Network Working Group	M. Nottingham
Internet-Draft	March 14, 2011
Intended status: Informational	
Expires: September 15, 2011	

Making HTTP Pipelining Usable on the Open Web
draft-nottingham-http-pipeline-01

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.

Status of this Memo

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 (POST 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 mechanisms that are designed to make its use more reliable and safe for browsers.

Note that this memo does not suggest drastic changes to HTTP, nor does it require that intermediaries change to better support pipelining. Instead, it takes the position that removing the responsibility for making pipelining decisions from browsers, as well as reduce associated risks for browsers, we make it more likely that browsers will support it.

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. Reviewers are encouraged to pay particular attention to items marked FEEDBACK.

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. Balking Servers - server implementations can stall pipelined requests, or close their connection when the client attempts to pipeline. This is one of the most commonly cited problems.

2. Confused Servers - a few server implementations can respond to pipelined requests in the wrong order. Even fewer might corrupt the actual responses. Because this has security implications (consider multiple users behind such a proxy, or multiple tabs in a browser), this is very concerning.

3. Head-of-Line Blocking - 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, including so-called "transparent" proxies (also known as intercepting proxies).
The remainder of this memo proposes mechanisms that can be used to mitigate one or more of these problems; not all of them will survive discussion and implementation.

4. Blacklisting Origin Servers

To address balking and confused origin servers, a client SHOULD maintain a blacklist of origins that it does not attempt pipelining with.
Such a blacklist MAY be populated by external information (e.g., a list of known-bad servers maintained by the browser vendor), and when pipelining has been detected to fail to the origin.

5. Discovering Faulty Proxies

When a balking or confused server is a proxy, pipelining won't work for any requests sent through it. Therefore, clients SHOULD test the network for such proxies periodically.
This can be done 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.
RECOMMENDED measures in such tests include:

*Sending a non-trivial number of pipelined requests (e.g., 10)

*Sending multiple pipelined requests in the same packet

*Inserting request bodies with various sizes

*Assuring that caching is disabled, so that requests are end-to-end

*Sending a variety of responses types that includes 100 and 304 responses

*Examining responses to assure that they all appear in the correct order

*Examining received requests and responses to assure that they aren't unduly modified

These tests SHOULD be performed by clients (both user agent and proxy) upon startup, as well as periodically afterwards to assure that a new intercepting proxy hasn't been interposed. They MAY be performed after a pipelining problem is detected, to determine whether the issue is proxy- related.
See https://github.com/mnot/pipeline-surveyor for an example implementation.

6. Correlating 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 confused origin and proxy servers (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.

A client wishing 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 it believes 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, and the origin server (identified by its (host, port) tuple) SHOULD be blacklisted.

A client MAY choose to blacklist any origin server that does not send the Assoc-Req header.

FEEDBACK: Omitting the URI scheme and authority (i.e., just making it the path and query components) would make the header easier to generate and avoid some false positives (e.g., when a "reverse proxy" or other URI rewriter is present), but may fail to identify cases where two requests are confused (consider requests for "http://example.com/style.css" and "https://foo.example.net/style.css").

7. Hinting Pipelinable Content

It's necessary to assist clients in determining what requests are suitable for pipelining, so that the sole responsibility for deciding what and when to pipeline isn't theirs. This can be done through origin server hinting.

Such 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 connection afterwards.

There are several possible ways that content could be hinted, including:

* The "quick" link relation type can appear on individual HTML elements such as "img", "script" and "link" to indicate that the link they contain has low overhead.

* A similar link relation could also be used in the HTTP link header to indicate "quick" links within the response in a format-neutral way.

* A server can indicate that all its resources are suitable for pipelining by returning a successful response status code (2xx) to requests for the path "/.well-known/pipeline". In the future, a format available at this location could give more fine-grained information.

FEEDBACK: thoughts on the suitability of these hinting mechanisms is encouraged, so that the list can eventually be narrowed down.

A user agent MAY have a policy of only pipelining to hinted resources.

8. Indicating Blocking Responses

An alternate way to avoid head-of-line blocking is for the origin server to aggressively indicate when a request would block.
This can be done by using a new HTTP status code, 430 WOULD BLOCK.
The meaning of "would block" is defined by the server; e.g., it may return this code when the response is known to be over a certain size, or when the code to generate the response is known to take a long time to execute.
When a client (user agent or intermediary) receives a 430 WOULD BLOCK, it SHOULD resubmit the associated request on a new or idle connection.
An origin server MUST NOT send a 430 WOULD BLOCK status code to clients that do not include a "PWB: 1" (mnemonic: Pipelining Would Block) request header. User Agents that support the status code SHOULD send this header, and intermediaries that are willing to handle its processing MAY append it to requests that do not already include it.
A cache MUST NOT store a 430 WOULD BLOCK response, and origin servers SHOULD mark them as explicitly uncacheable (e.g., with Cache-Control: no-store).
FEEDBACK: This is a relatively new idea; thoughts? In some ways it's easier to deploy, but it does add a certain amount of latency to requests that block. Theoretically, a Location header could be added to redirect the client to a place where the generated response will be waiting (if the blocking is caused by server-side think time), but this may be impractical to implement.

9. Handling Pipelining Problems

Upon encountering an indication of pipelining problems with a particular response (e.g., an incorrect Assoc-Req field-value, a pipelined response that stalls), user agents SHOULD discard the response in question, all subsequent responses on the same connection, and 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 (see "Blacklisting Origin Servers").

10. Security Considerations

TBD

11. IANA Considerations

TBD

12. References

12.1. Normative References

[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
[I-D.ietf-httpbis-p1-messaging]	Fielding, R, Gettys, J, Mogul, J, Nielsen, H, Masinter, L, Leach, P, Berners-Lee, T and J Reschke, "HTTP/1.1, part 1: URIs, Connections, and

	Message Parsing", Internet-Draft draft-ietf-httpbis-p1-messaging-12, October 2010.

12.2. Informative References

[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.
[I-D.ietf-httpbis-p6-cache]	Fielding, R, Gettys, J, Mogul, J, Nielsen, H, Masinter, L, Leach, P, Berners-Lee, T and J Reschke, "HTTP/1.1, part 6: Caching", Internet-Draft draft-ietf-httpbis-p6-cache-12, October 2010.

Appendix A. Acknowledgements

Thanks to Ilya Grigorik, Anirban Kundu, Patrick McManus and Julian Reschke. 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.

Appendix C. Changes

draft -00 to draft -01:

- *Add guidelines for blacklisting
- *Remove advice on signature checking (for now)
- *Clarified problem statement
- *Rearranged
- *Added 430 WOULD BLOCK

Author's Address

Mark Nottingham Nottingham EMail: mnot@mnot.net URI: http://www.mnot.net/