Network Working Group
Internet-Draft
Intended status: Informational
Expires: February 11, 2011

M. Nottingham
August 10, 2010

Making HTTP Pipelining Usable on the Open Web
draft-nottingham-http-pipeline-00

Abstract

Pipelining was added to HTTP/1.1 as a means of improving the performance of persistent connections in common cases. While it is deployed in some limited circumstances, it is not widely used by clients on the open Internet. This memo suggests some measures designed to make it more possible for clients to reliably and safely use HTTP pipelining in these situations.

This memo should be discussed on the ietf-http-wg@w3.org mailing list, although it is not a work item of the HTTPbis WG.

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 February 11, 2011.

Copyright Notice

Table of Contents

1. Introduction

HTTP/1.1 [RFC2616] added pipelining -- that is, the ability to have more than one outstanding request on a connection at a particular time -- to improve performance when many requests need to be made (e.g., when an HTML page references several images).

Although not usable in all circumstances (e.g., POST, PUT and other non-idempotent requests cannot be pipelined), for the common case of Web browsing, pipelining seems at first like a broadly useful improvement -- especially since the number of TCP connections browsers and servers can use for a given interaction is limited, and especially where there is noticeable latency present.

Indeed, in constrained applications of HTTP such as Subversion, pipelining has been shown to improve end-user perceived latency considerably.

However, pipelining is not broadly used on the Web today; while most (but not all) servers and intermediaries support pipelining (to varying degrees), only one major Web browser uses it in its default configuration, and that implementation is reported to use a number of proprietary heuristics to determine when it is safe to pipeline.

This memo characterises issues currently encountered in the use of HTTP pipelining, and suggests the use of mechanisms that, when used in concert, are designed to make its use more reliable and safe for browsers. It does not propose large protocol changes (e.g., out-of-order messages), but rather incremental improvements that can be deployed within the confines of existing infrastructure.

2. Requirements

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 [RFC2119].

3. HTTP Pipelining Issues

Anecdotal evidence suggests there are a number of reasons why clients don't use HTTP pipelining by default. Briefly, they are:

1. Server implementations may stall pipelined requests, or close their connection. This is one of the most commonly cited problems.

2. Server implementations may pipeline responses in the wrong order. Some implementations mix up the order of pipelined responses; e.g., when they hit an error state but don't "fill" the response pipeline with a corresponding representation.
3. A few server implementations may corrupt pipelined responses. It's been said that a very small number of implementations actually interleave pipelined responses so that part of response A appears in response B, which is both a security and interoperability problem.
4. Clients don't have enough information about what is useful to pipeline. A given response may take an inordinate amount of time to generate, and/or be large enough to block subsequent responses. Clients who pipeline may face worse performance if they stack requests behind such an expensive request.

Note that here, "servers" can also include proxies and other intermediaries.

The remainder of this memo proposes mechanisms that, together, can be used to mitigate these issues.

4. Discovering Faulty Proxies

Issues specific to proxies are limited to the network infrastructure currently used by the client, and it is reasonable to assume that testing the infrastructure at the beginning of a session will indicate how safe it is to pipeline while that infrastructure is in use.

Such issues can be detected by sending pipelined requests to a known server, and examining the responses for errors.

For example, if the ExampleBrowser implementation wishes to probe for faulty proxies, it can send a series of requests to "http://browser.example.com/pipeline-test/" and subresources. If the bodies of the resulting responses deviate from those it expects in any way, it is reasonable to assume that a faulty proxy is present, and pipelining SHOULD NOT be used through it.

Typically, user agents will do this upon startup and changes in the network, although they might periodically test to assure that a new proxy hasn't been interposed.

Note that proxies aren't always configured explicitly.

5. Identifying Responses

HTTP relies on the context of the connection to associate a given request with its intended response. In HTTP/1.0, this was a reasonable assumption, since only one request could be outstanding at a given time, and each request had exactly one response.

HTTP/1.1 made associating requests and responses in a given connection more complex (and therefore fault-prone). Not only does pipelining mean that multiple requests can be outstanding, but also the 1xx series of response status codes introduce the possibility of multiple response messages (syntactically) being associated with a single request.

To improve the client's ability to correlate responses with their requests and identify responses that are out of order (as well as serve other potential use cases), this memo introduces the "Assoc-Req" response header field.

```
Assoc-Req   = "Assoc-Req" ":" OWS Assoc-Req-v
Assoc-Req-v = Method SP absolute-URI
```

The field-value of the Assoc-Req header field is the method and effective request URI of the request associated with the response that it appears in. The URI used MUST be generated using the algorithm for finding the Effective Request URI in [I-D.ietf-httpbis-p1-messaging]. The header field MUST NOT be generated by proxies.

For example, given the following request over port 80:

```
GET /foo?it HTTP/1.1
Host: www.example.com
```

the appropriate Assoc-Req header field would be:

```
Assoc-Req: GET http://www.example.com/foo?it
```

Note that the Assoc-Req header field is not a perfectly reliable identifier for the request associated with a response; for example, it does not incorporate the selecting headers for content negotiation [I-D.ietf-httpbis-p6-cache], nor does it differentiate request bodies, when present. However, for the purposes of making pipelining more reliable, it is sufficient.

