Network Working Group	A. Silvas
Internet-Draft	GoDaddy
Intended status: Experimental	June 26, 2016
Expires: December 28, 2016	

Push-Assets Header Field

draft-asilvas-http-push-assets-00

Abstract

Push-Assets is a header field that provides the necessary client state in order for servers to utilize HTTP/2 Server Push with confidence in knowing what resources SHOULD or SHOULD NOT be sent, reducing waste, and ultimately providing an improved user experience. This document will provide an overview of Push-Assets requirements, and describes any implementation concerns.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

As described in [HighPerformance], transfer sizes and resource counts continue to increase. While network conditions continue to improve, resulting in lower latencies and increased bandwidth, HTTP/1.1 ([RFC7230] and [RFC7231]) fails to address the underlying problem of resource dependencies and the resulting "waterfall" of blocked requests.

HTTP/2 [RFC7540] aims to address some of these problems, by way of Streams and Multiplexing, combined with HTTP/2 Server Push [RFC7540]. A ruthless combination, addressing "head-of-line blocking" through Multiplexing, and optimistic pre-loading by way of Server Push.

Where Server Push begins to fall short is around client state, leaving it up to servers to leverage existing HTTP State Management Mechanism [RFC6265] with Cookies, which are not purpose built to solve the problem of resource dependency state. This lack of client state can result in HTTP/2 [RFC7540] RST_STREAM, where-in in-flight Server Push Streams will be cancelled, incurring client and server waste.

This document aims to address resource dependency state by looking to Caching [RFC7234] familiar with existing HTTP/1.1 requests (see [RFC7230] and [RFC7231]). By pulling this state data into the request, servers are able to intelligently and responsibly Server Push only missing or outdated resource.

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. Understanding The Problem

Here we can begin to see the problem with vanilla HTTP/2 [RFC7540] Server Push without client state management.

```
+--------------------------+------------------------+
| Client                   | Server                 |
+--------------------------+------------------------+
| [        TCP+TLS+HTTP/2 Negotiation             ]  |
|                          |                        |
| GET /A.html      =======> |                        |
|                <======= | PUSH_PROMISE GET /A1.js |
|                <======= | PUSH_PROMISE GET /AB.js |
|                <======= | GET /A.html             |
|                <======= | GET /A1.js             |
|                <======= | GET /AB.js             |
| [Page Complete]          |                        |
|                          |                        |
| [Navigate to "/B"]       |                        |
|                          |                        |
| GET /B.html      =======> |                        |
|                <======= | PUSH_PROMISE GET /B1.js |
|                <======= | PUSH_PROMISE GET /AB.js | (Server doesn't know Client has this)
|                <======= | GET /B.html             |
|                <======= | GET /B1.js             |
| RST_STREAM /AB.js =======> | (Hault transfer)       |
|                <======= | GET /AB.js             | (Cancelled pre-flight, may be fully
|                          |                        | transferred before RST_STREAM
| [Page Complete]          |                        | received)
+--------------------------+------------------------+
```

While in some situations cookie-based management will address the above, ultimately it'll vary depending on the complexity of the origin, including but not limited to the number of assets and the frequency of change.

Figure 1

With Push-Assets enabled both client and server adhere to a strict dependency state contract.

```
+--------------------------+------------------------+
| Client                   | Server                 |
+--------------------------+------------------------+
| [        TCP+TLS+HTTP/2 Negotiation             ]  |
|                          |                        |
| GET /A.html      =======> |                        |
|                <======= | PUSH_PROMISE GET /A1.js |
|                <======= | PUSH_PROMISE GET /AB.js |
|                <======= | GET /A.html             |
|                <======= | GET /A1.js             |
|                <======= | GET /AB.js             |
| [Page Complete]          |                        |
```

```
|                        |                        |
| [Navigate to "/B"]     |                        |
|                        |                        |
| GET /B.html      =======> |                      |
|                <======= | PUSH_PROMISE GET /B1.js |
|                <======= | GET /B.html          |
|                <======= | GET /B1.js           |
| [Page Complete]        |                        |
+------------------------+------------------------+
```

Avoiding needless waste, the benefits of Push-Assets far outweighs the additional header data needed to track client state.

Figure 2

3. Push Assets Use Cases

3.1. First Load Experience

Often the most import visit to a site is the first. Push-Assets provides the necessary client state for the server to confidently know which resources are missing or outdated.

3.2. Subsequent Load Experience

As users navigate to previously visited resources, or new resources where some shared resources have been cached, Push-Assets provides the necessary client state to make efficient use of Server Push, only sending what resources the client does not already have.

3.3. Proxy Optimization

On one end of the spectrum of proxies lies your server proxies, with CDN's on the other end.

```
[Client] <==[CDN]<======================[Proxy]<==[Origin]
```

Figure 3

With Push-Assets providing efficient communication between two points, this may lend to potential benefits between Proxies and their Origin server as well. While the Proxy nearest your Client SHOULD support Push-Assets for best results, it MAY elect not to also leverage Push-Assets between the Proxy and Origin.

For proxies with caching nearest to Client (namely CDN's), they may further benefit from Push-Assets by way of efficient use of Server Push.

3.4. Non-Browser Clients

By enabling Push-Assets between any two points, Server Push can be used to reduce waste and provide improved performance. The greater the shared resources, the greater the potential benefits.

