Towards an Automated Client-Side Framework for Evaluating HTTP/TCP Performance

Paul Davern, Noor Nashid, Ahmed Zahran, Cormac J Sreenan
Mobile and Internet Systems Laboratory,
Department of Computer Science
University College Cork
Cork, Ireland
p.davern@cs.ucc.ie
Phone: +353 21 420 5383 Fax: +353 21 420 5367

***Abstract*—Many different existing technologies and techniques have been developed to accelerate Hyper Text Transfer Protocol (HTTP) with the aim of improving the end-user's web browsing experience. However, there is no standard methodology for evaluating HTTP performance or for the evaluation of a given HTTP acceleration technology in a multi-user scenario. In this paper we present a framework HTTP-AE (HTTP Automated Evaluation) for measuring the end-user's web browsing quality of experience. We describe how HTTP-AE can be used to automate the evaluation of HTTP performance and for the evaluation of HTTP acceleration technologies in multi-user environments. We present three case studies in which we evaluate three different HTTP acceleration technologies for satellite systems using HTTP-AE. In our first two case studies we show that typical browser HTTP acceleration techniques operate sub-optimally on satellite systems. In our third case study we show that in multi-user scenarios, HTTP acceleration technologies, which break the end-to-end semantics of HTTP, can distribute network resources unfairly to the end-users.**

***Index Terms*—HTTP performance evaluation, HTTP acceleration, Testing Framework, High latency, TCP performance, TCP slow start, Satellite systems.**

I. Introduction

The performance of HTTP/TCP can be evaluated using different approaches, including analysis, simulation, emulation, and experimental testbeds. Heidemann [1] proposes an analytical model for web transfer over different protocols like persistent connections, transactional TCP and UDP. Typically, analytical models are limited to specific scenarios and can not fully accommodate all the dynamics of the protocol stack and testing environments. Simulation-based studies [2] [3] employ empirical models that are developed using network traffic to simulate HTTP behaviour. PackMime-HTTP [3] is a well-known HTTP traffic generator and is used in several simulation-based studies. However, such models and tools may not cope with the speed of evolution of web content and design. Hence, evaluating web browsing in real environments is expected to provide a more up-to-date and accurate result.

Real HTTP testing includes both emulated and real environments. In emulated environments, e.g. [4], [5] specific tools, such as Dummynet [6] are employed to simulate the behavior of network elements or an entire network between the HTTP client and server. This approach enables the testing of complex scenarios that involve large number of nodes or expensive devices without the complexity involved in real HTTP testing. Many studies consider tests for evaluating the performance of web browsing in real environments. For example, Chakravorty et al. [7], [8], [9] evaluate web browsing performance in GPRS using a commercial test bed under different scenarios in the presence of a network optimization proxy.

In [10] the authors present a common TCP testing framework, which allow researchers to quickly and easily evaluate their proposed TCP extensions. Similarly a generally accepted testing framework is necessary for HTTP-related technology. In this paper, our contribution is HTTP-Automated evaluation (HTTP-AE) as a framework to automate the evaluation of HTTP performance over a particular system under test. The framework does this by calculating the relative web browsing quality of experience for a set of end-users over the system under test.

Additionally, we present case studies in which HTTP-AE is used to evaluate HTTP acceleration techniques for satellite systems. Web browsers normally open multiple TCP connections to a Web server in order to retrieve content efficiently. We show in our first case study that this acceleration technique is inadequate for satellite systems. In our second case study, we show the performance benefits for web browsing when the TCP congestion algorithm is optimized for satellite systems. In our third case study we show the performance advantage of deploying a HTTP Performance Enhancing Proxy (PEP) into a satellite system. In our fourth case study, we show that in multi-user scenarios, HTTP acceleration technologies, which break the end-to-end semantics of HTTP, can distribute network resources unfairly to the end-users. In the fourth case study, we also show how HTTP-AE can be used to evaluate HTTP-based services, which provide user and service differentiation.

