Proceedings of the 2007 IEEE International Conference on Telecommunications and
Malaysia International Conference on Communications, 14-17 May 2007, Penang, Malaysia

A Performance Analysis of Page Retrieval with HTTP-MPLEX on Asymmetric Links

Robert Mattson and Somnath Ghosh
School of Computer Science and Electronic Engineering
La Trobe University, 3086
Australia
Email: rlmattson@ieee.org and somnath@cs.latrobe.edu.au

***Abstract*—HTTP-MPLEX [1] is a header compression and response encoding scheme for HTTP. It is intended to speed up response time for multiple simultaneous HTTP transactions and improve application layer use of TCP by reducing the number of parallel connections and sustaining response bursts. HTTP-MPLEX achieves these objectives by reducing request size and prioritizing responses during multiplexing.**

To evaluate the performance of HTTP-MPLEX relative to HTTP 1.1 [2] in an asymmetric network environment; we developed a client and server in C++ with plug-in versions of our HTTP 1.1 and HTTP-MPLEX engines. We present in this paper a performance evaluation of our hypertext transfer engines using both the simulated network environment ns (2.29-snapshot-20050921) and an Asymmetric Digital Subscriber Line (ADSL) connection to the Internet.

We used snap-shots of www.cnn.com, www.whitehouse.gov, www.latrobe.edu.au and a photo gallery of forty eight tiles as sample web pages to retrieve using our hypertext transfer engines.

I. Introduction

HTTP 1.1 is a transactional protocol that uses the Transmission Control Protocol (TCP) for ordered guaranteed delivery of web objects between agents. To reduce the impact of delay, loss or connection interruption between agents more than one TCP connection is commonly used, usually two [2], [3] or four [4]. The trade-off between using many connections or a minimum of connections is a dichotomy. Multiple connections allow for a degree of redundancy should a connection stall or reset. Too many connections introduce excessive management demands on the kernel TCP stack [5], increase the amount of overhead incurred for each object to be retrieved and introduce connection competition. Multiple connections do, however, allow for many objects to be retrieved simultaneously enabling smoother progressive rendering of multiple objects within web documents. As an HTTP server must respond to requests in the order in which they are received, many smaller important responses can be delayed by the transmission of one large response; multiple connections enable multiple objects to be retrieved simultaneously. In recognising that reducing the number of TCP connections used to download web pages is desirable [6], [7], HTTP has progressively improved its utilisation of TCP from using one connection for each request/response pair to the pipelined and persistent connections used by HTTP 1.1.

Systems exist for aggregating requests and responses to increase the utility HTTP derives from a single connection. Systems have been proposed for aggregating requests [8], [9], [10], and protocols do exist for condensing pages and their components into coherent single entities MHTML [11], Bundles [12], [13] and data URLs [14]. MHTML [11] and data URLs [14] are specifications that allow the data from referenced objects to be compiled into a single coherent HTML file. Bundles rely on the server volunteering the capacity to transmit an entity that contains all referenced objects for a page and/or a bundle that contains generic objects for a specific site. Page bundles can be supplemented with content encoding techniques like compression and delta encoding [15], [16], [17]. Assuming few objects on a site change frequently, combining delta encoding with page bundling techniques may be an optimum way of reducing the cost of retrieving pages that have been slightly modified.

This paper considers the performance of HTTP 1.1 and the HTTP-MPLEX response protocol in asymmetric networks. HTTP-MPLEX [1] is a protocol adoption of HTTP defining two inter-related mechanisms: a request compression system and a response multiplexing algorithm and encoding scheme. By aggregating queries and multiplexing responses HTTP-MPLEX can deliver more objects simultaneously than the established number of TCP connections would traditionally allow with the capability of prioritising responses or parts of responses, all over a minimum number of connections.

The HTTP-MPLEX response encoding scheme and algorithm is described in section II. The result of simulation with HTTP 1.1 and HTTP-MPLEX is described in section III. Results of experimentation with HTTP-MPLEX in symmetric networks is addressed in section IV. We conclude in section V.

