Network and User-Perceived Performance of Web Page Retrievals

Hans Kruse
Ohio University
hkruse1@ohiou.edu

Mark Allman, Paul Mallasch
NASA Lewis Research Center
{mallman,Paul.G.Mallasch}@lerc.nasa.gov

Abstract

The development of the HTTP protocol has been driven by the need to improve the network performance of the protocol by allowing the efficient retrieval of multiple parts of a web page without the need for multiple simultaneous TCP connections between a client and a server. We suggest that the retrieval of multiple page elements sequentially over a single TCP connection may result in a degradation of the perceived performance experienced by the user.

We attempt to quantify this perceived degradation through the use of a model which combines a web retrieval simulation and an analytical model of TCP operation. Starting with the current HTTP/1.1 specification, we first suggest a client-side heuristic to improve the perceived transfer performance. We show that the perceived speed of the page retrieval can be increased without sacrificing data transfer efficiency. We then propose a new client/server extension to the HTTP/1.1 protocol to allow for the interleaving of page element retrievals. We finally address the issue of the display of advertisements on web pages, and in particular suggest a number of mechanisms which can make efficient use of IP multicast to send advertisements to a number of clients within the same network.

Problem Statement

Many electronic commerce applications are built upon the concept of web pages, We define a web page as a multi-part document consisting of an HTML file describing the general document layout, and a number of multimedia files referenced by the HTML base document. Functionally, the user's client first retrieves the base HTML document. The client then issues additional retrieval commands as it encounters embedded multimedia elements (images, JAVA applets, sound and movie files, etc.). Once a document has been presented to the user, the user selects a navigational option from the page which may transmit additional information (e.g., from a form), and which will usually result in a retrieval request for a new multi-part page.

It is important to note that in many cases, users can and do select a navigational option after some, but not all of the page elements have been loaded and displayed. Users perceive application performance based in part on the time needed to display enough of the page to allow the user to make this selection. For our discussion, it also is important to note that many providers of electronic commerce applications have found it desirable to display advertisements to users as part of the pages making up applications. The application provider has an interest in displaying advertisements before the user moves on to another page. However, users will have a negative reaction to a page if the loading of advertisements is perceived as slowing down the loading and display of information and navigational controls.

Today, many user clients are based on version 1.0 of the HTTP protocol. This version of HTTP permits only one document request to be pending on a TCP connection between a client and server. For performance reasons, most client implementers have chosen to open multiple TCP connections between the client and the server in order to request a number of page elements at the same time. As a side effect of this implementation, the client is able to quickly begin the actual display of multiple page elements for the user. Therefore, users now expect the appearance that goes along with using HTTP/1.0 with multiple simultaneous TCP connections.

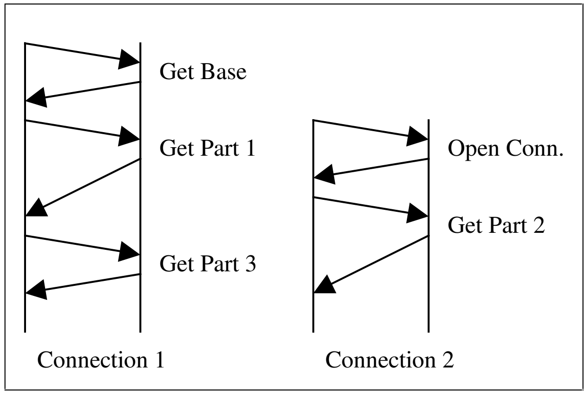

Figure 1

Figure 1 shows the information flow for the case of two TCP connections. Following the retrieval of the base HTML, the client requests one page element on the original TCP connection. The client then opens a second connection and issues another retrieval request. The third request will occur on the connection that returns to an idle state first.

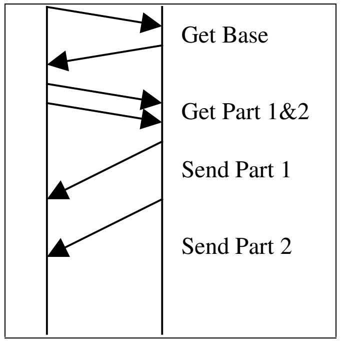

Figure 2

The transition to version 1.1 of HTTP [FJGFBL97] has begun. This version is designed to permit pipelining of requests for page elements, i.e., a request for an element may be issued on a TCP connection while the previous element is still being transmitted. This approach has been shown to exhibit better network utilization and better overall performance than the multi-connection approach in HTTP 1.0. [Hei97]. In addition, HTTP/1.1 is more network friendly, as using multiple simultaneous TCP connections has been shown to increase congestion levels on networks and may contribute to congestion collapse [FF98]. Figure 2 shows the information flow for a page retrieval with HTTP 1.1. Once the base document has been retrieved, a single transmission may request multiple page elements, which the server will return sequentially on the TCP connection.

