The experimentation and modeling of "satellite-friendly" HTTP (*)

Aniruddha Bhalekar and John S. Baras

Electrical and Computer Engineering Department
and the Institute for Systems Research
University of Maryland College Park
College Park, MD 20742
{anibha, baras}@isr.umd.edu

Abstract: **The demand for Internet access has been characterized by an exponential growth. The introduction of high-speed satellite communication systems providing direct-to-home Internet services is a response to this increasing demand. However, such systems suffer from high delay-bandwidth product and high bit-error-rate which causes degradation in the performance of HTTP, which is a request-response type, application-layer protocol over TCP. In this paper we investigate HTTP/1.0, HTTP/1.1 and analyze different mechanisms to reduce the user-perceived delay and improve the performance of Internet delivery over satellite links. The paper also suggests other mechanisms, such as multiple parallel connections and a delta encoding implementation to overcome the above-mentioned problem. We show that there is a marked improvement in Internet browsability and user-perceived delay, using these methods.**

1. Introduction

This paper presents a performance evaluation of World Wide Web (WWW) page retrievals, over a network path with a satellite component. The delay caused by this component, emphasizes the importance of the different developments of HTTP, along with the ones proposed here.

Our first goal is to present a comparison of the performance of the different versions of HTTP and its various developments. We tested both HTTP/1.0 and HTTP/1.1 in our simulation scenario and describe here the pros and cons of the various developments, including persistent HTTP, the use of conditional statements, changing the underlying TCP protocol, changing the nature of the content of the WWW, data compression along with delta encoding and adaptive connections.

(*) Research supported by NASA under cooperative agreement NCC8235, Hughes Network Systems and the Maryland Industrial Partnerships Program.

Our second goal is to focus on the most promising possible development, i.e. adaptive connections. We keep in mind the fact that the current and previous versions of the most popular internet browsers, namely Netscape and Internet Explorer, do not support pipelining [Wca98]. In our simulation, we use an underlying Split-Connection TCP Reno network. For the satellite segment, we used the Receiver Window Backpressure Protocol [ZBa02]. It is similar to TCP but does not have slow start and congestion avoidance. We show via our results that there is an improvement in the performance of the system. We note that the effect of this mechanism on the web server performance remains an open question. We also analyze the delta encoding of web pages and a method for its implementation.

2. HTTP/1.0 and /1.1

HTTP/1.0 was developed as a better application-layer protocol for web transfers. The main problems with TCP over a satellite link are the Slow Start (SS) algorithm and accumulation of "time wait" states of the various connections [Spe94]. TCP requires a 3-way handshake and TCP packets are generally segmented. Typically, the MSS is 536 bytes. Now, TCP uses the SS algorithm to avoid traffic congestion. The HTTP requests are often longer than the TCP segment size. Thus the requests can take more than 1 round trip to be fully at the receiving end. The SS algorithm aggravates this delay as the second packet cannot be sent until the first has been acknowledged. Hence HTTP responses may require multiple round trips. This round trip delay in satellites is in the order of 500*ms* which is a major issue. Objects in HTTP/1.0 are downloaded back-to-back with each object requiring a separate HTTP connection. This method is not suitable for a network characterized by high delay-bandwidth product, such as ours. As most web-pages comprise of many embedded objects, the additional overhead of setting up a new connection for fetching each object individually, has an adverse effect on how much time it takes to fetch the entire web page. HTTP/1.1, on the other hand, incorporates pipelining and persistent

connections (p-HTTP). Pipelining does not require objects on a web page to be downloaded back-to-back with each object requiring a separate HTTP connection. All the objects embedded on the webpage are pipelined and can be sent over even a single HTTP connection. The flip side to this development, as mentioned earlier is the fact that none of the versions of the most popular browsers, namely, Netscape and Internet Explorer actually have implemented pipelining [Wca98]. We must remember that being HTTP/1.1 compliant does not mean implementing pipelined connections but implies only support for persistent connections.

3. HTTP Developments

3.1 p-HTTP

p-HTTP or persistent HTTP connections, which is used along with HTTP/1.1, avoids the cost of multiple opens and closes and reduces the impact of SS. Since the connections are now long lived, hence, "time wait" states required to be maintained are fewer in number. Also, the RTT's are also fewer in number due to fewer requests and hence fewer number of packets. This directly implies lower latency for the end user.

