Network Working Group
Internet-Draft
Intended status: Experimental
Expires: November 29, 2016

A. Silvas
GoDaddy

June TBD, 2016

Push-Assets Header Field
draft-asilvas-http-push-assets-00

Abstract

"Push-Assets" request header combined with response headers "Push-Asset-Key" and "Push-Asset-Match" provide necessary state to be negotiated between client and server, offering greatly improved usage HTTP/2 "Server Push" on every request.

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 November 29, 2016.

Table of Contents

1. Introduction 2
1.1. Terminology 3
2. Understanding The Problem 3
3. Push Assets Use Cases 4
3.1. First Load Experience 4
3.2. Subsequent Load Experience 4
3.3. Proxy Optimization 4
3.4. Non-Browser Clients 5
3.5. Alternative Content Types 5
4. Push-Assets Header 5
4.1. Caching Headers 5
4.2. Empty Cache Request 5
5. Push-Asset-Key Header 6
5.1. Named Key 6
5.2. Key from URI Path 6
6. Push-Asset-Match Header 6
6.1. Match Similar Requests 7
6.2. Match All Requests 7
7. References 7
7.1. Normative References 7
7.2. Informative References 7
Author’s Address 8

1. Introduction

As described in [HighPerformance], transfer sizes and document requests 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 document 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 [RFC7540] Server Push. 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 document 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 document 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 parent request, servers are able to intelligently and responsibly "Server Push" only missing or outdated dependencies.

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 using one of many custom cookie-based implementations to manage state.

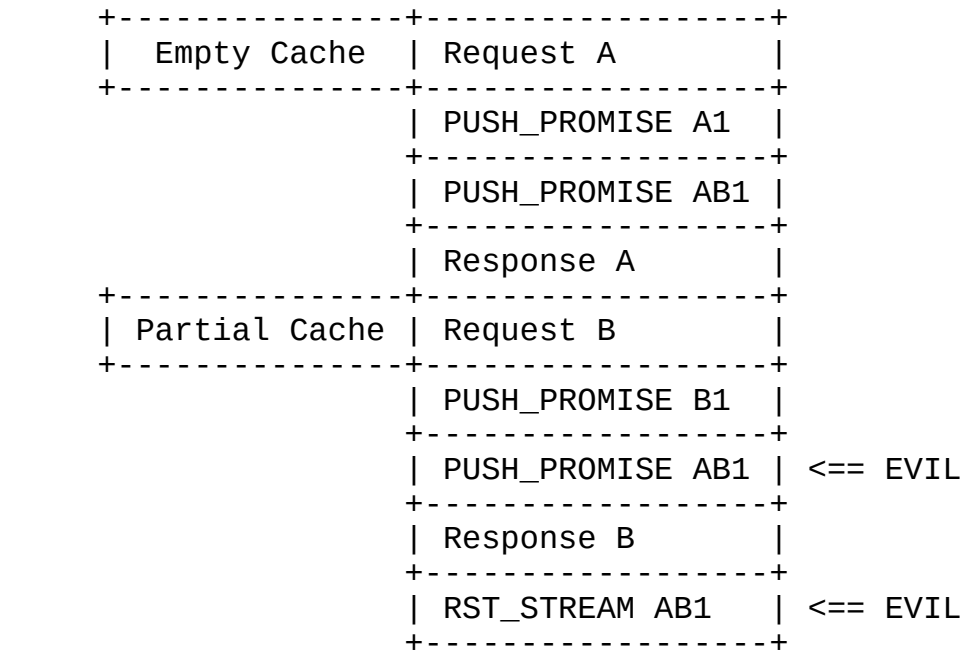

Figure 1

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

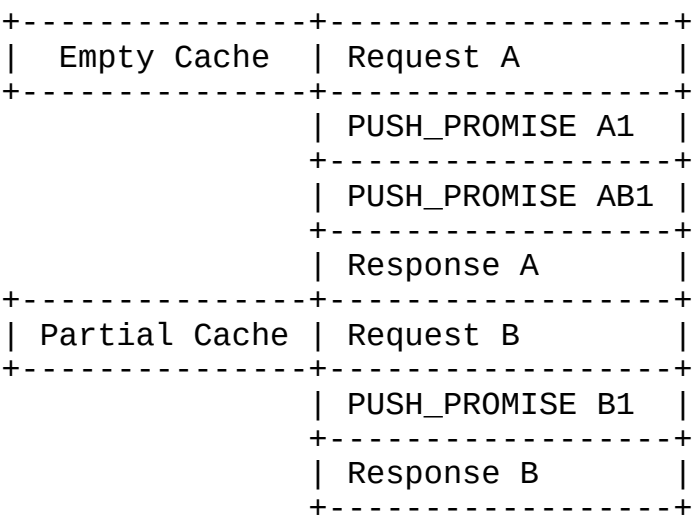

Figure 2

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 pages, or new pages where some shared resources have been cached, "Push-Assets" provides ample client state to continue making 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.

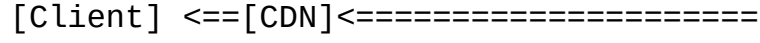

Figure 3

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" for an efficient utilization of "Server Push".

3.4. Non-Browser Clients

Leveraging the benefits of "Push-Assets" allows for more efficient communication with compatible servers. The more complex the data, 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 document, which signals relationships with other documents. "Push-Assets" reduces waste and enables better user experiences regardless of the Content-Type.

4. Push-Assets Header

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

A REQUIRED Request Header.

Comprised of zero or more "Assets" addressed by their "Asset Key".

An "Asset-Key" is the name of the asset uniquely identifiable by the document or matching documents.

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 Push all push-enabled dependencies, if "Push-Assets" is enabled. Servers are NOT required to push any or all dependencies, but MUST push all missing or outdated push-enabled assets.

5. Push-Asset-Key Header

Push-Asset-Key = [$][Asset-Key]

A REQUIRED "PUSH_PROMISE" Response Header.

Unlike the "Asset-Key" in a Request, the "Push-Asset-Key Header" corresponds to the Key of the "PUSH_PROMISE" Response.

5.1. Named Key

Push-Asset-Key = core-bundle.js

By naming an asset, you MAY share that asset across multiple documents, and MAY change the URI [RFC3986] as necessary.

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/document?some=thing" would by keyed as "/my/document".

6. Push-Asset-Match Header

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

An OPTIONAL "PUSH_PROMISE" Response Header.

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 = *
```

"*" by itself corresponds to "match all requests". This is the equivalent of "/*".

7. References

7.1. 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, <http://www.rfc-editor.org/info/rfc2119>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <http://www.rfc-editor.org/info/rfc7540>.

7.2. 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, <http://www.rfc-editor.org/info/rfc2045>.

[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, <http://www.rfc-editor.org/info/rfc3986>.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <http://www.rfc-editor.org/info/rfc6265>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

[RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <http://www.rfc-editor.org/info/rfc7234>.

Author’s Address

Aaron Silvas
GoDaddy

Email: asilvas@godaddy.com