In this paper we show that the pipelined HTTP/1.1 approach can lead to a decrease in user-perceived performance. This is caused by the fact that the client only receives layout information and actual data for one page element at a time, so the user sees a sequential build of the overall page. Let I_1 and I_2 be the byte counts for two page elements. Let C be the effective speed, in bytes per seconds, of the channel between the server and the client. We assume further that the client will issue a request for I_1 before the request for I_2, while the user will wait for I_2 to display before acting on the page content. If we ignore the time needed to issue the request for a page element (a request is usually small, typical a partial TCP segment), and further assume that multiple TCP connections will share the effective bandwidth in a fair manner, we can estimate the time required before I_2 is displayed to the user.

In HTTP/1.0, I_1 and I_2 will be transmitted on two separate TCP connections sharing C, so that each element will be transmitted at speed C/2. I_2 will be received at

$$T_2 = \frac{2 \cdot I_2}{C} \qquad (1)$$

In HTTP 1.1, using a single pipelined connection, the corresponding time is

$$T_2' = \frac{I_1 + I_2}{C} \qquad (2)$$

In all cases where $I_1 > I_2$, the user will perceive a longer wait for the desired page element. In cases where $I_1 >> I_2$, the performance of the application may be judged as deficient by the user. We also note that the simple estimates above apply only if the HTTP/1.0 client is able to immediately access available TCP connections for the transfer of I_1 and I_2. For pages with large numbers of elements, HTTP/1.0 will exhibit some sequential behavior.

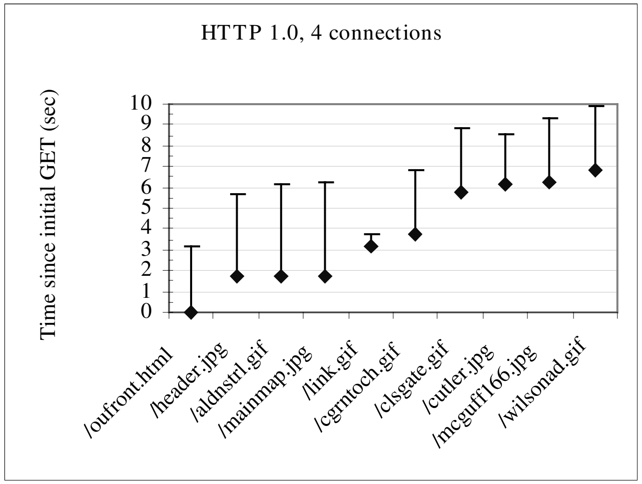

Figure 3

Figures 3 and 4 illustrate this effect on an actual web page retrieval [Kru98]. Each vertical line in the figures indicates the time of the element request (bottom of the line) and the time of the retrieval completion (top of the line), relative to the time when the base HTML request was issued. Notice that the 5th element ("link.gif") in this case is rather small. In HTTP/1.0 the element requires very little time between the GET request and the complete arrival and display of the element. The same page retrieval with HTTP/1.1 delays this small element behind a number of larger ones and delays the display of this page element.

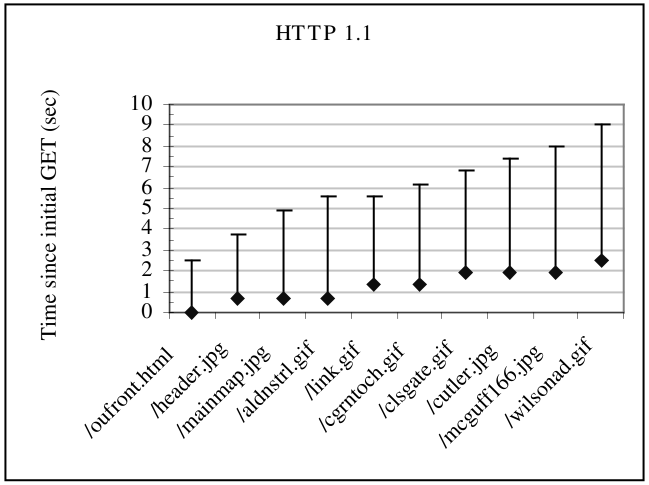

Figure 4

In the following section, we outline a model which quantifies the perceived delay effect caused by the use of HTTP/1.1. We then present a number of strategies to make HTTP both network friendly and mitigate the problems associated with the user-perceived performance that a network-friendly protocol may introduce. Our analysis is based on both the existing HTTP/1.1 protocol and several possible protocol additions.