Unfortunately, this does not substantially affect web access as only 11% improvement is observed for web-access slower than 200 Kbps [THO96]. Also, connection establishment algorithms require that the requested file size ≤ delay-bandwidth product. p-HTTP also causes complexity at the application layer, since the responsibility of segmentation and reassembly now lies at that layer [KAG99].

3.2 Conditional Statements

Using HTTP conditional statements along with cache validation instead of regular statements can save redundant data from being transmitted along the network. Considering our network, this results in a direct reduction in latency. For instance, using a Conditional GET instead of a normal GET statement along with cache validation saves up to 90% of the data from being transferred [PMo94]. Timestamps are used for cache-validation in HTTP, e.g. "if-modified-since" is used for a "conditional GET" in HTTP/1.1. There are other options such as, the GETALL, GETLIST statements. The GET statement gets just one object. ALL adds all the contents in one continuous connection. The GETALL statement can be easily implemented using a pragma. Since web pages are composed of several files, GETALL is more effective than using the GET command persistently [THO96]. Unfortunately, this is useful only when no images from that page are cached, hence for the first time only [PMo94]. The GETLIST statement is useful if we don't know what we are getting. This allows us to select what we want, instead of getting what we have cached, over and over again [PMo94]. A multitude of options in GET statements, however, lead to lack of an optimal algorithm. Also, cache bursting may occur.

3.3 Transport Layer Protocol and Web Content

Is there a transport layer protocol that HTTP performs better over? Transaction TCP (T/TCP) caches the state information of the session (i.e. RTT, control block parameters, etc.) and hence the SS algorithm is quicker the next time, hence permitting early delivery of packets [HOT97]. Shared TCP Control Blocks (S-TCB) functions on the same principle as T/TCP except that S-TCB can be used for both concurrent and serial connections, whereas T/TCP can be used only for the latter. Note that connection-caching protocols are very useful even for single web-page hits [JKJ97]. Unfortunately, this approach suffers from "change-the-world" syndrome!

Altering the nature of web-content suitably, in order to make the web-page satellite friendly, will help reduce latency. These new technologies are typically smaller in size than the existing ones. Hence using Cascading Style Sheets (CSS) and Portable Network Graphics (PNG) embedded in our webpage instead of JPEG images will decrease the download time of the web page. This is being widely implemented these days and the advantages are obvious but are not immediately gratifying [NGS97].

3.4 Data Compression

We believed that it might help to decrease the user-perceived latency if the HTTP headers and/or web content are compressed using a simple compression utility such as gzip. Gzip is the most common coding accepted by most browsers [TMc02]. Both HTTP/1.0 and HTTP/1.1 support compression via "Accept Encoding" header fields. HTTP/1.1 accepts both, hop-by-hop encoding via transfer encoding and also end-to-end compression via content encoding whereas HTTP/1.0 supports only the latter [FGM99]. Gzip currently uses the Lempel Ziv (LZ-77) compression algorithm. Plain text and hence HTML, has tremendous potential for compression. The resulting page sizes have been shown to be 27% of the original using simple level 1 gzip compression [THO96]. Though images are pre-compressed, pages with several embedded images benefit immensely using simple gzip compression. The resulting page size is 33% of the original page. The advantages of data compression are as follows [TMc02]:

1. Data compression along with T/TCP amounts in enormous savings in time and bandwidth. With T/TCP 93% reduction in the number of packets transferred and 83% savings in transmission time occurred [Sta98].

2. It alleviates the problems caused by TCP SS and the client is able to send newer requests for embedded images quicker than without compression.
3. Most browsers perform streaming decompression and hence do not have to wait for the entire base HTML page to arrive before it can be decompressed and requests for the embedded objects are sent.

[NGS97] shows that using high-level compression rather than standard modem compression (V.42bis /V.44 protocols) resulted in a 64% reduction in the total transfer-time and a 68% reduction in the total number of packets transferred; hence was much more efficient. The HTML file, used in this experiment, was generated using data combined from the Microsoft and Netscape home pages and was transferred in an uncompressed and compressed form over a 28.8 kbps modem.

HTTP/1.0 has a problem with the caching of multiple versions of the same resource i.e. compressed and uncompressed ones, in this case. This problem was solved by the inclusion of the "Vary" header in the HTTP/1.1 response as this field was used to distinguish between the two. The drawbacks of using data compression are the following:

1. Server and client computing overheads. A trade-off between the price of paying extra CPU cycles for reduced network latency must be struck.
2. The lack of a specific standardized algorithm built specifically for web-content compression.
3. The lack of expanded support for compressed transfer coding. Most browsers do not support this.
4. The lack of expanded proxy support for the "vary" header. As of now, compression comes at the price of uncachebility in most instances [TMc02].

