Accept-Push-Policy Header Field
draft-ruellan-http-accept-push-policy-00

Abstract

The "Accept-Push-Policy" and "Push-Policy" header fields enable a client and a server to negotiate the behaviour of the server regarding the usage of push on a per-request basis.

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 April 18, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.
1. Introduction

HTTP/2 [RFC7540], the new version of the HTTP protocol, not only provides significant improvements compared to HTTP/1.1 (see [RFC7230] and [RFC7231]), but also provides several new features. Among these is Server Push, which enables a server to send responses to a client without having received the corresponding requests.
The range of possibilities offered by Server Push is a new domain wide open for experimentation. A first usage was foreseen early in the addition of this feature into HTTP/2, which is to replace the inlining of sub-resources inside a main resource, by pushing these sub-resources in response to the request for the main resource. As described in [HighPerformance], with HTTP/1.1 a web designer may want to optimize the page load time by packing a whole web page into a single HTTP response. This can be achieved by inlining the CSS, JavaScript, and images inside the HTML document. By removing the need for the client to send requests for these sub-resources, this inlining technique can reduce the page load time by roughly a RTT. With HTTP/2, the same results can be obtained by pushing the sub-resources instead of inlining them. Using push has the advantage of keeping each sub-resource independent.

HTTP/2 provides a few ways of controlling Server Push from the client side. First, the SETTINGS parameter "SETTINGS_ENABLE_PUSH" allows a client to globally enable or disable push on a HTTP/2 connection. In addition, HTTP/2 Flow Control can be used to limit the bandwidth used by pushed resources.

These options provide only a coarse control of the usage of Server Push from the client side. In some cases, a more fine-grained control would be useful. This document describes several use cases where controlling Server Push would be useful for the client. It then proposes new header fields for realizing this control.

1.1. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [RFC2119] and indicate requirement levels for compliant implementations.

This document uses the Augmented BNF defined in [RFC5234].

2. Push Control Use Cases

2.1. Adapting Push Behaviour
A browser may want to ask the server to adapt its behaviour for pushing resources depending on the user's actions. For example, after navigating through a site for some time, the browser may have many sub-resources in its cache and may prefer that the server doesn't push sub-resources anymore to prevent wasting bandwidth. This could be further optimized with the browser asking the server to push only response metadata (i.e., the responses pushed by the server correspond to requests made with the HEAD method instead of requests made with the GET method). By receiving in advance the list of sub-resources corresponding to a specific request, the browser would be able to fetch early on any missing sub-resource.

As another example, when a user opens many pages on the same site, the browser may want to receive pushed sub-resources only for the foreground tab and not for any background tab. This results in a better optimization of the page load time for the tab that is visible to the user.

2.2. Load Balancer

A second use case is a load balancer serving both HTTP/1.1 and HTTP/2 clients, and using HTTP/2 to connect to the backend servers, as described in [LoadBalancer].

The load balancer uses the same HTTP/2 connection towards a backend server to forward the requests received from several clients. When the client is a HTTP/1.1 client, the load balancer doesn't want the backend server to push any resource in response to the client's request. On the contrary, when the client is a HTTP/2 client, the load balancer would like the backend server to push sub-resources associated to the client's request.

The load balancer would like to be able to enable or disable push on a per-request basis. This would enable it to optimize the server behaviour depending on the client's capacity.

2.3. DASH Fast Start

Controlling the server behaviour regarding push may also be useful for specific applications. As an example, DASH [DASH] is a technology for streaming media content over HTTP. The media content
is split into small file-based segments that can be retrieved through HTTP requests. Potentially, the media content is made available with different quality levels. A media presentation description (MPD) describes the organization of the media.

To render a media, a DASH client needs to first download the MPD, process it, and then request the necessary media segments. When requesting a MPD to play the associated media content, it would be useful for a DASH client to be able to ask the server to push some initial content (for example, the initialization segments, and possibly the first content segments).

However, there are also cases when it is not useful for the DASH client to receive in advance this initial content. For example, in a video program guide, the DASH client may want to download several MPDs corresponding to different media content, but doesn't want to receive the initial content for all of these. Therefore, it is useful for the DASH client to be able to specify in a request for a MPD whether it wants the server to push some initial content.

2.4. Fast Page Load

The previous use case can be expanded to the more generic use case of downloading quickly a web page. As described in [Breaking1000msBarrier], it is important for the user perception to keep the perceived latency of loading a web page under 1000 ms. This can be difficult when using a mobile connection with a high latency. Part of the solution proposed in [Breaking1000msBarrier] for HTTP/1.1 is to inline all the sub-resources necessary for achieving a first rendering of the web page. With HTTP/2, the inlining of these sub-resources can be replaced by having the server push them.

