Analysis of the effects of removing redundant header information in persistent HTTP connections

Timothy Bower, Daniel Andresen*, David Bacon
Department of Computing and Information Sciences
234 Nichols Hall
Kansas State University
Manhattan, KS 66506
voice: 785-532-6350
fax: 785-532-7353
{*tim, dan, dbacon*}*@cis.ksu.edu*

Abstract

This paper examines the potential benefits of an extension to the HTTP protocol that allows the web server and client to establish a session of HTTP messages as opposed to a sequence of independent connections. We report results from experiments which measure the potential bandwidth saving achieved by removing redundant headers when possible. We report findings from an actual implementation of the protocol extension, from which we derive a model for calculating the potential bandwidth savings based on statistics of the HTTP traffic. Based on our implementation, we propose an architecture which can be used to deploy the protocol extension on point–to–point network connections without interfering with the operation of either the web client (browser) or server. Next, we examine the access logs from our departmental web server and calculate the expected savings based on statistics from observed HTTP traffic. The impact of removing redundant header information in HTTP replies is fairly minimal (1–5% savings) because the size of the content is much larger than the size of the header. However, we found that, on average, HTTP requests from clients to servers are reduced to 55 to 60 percent of their normal size.

Keywords*: http, WWW, performance, bandwidth*

Presenting author: *Daniel Andresen*

Conference: *IC'01*

1 Introduction

For reasons of robustness and simplicity, the HTTP protocol was designed as a stateless protocol. That is to say, with HTTP/1.0 [1] each client request contains all the information the server will need to reply as if the server had never communicated with this client before. This simple design fits well with the distributed nature of the World Wide Web and has indeed proven to be a robust design. However, given the nature of TCP/IP and observed statistics of HTTP requests, this design is suboptimal in terms of performance [4, 3].

The problems with HTTP/1.0 relative to TCP/IP were quickly observed by Simon Spero [4]. With HTTP/1.0, each request is a separate network connection, which is inefficient. The HTTP/1.1 version of the protocol addressed these shortcomings by introducing persistent (keep-alive) connections [2]. Given the advent of persistent connections, it is natural to consider the possible benefits of reducing some of the redundancy in the HTTP headers. A framework for establishing sessions, and hence states, to reduce header redundancy was proposed in the so-called sticky header proposal [3] to W3C, which was deferred from HTTP/1.1.

The potential benefits from reducing the redundancy in HTTP headers seems to be greater today than in the past. Many web servers are receiving an increasing number of requests from web-caching proxies and search engines. Caching proxies and search engines tend to make a large number of requests in a short period of time, and a high percentage of their requests are HEAD or If-Modified-Since GET requests for which the reply often consists of only a header. Traffic of this nature will especially

benefit from a reduction in redundant HTTP headers.

In this paper, we report on the results of experiments aimed at establishing the feasibility and potential bandwidth saving of a HTTP extension which establishes the notion of a session between client and web server and removes redundant headers when possible. We report findings from actually implementing the protocol from which we derive a model for calculating the potential bandwidth savings based on statistics of the HTTP traffic. Next, we examine over 670,000 HTTP requests and replies from the access logs from our departmental web server and calculate statistical metrics to estimate the savings based on observed HTTP traffic.

The paper is organized as follows: In Section 2, we present the background on the HTTP protocol and typical web server workloads in today's e-commerce environment. In Section 3, we discuss our system design for an experimental implementation of the extended HTTP protocol. Based on our implementation, we show a model for the expected gains from reducing redundancy in the HTTP headers. Section 4 presents the experimental results, and section 5 discusses conclusions.

2 Background and motivation

Web usage usually consists of multiple requests to the same provider in a short amount of time. The most common resource requested via HTTP is an HTML page, which itself usually contains embedded objects (images, frames, style-sheets) that also need to be requested. These additional resources usually reside on the same provider, or alternately on a dedicated image server or some similar solution. Users selecting links off a web page to other pages/resources from the same server and search engines probing the network for new and changed documents further add to the frequency of traffic between the server and client in a short period of time.

When a client and server exchange multiple messages in a short period of time, then the set of messages exchanged can be called a *session*. For the client and server to establish a session, then each must maintain a dictionary (cache) of hosts with whom it has an established session. Of course, the host dictionary would have a timeout value associated with the idle time until host entries expire and are removed from the dictionary. A longer timeout increases the probability of future requests being a part of an established session. This is reflected in Figure 1, which shows statistics from our departmental web server. Figure 1 shows the percentage of requests which arrived within a specified timeout of a previous request from the same client.[1] That is to say, if the server had kept a host dictionary of clients with which it had an established session, then the percentage of requests that are part of an established session are shown in Figure 1 over a range of timeout values.

