Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 26, 2015

H. Ruellan
Y. Fablet
R. Bellessort
Canon CRF
J. Fujisawa
Canon, Inc.

January 22, 2015

HTTP/2 Priority Tree Synchronization
draft-ruellan-priority-tree-sync-01

Abstract

This specification describes an issue in HTTP/2 linked to the synchronization of priority trees between a client and a server. It outlines possible solutions to this issue.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on July 26, 2015.

Copyright Notice

Table of Contents

1. Introduction

HTTP/2 [HTTP2] allows multiplexing messages over a single connection. A client can express the processing order it expects from the server for its requests, by using HTTP/2 priority mechanism. Using this mechanism, the client requests are organized in a priority tree.

The priority tree evolves as new requests are sent by the client, and as older requests are fulfilled by the server. Due to this dynamic nature, the client and the server can have different views of the priority tree. A discrepancy can cause issues, mainly due to the removal of requests from the priority tree.

Section 2 details this synchronization issue and its possible consequences.

Section 3, Section 4, and Section 5 draw rough sketches of possible solutions to this synchronization issue.

1.1. 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 RFC 2119 [RFC2119].

All numeric values are in network byte order. Values are unsigned unless otherwise indicated. Literal values are provided in decimal or hexadecimal as appropriate. Hexadecimal literals are prefixed with "0x" to distinguish them from decimal literals.

2. Problem Overview

2.1. HTTP/2 Priorities

HTTP/2 [HTTP2] allows multiplexing concurrent messages on the same connection. Each message exchange is carried by a stream. A client can express how it would prefer the server allocate resources for the concurrent streams, by using HTTP/2 priority mechanism (Section 5.3 of [HTTP2]).

Streams are organized into a priority tree by making each stream depend on another stream. A stream is processed only when all its parents in the priority tree have been processed.

Each stream is allocated a weight. This weight is used to determine the relative share of resources that are allocated to streams depending on the same parent.

A priority is set for a stream by defining its parent stream (i.e., the stream it depends on), and its weight (a value between 1 and 256). By default, the priority for a stream is to depend on no stream, and to have a weight of 16.

A client can define the priority for a stream when creating it. It can later change this priority to reflect new expectations regarding the allocation of resources by the server.

2.2. Priority Usage

A server needs to control the amount of memory used by a HTTP/2 connection. To this end, it can limit the maximum number of simultaneous streams that a client is allowed to create. It also needs to remove streams from the priority tree once they are closed. However, the client could rely on these closed streams to place new streams in the priority tree. If the server receives a priority for a stream referencing a stream no longer in its priority tree, the

default priority is assigned to the stream. This can lead to suboptimal behaviour.

For example, when downloading a web page, a client can prioritize the resources used by the page to optimize the download speed as perceived by the user. To this end, the client organizes its priority tree to download less important resources after the more important ones. The stream for a less important resource is prioritized as depending on a stream for a more important resource. If the server is not able to apply this priority, because it has not kept priority information for the latter stream, it will use a default priority for the less important resource. As a result, this less important resource will be downloaded concurrently with much more important resource, and the downloading of the web page will not be optimized according to the client expectations.

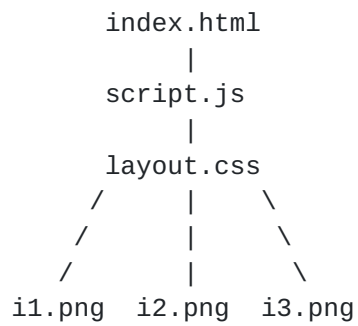
Another example is the downloading of two web pages in parallel, one in the foreground, the other in the background. The client can prioritize the resources to ensure the web page in the foreground is downloaded faster that the web page in the background. To be able to react to the user inverting the foreground and the background web pages, the client can organize the resources corresponding to each web page in a different branch of the priority tree. By changing the weights of the root of each branch, the client can change the relative download speeds of the two pages. However, if the server does not keep priority information for these roots, it will not be able to apply the weight changes sent by the client, and the client will not be able to change the relative download speed of the pages.

As seen in these examples, not all streams are of the same importance to the client for defining new priorities. As a general rule, recent streams are more useful to the client as it will use them to define the priorities of new streams. However, there are two particular cases that can be used by the client to structure the priority tree.