II. The HTTP-MPLEX Protocol

HTTP-MPLEX [1] is a dual purpose enhancement of HTTP. Firstly, HTTP-MPLEX specifies a header compression scheme using an inferred inheritance hierarchy to minimise redundancy in requests and secondly, uses a response multiplexing scheme for reducing request response times and accelerating the delivery of small objects. HTTP-MPLEX achieves high degrees of request compression by eliminating the large amount of redundancy present in an HTTP request stream.

To compress requests a syntax and algorithm described in [1] is applied; the most redundant request is selected from a cache of requests (or queue of outgoing requests) and all other requests are delta-encoded using the selected request as a template. As many requests are condensed into one compressed request, a server can farm out de-compressed requests to sub-processes/threads/CPU's to exploit modern highly-parallel architectures. This technique may cause responses or partial responses to be ready for transmission from the server out of order. Multiplexing provides the capability to transmit responses on a first come, first served basis. Similarly, for en-route caching, if an intermediate cache has a fresh copy of an object, that object can be transmitted while responses for other objects are pending or slow to receive. These techniques relieve the head of the line blocking problem common to input queued switches and present in the stream/FIFO oriented HTTP.

Using the request compression benefits of HTTP-MPLEX we intend to reduce congestion at the upstream client side link and improve the overall download time for a web page.

To compose a compressed request (shown in Figure 1) an initial parent request is chosen from the request pool that has the most number of elements in common with all other requests and subsequent requests are nested using 'GET⟨int⟩:' headers directly below the 'Host:' header. The amended GET line contains three important elements. Firstly, the unique integer serialising the nested requests which is used to associate response channels in the multiplex streams with a specific request which must be atomic. Secondly, the URI identifying the object to be retrieved and lastly the parenthesised expression may optionally be appended to the end of the header. The parenthesised expression is a mechanism to control which attributes of a parent request the nested request will inherit. Absence of a parenthesised expression infers total inheritance for the nested request. This parenthesis is a semicolon separated list of elements, the first of which is a bitmap of variable length. Each bit in the bitmap corresponds to each header in the parent request in order. A value of one forces inheritance and a value of zero will inhibit inheritance. If the first element of the parenthesis is whitespace (the bitmap is not present) total inheritance is inferred. The request engine may provide an incomplete bitmap, enumerating bits sufficient to exclude a particular element and leaving the other elements out (enforcing inheritance for those elements). After the initial bitmask, the parenthesis allows for a list of headers separated by semicolons to supplement those inherited from the parent request. A header present in the parent request and parenthesis is overridden by the value in the parenthesis.

The compressed request format is backwards-compatible with HTTP because an origin server not compliant with HTTP-MPLEX will ignore the nested GET requests and respond to the parent request with a standard HTTP 1.1 response. On the arrival of an HTTP-MPLEX request at an origin server, the FIFO serialised nature of HTTP requests and responses does not have to be enforced. Having more than one request arrive at the same time places the server in an advantageous position. It has four options available to it to decide how to respond. It can transmit an error response, respond with a conventional response signalling non-compliance with HTTP-MPLEX (and causing the client exchange to revert to HTTP 1.1 for the remaining interaction), transmit a response to each nested request in succession or transmit an HTTP-MPLEX compliant response. HTTP-MPLEX maintains the conventional HTTP response message format of a message header followed by a message body. In the instance of an HTTP-MPLEX response, the message header has a unique response code of 211 and contains generic headers pertaining to all responses in the message body. The response body contains the individual responses to each nested request in the compressed request multiplexed (or interleaved) amongst each other in the size of a pre-determined variable *ChunkSize* which can be context dependant. Each multiplexed response contains message headers and a message body for an uncompressed request. Multiplexed responses contain only headers specific to that stream, inheriting the generic headers from the response headers of the multiplexed response. A basic scheme has been prescribed in which responses are prioritised by size, divided into equal size chunks and transmitted with a light-weight header by round robin.