3.5. Alternative Content Types

With Push-Assets being nothing more than an HTTP Header, extending the benefits to other Content Type's [RFC2045] is entirely up to the Client and Server. Consider circumstances where you retrieve a JSON resource, which signals relationships with other resources. Push-Assets reduces waste and enables better

user experiences irrespective of Content-Type.

4. Push-Assets Header

```
Push-Assets = [*][Asset-Key=Caching-Headers][;Asset-Key=Caching-Headers]
```

A request header field SHALL be sent by the client when requesting the server to support Push-Assets.

Comprised of zero or more resources addressed by their Asset-Key.

An Asset-Key is the name of the resource uniquely identifiable by the resource or matching resources.

4.1. Caching Headers

```
Push-Assets = Asset-Key=[etag(etag-value),][last-modified(date)][no-push]
```

Caching MAY include an etag, and/or last-modified, or no-push. This provides necessary client state of dependencies to server.

4.2. Empty Cache Request

```
Push-Assets = *
```

Where * informs server to Server Push all push-enabled dependencies, if Push-Assets is enabled. Servers MUST push all missing or outdated push-enabled resources.

5. Push-Asset-Key Header

```
Push-Asset-Key = Asset-Key
```

A PUSH_PROMISE response Header field MAY be sent to inform the client that the resource should be tracked as a Push-Asset.

The Asset-Key MUST be stored in the header field as an MD5 representation of the desired Key.

Unlike the Asset-Key in a request, the Push-Asset-Key header field corresponds to the Key of the PUSH_PROMISE response.

5.1. Named Key

```
Push-Asset-Key = core-bundle.js
```

By naming a resource, you MAY share that resource across multiple resources, and MAY change the URI [RFC3986] as necessary without resulting in wasted requests.

5.2. Key from URI Path

```
Push-Asset-Key = $
```

Where $ is reserved as a short-hand for the client to recognize the key as the URI Path [RFC3986], and MUST NOT include the query string.

Example URI Path [RFC3986] of /my/resource?some=thing would by keyed as /my/resource.

If there are more than one cached resources on the client for a given Push-Asset-Key, the client MUST treat the most recent Key as the current version.

6. Push-Asset-Match Header

```
Push-Asset-Match = Asset-Path[;Asset-Path]
```

An OPTIONAL PUSH_PROMISE response header field.

An Asset-Match supports the lexical matching of the URI Path [RFC3986], and MAY end with reserved wildcard * to indicate matching all requests "equal or greater than" the URI Path. While one or more Asset-Path's may be provided, they SHOULD be consistent between requests to avoid any caching proxies from serving varying responses. Usage of Vary header field (Section 7.1.4 of [RFC3986]) MAY be applied with Push-Asset-Match to permit varying responses, but SHOULD NOT be used in most scenarios to avoid unnecessary complexity.

6.1. Match Similar Requests

```
Push-Asset-Match = /some-path/*
```

Where all requests with URI Path [RFC3986] greater than or equal to /some-path/ will be matched.

6.2. Match All Requests

```
Push-Asset-Match = *
```

* is reserved to indicate "match all requests". This is the equivalent of /*, matching all from root.

7. Responsible Usage

State management can be simple for simple origins, but complex for complex origins. Following is a set of usage scenarios and suggested tactics to combat unnecessary waste.

7.1. Frequently Changing Paths

Not uncommon amongst websites are changes to the URI Path [RFC3986] of a resource when contents change. For these assets, utilizing the default Push-Asset-Key of $ MAY result in excessive waste by way of the client sending state of matching resources that are no longer applicable.

An effective measure is to leverage a uniquely named Push-Asset-Key, enabling the client and server to understand that the resource has effectively been renamed.

7.2. Excessive Matching

Leveraging the power of the Push-Asset-Match header field MAY greatly improve the efficiency of resources shared amongst many resources. If used excessively where-in many requests do not depend on the matched resource MAY lead to waste, as the state of matching resources are sent via Push-Assets header field.

The server MAY improve effectiveness by way of highly specific Push-Asset-Match definitions, breaking an origin into multiple sub-paths to permit parts an Origin to operate without negative affect from other parts of Origin. For example, /some-path/ does NOT need to use the same shared resources as /some-other-path/, as they MAY NOT know about one another.

In cases where even with highly specific Push-Asset-Match do not address excessive matching, the client MAY track historical false positives where-in matching resources are not served from requested resources, and MAY determine a threshold from which the client MAY elect NOT to send alongside Push-Assets requests.

7.3. Varying Content Types

With Push-Assets not being specific to html resources, clients MUST NOT match resources across requests with varying Content Type's [RFC2045].

If a server has enabled Push-Assets for more than one Content Type, the client MUST only notify the server of matching resources that were from the same Content Type of the parent resource.

As an example, "/home.html" with a dependent resource that matches all URI Paths, MUST NOT be sent via Push-Assets when making a request for "/file.js" as the parent resources differ in Content Type.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997.

[RFC3986] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008.

[RFC7230] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014.

[RFC7231] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014.

[RFC7234] Fielding, R., Nottingham, M. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014.

[RFC7540] Belshe, M., Peon, R. and M. Thomson, "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015.

8.2. Informative References

[HighPerformance] Grigorik, I., "High Performance Browser Networking", September 2013.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011.

Author's Address

Aaron Silvas
GoDaddy
EMail: asilvas@godaddy.com