

Optimizing Latency of the HTTP Protocol

Franz J. Hauck

October 1994

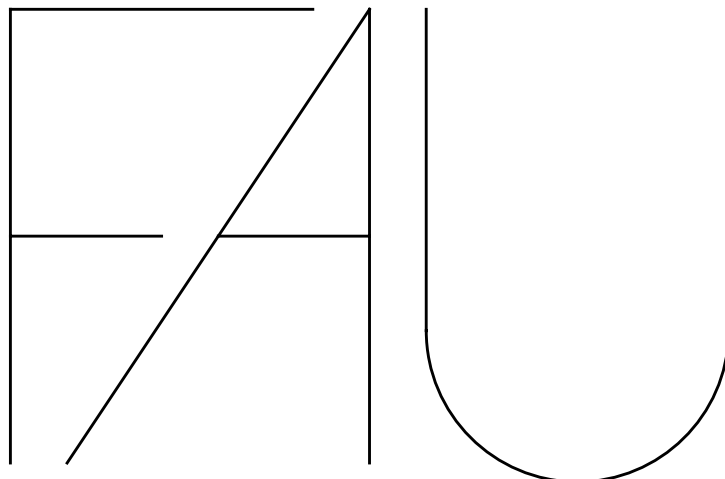
TR-14-94-23

Technical Report

Computer
Science Department

Operating Systems — IMMD IV

Friedrich-Alexander-University
Erlangen-Nürnberg, Germany



This paper is also published as:

submitted for publication

Optimizing Latency of the HTTP Protocol

Franz J. Hauck

hauck@informatik.uni-erlangen.de
University of Erlangen-Nürnberg, IMMD 4
Martensstraße 1
D-91058 Erlangen, Germany

Abstract. The World-Wide Web (WWW) hypertext system uses TCP/IP connections for network-transparent access to hypertext documents. Documents often contain several inlined images which have to be transferred as individual documents from the server to the client. For each transfer a separate TCP/IP connection has to be established. A significant amount of the total response time is caused by the latency due to the opening procedure of a TCP/IP connection. This paper presents a small extension of the HTTP protocol used by WWW servers and clients. The extension allows the reuse of established connections and, thus, reduces latency. At the same time, the extension is transparent to servers and clients using the standard HTTP protocol.

1 Introduction

The World-Wide Web (called WWW) is a distributed hypertext system [1] which is increasingly being used throughout the world. Users can browse hypertext documents provided by a WWW server. The browser or WWW client manages the interconnections to servers needed to present a hypertext document. The communication is ruled by the Hypertext Transport Protocol HTTP [2] which in turn uses TCP/IP connections [3] opened by the client and accepted by the server.

One of the drawbacks of the WWW system is the bad response time of browsers when a user wants to view a new document: for each document a separate TCP/IP connection to a server is established, a request is sent, and a reply is received. Finally, the connection is closed. The response time depends on the bandwidth of the network connection from client to server site. To increase bandwidth and, thus, to shorten response time, cache servers have been introduced which cache hypertext documents on locally available servers [4,5]. The

connections from clients to cache servers have usually a quite high bandwidth.

Another factor contributing to the total response time is that most of the documents contain several inlined images which need to be loaded by the browser to present the document to the user. These images are to be loaded through individual requests to the server. Each request requires its own TCP/IP or HTTP connection respectively. The latency of establishing a TCP/IP connection is not to be neglected, even when using cache servers.

We present a solution to this problem introducing reusable TCP/IP connections. Chapter 2 presents theoretical aspects of the proposed HTTP extension. In chapter 3 a prototype implementation is discussed. We added the new feature to our WWW server and to a *Mosaic* client. Chapter 4 introduces a server management algorithm for time-out handling. Chapter 5 offers the patches to add the new feature to *Mosaic* 2.4. Finally, chapter 6 concludes with a summary.

2 HTTP Hold-Line Extension

To overcome the problem of high latency caused by hypertext documents with several inlined images we propose a HTTP extension which allows to hold the established TCP/IP connection after the server replied to the initial request of the client. If the client wants to send another request to the same server it proceeds as if the connection has been established just for this request. The server replies as usual.

2.1 The Hold-Line Header

The main obstacle of this scheme is that the server usually forks a process for each request for processing it. This forked process needs to hold the line to the client. It is not appropriate to keep forked processes alive if the client never wants to reuse the connection. Thus, the HTTP extension has to provide a way for clients to signal to servers that they are capable of holding the line and that they are going to do so in the current situation. Therefore, a new request header is introduced. Its name is `Hold-Line`. After the header name a value may be added. The value should be an integral number indicating the time-out when the client will drop the connection on not receiving any further requests from the user, e.g.:

`Hold-Line: 5`

In this example, the header signals to the server that the client will drop the line after five seconds if there is no further request from the user. If the server does not know about the hold-line feature it will simply ignore the new header. If the client does not provide the number of seconds it only signals to the server that it wants to hold the line.