Perceived-Delay Analysis

To quantify the perceived delay effect, we simulate the retrievals of randomly constructed sample pages. These pages have the following attributes:

- 80% of the page elements are between 1000 and 5000 bytes in size. Element sizes are uniformly distributed within this range.
- 20% of the page elements are between 35,000 and 65,000 bytes in size. Element sizes are uniformly distributed within this range.
- The base HTML document size is always selected from the small size range.
- The total number of elements is 25.

We simulate each page retrieval in an analytical model using the following assumptions:

- The HTTP 1.0 browser uses up to 4 connections.
- The Round Trip Time (RTT) for all connections is 200msec.
- The effective aggregate channel speed is 56kbits/sec
- The TCP window size was large enough (64kbytes) to insure that the transfers are limited by the channel speed.
- We assume that the channel capacity is split evenly over all connections in the HTTP 1.0 case.
- All data flow is limited to the channel rate, i.e. the TCP loss and subsequent congestion avoidance is not modeled.
- TCP slowstart is modeled, using the traditional approach of starting the transmission with a single TCP segment.
- The TCP segment size is 1460 bytes.
- We assume that every segment is acknowledged. Our model also permits the use of delayed acknowledgements, but we did not chose that option for this analysis.
- Both HTTP 1.0 and HTTP 1.1 will retrieve elements in the order in which they appear in the base HTML document. Note that commercial browsers may sometimes chose a more arbitrary retrieval order.
- HTTP 1.0 will use the next available connection to retrieve each new element.

During the simulated transfers, we note the time at which each element arrives, as well as the number of bytes represented by that element. We use all page transfers (100 in these runs) to compute:

1. P(n,t), the probability that n elements have arrived at time t, and
2. P(s,t), the the probability that s bytes have arrived at time t.

While a steep slope of P(s,t) indicates an efficient use of the network, a steep increase in P(n,t) suggests that the users is observing the display of many page elements, which should further indicate an increased arrival probability of a data or navigation item desired by the user.

Figure 5 shows a comparison of element arrival probabilities for HTTP 1.1 and HTTP 1.0 with 4 connections. The figure clearly shows that HTTP 1.0 is somewhat better able to quickly retrieve small elements. Figure 6 shows the byte retrieval probabilities for the same two cases. HTTP 1.1 is clearly more efficient at using the network. This is at least in part due to the fact that HTTP 1.0 cannot request a new element until all data has "drained" from a connection; in this case we are forced to idle the connection for one RTT.