In addition to the existing protocol specification, we implemented the following amendments from our experience with HTTP-MPLEX. As it is important for browsers to receive response headers as early as possible, especially for conditional requests, we transmit only the HTTP response headers in the first round of multiplexing to ensure responses are received as early as possible. If some proportion of the total number of multiplexed responses are positively cached acknowledgements, the browser will receive the positive acknowledgement before any object data is received.

The header compression scheme described in [1] fails in part to recognise that multiple HTTP requests for different origin servers may traverse a common proxy, and therefore fails to recognise the possibility that multiple requests destined for a common origin server configured for multiple virtual servers or a proxy server could and should be compressed together. Such flexibility must provide for the overriding of the mandatory *'Host:'* header for nested/derived requests.

The use of a simple round robin scheme for multiplexing outgoing responses was proposed. As responses are multiplexed in the order provided by the compressed request, some smaller responses can be unfairly delayed when response size is less than *ChunkSize*, or if the number of responses is large. To reduce unfair competition between response streams we order all responses by size from smallest to largest before transmission. Streams for which a response size is unknown are transmitted last until a response size is known.

A browser's ability to request objects in a compressed form is a function of two things; the amount of data that is passed to the browser's HTML parser from the transport layer at a discrete point in time and the sparseness of references in the HTML page. A page with high link density will allow the browser to compress many requests together. HTML with

```
GET␣/ltu_assets/images/interface/home_logo.gif␣HTTP/1.1\r\n
Host:␣www.latrobe.edu.au\r\n
GET1:␣/ltu_assets/images/interface/home_random3.jpg␣{11110}\r\n
GET2:␣/images2/home_campuspages.gif␣{11111111;
␣␣␣␣If-None-Match:␣"17b477-16ac-3fce87c2"}\r\n
GET3:␣/images2/text_00.gif␣{11111111}\r\n
User-Agent␣Mozilla/5.0␣(Windows;␣U;␣Windows␣NT␣5.0;␣en-US;␣rv:1.7.6)
␣␣␣␣Gecko/20050319\r\n
Accept:␣image/png,*/*;q=0.5\r\n
Accept-Language:␣en-us,en;q=0.5\r\n
Accept-Encoding:␣gzip,deflate\r\n
Accept-Charset:␣ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
Keep-Alive:␣300\r\n
Referer:␣http://www.latrobe.edu.au/\r\n
Cookie:␣phpbb2mysql_data=a\%3A0\%3A\7B\%7D;
␣␣␣␣ltusitevisitor=131.172.xx.xxx.31990109779449440\r\n\r\n
```

Fig. 1. Four HTTP requests compressed with HTTP-MPLEX (658 characters).

sparse references may force the browser to fall back to one object per request. The parser should have the option of caching references to maximise the number of objects in a request. Our version of HTTP-MPLEX used for this paper includes a request staging algorithm to maximise the number of objects requested in a compressed request. Our parsing engine would cache references until one of four thresholds were met:(1) a minimum number of bytes have been received (set to 4380 bytes); (2) a minimum number of segments have been received from the transport layer; (3) a minimum number of references have been discovered in the HTML or (4) the end of the HTML page had been received. A time out would be a good supplement to the reference caching algorithm however packets traced in the simulation environment and experimental evaluation did not suffer delay to warrant implementation nor was time available to implement timers in both native Linux and ns. Any real world implementation would make good use of such a feature. To maintain maximum link utilisation, our engine would not cache requests if there were less than four outstanding responses for all connections.

Limited experimentation was done with caching all references until the entire HTML page was retrieved. Those results are presented in section III. Delaying all requests allowed us to achieve a maximum amount of compression from HTTP-MPLEX. Conversely, the delay introduces a round trip time delay to the total time taken to retrieve the page.