Clients who wish to use the Assoc-Req response header field to aid in identifying problems in pipelining can compare its values to those of the request that they believe it to be associated with (based upon

HTTP's message parsing rules, defined in [I-D.ietf-httpbis-p1-messaging]). If either the method or the URI differ, it indicates that there may be a pipelining-related issue.

6. Signing Content for Integrity

Another means of protecting against server issues (whether proxy or origin server) is to sign the response content for integrity, so that any corruption becomes apparent.

One existing way to do this is to use the Content-MD5 header field [I-D.ietf-httpbis-p3-payload].

Clients who wish to use the Content-MD5 response header field to aid

in identifying corrupted content due to pipelining issues can compare the hash to a calculated hash of the content.

In some circumstances, it may be impractical for the server to buffer the response content in order to calculate a hash before sending it. In these cases, the Content-MD5 response header can be send in an HTTP trailer, provided that the connection is HTTP/1.1 from end to end, and the client is able to process trailers.

Additional means of verifying HTTP response integrity may become available in time.

7. Hinting Pipelinable Content

Finally, to assist clients in determining what requests are suitable for pipelining, we define extensions to allow hinting by origin servers.

Each of these hints indicates URLs that, when dereferenced, will likely not incur significant latency on the server in generating the response, nor significant latency on the network in transferring the response.

What is "significant" is determined by the server. Clients will use these hints to determine what request(s) it is safe to pipeline something else after.

For example, if "http://example.com/a" is hinted, a client can be more confident pipelining another request (e.g., to "http://example.com/b") on the same request afterwards.

To allow flexibility and ease of administration, different kinds of

hints are defined:

- The "quick" link relation type [TBD: ref] can appear on individual HTML elements such as "img", "script" and "link" to indicate that the link they contain has low overhead. Additionally, it can be used in the HTTP link header to indicate links within the response in a format-neutral way.
- A document can indicate that all links from the indicated elements have low overhead by using the HTML META "quick" element, with the

content indicating the element names that are "quick". For example, "<META name='quick' content='img script link'/>".
- [DISCUSS: is it worthwhile to define a /.well-known lookup mechanism for quick links?]

8. Origin Server Considerations for Pipelining

To maximise the potential for request pipelining from clients that support this specification, origin servers:

- SHOULD send the Assoc-Req response header field in all potentially pipelinable responses (keeping in mind that downstream caches might be serving responses in the future).
- SHOULD send the Content-MD5 response header (or trailer) field in potentially pipelinable responses.
- SHOULD hint potentially pipelinable requests as outlined above.

9. User Agent Considerations for Pipelining

To take advantage of the server-side mechanisms defined in this specification, user agents:

- SHOULD ascertain whether any proxies present (either configured or interposed by interception) support pipelining by following the protocol described above. If they do not, pipelining SHOULD NOT be used.
- SHOULD check the Assoc-Req response header field-value, when present, on all pipelined responses.
- SHOULD check the Content-MD5 response header (or trailer) field-value, when present, on all pipelined responses.
- MAY use content hints for pipelining to assist in determining whether to pipeline a given request.

Upon encountering an indication of pipelining problems with a particular response (e.g., an incorrect Assoc-Req field-value, an incorrect Content-MD5 field-value), user agents SHOULD discard the response in question, all subsequent responses on the same connection, close the connection. Unsatisfied requests can be resubmitted, without pipelining, and the implementation can choose not to use pipelining to the same server in the future.

10. Security Considerations

TBD

11. IANA Considerations

TBD

12. References

12.1. Normative References

[I-D.ietf-httpbis-p1-messaging]
Fielding, R., Gettys, J., Mogul, J., Nielsen, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., and J. Reschke, "HTTP/1.1, part 1: URIs, Connections, and Message Parsing", draft-ietf-httpbis-p1-messaging-11 (work in progress), August 2010.

[I-D.ietf-httpbis-p3-payload]
Fielding, R., Gettys, J., Mogul, J., Nielsen, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., and J. Reschke, "HTTP/1.1, part 3: Message Payload and Content Negotiation", draft-ietf-httpbis-p3-payload-11 (work in progress), August 2010.

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

12.2. Informative References

[I-D.ietf-httpbis-p6-cache]
Fielding, R., Gettys, J., Mogul, J., Nielsen, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Nottingham, M., and J. Reschke, "HTTP/1.1, part 6: Caching", draft-ietf-httpbis-p6-cache-11 (work in progress), August 2010.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

Appendix A. Acknowledgements

Thanks to Julian Reschke for help in defining the Assoc-Req field-

value. The author takes all responsibility for errors and omissions.

Appendix B. Frequently Asked Questions

Isn't full multiplexing better?

While "full" multiplexing is theoretically better, pipelining -- once usable -- is adequate for almost all common Web browsing cases. Since the browser needs to download HTML first, it has an opportunity to receive hints about subsequent requests and pipeline them appropriately. Likewise, by far the most common case for multiplexing on the Web is when a large number of images and other page assets need to be fetched with GET; a perfect use of pipelining, provided that the client has enough information to avoid head-of-line blocking.

Why not have the client generate a unique request identifier?

While in some ways this would be easier than the approach that the Assoc-Req header takes, it would be more difficult to deploy, because existing caching proxies wouldn't be able to serve the correct identifier when using a cached response.

Author's Address

Mark Nottingham

Email: mnot@mnot.net
URI: http://www.mnot.net/