If the server gets a request without a `Hold-Line` header it will proceed as usual. If a client sends a request with a `Hold-Line` header the server decides whether it wants to hold the line after the reply or not. This decision is signalled to the client by a `Hold-Line` header in the reply message. The value part is the number of seconds the server wants to hold the line. This might be the number recommended by the client, but the server may change the number of seconds to its needs. If the server does not provide a number of seconds in the header line it signals to the client that it will hold the line for an arbitrary amount of time.

If a client gets no `Hold-Line` header in the reply message it knows that the server is not capable of the feature or does not want to hold the connection. If a client gets the header it can hold the line up to the indicated number of seconds.

A prerequisite is that the client can detect the end of the reply message. In the HTTP Version 0.9 the end of the message is simply signalled by dropping the connec-

tion. HTTP Version 1.0 provides a `Content-Length` header line which indicates the exact length of the reply message [2]. If the server cannot guarantee that the client can detect the end of the message it must not return the `Hold-Line` header to the client.

2.2 Compatibility

The extension of the HTTP protocol is compatible to the standard HTTP protocol, because clients' request messages which do not include the new header line are processed as usual. Servers which do not support the extension simply ignore the new header and will not return the new header to the client.

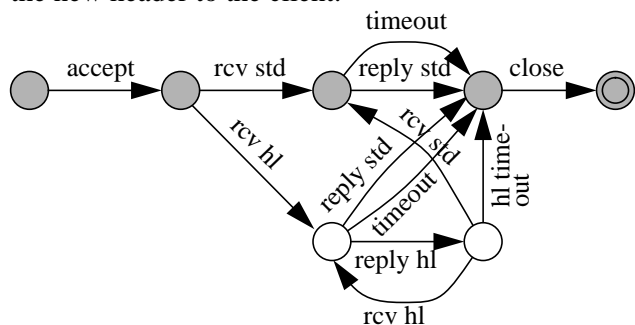


Fig. 2.1 Server's state diagram

Fig. 2.1 shows the server's state diagram. Grey shaded states are the usual states of a standard HTTP server. Only two further states are necessary to implement the server behavior. The state transitions mean:

- accept** The server accepts the client's open of a new connection.
- rcv std** The server receives a standard HTTP request.
- reply std** The server replies a standard HTTP message.
- rcv hl** The server receives a HTTP request with `Hold-Line` header.
- reply hl** The server replies a HTTP message with `Hold-Line` header.
- timeout** The server gets a time-out.
- hl timeout** The hold-line time-out exceeded.

The client's state is a little bit more complex. Fig. 2.2 shows the client's state diagram. Again, shaded states

are the states of a standard HTTP client. The transitions mean:

- open** The client opens a connection to a server.
- send std** The client sends a standard HTTP request.
- get std** The client gets a standard HTTP reply message.
- send hl** The client sends a HTTP request with a Hold-Line header.
- get hl** The client gets a HTTP reply message including a Hold-Line header.
- error** The client detects an error.
- hltimeout** The hold line time-out exceeds

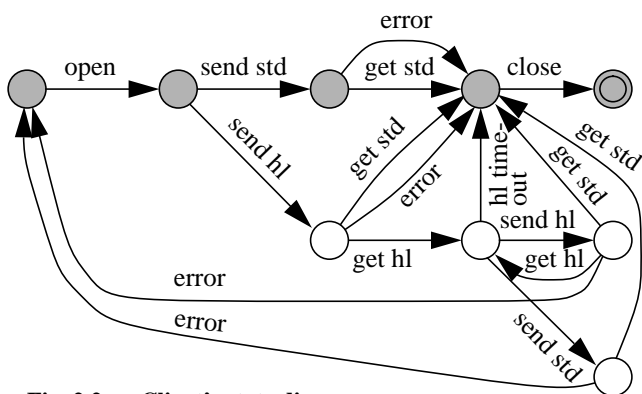


Fig. 2.2 Client's state diagram

When client and server hold the line and the client decides to send another request this request may interfere with the server's wish to drop the line. Thus, the server will not get the new request and the client will detect an error. This error should be handled by a silent retry opening a new connection to the same server.

3 Implementation

The proposed HTTP extension was implemented and tested. The server side was easy to implement. We used a prototypical version of the *Perplex* WWW server [6]. This server software is under development. It is derived from the *Plexus* [7] server software and as this written in the *Perl* programming language [8,9]. Indeed, the extension needed only a few lines of code. The minimal functionality of the HTTP protocol extension was implemented. The server returns a Hold-Line header if the corresponding request had one. Eventually, the server holds the connection up to a defined time-out period.