3.5 Delta Encoding

A "delta" is the difference in the web page on the server and the page that is cached locally. Delta encoding by itself has proven to be an effective technique in reducing network latency. This mitigates TCP-SS for small HTTP responses and reduces the average number of bytes for large HTTP responses. The different possible delta generators are:

1. diff-e: A compact format generated by UNIX diff
2. compressed diff-e: Diff plus compression using gzip
3. vdelta: A good delta algorithm that inherently compresses its output [MDF97]

On the arrival of a request for a certain page, instead of transmitting the entire page, a delta is generated and sent by the server, if the client already has the page cached and there has been a change in the page since the time the page was cached. Cache validation is done using timestamps. The client, then locally generates the entire page using the cached copy and the delta. If there is no change in the web page, i.e. the cached copy is fresh, the server validates the copy in the cache and that is the one displayed. Due to this scenario, the server has to maintain several old versions of the document. The issue here is in how much time and how many updates since, should the old document be saved at the server for generating deltas [Sta98]. Deltas are useful only if the page is going to be accessed multiple times and the percentage of Internet traffic that is delta-eligible [MKF02] is high. The deltas can be further compressed, though repeated compression of deltas must be avoided as compression consumes time [MDF97]. Deltas are easy to generate as the generators exist as standard library functions. The deltas can also be pre-created and cached at the server, with each upgrade of the web page. The HTTP requests sent from the client to the server can also be delta-encoded. Calculating the delta between URL requests does this, as many requests (successive or otherwise) have the same prefix [MDF97]. Moreover, the deltas must be at least less than half the original size of the object for delta encoding to be useful. To calculate deltas, the server has to cache various versions of updates of a document. This again may lead to cache bursting.

Depending on the statistics obtained from traces, we can predict to a high degree of accuracy what web page a client will ask for next while viewing a current page. Also, a specific user tends to visit certain web-pages daily, if not just several times in a particular Internet session. Hence, in such cases we could let the client ask for the expected following pages while the user is looking at the current page displayed on his screen. On receiving the following pages, the client could cache them (if they have not been received earlier) or calculate the delta (if the same page has been displayed earlier) and cache the delta. When the user requests that expected following page, the page can be generated locally and almost instantaneously, using the cached delta, while the pre-creating of further deltas is already again underway. This scheme leads to an increase in network traffic since some of the deltas may be unnecessarily created and cached. We believe that this pre-fetching scheme may prove to be extremely useful in case of high latency, high bandwidth satellite links, as in our case. The process at the server side needs to differentiate between requests that are being made for the future and requests that are made directly by the user and hence giving the latter, higher priority.

Another possibility is to let an intelligent cache which supports many users at the client end, have the responsibility of regenerating pages from deltas and sending them to the respective user. This takes away the processing load on the user's machine. When the client requests a page, the locally cached version will be displayed. At the same time, the timestamp of this delta will be sent over the satellite segment for validation from the cached delta at the hybrid gateway (HG). If the HG

has a delta with a higher time-stamp, it sends it over the satellite link to the client, where the client generates the new page and the browser auto-refreshes. At the same time the HG checks with the web-server and refreshes/upgrades the delta it has cached. If the delta cached at the HG is upgraded by the web server, the HG sends the newer delta to the client over the satellite segment resulting in the client generating the web page and the auto-refreshing of the browser. The HG refreshes/upgrades the delta in its cache irrespective of whether the client has the same delta or otherwise. This ensures the freshness of the cached delta at the HG and hence obviates the 2nd auto-refreshing of the client browser even in the worst-case scenario. A time-out scheme, an on-demand scheme and figuring out which would be the best suited for this type of hybrid gateway refreshing/upgrade along with an algorithm for the pre-creating of deltas are open problems.

4. HTTP Model and Implementation

We intended to use the meta-information field in the HTTP response header to tell us the number of embedded images on a page. This information could also be pre-fetched. Upon this, the client decides how many connections it wants to set up with the server for the transfer of data, depending on the nature of the page. If the web page has several moderate-sized elements, we tested the use of multiple connections with a regular cwnd size. We expected that the set-up time for each connection would be the same since the setup would happen concurrently [KAG99]. Hence the maximum time for the entire page data to transfer to the client, in this case, would then be equal to the time required for the single largest element on the page to transfer and this scheme might prove to be beneficial. A larger cwnd helps in reducing time by 1-1.5 RTT and is hence better for those links where RTT is high, i.e. networks with a high delay-bandwidth product segment, for pages with large objects. This can have adverse effects on the terrestrial TCP link performance [ELR02].