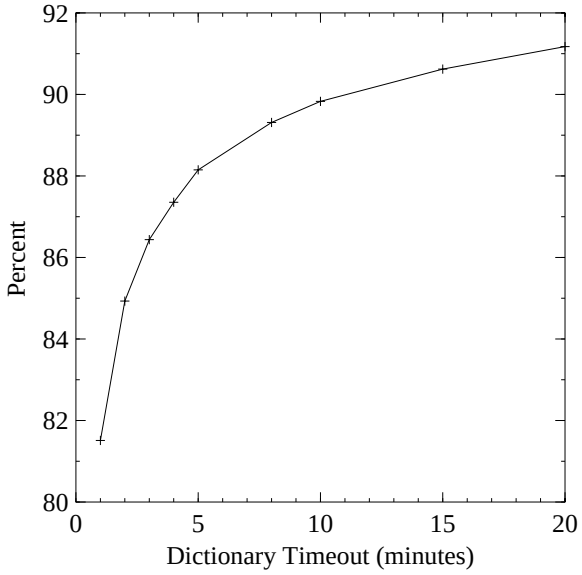

Figure 1. Percentage of HTTP requests in host dictionary

Each HTTP request contains a list of header tags denoting the client's host information and capabilities. Each response contains information about the server, the object attached, or error information. Figure 2 shows a typical example of a HTTP request and reply. Table 2 lists the header tags and the size of each tag which may removed from messages which are a part of an established session.[2]

```
GET / HTTP/1.1
Accept: image/gif, image/x-xbitmap,
image/jpeg, image/pjpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Host: www.cis.ksu.edu
Connection: Keep-Alive
```

Figure 2. Example HTTP request headers

[1] Statistics taken from 14 days of access logs, 670,113 HTTP requests.

[2] Header tag sizes are based on a Netscape 4.7 browser running on a Linux platform.

	Header Tag	Size (bytes)
Request Headers	Accept	70
	Charset	34
	Encoding	21
	Language	19
	User-Agent	61
	Total	205
Reply Headers	Server	40

Table 1. Redundant headers tags which may be removed in an established session

3 System architecture

We implemented a protocol similar to that proposed in the sticky headers proposal [3] to validate the feasibility of the protocol and to establish a framework for modeling the performance benefits. So as to not affect the web client or server, the extensions to HTTP were implemented between two proxy servers. This design provided a convenient means to test the implementation with real clients and a real server without the need to modify either the client or server. This also demonstrates an architecture allowing for a point-to-point use of the protocol extension on bandwidth-constrained routes regardless of if the clients and servers support the protocol extension. Figure 3 shows a block diagram of the implementation.

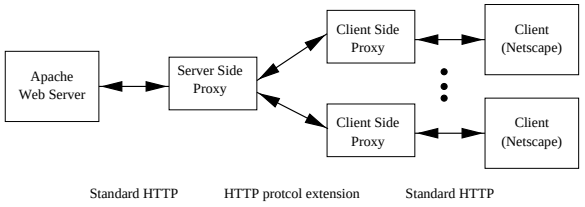

Figure 3. Block diagram of experimental environment

In our implementation, we built on Jigsaw, the W3's easily extendible Java web server. This web server is designed such that filters can be placed on any resource. To implement our host/header dictionary and filtering on the server side, we wrote a ServerStateFilter which was placed on the root resource of the server. Thus, every client was entered into the dictionary. For the client side, we configured the Jigsaw web server as a proxy, and placed a ClientStateFilter on the proxy resource. Each of these filters uses a HostDict to keep track of peers and their header values. The only difference is the Request and Response objects are opposite.

The HostDict is responsible for maintaining a dictionary of hosts and their current header values. Based on these values, the HostDict is also responsible for stripping the outgoing messages and reconstructing the incoming messages.

The dictionary categorizes headers into three tiers according to how their values might change: fixed, sometimes, always. "Fixed" headers are those related to the host (e.g., its host-name or IP), which will not change for a particular peer. "Sometimes" headers are those that change state but possibly maintain the same value for multiple consecutive messages. "Always" denotes headers whose values always change (e.g., Content-Lengths or hashes). These are usually headers related to the particular resource requested.

The "Fixed" headers are exchanged only once. The "Sometimes" headers are only sent when their value is changed. The "Always" headers are not touched and are sent with each request.

With respect to the three categories of headers, the nature of requests and responses is quite different. The Request contains much information about the host and its capabilities; hence, many of the headers can be filtered. However, the Response headers are almost all related to the resource it is returning, which fall into the always category. Savings on Response messages are significantly less than that of Request messages.