The rest of the paper is organized as follows. Section II is dedicated to background and related work on both HTTP Evaluation and acceleration. In Section III, we present the HTTP-AE framework followed by our case studies in Section IV. Finally, conclusions and future work are presented in Section V.

II. Background and Related Work

The purpose of this section is to provide background for the specific scenarios, which can be evaluated by our proposed tool. Figure 1 indicates that a diversity of HTTP/TCP acceleration techniques can be applied at the client, at the server or at intermediaries in the network itself. In section II-A we introduce such HTTP acceleration techniques. In section II-B we outline such TCP acceleration techniques, which improve the performance of HTTP. In section II-C we outline such HTTP/TCP acceleration techniques specific to satellite systems, which is the scenario associated with our case studies. In section II-D we outline the contribution of our proposed tool in relation to other such tools.

A. HTTP acceleration techniques

HTTP is a request/response protocol in which web clients send a GET request for a particular web page and web servers send back this page in hyper-text markup language (HTML) format. Nowadays, web-pages are very complex and usually contain many references for other nested content required to display the page to the end-user. This complexity was never envisaged when HTTP was designed and sometimes it deteriorates the end-user's browsing experience due to the sequential request-response operation of the protocol.

Several optimizations are proposed in the literature to accommodate the complexity of web-pages and improve the sequential operation of HTTP. Examples include:

1) Simultaneous download of multiple objects over multiple TCP connections and pipelining multiple GET requests are currently employed by many browsers.
2) HTTP-MPLEX [11] has been proposed, which combines a request compression and response multiplexing scheme to HTTP.
3) SPDY [12] allows multiple HTTP requests to be streamed over one TCP connection, it provides a mechanism for a server to push resources to the client, and it automatically compresses HTTP headers.

B. TCP acceleration techniques

In the transport layer, Transmission Control Protocol (TCP) also introduces several performance issues, which given the nature of HTTP, magnify the web page load latency. For example, each TCP connection requires a three-way handshake (SYN-SYN.ACK-ACK) for session establishment before sending any data over this connection. This behavior not only delays the data transmission but also introduces a data transmission overhead when opening multiple TCP connections. Additionally, TCP depends on a preventive congestion control mechanism by which a sender should conservatively increase its transmission window during the slow start phase. In many scenarios, this behavior leads to slowing down the TCP session and under-utilization of network resources. Numerous algorithms have been designed to maximize the data throughput while avoiding congestion during the TCP slow start phase. For example, TCP Hybla [13] was specifically designed for high latency links, where standard TCP confuses the delay for congestion and backs off transmission unnecessarily.