The client side was more complex to implement. The `libwww` library of the *Mosaic* 2.4 software [10] for UNIX is not prepared for such an extension. We implemented a very simple subset of the extension. This subset does not consider time-out values, but holds the connection until it is recognized as closed by the server. Our implementation may not be correct in several cases of errors but demonstrates that the feature is implementable and provides the desired benefits. A final implementation of the client side of the HTTP extension may hold a few connections to servers at once and may manage time-outs.

We tested a patched *Mosaic* 2.4 client together with our *Perplex* 0.91 server on a SparcStation 10/40. The loading time of two documents of our WWW tree have been compared running server and client with and without the hold-line feature. Document 1 is the home page of the computer science department in Erlangen. This document is about 2k large and contains 4 small and 2 medium size inlined images totalling a size of 18k. Document 2 presents the Erlangen Icon Collection. It is about 3.7k large and contains 35 small inlined images with a total size of about 11k. The URLs of the documents are:

Document 1:

<http://www4.informatik.uni-erlangen.de/>

Document 2:

<http://www4.informatik.uni-erlangen.de/images/>

The measured response times are presented in the following table:

	standard	hold-line 5 s
Document 1	5.6 s	3.5 s
Document 2	28.1 s	14.6 s

All values are measured at the client side without the communication with the X11 server which usually presents a hypertext document to the user. We used the `ftime(3)` system call for measurement placed at the first `open(2)` or `socket(2)` and the last `close(2)` system call.

There is a significant optimization of the response time with the new feature. We believe that further optimization in the client software could be done to improve the new feature. In the current implementation of *Mosaic* 2.4, each character is copied about four to five times before it can be used for display. This does not matter when the software spends a lot of time in the initialization of the HTTP connection. With the new feature, the initialization is very short and data processing needs a significant portion of the overall response time and is, thus, to be further optimized. The server software is also a matter of further optimizations.

4 Server Management

If there are a lot of clients using the HTTP extension the server may be heavily overloaded with spawned server processes holding the TCP connections to their clients. We designed a server management component for our prototypical server which adapts the maximum of the holding time for new requests according to the servers load.

The server daemon counts the number of forked processes. If the number exceeds a certain threshold the hold-line time-out for new requests is decreased. If the number falls below a certain minimum the time-out is increased again. When there are too many processes no hold-line feature is granted to clients at all. Therefore, no `Hold-Line` header is returned. This allows users to hold the line to the WWW server for a long period of time (e.g. 20s) if the server is idle most of the time. Such a long holding time will increase efficiency not even for inlined images but also for the hypertext document which a user wants to access next. With our test environment, we noticed that even a time-out of 5s is sufficient to load the next document through the held connection because often the path to a target document is well known, but needs navigation through several other documents.

5 Availability

The *Perplex* server software is not yet available, but we have an early version of the *Perplex* server running which also provides the proposed HTTP extension. For tests use the URL:

<http://camelot.informatik.uni-erlangen.de/>

The client software is available as patches for *Mosaic* version 2.4 as WWW document:

<http://www4.informatik.uni-erlangen.de/IMMD-IV/Persons/fzhauck/holdline.patch>

6 Conclusion

We presented a small extension to the HTTP protocol which allows the reuse of established HTTP connections to servers. This reuse reduces latency caused by multiple connections to a server while loading inlined images of a document. Combined with a cache server, the new feature provides a significant reduction of the total response time.

The extension is compatible to the existing HTTP standard, i.e. old and new client or server software can interact with each other.

7 References

- [1] *The World-Wide Web Initiative: The Project* <URL: <http://info.cern.ch/hypertext/WWW/TheProject.html>>
- [2] *HyperText Transfer Protocol* <URL: <http://info.cern.ch/hypertext/WWW/Protocols/HTTP/HTTP2.html>>
- [3] J. Postel; *Transmission Control Protocol*; RFC 793, Sept. 1981 <URL: <ftp://src.doc.ic.ac.uk/pub/rfc/rfc793.txt.gz>>
- [4] Ari Luotonen, Kevin Altis; *World-Wide Web Proxies*, April 1994 <URL: <http://info.cern.ch/hypertext/WWW/Proxies/>>
- [5] Steven Glassman; *A caching relay for the World-Wide Web*; 1994 <URL: http://www.research.digital.com/SRC/people/Glassman_Steve/CachingTheWeb.html>
- [6] *The Perplex HTTP Server* <URL: <http://camelot.informatik.uni-erlangen.de/admin/doc/>>
- [7] *Plexus Server* <URL: <http://www.bsdi.com/server/doc/plexus.html>>
- [8] *The Perl programming language* <URL: <http://www.metronet.com/1h/perlinfo>>
- [9] Larry Wall, Randal Schwartz; *Programming PERL*; O'Reilly; 1992
- [10] *Mosaic 2.4* <URL: <http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/help-on-version-2.4.html>>