Therefore, a client detecting that it is using a high-latency network could improve the user perceived latency by asking the server to push
all the sub-resources necessary for a first display of a web page.

2.5. Use Cases Requirements

The analysis of these use cases enables to build a list of requirements for defining a fine-grained control over the usage of push by a server.

- The client can ask the server not to push any resource in response to a request.
- The client can ask the server to only push response metadata.
- The client can ask the server to use an application-defined behaviour regarding push.
- The server can indicate to the client its behaviour regarding push when processing a request.

3. Push Policy

A push policy defines the server behaviour regarding push when processing a request. Different push policies can be used when processing different requests.

This section defines new header fields enabling a client and a server to negotiate the push policy used by the server to process a given request.

The new "Accept-Push-Policy" header field enables a client to express its expectations regarding the server's push policy for processing a request.

The "Push-Policy" header field enables a server to state the push policy used when processing a request.

3.1. The Accept-Push-Policy Header Field

A client can express the desired push policy for a request by sending an "Accept-Push-Policy" header field in the request.
Accept-Push-Policy = token ; a push policy name

The header field value contains the push policy that the client expects the server to use when processing the request.

Possibly, the "Accept-Push-Policy" header field could be extended to support carrying multiple policies, as a comma-separated list of tokens. The server could choose its preferred policy among those proposed by the client.

3.2.  Push-Policy Header Field

A server can indicate to a client the push policy it used when processing a request by sending a "Push-Policy" header field in the corresponding response.

Push-Policy = token ; a push policy name

The server MUST follow the indicated push policy when processing the client request associated to the response.

The "Push-Policy" header field can be used as an acknowledgement from the server after receiving a request containing the "Accept-Push-Policy" header field.

If the "Accept-Push-Policy" header field can contain a list of push policy names, the "Push-Policy" header field can be used to express which push policy was selected by the server.

3.3.  Push Policy Values

This section defines some generic push policies. Other push policies
can be standardized for either a generic usage, or for an application-specific usage. In addition, private push policies can be used by a web application.

TBD: select the form of private push policies (URN, "X-") values...).

### 3.3.1. None Push Policy

The "None" push policy value indicates that no resource is pushed when processing a request.

For example, a browser sending a request for a background tab could ask the server not to push any resources in response to this request by sending an "Accept-Push-Policy" header with the "None" value. This would result in the following HTTP/2 header block:

```
:method = GET
:scheme = https
:path = /index.html
host = example.org
accept = text/html
accept-push-policy = none
```

### 3.3.2. Head Push Policy

The "Head" push policy value indicates that only response metadata are pushed (the server is pushing responses corresponding to requests made with the HEAD method).

For example, a browser may already have many resources from a web site in its cache. It could ask the server to push only response metadata. This would allow the browser to know early on the resources useful for rendering a web page (i.e., before receiving and parsing the HTML document), without taking the risk of wasting bandwidth with resources already in its cache. In this example, the browser's request would contain the following HTTP/2 header block:

```
:method = GET
:scheme = https
:path = /index.html
```
3.3.3. Default Push Policy

The "Default" push policy value indicates that the server is using its default behaviour for pushing resources when processing a request.

For example, a server not fulfilling a client's expectation regarding the push policy could indicate this with the "Default" push policy. It would send the following HTTP/2 header block in its response:

```
:status 200
push-policy = default
```

3.3.4. Fast-Load Push Policy

The "Fast-Load" push policy value indicates that the sub-resources necessary for a first rendering of a main resource are pushed alongside the response containing this main resource.

A server using the "Fast-Load" push policy while processing a request can push sub-resources not necessary for a first rendering, but SHOULD prioritize sub-resources necessary for this first rendering.

For example, a client detecting that it is using a high-latency network can try to improve the user perceived latency by asking the server to push the sub-resources necessary for a first rendering of a main page by including an "Accept-Push-Policy" header with the "Fast-Load" value. This would result in the following HTTP/2 header block:

```
:method = GET
:scheme = https
:path = /index.html
host = example.org
accept = text/html
accept-push-policy = fast-load
```

4. IANA Considerations

TBD
5. Security Considerations

TBD

6. References

6.1. Normative References


6.2. Informative References


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

Franck Denoual
Canon CRF
Email: francoz.denoual@crf.canon.fr

Frederic Maze
Canon CRF
Email: frederic.maze@crf.canon.fr