A certain threshold needs to be decided where we limit the number of connections and have more than one element on some connections using pipelining. In the implementation we assumed that the server knows the number of embedded images in the page beforehand. We did this to validate our conjecture that there would be an improvement in performance using this method, and then to follow up with the idea of exploiting the meta-data that we can get about the requested web page through the HTTP response headers.

We used the OPNET Modeler 8 simulator to run experiments using an underlying Split-connection TCP Reno Network. The RWBP (Receiver Window Backpressure Protocol) protocol on the satellite segment, is based on TCP but does not have slow start and congestion avoidance. For simplicity, our network was a single server, a single client, and the satellite link comprised of the hybrid gateway and satellite gateway. The index page of one of the most popular websites, www.espn.com was the sample web page taken and modeled. Several other web pages were generated, which varied in the number of embedded objects and the sizes of these objects. Experiments were run using these pages and FTP, HTTP/1.0, HTTP/1.1 as the application layer protocols and the expected results were validated. We observed the active connection count, traffic sent and received over the link in packets/sec and bytes/sec, object and page response times as the global statistics. We also observed the link utilization, the throughput (bits/sec) and the queuing delay as the object statistics.

The first experiment was run using HTTP/1.0 as the application layer request-response protocol. The simulation where the client requested the page from the server and this transfer was completed, was then run, and the desired statistics were collected. The existing scenario was then duplicated and the application layer protocol changed to HTTP/1.1. The simulation was then run again.

We then changed the application layer protocol and defined the “MaxConn” and “ExactConn” scenarios in OPNET. MaxConn is defined as a scenario where the maximum number of TCP connections that can be set up between the client and the server is equal to the number of embedded objects in the web page. In our setup, this is set to be 50. The SS starts with 1 segment (small cwnd). The ExactConn scenario is defined as exactly one connection with a larger congestion window (cwnd) i.e. the SS algorithm starts with 4 segments instead of just 1. The idea of MaxConn was to reduce the latency in the network, by getting exactly one element per connection. Since, no pipelining or persistence was required; we used HTTP/1.0 for these connections. Theoretically HTTP/1.0 can support a large number of connections. We simulated two web pages as two extreme cases, i.e. one with 1 large sized element and another page with 50 small sized elements.

In the second experiment we transferred these two simulated web pages using the MaxConn and ExactConn scenarios as the application layer protocol instead of HTTP/1.0 or /1.1, respectively. Modifying the “profiles” module, in the simulation setup, accomplished this.

In the third experiment we simulated a sample web page, i.e. www.espn.com, and transferred it from the web server to the client using HTTP/1.0, HTTP/1.1, MaxConn and ExactConn and compared the metrics.

5. Observations and Results

The statistics from the two simulation runs of the first experiment were compared and the results stated in the literature survey were validated.

The following observations were made with respect to the results of the second experiment using the MaxConn scenario and the page with many small elements.

1. Traffic received per second was higher than with HTTP/1.1 and was much higher than with HTTP/1.0
2. Traffic sent was the same as when using HTTP/1.1
3. Page response time was lower than with HTTP/1.1 and much lower than when HTTP/1.0 was used
4. Object response time was as low or lower than HTTP /1.1 and much lower than HTTP/1.0

In the ExactConn scenario of the second experiment, for the page with many small elements, traffic was high and moderate, though over a longer time period. The page response time was slightly higher than with HTTP/1.1 and MaxConn but much lower than with HTTP/1.0.

In the second experiment, for the page with one large element, we observed unexpected results. We noted that all the 4 application layer protocol variations had identical performances in all considered metrics. This obviously meant that HTTP/1.0, HTTP/1.1, MaxConn and ExactConn use the same connection to transfer the page and its contents. We also noted a drastic improvement in the utilization, throughput (*bits/sec*) and especially the queuing delay, in object statistics. We hence concluded that multiple parallel connections are better for a page with multiple elements. Also, increasing the initial cwnd does not contribute to performance enhancement substantially for a page with a small number of larger objects.