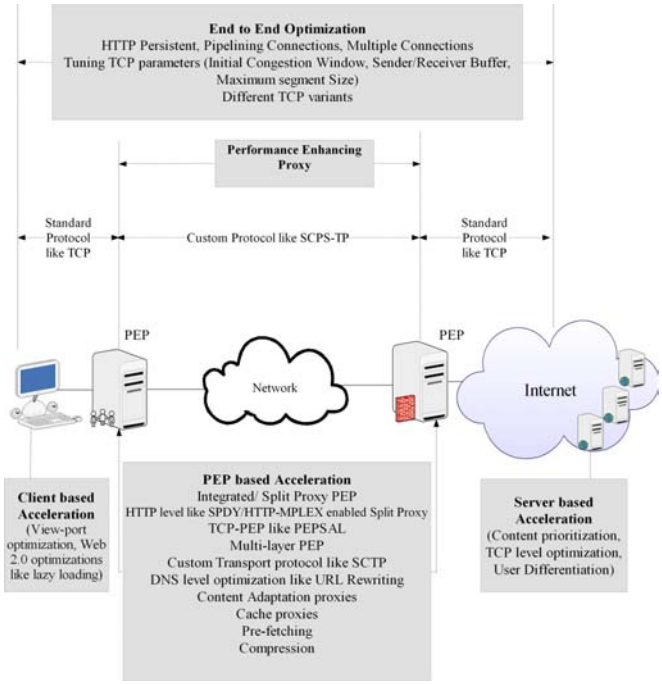

Figure 1. HTTP acceleration techniques

C. HTTP/TCP acceleration over satellite

The aforementioned issues with HTTP/TCP are greatly accentuated when it comes to high latency links such as broadband satellite systems. Broadband satellite is becoming more and more pervasive to deliver an Internet access facility to remote communities [14]. However, the performance of Internet protocols like HTTP and TCP are sub-optimal for satellite links where long delay, high bit error rate (BER) and limited resources are inherent in their nature. Hence, many different technologies for accelerating HTTP/TCP have been developed to improve end-user web browsing quality of experience.

Technologies/techniques such as cache proxies [15], pre-fetching of HTTP content, and persistent TCP connections are used to improve the end-user's web browsing experience over satellite. An HTTP Performance Enhancing Proxy (PEP) [16], can employ all three of these techniques. An HTTP PEP can be deployed as shown in Figure 1 in a high latency segment of a network between the end-users and the web server. TCP PEPs [17] can also be similarly deployed in a high latency link to locally acknowledge a TCP sender and thus cause the congestion window to grow artificially.

D. Contribution of HTTP-AE in relation to other tools

Currently, there exist few tools that evaluate the performance of HTTP from the perspective of end-user quality of experience. For example, Firebug [18] gives client side statistical information about web page load time. But Firebug does not provide a method of modifying the conditions of the test or automating a series of tests. YSlow [19] and Page Speed [20] test web application performance while JMeter [21] and httperf [22] were designed to test the performance of a web

Figure 2. Configure an Evaluation Object

application under traffic load. One supplies these tools with a set of URLs and HTTP commands (for example a GET request, or a POST request) to be executed on these URLs. These tools were not designed to evaluate the performance of HTTP/TCP between the client and server from the perspective of the end-user's quality of experience. Google's Chromium Benchmarking Extension [23] was designed to evaluate SPDY. Further functionality is required to compare and contrast two different HTTP acceleration technologies with the Chromium Benchmarking Extension tool. More importantly, all these tools only consider obtaining performance metrics for a single user.

In the following section, we present HTTP-AE as a multi-user web browsing client-based framework for automating the performance evaluation of HTTP/TCP. The multi-user aspect of the framework, represents a means for determining, for example, the relative HTTP performance for each end-user, in combination with other users and a particular HTTP acceleration technology. Further, by operating at the client side, HTTP-AE allows HTTP acceleration technologies to be evaluated independently and concurrently. For example, HTTP-AE can evaluate the effect on end-user quality of experience when a SPDY proxy alone or the proxy in combination with a TCP PEP has been deployed into the network [24]. Thus, HTTP-AE can be used in a generic manner to evaluate the performance of any type of system that supports HTTP or optimizes HTTP.

III. HTTP-AE Framework

HTTP-AE is a multi-user web browsing client-based framework for automating the performance evaluation of HTTP/TCP. Figure 2 shows a view of an *Evaluation* object namely *Evaluation of a HTTP PEP*, which was created using the HTTP-AE front end. An *Evaluation* object contains a set of configuration parameters, a set of *user agents* and a set of *scenarios*. We used this *Evaluation* object in sections IV-D and IV-E to evaluate a HTTP PEP. The HTTP-AE user

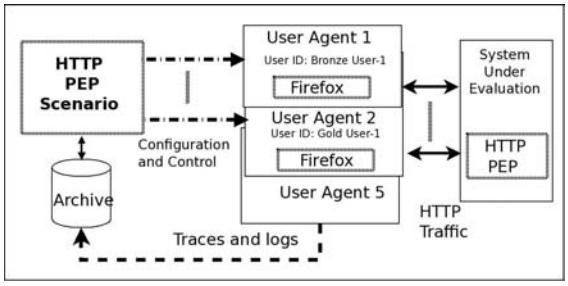

Figure 3. HTTP PEP scenario logical view

interface has three main options, which are associated with the *Evaluation* object: Configure, