes signaling that is simpler and more constrained, requiring less interpretation and allowing less variation.

RFC 7540 does not define a method that can be used by a server to provide a priority signal for intermediaries.

RFC 7540 stream priority is expressed relative to other requests sharing the same connection at the same time. It is difficult to incorporate such a design into applications that generate requests without knowledge of how other requests might share a connection, or into protocols that do not have strong ordering guarantees across streams, like HTTP/3 [HTTP/3].

Experiments from independent research [MARX] 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 Stream Priorities

The problems and insights set out above provided the motivation for an alternative to RFC 7540 stream priority (see Section 5.3 of [HTTP/2]).

The SETTINGS_NO_RFC7540_PRIORITIES HTTP/2 setting is defined by this document in order to allow endpoints to omit or ignore HTTP/2 priority signals (see Section 5.3.2 of [HTTP/2]), as described below. 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 [HTTP/2]) of type PROTOCOL_ERROR. The initial value is 0.

If endpoints use SETTINGS_NO_RFC7540_PRIORITIES, they MUST send it in the first SETTINGS frame. Senders MUST NOT change the SETTINGS_NO_RFC7540_PRIORITIES value after the first SETTINGS frame. Receivers that detect a change MAY treat it as a connection error of type PROTOCOL_ERROR.

Clients can send SETTINGS_NO_RFC7540_PRIORITIES with a value of 1 to indicate that they are not using HTTP/2 priority signals. The SETTINGS frame precedes any HTTP/2 priority signal sent from clients, so servers can determine whether they need to allocate any resources to signal handling before signals arrive. A server that receives SETTINGS_NO_RFC7540_PRIORITIES with a value of 1 MUST ignore HTTP/2 priority signals.

Servers can send SETTINGS_NO_RFC7540_PRIORITIES with a value of 1 to indicate that they will ignore HTTP/2 priority signals sent by clients.

Endpoints that send SETTINGS_NO_RFC7540_PRIORITIES are encouraged to use alternative priority signals (for example, see Section 5 or Section 7.1), but there is no requirement to use a specific signal type.

2.1.1. Advice when Using Extensible Priorities as the Alternative

Before receiving a SETTINGS frame from a server, a client does not know if the server is ignoring HTTP/2 priority signals. Therefore, 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 5 and 7.1).

Once the client receives the first SETTINGS frame that contains the SETTINGS_NO_RFC7540_PRIORITIES parameter with a value of 1, it SHOULD stop sending the HTTP/2 priority signals. This avoids sending redundant signals that are known to be ignored.

Similarly, if the client receives SETTINGS_NO_RFC7540_PRIORITIES with a value of 0 or if the settings parameter was absent, it SHOULD stop sending PRIORITY_UPDATE frames (Section 7.1), since those frames are likely to be ignored. However, the client 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 is primarily focused on the prioritization of HTTP response messages (see Section 3.4 of [HTTP]). It defines new priority parameters (Section 4) and a means of conveying those parameters 5 and 7), which is intended to communicate the priority of responses to a server that is responsible for prioritizing them. Section 10 provides considerations for servers about acting on those signals in combination with other inputs and factors.

The CONNECT method (see Section 9.3.6 of [HTTP]) can be used to establish tunnels. Signaling applies similarly to tunnels; additional considerations for server prioritization are given in Section 11.

Section 9 describes how clients can optionally apply elements of this scheme locally to the request messages that they generate.

Some forms of HTTP extensions might change HTTP/2 or HTTP/3 stream behavior or define new data carriage mechanisms. Such extensions can themselves define how this priority scheme is to 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 priority parameters when a request or a response is issued. After sending a request, a client can change their view of response priority (Section 6) by sending HTTP-version-specific PRIORITY_UPDATE frames as defined in 7.1 and 7.2. Frames transmit priority parameters on a single hop only.

Intermediaries can consume and produce priority signals in a PRIORITY_UPDATE frame or Priority header field. An intermediary that passes only the Priority request header field to the next hop preserves the original end-to-end signal from the client; see Section 14. An intermediary could pass the Priority header field and additionally send a PRIORITY_UPDATE frame. This would have the effect of preserving the original client end-to-end signal, while instructing the next hop to use a different priority, per the guidance in Section 7. An intermediary that replaces or adds a Priority request header field overrides the original client end-to-end signal, which can affect prioritization for all subsequent recipients of the request.

For both the Priority header field and the PRIORITY_UPDATE frame, the set of priority parameters is encoded as a Dictionary (see Section 3.2 of [STRUCTURED-FIELDS]).

This document defines the urgency (`u`) and incremental (`i`) priority parameters. When receiving an HTTP request that does not carry these priority parameters, a server SHOULD act as if their default values were specified.

An intermediary can combine signals from requests and responses that it forwards. Note that omission of priority parameters in responses is handled differently from omission in requests; see Section 8.