III. Simulation

There are also some important differences between the behaviour of our client and traditional browsers. Browsers normally restrict the number of outstanding unfulfilled requests per connection to reduce the effort required to recover from a communications error. The HTTP 1.1 browser would transmit a request to the server immediately upon parsing a reference to an object without delay. Browsers also typically cap the number of simultaneous connections per server to two [2], [3] or four [4] to reduce excess overhead, connection competition and interference. One connection was chosen for clarity when analysing simulated protocol performance. Two connections were also used when both simulating and experimenting to give results that are comparable with contemporary browsers.

The synthetic network was configured with three nodes: a client, a server and an intermediate node. The server was linked to the intermediate node via a 10mb bi-directional 'duplex' link, 60ms propagation delay and a DropTail queuing scheme. The client node was connected to the intermediate node via two uni-directional links with 10ms propagation delay, 1.5mb/sec of bandwidth from the intermediate node to the client and 256kb/sec of bandwidth from the client to the intermediate node. The network was not configured to drop packets or to synthesize cross-traffic. Our ns simulator was configured with a network MTU of 1,500 bytes (*Agent/TCP/FullTcp set segsize_ 1460;*). HTTP was configured to append 385 bytes of overhead per request, which is consistent with the amount of overhead appended by Mozilla 1.7.8. Recommendations in [18] and [19] were adhered to such that the TCP_NODELAY flag should be set on both the ns TCP stack and Linux sockets for later experimentation.

Snapshots of four web pages were taken on 20th July, 2005 to be used as samples for retrieval. The pages were selected as they had moderate HTML complexity, varying size and a high number of references. Web page construction (on the web) varies widely, hence it is difficult to capture a page set representative of the wider web; at the very least the pages used are highly popular and frequently accessed. We did not include dynamically created references in our pages, and references to alternate servers were mapped to the same site as the web server.

1) **Picture gallery** - A photo gallery of 48 JPEG tiles and CSS built with Mino Studio. Image references are uniformly spaced at 100 bytes (with the exception of end of row references and start of row references which

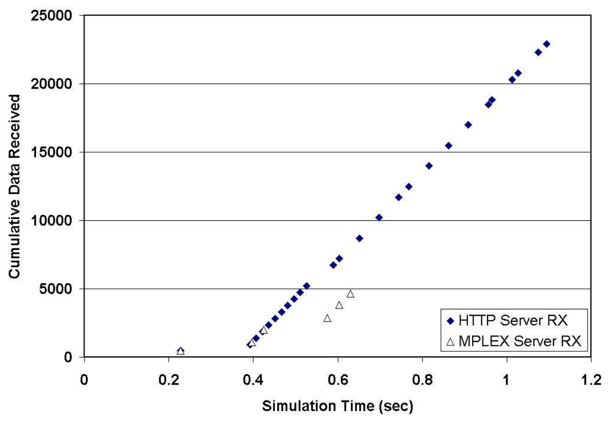

Fig. 2. Arrival of data request packets at the simulated server using HTTP 1.1 and HTTP-MPLEX

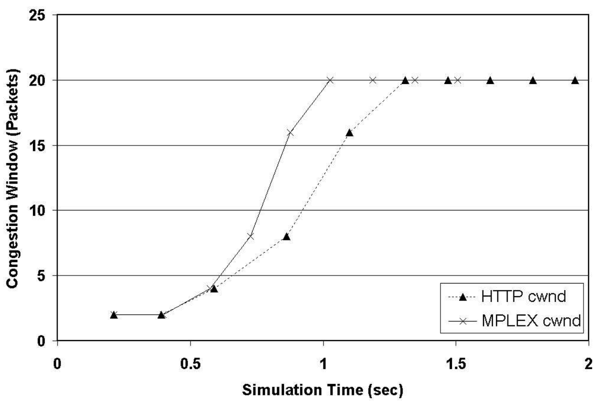

Fig. 3. Client side congestion window (cwnd) value of the server side TCP stack for HTTP 1.1 and HTTP-MPLEX

are separated by 143 bytes). Tiles are on average 3KB. A minimum of 1,997, maximum of 3,776 and standard deviation of 468 bytes.