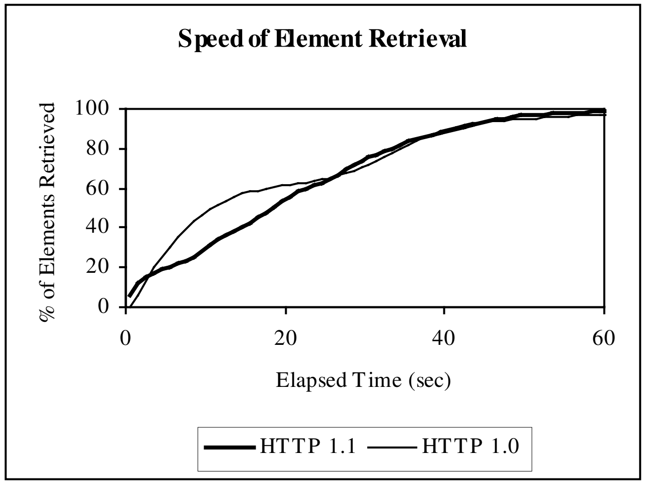

Figure 5

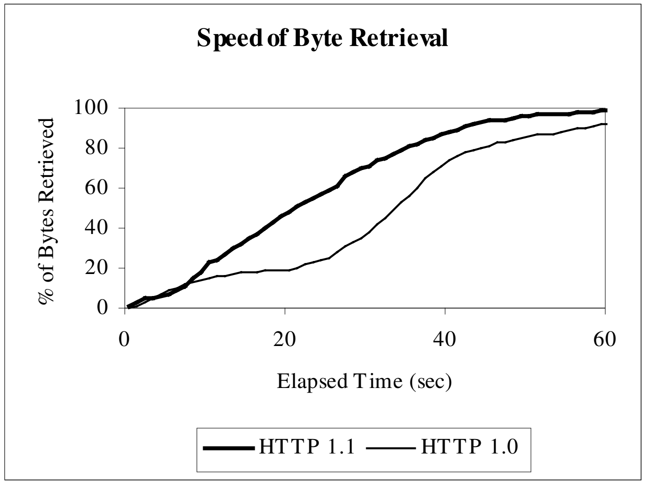

Figure 6

Effective use of HEAD Requests

In this approach, the client sends requests for the header information (using the HEAD request type) of all page elements before actually retrieving any of them. This type of request is routinely used for elements already in the client's temporary cache. We extend this scheme to all page elements. After receipt of the header information, the client uses information from the headers to determine the most appropriate order in which to request page elements. The existing standard defines the Content-Length header field. Servers that support HTTP/1.1 are likely to include this field for static pages, since the client must be able to reliably determine the end of a page element, preferably without the need for the server to close the TCP connection. For dynamic pages, HTTP/1.1 defines the new Transfer-Encoding=chunked option. In this case the HEAD request will not provide information that the client can use to order the elements. We propose a new entity header, "Content-Priority", which would be included by the server for every page element for which an early retrieval may benefit the user. Note that our proposal will require a HEAD request (and a delay of about one RTT before data retrieval begins) in some cases where current clients would not normally issue this requests.

For this paper we have simulated only the simplest heuristic, namely retrieving the elements in the order of size, from smallest to largest. Figure 7 compares the element retrieval probability, P(n,t), for a normal HTTP 1.1 transfer, and a transfer in the order of element size. As one would expect, many more page elements will arrive early in this approach. Figure 8 compares the connection utilization, using P(s,t) for the same two cases.

If a web page is designed to take advantage of the heuristic (and it is implemented in the user's client), the information provider can insure that crucial navigation items - e.g. links to on-line catalogs or current news headlines - appear quickly in the browser window allowing some users to move on, while large images and site maps are displayed later (and more slowly) for users who want to view the entire page. Implementation of this heuristic will require a modification of the client software. The extension of this heuristic would be to retrieve elements in priority order if the server provides the Content-Priority header we have proposed. This would require a server modification as well as a mechanism for the web designer to assign priorities to page elements at the time of page construction.

Retrieval of header information becomes even more useful to the client if layout information (such as the dimensions of an image) either is given in the base HTML, or is sent in optional entity header fields