The simulation was performed using as close to real world conditions as possible. A state-enabled proxy was set up as the server. A second proxy was set up to act as a state-enabled browser using the first proxy as its proxy server. (See Figure 3.) Each proxy gathered the statistics of byte savings. For each HTTP request, a simple shell script was used to iterate through a list of URLs remote-controlling a Netscape Navigator process to load each page. The list of URLs was obtained from the Apache logs of our departmental web server. The simulation was run to verify the functionality of the extended protocol and to ensure that the observed results match with our model for the bandwidth savings.

Modeling the Bandwidth Savings

In our implementation, we add an additional header tag called "has state" to each header. This communicates whether the client and server have the other in their local host dictionary (cache) of known hosts. This message adds 15 bytes to each header. As was discussed in Section 2, if the client proxy knows that the client is in the server side proxy dictionary, it strips out 205 bytes

of redundant fixed headers. So the net savings for a message in an established session is at least 190 bytes. Note that we only account for the fixed headers and not the sometimes changing headers; thus, our calculation is a worst-case estimate.

For reply messages, the "has state" header is still added, but 40 bytes for the "server" header is removed from messages in an established session.

Expected Savings on HTTP requests:

Let S_r be the size of a normal HTTP request, while $S_{r'}$ represents the new HTTP request size.
Let N_s represent the number of a requests within a given time interval that are part of an established session.
Let N_t represent the number of all HTTP requests within a time interval.
Then the expected value of the ratio of the size of the new requests to the size of a normal request is given by:

$$E\left\{\frac{S_{r'}}{S_r}\right\} = \frac{E\{S_r\} - E\left\{\frac{N_s}{N_t}\right\}190 + \left(1 - E\left\{\frac{N_s}{N_t}\right\}\right)15}{E\{S_r\}} \quad (1)$$

Let S_h be the size of a normal HTTP reply header, while $S_{h'}$ is the size of the new HTTP reply header.
Let S_c be the size of the reply content.
Let N_s represent the number of replies within a given time interval that are part of an established session.
Let N_t represent the number of all HTTP replies within the time interval on the network link.
Then the expected value of the ratio of the size of the new replies to the size of a normal reply is given by:

$$E\left\{\frac{S_{h'} + S_c}{S_h + S_c}\right\} = \frac{E\{S_h\} + E\{S_c\} - E\left\{\frac{N_s}{N_t}\right\}27 + \left(1 - E\left\{\frac{N_s}{N_t}\right\}\right)15}{E\{S_h\} + E\{S_c\}} \quad (2)$$

4 Experimental results

The implementation of the protocol extension demonstrated the feasibility of eliminating redundant header tags. It also gave us an empirical basis for a model of the performance benefits based on HTTP statistics. Using the performance model, we analyzed access logs from our departmental web server to calculate the expected bandwidth savings. We calculated the statistical measures needed for equations 1 and 2 based on 14 days of access logs, containing 670,113 HTTP requests. Some statistics regarding the HTTP requests and the size of the replies are listed in Table 4.

We calculated the statistical metrics for dictionary timeout values of 1, 2, 3, 4, 5, 8, 10, 15 and 20 minutes. The

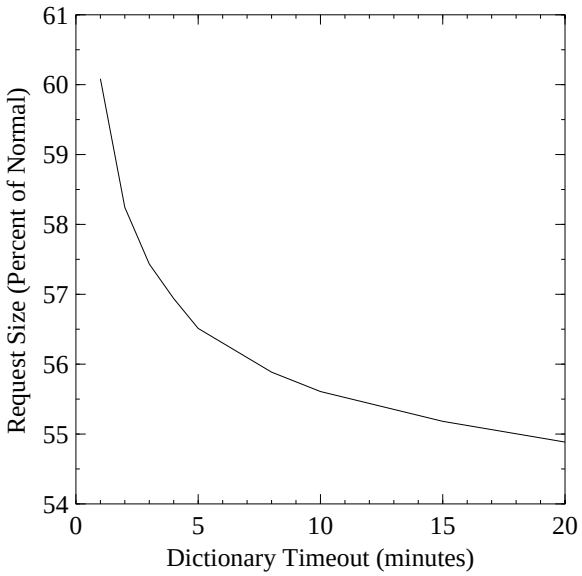

Figure 4. Average Request Header Size

Request Type	200		304	other	
	%	Size	%	%	Size
GET	66.83	14833	17.22	11.8	15674
GET (cgi)	1.74	4042	0	0.65	502
POST	0.62	3139	0	0.068	354
HEAD	0.88	0	0.002	0.128	0
other	0.015	0	0	0.05	151

Table 2. HTTP Statistics of Observed Server Logs

measured probability of a request being a part of an existing session ($E\{N_s/N_t\}$) is shown in Figure 1. The metrics which are constant for all timeout values are the normal header and content sizes (S_r = 381.04 bytes; S_h = 223.35 bytes; and S_c = 11,856 bytes). The savings on requests are significant — new request sizes are 55–60% of normal requests sizes, which is a savings of 40–60%. The expected value of the new request header size ($E\{S_{r'}/S_r\}$) is shown in Figure 4.