2) **www.cnn.com** - An index page of 55.4KB, with 70 widely dispersed references. Referenced objects average 1,720 bytes, with a minimum of 37 and maximum of 28,579 bytes. A standard deviation of 4,070 bytes.
3) **www.whitehouse.gov** - An index page of 41KB, with 57 references. A tightly clustered number of references are present in the first 8,746 bytes. A common feature of many web pages, these images are used to compose a page banner with different link targets. Images average 3,649 bytes, with a minimum of 43 and maximum 17,729 bytes. A standard deviation of 3,852 bytes.
4) **www.latrobe.edu.au** - An index page of 21,398 bytes. 25 references to objects are tightly clustered in the middle of the HTML. Referenced objects average 3,042 bytes. A minimum of 43 bytes, maximum of 50,729 bytes and standard deviation of 10,101.

The retrieval of the above described picture gallery was simulated in ns 2.29-snapshot-20050921 to evaluate the performance of HTTP-MPLEX relative to HTTP 1.1.

The initial request for the index file was 560 bytes for both protocols. HTTP transmitted requests in 26 segments, totalling 21,876 bytes, with packets averaging 841 bytes and a mode of 438 bytes. 10 packets were 1,460 bytes.

HTTP-MPLEX was successful at reducing the total size of requests from 21,876 bytes to 4,412 bytes. HTTP-MPLEX requests contained 5, 11, 11, 12 and 10 objects referenced per request. Figure 2 shows the arrival time for packets containing requests for both HTTP-MPLEX and HTTP 1.1. Also shown is the reduction in number and size of packets for HTTP-MPLEX.

In the simulation and experimentation we conducted HTTP-MPLEX, performed better than HTTP 1.1. Figure 1 shows the arrival of data request packets at the server interface for both protocols. All requests for HTTP-MPLEX had completed in the sixth data packet arriving at 0.63 seconds. The last request for HTTP 1.1 was received in the twenty seventh data packet at 1.094 seconds. When contention for the upstream link was alleviated, as all requests had been sent, acknowledgement packets can be sent on the upstream link with less competition for bandwidth.

Indeed, if packets are acknowledged faster with HTTP-MPLEX than with HTTP 1.1 the time and value of the congestion window (cwnd) and round trip time (RTT) variables calculated by the TCP stack should differ dramatically between the two protocols. The congestion window and RTT values calculated by the server TCP stack are demonstrated in Figures 3 and 4. At the third calculation of RTT and cwnd (after calculations made with synchronisation and the initial index page request) the values calculated by TCP begin to differ dramatically between the two protocols. HTTP 1.1 calculates a maximum RTT value of 270ms 862ms into the simulation, while HTTP-MPLEX calculates a maximum value of 180ms at 547ms. Consistent with the slow start algorithm, the TCP stack will open up the congestion window to a larger value earlier with a smaller RTT. HTTP-MPLEX achieves a congestion window size of 20 283ms before HTTP 1.1. Both protocols reach a stable RTT of 160ms though HTTP 1.1 reaches this time 280ms after HTTP-MPLEX.

Figures 5 and 2 show the arrival time of packets at the client and server interfaces respectively. All packets arriving at the client are 1,500 bytes (1,460 at the application layer), except for the last packet of the transfer which is 45 bytes for HTTP and 675 for HTTP-MPLEX. 168,045 bytes in 113 packets are used to transmit the page using HTTP and 167,715 bytes in 112 for HTTP-MPLEX. The page was transmitted by HTTP-MPLEX with 330 bytes less than HTTP 1.1. An HTTP-MPLEX compliant response stream can be smaller than the accumulated size of traditional responses. As MPLEX response streams are nested within the body of a larger request, we can again infer that nested responses in the stream inherit headers from the nesting response. This allows us to again reduce or possibly eliminate redundant information in response headers and reduce total response size. It is unlikely