First, some streams are used as branching points in the priority tree. A branching point has several children that are intended to be processed in parallel. A branching point is useful to the client for adding further streams to be processed in parallel, alongside the existing children of the branching point.

[[CREF1: Revise the figures to include Idle streams.]]

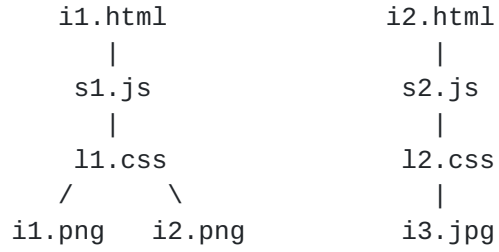
For example, the following priority tree allows downloading the three images in parallel:



Branching Point

Second, some streams are at the root of priority tree branches. These streams are useful to the client for changing the general priority of a whole branch of the priority tree.

For example, the following priority tree contains two branches, each corresponding to a web page:



Branch Root

3. Priority Retention

3.1. Overview

The client can ask the server to keep the priority state for a stream for some time after the stream is closed. A new frame PRIORITY_RETENTION (Section 3.2) is defined to allow this.

This frame can also be used by the client to indicate that it no longer needs the server to keep the priority state corresponding to a stream.

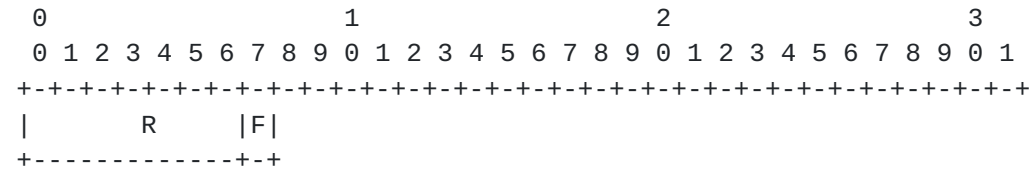
3.2. The PRIORITY_RETENTION Frame

The PRIORITY_RETENTION HTTP/2 frame (Section 4 of [HTTP2]) allows an endpoint to transmit priority state retention information to its peer.

The PRIORITY_RETENTION frame is a non-critical extension to HTTP/2. Endpoints that do not support this frame can safely ignore it.

An endpoint willing to support receiving the PRIORITY_RETENTION frame from a peer can announce it by sending a PRIORITY_RETENTION frame on stream 0.

The PRIORITY_RETENTION frame type is TBD.



PRIORITY_RETENTION frame PAYLOAD

The PRIORITY_RETENTION frame contains the following fields:

R: A reserved 7-bit field.

F: A 1-bit field indicating the retention status of priority information for the stream.

The value 1 means that the priority state for the stream is retained.

The value 0 means that the priority state for the stream is not retained.

The PRIORITY_RETENTION frame does not define any flags.

An endpoint can request its peer to retain priority information for a stream by sending a PRIORITY_RETENTION frame with the F field set to the value of 1 on this stream.

An endpoint can inform its peer that it no longer needs to retain priority information for a stream by sending a PRIORITY_RETENTION frame with the F field set to the value of 0 on this stream.

[[CREF2: The detailed usage of the PRIORITY_RETENTION frame needs to be defined.]]

3.3. Evaluation

This extension enables the client to ask the server to retain some specific priority information. As such, the client has a good control over the priority tree of the server and can use many possible strategies for organizing the shape of the priority tree.

The client is however limited in that it can't ask the server to retain a too large number of streams, otherwise the memory consumption on the server side would be too large.

4. Priority Pruning Algorithm

4.1. Overview

The server can use a well-defined algorithm for selecting which priority states to keep for closed streams, and which to delete from memory. The client can replicate this algorithm to know on which streams to rely for defining new priorities.

The algorithm defines the number of priority states kept by the server. By default, it is the same number as the maximum number of streams the client can open. This can be changed through a new setting parameter, SETTING_PRIORITY_STATES (Section 4.3).

The priority states are by default deleted in the stream creation order. However, this order is modified to keep longer two types of streams:

- Streams that are branching points in the priority tree: those that have several child streams. These streams are useful to define parallel processing.