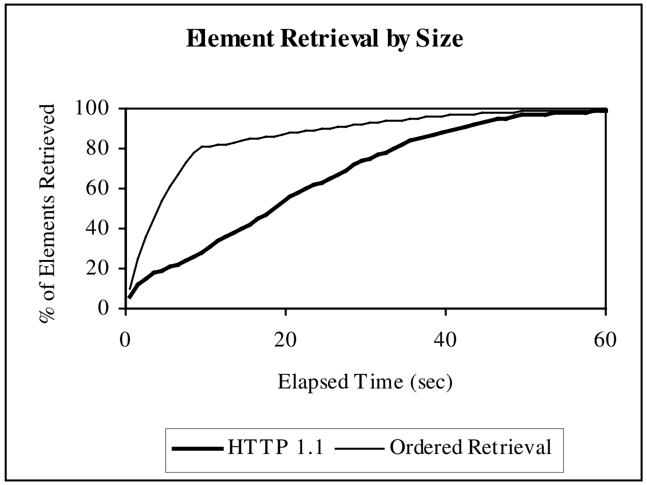

Figure 7

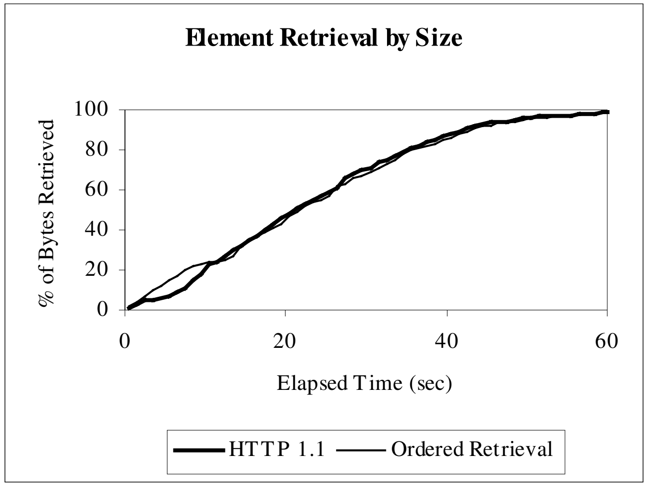

Figure 8

Interleaving of retrieved elements

Even if the client can retrieve page layout information and an appropriate ordering of page elements, retrieval and display of elements remains sequential. And, as outlined above this can cause the user's perceived performance to degrade. In an attempt to keep the perceived load time low, we propose a new interleaved pipelining approach. As the server receives multiple pipelined requests for page elements, new requests are multiplexed into the current data stream. This allows the use of a single TCP connection, making the transfer network friendly, while also loading multiple objects in parallel. Since these objects can be partially displayed as they arrive, the user will perceive roughly the same behavior as shown by using multiple parallel TCP connections under HTTP/1.0.

To accomplish interleaving, the client and server must negotiate its use during the transfer of the base HTML document. The client will send an advertisement to the server stating that interleaving is supported. This advertisements could be handled as a protocol extension to the Connection header field. Alternatively, a new entity header field could be defined (this field would be used in the header for the base HTML document). The latter approach could be introduced without a protocol version change, while a non-experimental implementation of the former option does require such a version number change.

In the server's response, it will indicate whether or not it supports interleaving. If both the client and server agree to use interleaving, the HTTP client will send an "Object-number" entity header in the request to the server for a given object. The server then returns a stream of "object records" using the object numbers given by the client request. Each record contains a maximum of several kilobytes of actual data, prepended with the object number and the length of the current record. We are assuming here a structure imposed internally on the data stream. Future work will investigate the inclusion of this approach in the "chunked" transfer encoding method.

For example, consider a simple round-robin algorithm and a web page with 3 objects other than the base HTML. The pipelined requests would be sent to the server with object numbers 1 to 3. The server would return a record of data for each of the objects, in turn. Following the transmission of the record of data for object 3, a record of data for object 1 would be transmitted. An optimal algorithm for the selection of which object to service at a given time is an open question (for instance, it may be better to service large object more often than small objects).

This method of transmitting web pages will slightly increase the overhead of transferring web pages. For instance, consider a 1 byte object number (providing up to 256 objects per web page) and a 4 byte length number (allowing objects to be as big as the largest file under many Unix systems) . If the data portion of the record is 2048 bytes the additional bytes constitute 0.002% of each record. In a recent study [PGA98], it was observed that most of the web objects requested on the NASA Lewis Research Center connection to the Internet were under 5,000 bytes. With the conditions given above the overhead associated with retrieving each of these objects would be approximately 0.003% on the studied network.

Multicasting Considerations

While the merits of Web-based advertising are arguable, advertisements do serve to help support much of the high-quality content found on Web pages. Whether supporting Web developers, networking or hardware infrastructure, or the collection and maintenance or data archives, advertisements subsidize the growing variety of sophisticated Web-based tools.

Many large organizations, such as companies or universities, connect large numbers of users to the Internet via local Intranets. It can be expected that a large number of the web pages loaded by people in these organizations may include advertisements along with the desired content. Therefore, it makes sense to transmit these advertisements in an efficient manner. We believe that IP multicasting can be used to transmit a constant stream of advertisements from the Internet to an Intranet in a bandwidth efficient manner. This will limit the number of incoming advertisements to one at any given time. Although one could reasonably assume that several streams of advertisements might be desirable at any one time, as long as this number is well under the number of people simultaneously browsing the web, the network is being used more efficiently.

There are two ways to utilize IP multicasting for the distribution of ads. The first is to use it to send advertisements to a proxy local to a given organization. Research at NCSA suggests employing the MBone to multicast a web page to a publisher-subscriber process group. Participants join a group where Webcast receives the full text and inline images from Mosaic, then multicasts them to the group. A locally running browser is then notified that the web page is available locally as a file://localhost URL [Bur95].

An alternative is to use multicasting to transmit advertisements right to user's browsers. Both of these options save bandwidth on an organization's link to the Internet (which most likely does not have as much bandwidth as the organization's internal network). If each browser is to receive advertisements via multicast, a new URL must be used to instruct the browser to show the next advertisement coming across the multicast stream. A new URL might look similar to that given in [Shi97] (i.e., multicast://...). If an organization-wide proxy is used, browsers will not need to be changed, as they will make requests to the cache server as is done currently. The following discussion is mostly general enough to encompass either of the two multicasting methods outlined above.