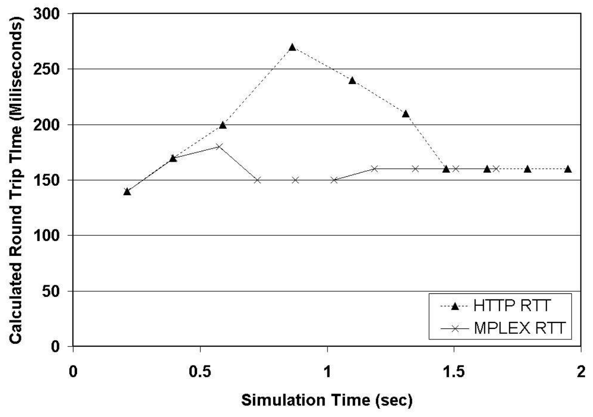

Fig. 4. Round trip time (RTT) calculated value of the server side TCP stack for HTTP 1.1 and HTTP-MPLEX

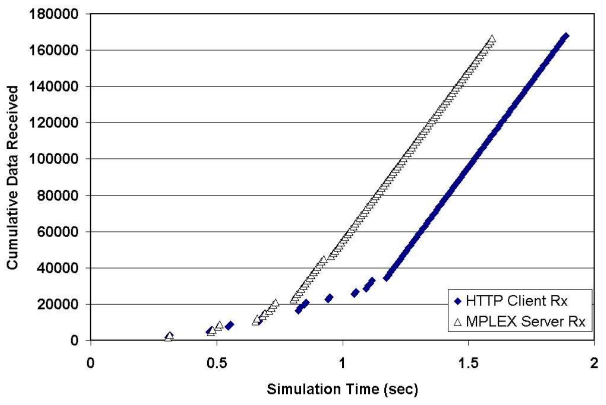

Fig. 5. Arrival of data response packets at the simulated client using HTTP 1.1 and HTTP-MPLEX

that a multiplexed response stream that is large with many interleaved responses will be comparatively smaller because of the penalty incurred including the stream and chunk headers.

The total retrieval times required for each page and protocol using both one and two connections are shown in table I. Using a single connection, the time to retrieve pages with HTTP-MPLEX was an average of 16.1% faster than HTTP 1.1. Using two simultaneous connections, HTTP-MPLEX retrieved pages 12.9% faster than HTTP 1.1. The time required to retrieve the university web page (page 4) was only minimally improved by 79 milliseconds (5.9%) and retrieval of the picture gallery was 50 milliseconds faster (2.7%). HTTP-MPLEXs best improvements were with page 2 using one connection, and page 4 using two connections, improving total retrieval time by 28.7% and 19.1% respectively. HTTP-MPLEX retrieved all our sample pages faster than HTTP 1.1 for all simulations over an asymmetric link.

Delaying the transmission of HTTP-MPLEX requests until receipt of the entire HTML page and transmitting a single request took 1.67 seconds. This method is somewhat faster than that of traditional HTTP, though as the cost of transmitting requests is not hidden behind the delay of receiving responses, this time is slightly longer than that for HTTP-MPLEX. The single request of HTTP-MPLEX was 2,432 bytes. This compares with a cumulative total of 21,876 bytes for HTTP 1.1 from the previous simulation.

TABLE I
PAGE RETRIEVAL TIMES IN SECONDS FOR HTTP 1.1 AND HTTP-MPLEX IN THE NS SIMULATED NETWORK ENVIRONMENT.

Page	Protocol	1 Connection Limit (seconds)	2 Connection Limit (seconds)
1	HTTP 1.1	1.886	1.829
	HTTP-MLPEX	1.599	1.778
2	HTTP 1.1	2.245	2.168
	HTTP-MLPEX	1.744	1.776
3	HTTP 1.1	2.487	2.414
	HTTP-MLPEX	2.127	2.133
4	HTTP 1.1	1.355	1.575
	HTTP-MLPEX	1.276	1.273

IV. EXPERIMENTATION

To evaluate the effectiveness of HTTP-MPLEX over a network we experimented with retrieving the sample pages over an ADSL link. The client was executed on a RedHat 7.3 GNU/Linux (2.4.18-3 Kernel), D-Link 504/Watchguard SOHO 6tc combination to a residential 256/1,512kb ADSL link. The server was another RedHat 7.3 operating system in our computer science laboratory connected via the university WAN to the Internet. To compare and evaluate the total time required to retrieve the sample pages, each page was retrieved 75 times for each protocol. The experimentation was conducted late at night to reduce unpredictable and variable delay and loss at intermediate routers. Retrieval with HTTP 1.1 and HTTP-MPLEX was interleaved by one second.