The overall savings on the replies is fairly insignificant; average reply is 99.8% of the normal size. The savings was the same for any timeout value. This is because the average content size, S_c, is so large that the header savings does not have much impact. However, some clients, such as proxy cache servers or search engines, often make HEAD requests or If-Modified-Since GET requests, which lowers the average content size. To examine traffic between a search engine and our web server, we looked at the requests from several search engines in isolation. Some search engines make extensive use of HEAD and If-Modified-Since requests, so the savings on the replies was more. The best case found was a 12% reduction (88% of normal) with a 20-minute dictionary timeout. Other search engines take a more brute force approach and the savings on reply messages is again minimal (1–5%).

One concern with such an implementation is the amount of memory which would be required to hold the host dictionary for a given timeout value. We found that the maximum number of active sessions increases linearly with the timeout value. Figure 5 shows the maximum number of active sessions for a given timeout value. Note from Figure 1 that with a timeout of 10 minutes, nearly 90% of the requests are part of a previously established session, requiring less than 150 sessions to be saved. Hence, memory constraints do not pose a significant obstacle to implementing the protocol extension.

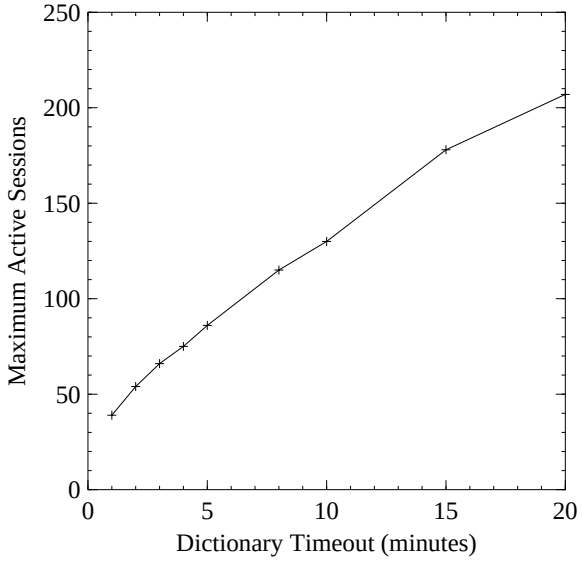

Figure 5. Maximum Number of Active Sessions

5 Conclusions

We establish that it is feasible to implement an extension to the HTTP protocol which allows web clients and servers to, without modification, establish sessions and remove some of the redundant header tags in HTTP messages. Based on our implementation of such a protocol on a point–to–point network link, we show a model for calculating the savings associated with such a protocol based on measurements of statistical metrics of the HTTP traffic. We then calculate the expected saving for our own departmental web server.

Because the content size is on average much larger than the headers, the overall savings for HTTP replies is minimal (1–5% in most cases). However, the bandwidth savings of HTTP requests is significant (40–45% savings — message lengths are 55–60% of normal size depending on the timeout value used). In the case of the server logs we examined, the potential saving on requests alone was 102–115 Mbytes for the 670,113 requests considered.

Other implementations of such an extension to the HTTP protocol may vary slightly in terms of the number of bytes removed from the headers, and other sites may have a different statistical mix of HTTP traffic, but it is hoped that the information presented here will give perspective to the discussion of the potential benefits of such an extension to the HTTP protocol.

References

[1] T. Berners-Lee, R. Fielding, and H. Frystyk. *Hypertext Transfer Protocol – HTTP/1.0*, May 1996. http://www.ics.uci.edu/pub/ietf/http/rfc1945.html.

[2] J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*, June 1999. http://www.w3.org/Protocols/rfc2616/rfc2616.html.

[3] A. Hopmann. *Persistent HTTP Connections*. ResNova software, Inc., Feb. 1996. http://oradb1.jinr.ru/Info/std/draft-ietf-http-ses-ext-01.txt.

[4] S. E. Spero. *Analysis of HTTP Performance Problems*. http://sunsite.unc.edu/mdma-release/http-prob.html.