A multicasted advertisement makes sense from a network point of view, but may not always make sense from a business standpoint. Some web sites display advertisements based on a search made by the user (e.g., if the user is searching for information about Warren Zevon, an advertisement for an on-line record store may appear, telling the user where they can obtain Mr. Zevon's latest CD online). More specialized advertisements can continue to be unicast to the user. The "price" of putting specific advertisements on web pages then becomes a longer transfer time (and therefore, less time for the user to look at the advertisement), when compared to displaying an advertisement from the multicast stream. To lower the load time of specific advertisements these should be cached and aged by an organization proxy, just as other web objects are cached.

A multicast advertisement stream would waste bandwidth when the rate of advertisement arrivals was higher than the rate of browser requests for advertisements. For example, a multicast advertisement stream probably makes less sense in the middle of the night than it does during lunch hour. Therefore, it makes sense to be able to turn the stream on and off in some fashion (either by a policy of turning it off everyday at 5 PM or dynamically when the demand decreases). This leaves more bandwidth available for other traffic during non-peak hours (such as NNTP downloads, for example). Dynamically determining when to turn off a multicast stream is an open question. If a user's browser received a multicasted advertisement directly (as opposed to a proxy server receiving the stream, as discussed above), the stream could not be completely turned off. However, the rate of the incoming advertisements may be reduced.

Another important consideration is whether reliable multicast is required. An advertisement delivered with portions missing will not be very useful. The tradeoff is between the time required to repair loss versus discarding the advertisement and waiting for the next one. In organizations connected to the Internet with highly congested links, reliable multicast will probably be necessary, whereas in lightly congested network it is probably best to just discard advertisements in which data is missing. A possible approach is to use a reliable multicast protocol that can turn on and off the ability to request retransmissions. If an advertisement loss rate is high, hosts or proxies can begin to request retransmissions of missing packets. However, if the advertisement loss rate is low, hosts and proxies can operate in a mode where incomplete advertisements are discarded in favor of the next advertisement in the stream.

Several researchers [Cla95, Don95, Ham95] have recommended various multicast approaches to web caching or delivery of entire web pages. These mechanisms may work well and be more bandwidth efficient than the current delivery of web documents. However, these techniques do not take into account the increasing popularity of web portals. Web portals have introduced user customization or home page personalization features, which increases the type and variety of content delivered to the user. However, using multicast distribution techniques to deliver somewhat static content or update proxies may be a way to reduce the overall network traffic and congestion. With this reduction, unique web pages that must be retrieved directly from the source will also transfer more quickly. In addition, unique pages may also contain a number of common elements, which could benefit from some form of multicasting, as discussed above.

Bibliography

[Bur95] E. Burns. Webcast documentation. The National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, May 1995. http://www.ncsa.uiuc.edu/SDG/Software/Xmosaic/CCI/webcast.html

[Cla95] R.J. Clark and M.H. Ammar. Providing Scalable Web Service Using Multicast Delivery. Proceedings of 2nd International Workshop on Services in Distributed and Networked Environments (SDNE 95), June, 1995.

[Don95] J.E. Donnelley. WWW Media Distribution via Hopwise Reliable Multicast. Third International World-Wide Web Conference, April 1995.

[FJGFBL97] R. Fielding, J. C. Mogul, J. Gettys, H. Frystyk, T. Berners-Lee. Hypertext Transfer Protocol -- HTTP/1.1, January 1997. RFC 2068.

[FF98] S. Floyd and K. Fall. Promoting the Use of End-to-End Congestion Control in the Internet. Technical report, LBL, February 1998. Submitted to IEEE Transactions on Networking.

[Ham95] M. Hamilton. Multicast Approaches to World-Wide Web Caching. Technical Report 988, Department of Copmuter Studies, LUT, August 1995.

[Hei97] John Heidemann. Performance Interactions Between P-HTTP and TCP Implementations. Computer Communications Review, 27(2):65--73, April 1997.

[Kru98] H.Kruse, M.Allman, J. Griner, HTTP Page Retrieval over Geostationary Satellite Links, in preparation.

[PGA98] J. Pugsley, J. Griner, and M. Allman. NASA LeRC Network Traffic Analysis. Technical report, NASA Lewis Research Center, October 1998. In preparation.

[Shi97] M.K. Shin. Extending the World Wide Web for Multicasting an HTML Document. Proceedings of INET'97, Internet Society, June 1997.