The results are summarised for the performance of HTTP 1.1 and HTTP-MPLEX in table II.

The experimentation and simulation environments are not absolutely consistent, as the propagation delay for the simulation network (140ms RTT) was many times greater than the delay experienced when using the Internet (approx RTT of 20ms). The simulation configuration was selected to emulate delay experienced by international traffic, and we could only utilise delay in our experiment over metropolitan distance. Difference in propagation delay would help to explain the difference in time between the simulation results and experimental results. Experimentation results are very low in variance, attributed to the fact that a time was chosen when it was less likely that peak traffic would affect results. A low value in the confidence intervals and standard deviation calculations for both protocols is a reflection that there was a lack of variance in background traffic impacting retrieval.

The average performance improvement seen in experimentation (25.64%) is almost twice that seen in simulation (12.9%) when using two TCP connections. Performance difference is most likely due to the reduction in propagation delay, as the time required to transmit data contributes more to the total

TABLE II
THE TIME TAKEN IN SECONDS FOR HTTP 1.1 AND HTTP-MPLEX TO REQUEST AND RETRIEVE FOUR WEB PAGES. STASTICS ARE DERIVED FROM 75 PAGE RETRIEVALS.

Page	Measure	HTTP 1.1	HTTP-MPLEX	Improvement
1	Average Time in seconds (τ)	1.79	1.23	0.56sec (31.27%)
	Standard Deviation (σ)	0.08	0.05	
	95% Confidence (α)	$\pm$0.02	$\pm$0.01	
2	τ	1.95	1.39	0.56sec (28.7%)
	σ	0.05	0.05	
	α	$\pm$0.01	$\pm$0.01	
3	τ	2.29	1.82	0.47sec (20.5%)
	σ	0.03	0.01	
	α	$\pm$0.00	$\pm$0.01	
4	τ	1.01	0.79	0.22sec (22.05%)
	σ	0.03	0.04	
	α	$\pm$0.00	$\pm$0.00	
	Average performance improvement:			0.45sec (25.64%)

delay than propagation. As HTTP-MPLEX reduces the amount of request data transmitted, the proportion of time saved by reducing data transmitted is greater.

V. CONCLUSION

TCP relies on the slow-start algorithm to increase the rate at which packets are injected into the network from a minimal initial rate. As HTTP transmits many verbose requests over a relatively slow upstream link, the rate of acknowledgement is restricted. HTTP-MPLEX alleviates contention of the upstream link by eliminating request verbosity, which in turn, reduces the delay in acknowledging response traffic. Faster responses enable slow-start to grow the rate of packet injection faster.

This increased rate of growth allows HTTP-MPLEX to retrieve pages on average 25% faster on a residential 256/1,512kb ADSL connection and 12.9% faster in simulation when using two parallel TCP connections. In addition to the reduction in time required to retrieve a web page, the results demonstrate that there is effective reduction in the size and number of request packets sent from our client to our server which results in a reduction of bandwidth consumption.

REFERENCES

[1] R. L. Mattson and S. Ghosh, *New Trends in Computer Networks*, ser. Advances in Computer Science and Engineering Reports. Imperial College Press, 2005, vol. 1, ch. HTTP-MPLEX: An Application Layer Multiplexing Protocol for the Hypertext Transfer Protocol (HTTP), pp. 190–201.

[2] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, *Hypertext Transfer Protocol – HTTP/1.1, RFC 2616*. Internet Engineering Task Force, 1999.