- Streams that are at the root of a branch of the priority tree. These streams are useful for changing priorities on a large scale.

4.2. Algorithm

To select the priority states to keep for closed streams, the server applies the following algorithm:

1. The server creates a list containing all the closed streams and orders it according to the stream creation order. The oldest stream is the first in the list, while the newest one is the last.

2. Each closed stream that has at least two children is moved after the latest of its child present in the list.

3. Each closed stream that depends on no other stream and that has at least one descendant is moved after the latest of its descendant in the list.

4. Priority states are kept only for the streams at the end of the list, such that the number of kept priority states is lower than or equal to the value of the SETTING_PRIORITY_STATES (Section 4.3) parameter.

4.3. The SETTING_PRIORITY_STATES parameter

The SETTING_PRIORITY_STATES SETTINGS parameter (Section 6.5.2 of [HTTP2]) indicates the number of priority states kept for closed streams by the endpoint.

This parameter identifier is TBD.

The initial value for this parameter is 100. It is recommended that the value for this parameter be at least the same as the value of the SETTING_MAX_CONCURRENT_STREAMS parameter.

The usage of this new setting parameter doesn't require any negotiation between peers. Upon sending this setting parameter, an endpoint informs its peer that it uses the pruning algorithm described above (Section 4.2) for selecting for which closed streams priority states are kept.

A peer receiving this setting parameter and understanding it can choose to take advantage of it to compute the priority state information kept by the sending endpoint.

[[CREF3: The detailed usage of the SETTING_MAX_CONCURRENT_STREAMS parameter needs to be defined.]]

4.4. Evaluation

This extension enables the client to have a good knowledge of the closed streams for which priority information is kept by the server. Using this information, the client can define priorities knowing reliably that the server will be able to apply them.

However, this extension is based on assumptions on which streams are the most useful to the client for defining priorities. If these assumptions don't hold, then the client may not be able to fully express its expectations for the processing order of its requests by the server.

5. Server Feedback

5.1. Overview

When the server is not able to apply a priority sent by the client, it fails silently. To mitigate the consequences of this failure, the server could send feedback to the client.

A new frame UNAPPLIED_PRIORITY (Section 5.2) is defined to allow the server to inform the client that a priority has not been applied.

5.2. The UNAPPLIED_PRIORITY Frame

The UNAPPLIED_PRIORITY HTTP/2 frame (Section 4 of [HTTP2]) allows an endpoint to inform its peer that the priority it received was not applied. The UNAPPLIED_PRIORITY frame is sent on the stream for which the priority was not applied.

The UNAPPLIED_PRIORITY frame is a non-critical extension to HTTP/2. Endpoints that do not support this frame can safely ignore it.

The UNAPPLIED_PRIORITY frame type is TBD.

The UNAPPLIED_PRIORITY frame has no payload.

The UNAPPLIED_PRIORITY frame does not define any flags.

[[CREF4: The detailed usage of the UNAPPLIED_PRIORITY frame needs to be defined.]]

5.3. Evaluation

This extension provides a lightweight way for the server to inform the client when it cannot apply a priority sent for a stream.

While this feedback enables the client to know that a priority has not been applied by the server, it provides little information on how to change the priority in order for the server to able to apply it.

6. Security Considerations

The different extensions proposed in this specification introduce new HTTP/2 setting parameters, or new HTTP/2 frames that could be abused in the same way as existing setting parameters and frames.

The PRIORITY_RETENTION (Section 3.2) frame can be abused to cause a peer to retain a large amount of priority state by sending only priority retention requests.

7. Normative References

[HTTP2] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol version 2", draft-ietf-httpbis-http2-13 (work in progress), June 2014.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

Appendix A. Change Log (to be removed by RFC Editor before publication)

Authors' Addresses

Herve Ruellan
Canon CRF

EMail: herve.ruellan@crf.canon.fr

Youenn Fablet
Canon CRF

EMail: youenn.fablet@crf.canon.fr

Romain Bellessort
Canon CRF

EMail: romain.bellessort@crf.canon.fr

Jun Fujisawa
Canon, Inc.

EMail: fujisawa.jun@canon.co.jp