In the third experiment, with the ESPN page, we observed (Fig.1) that the traffic sent and traffic received per second (*bytes/sec* and *packets/sec*) was highest for MaxConn and considerably lower for HTTP/1.1. Both the scenarios required very little time for the entire transfer, though. The time required for this transfer using HTTP/1.0 was much higher and was extremely high when using the ExactConn scenario.

The page response time (Fig.2) was lowest for MaxConn is was much better than all other scenarios including HTTP/1.1. The object response times (Fig.3) are identical for HTTP/1.0 and ExactConn and are slightly higher for MaxConn and higher than that for HTTP/1.1, but the performance of MaxConn is still by far better as most of the object response times (if not all) overlap. This is completely unlike the HTTP/1.0 or the ExactConn scenarios which bring in objects, back-to-back.

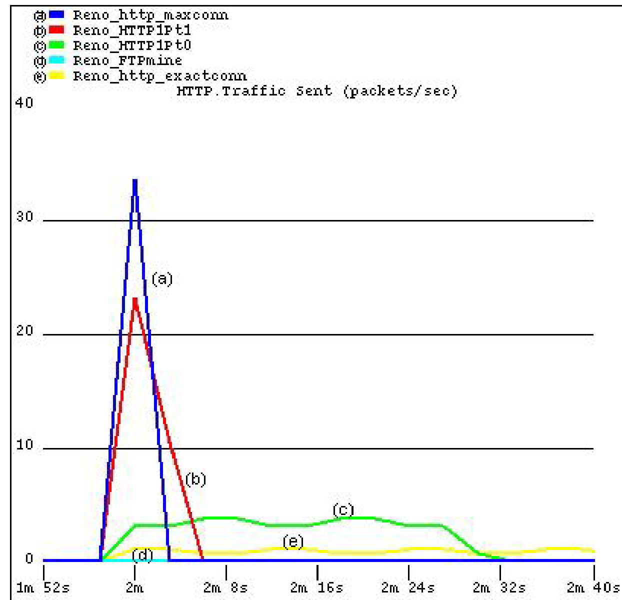

Fig. 1 – HTTP traffic sent (packets/second)

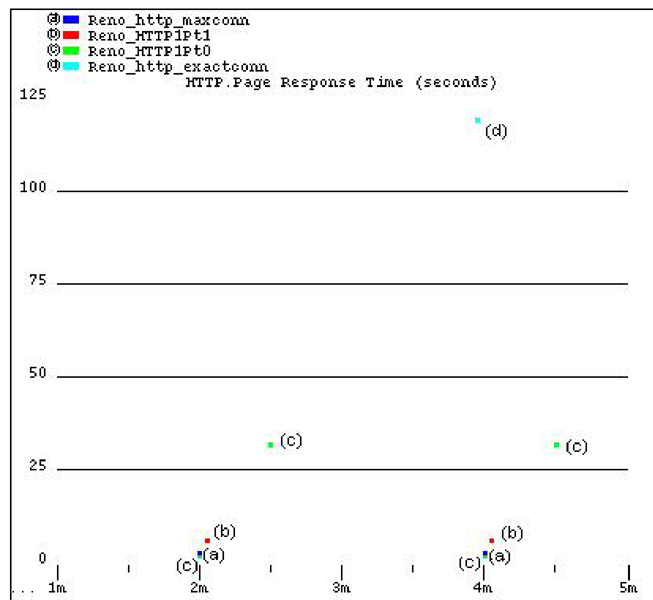

Fig. 2 – Page Response Time

We also ran experiments changing the number of elements on the page and observed that there was a definite and appreciable improvement in performance when a maximum of connections were used. When we increased the number of connections from zero through the number of elements on the page, the performance kept getting better and better as the number of connections approach the actual number of elements on the page. Identical results were observed even on increasing the number of connections above the number of elements on the page. A page with different number of elements has further validated these observations.