[3] mozillaZine, "Network.http.max-persistent-connections-per-server," Wiki, Jan 2006. [Online]. Available: http://kb.mozillazine.org/Network.http.max-persistent-connections -per-server

[4] W. Stevens, *TCP/IP, Illustrated, Volume 3: TCP for Transactions, HTTP, NNTP and the UNIX Domain Protocols*, 1st ed. Massachusetts: Addison-Wesley, 1996, vol. III.

[5] T. Faber, J. Touch, and W. Yue, "The time-wait state in tcp and its effect on busy servers," in *Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3, INFOCOM '99. IEEE, March 1999, pp. 1573–1583.

[6] J. M. Almeida, V. Almeida, and D. J. Yates, "Measuring the behavior of a world-wide web server," in *High Performance Networking VII. IFIP TC6 Seventh International Conference on High Performance Networks (HPN'97).*, A. Tantawy, Ed. White Plains, NY, USA. 28 April-2 May 1997: Chapman & Hall, 1997, pp. 57–72.

[7] J. C. Mogul, "The case for persistent-connection http," in *SIGCOMM '95 Conference on Communications Archetectures and Protocols*, vol. 25. Cambridge, MA: ACM. Computer Communication Review, 1995, pp. 299–313, ACM SIGCOMM '95. Cambridge, MA, USA. 28 Aug.-1 Sept. 1995.

[8] J. Franks, "Proposal for an HTTP MGET method," IETF HTTP Internet Mailing List, Dec 1994. [Online]. Available: http://ftp.ics.uci.edu/pub/ietf /http/hypermail/1994q4/0260.html

[9] V. N. Padmanabhan and J. C. Mogul, "Improving HTTP latency," *Computer Networks and ISDN Systems*, vol. 28, no. 1-2, pp. 25–35, Dec 1995.

[10] S. Bin, "Outline of initial design of the structured hypertext transfer protocol," *Journal of Computer Science and Technology (English Language Edition)*, vol. 18, no. 3, pp. 287–98, May 2003.

[11] J. Palme, A. Hopman, and N. Shelness, *MIME Encapsulation of Aggregate Documents, such as HTML (MHTML), RFC 2557*. The Internet Society, March 1999.

[12] C. E. Wills, M. Mikhailov, and H. Shang, "N for the price of 1: bundling web objects for more efficient content delivery," in *World Wide Web*, 2001, pp. 257–265. [Online]. Available: http://citeseer.ist.psu.edu/article/wills00for.html

[13] C. E. Willis, G. Trott, and M. Mikhailov, "Using bundles for web content delivery," *Computer Networks*, vol. 42, no. 6, pp. 797 – 817, 21 Aug 2003.

[14] L. Masinter, *The "data" URL scheme, RFC 2397*. The Internet Society, 1998.

[15] J. C. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy, "Potential benefits of delta encoding and data compression for HTTP," in *SIGCOMM '97: Proceedings of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication*. New York, NY, USA: ACM Press, 1997, pp. 181–194.

[16] ——, "Errata for Potential benefits of delta encoding and data compression for HTTP," *SIGCOMM Comput. Commun. Rev.*, vol. 28, no. 1, pp. 51–55, 1998.

[17] J. Mogul, B. Krishnamurthy, F. douglis, A. Feldman, Y. Goland, A. van Hoff, and D. Hellerstein, *Delta Encoding in HTTP, RFC 3229*. The Internet Society, 2002.

[18] H. F. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. W. Lie, and C. Lilley, "Network performance effects of HTTP/1.1, CSS1, and PNG," in *ACM. Computer Communication Review, vol.27, no.4, Oct. 1997. USA.*, 1997, pp. 155–66.

[19] J. Heidemann, "Performance interactions between P-HTTP and TCP implementations," *ACM Computer Communications Review*, vol. 27, no. 2, pp. 65–73, April 1997. [Online]. Available: citeseer.ist.psu.edu/heidemann97performance.html