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

4.1. Urgency

The urgency (`u`) parameter value is Integer (see Section 3.3.1 of [STRUCTURED-FIELDS]), between 0 and 7 inclusive, in descending order of priority. The default 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 website (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 any impact on user interaction.

4.2. Incremental

The incremental (`i`) parameter value is Boolean (see Section 3.3.6 of [STRUCTURED-FIELDS]). It 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 in 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 Priority Parameters

When attempting to define new priority 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 priority parameters. Since unknown priority parameters are ignored, new priority parameters should not change the interpretation of, or modify, the urgency (see Section 4.1) or incremental (see Section 4.2) priority 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 priority 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` priority parameter to indicate if the resource being requested is within the viewport.

Generic priority 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" registry. This registry governs the keys (short textual strings) used in the 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 extensions while avoiding unintended conflict among attractive key values, the "HTTP Priority" registry operates two registration policies, depending on key length.

- Registration requests for priority parameters with a key length of one use the Specification Required policy, per Section 4.6 of [RFC8126].
- Registration requests for priority parameters with a key length greater than one use the Expert Review policy, 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 the parameter key]

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

Reference: [to a specification defining this priority parameter]

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

5. The Priority HTTP Header Field

The Priority HTTP header field is a Dictionary that carries priority parameters (see Section 4). It can appear in requests and responses. It is an end-to-end signal that indicates the endpoint's view of how HTTP responses should be prioritized. Section 8 describes how intermediaries can combine the priority information sent from clients and servers. Clients cannot interpret the appearance or omission of a Priority response header field as acknowledgement that any prioritization has occurred. Guidance for how endpoints can act on Priority header values is given in 9 and 10.

An HTTP request with a Priority header field might be cached and reused for subsequent requests; see [CACHING]. 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 that 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 [HTTP/2] and HTTP/3 [HTTP/3]. 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 independently of the stream that carries the response. This means they can be used to reprioritize a response or a push stream, or to signal the initial priority of a response instead of the Priority header field.

A PRIORITY_UPDATE frame communicates a complete set of all priority parameters in the Priority Field Value field. Omitting a priority parameter is a signal to use its default value. Failure to parse the Priority Field Value MAY 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_GENERAL_PROTOCOL_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 be bounded by local implementation policy. Although there is no limit to the number of PRIORITY_UPDATE frames 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 [HTTP/2]) 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 (8) = 0x10,

  Unused Flags (8),

  Reserved (1),
  Stream Identifier (31),

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

Figure 1: HTTP/2 PRIORITY_UPDATE Frame Format

The Length, Type, Unused Flag(s), Reserved, and Stream Identifier fields are described in Section 4 of [HTTP/2]. The PRIORITY_UPDATE frame payload contains the following additional fields:

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 (i.e., streams where data might still be received). Servers can discard frames where the prioritized stream ID refers to a stream in the "half-closed (local)" or "closed" state (i.e., streams where no further data will be sent). The number of streams that have been prioritized but remain in the "idle" state plus the number of active streams (those in the "open" state or in either of the "half-closed" states; see Section 5.1.2 of [HTTP/2]) 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.

Servers MUST NOT send PRIORITY_UPDATE frames. 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 that uses 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 [HTTP/3]). 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 of 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 that has not yet been promised, this MUST be treated as a connection error of type H3_ID_ERROR.

Servers MUST NOT send PRIORITY_UPDATE frames of either type. 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 Priority 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; the 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 priority 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.

The 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 request header field, in which omission of a priority parameter implies the use of its default value (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`, i.e., 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

It is generally beneficial for an HTTP server to send all responses as early as possible. However, when serving multiple requests on a single connection, there could be competition between the requests for resources such as connection bandwidth. This section describes considerations regarding how servers can schedule the order in which the competing responses will be sent when such competition exists.

Server scheduling is a prioritization process based on many inputs, with priority signals being only one form of input. 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, it is not possible to describe a universal scheduling algorithm. This document provides some basic, non-exhaustive recommendations for how servers might act on priority parameters. It does not describe in detail how servers might combine priority signals with other factors. Endpoints cannot depend on particular treatment based on priority signals. Expressing priority is only a suggestion.

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 responses of the same urgency SHOULD be served by prioritizing bandwidth allocation in ascending order of 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 among them. The message content of incremental responses is used as parts, or chunks, 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 by the scheduling guidance based on urgency and request generation order might lead to suboptimal results at the client, as early non-incremental responses might prevent the 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 for doing so is an implementation decision. For example, a server might preemptively 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-signaled 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 in flight. 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 stream carries a CONNECT request, 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 recommendations for handling of the incremental parameter (Section 10) are likely to schedule these fairly, preventing one CONNECT stream from blocking others.