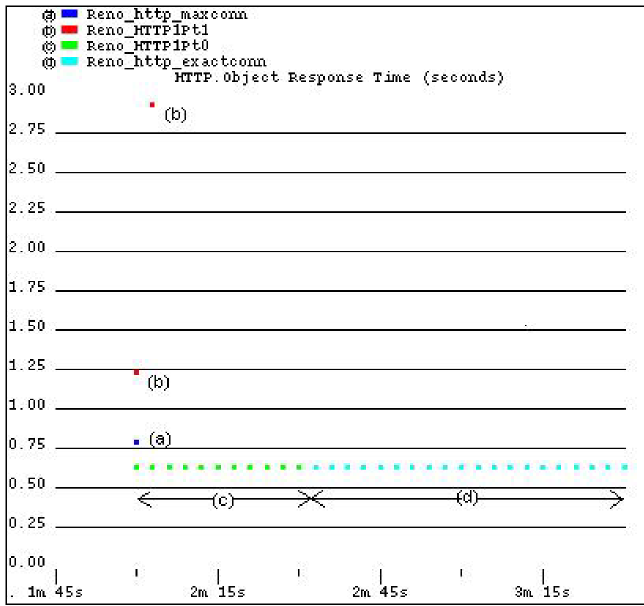

Fig. 3 - Object Response Time

6. Conclusions

By means of our various experiments using different flavors of HTTP we observe that the setting up or multiple parallel connections achieved the best results in terms of link utilization, transfer time, page and object response time, queuing delay and throughput. When we do not use connections that support pipelining, setting up many connections is better than increasing the initial congestion window size of the connections. The performance of the system keeps increasing as the number of parallel connections approaches the number of embedded images on the web page. This is especially important since Netscape and IE browsers do not implement HTTP pipelining in spite of being HTTP/1.1 compliant. Thus we conclude that using multiple parallel non-persistent connections, without pipelining, leads to best overall results, in the current scenario, where web pages typically have several embedded objects.

7. Future Work

Future work in this area consists of, but is not limited to studying closely the effect of multiple parallel connections on server load and bandwidth utilization. We will also look into finding an optimal number of connections to be set up depending on the number of elements on the web page.

References

[AFP98] IETF-RFC 2414, M. Allman, S. Floyd, C. Partridge, "Increasing TCP's Initial Window", September 1998

[CDi01] IETF-RFC 3143, I. Cooper, J. Dilley, "Known HTTP Proxy/Caching Problems" June 2001

[DFK97] Fred Douglis, Anja Feldmann, Balachander Krishnamurthy, "Rate of Change and other Metrics: a Live Study of the World Wide Web", December 1997

[ELR02] Navid Ehsan, Mingyan Liu, Rod Ragland, "Measurement Based Performance Analysis of Internet over Satellites", 2002

[FGM99] IETF-RFC 2616, R. Fielding, J. Gettys, J. Mogul, "Hypertext Transfer Protocol -- HTTP/1.1" June 1999

[GPC98] A.McGregor, M. Pearson, J. Cleary, "The effect of multiplexing HTTP connections over asymmetric high bandwidth-delay product circuits", November 1998

[HOT97] John Heidemann, Katia Obraczka, Joe Touch "Modeling the Performance of HTTP over Several Transport

[JKJ97] Protocols" June 1997

[KAG99] Hans Kruse, Mark Allman, Jim Griner, Diepchi Tran "Experimentation and Modeling of HTTP over Satellite Channels" 1999

[KMK98] Balachander Krishnamurthy, Jeffrry Mogul, David Kristol, "Key Differences between HTTP/1.0 and HTTP/1.1", December 1998

[MDF97] Jeffrey Mogul, Fred Douglis, Anja Feldmann, "Potential Benefits of Delta Encoding and Data Compression for HTTP" December 1997

[MKF02] IETF-RFC 3229, J. Mogul, B. Krishnamurthy, A. Feldmann, "Delta encoding in HTTP", January 2002

[NGS97] Henrik Nielsen, Jin Gettys, Anselm Baird-Smith "Network Performance Effects of HTTP/1.1, CSS1 and PNG" 1997

[NLL00] IETF-RFC 2774, H. Nielsen, P. Leach, S. Lawrence, "An HTTP Extension Framework" February 2000

[PMo94] Venkata Padmanabhan, Jeffrey Mogul "Improving HTTP Latency" 1994

[PMo98] Venkata Padmanabhan, Jeffrey Mogul, "Using Predictive Prefetching to Improve World Wide Web Latency" 1998

[Spe94] Simon Spero “Analysis of HTTP Performance Problems" July 1994

[Sta98] Mark Stacy "Suggested Performance Improvements for HTTP" 1998

[TMc02] Timothy McLaughlin “The Benefits and Drawbacks of HTTP Compression” 2002

[THO96] Joe Touch, John Heidemann, Katia Obraczka "Analysis of HTTP Performance" August 1996

[Wca98] Zhe Wang, Pei Cao, "Persistent Connection Behavior of Popular Browsers" 1998

[ZBa02] Xiaoming Zhou, John S. Baras, “TCP over satellite hybrid networks”, February 2002