12. Retransmission Scheduling

Transport protocols such as TCP and QUIC provide reliability by detecting packet losses and retransmitting lost information. In addition to the considerations in Section 10, scheduling of retransmission data could compete with new data. The remainder of this section discusses considerations when using QUIC.

Section 13.3 of [QUIC] states the following: "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 considerations regarding 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

Typically, HTTP implementations depend on the underlying transport to maintain fairness between connections competing for bandwidth. When an intermediary receives HTTP requests on client connections, it forwards them to backend connections. Depending on how the intermediary coalesces or splits requests across different backend connections, different clients might experience dissimilar performance. This dissimilarity might expand if the intermediary also uses priority signals when forwarding requests. 13.1 and 13.2 discuss mitigations of this expansion of unfairness.

Conversely, Section 13.3 discusses how servers might intentionally allocate unequal bandwidth to some connections, depending on the priority signals.

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 carry signals indicating higher priority than those coming from others.

It is sometimes beneficial for the server running behind an intermediary to obey Priority header field values. As an example, a resource-constrained server might defer the transmission of software update files that have the background urgency level (7). However, in the worst case, the asymmetry between the priority declared by multiple clients might cause all responses going to one user agent to be delayed until all responses going to another user agent have been sent.

In order to mitigate this fairness problem, a server could use knowledge about the intermediary as another input 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 can check to see if the 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 Content Delivery Network (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 backend servers is done using HTTP/1.1. Unlike connection coalescing, the CDN will "demux" requests into discrete connections to the back end. Response multiplexing in a single connection is not supported by HTTP/1.1 (or older), so there is not a fairness problem. However, backend servers MAY still use client headers for request scheduling. Backend 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?

In contrast to the prioritization scheme of HTTP/2, which uses a hop-by-hop frame, the Priority header field is defined as "end-to-end".

The way that a client processes a response is a property associated with the 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.

15. Security Considerations

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 responses.

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

16. IANA Considerations

This specification registers the following entry in the "Hypertext Transfer Protocol (HTTP) Field Name Registry" defined in [HTTP/2]:

Field Name: Priority
Status: permanent
Reference: This document

This specification registers the following entry in the "HTTP/2 Settings" registry defined in [HTTP/2]:

Code: 0x9
Name: SETTINGS_NO_RFC7540_PRIORITIES
Initial Value: 0
Reference: This document

This specification registers the following entry in the "HTTP/2 Frame Type" registry defined in [HTTP/2]:

Code: 0x10
Frame Type: PRIORITY_UPDATE
Reference: This document

This specification registers the following entry in the "HTTP/3 Frame Types" registry established by [HTTP/3]:

Value: 0xF0700-0xF0701
Frame Type: PRIORITY_UPDATE
Status: permanent
Reference: This document
Change Controller: IESG
Contact: ietf-http-wg@w3.org

IANA has created the "Hypertext Transfer Protocol (HTTP) Priority" registry at <https://www.iana.org/assignments/http-priority> and has populated it with the entries in Table 1; see Section 4.3.1 for its associated procedures.

Name	Description	Reference
u	The urgency of an HTTP response.	Section 4.1
i	Whether an HTTP response can be processed incrementally.	Section 4.2

Table 1: Initial Priority Parameters

17. References

17.1. 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/info/rfc9110>.

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

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

[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/info/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/info/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/info/rfc8126>.

[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/info/rfc8174>.

[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/info/rfc8941>.

17.2. Informative References

[CACHING] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Caching", STD 98, RFC 9111, DOI 10.17487/RFC9111, June 2022, <https://www.rfc-editor.org/info/rfc9111>.

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

[MARX] Marx, R., De Decker, T., Quax, P., and W. Lamotte, "Of the Utmost Importance: Resource Prioritization in HTTP/3 over QUIC", DOI 10.5220/0008191701300143, SCITEPRESS Proceedings of the 15th International Conference on Web Information Systems and Technologies (pages 130-143), September 2019, <https://www.doi.org/10.5220/0008191701300143>.

[PRIORITY-SETTING] Lassey, B. and L. , "Declaring Support for HTTP/2 Priorities", Work in Progress, draft-lassey-priority-setting-00, Work in Progress, 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/info/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/info/rfc7540>.

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

Acknowledgements

Roy Fielding presented the idea of using a header field for representing priorities in <https://www.ietf.org/proceedings/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 [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, Matthew Cox, Mike Bishop, Roberto Peon, Robin Marx, Roy Fielding, and the authors of this document.

Yang Chi contributed the section on retransmission scheduling.

Authors' Addresses

Kazuho Oku
Fastly
EMail: kazuhooku@gmail.com

Additional contact information:
##
Fastly
EMail: kazuhooku@gmail.com

Lucas Pardue
Cloudflare
EMail: lucaspardue.24.7@gmail.com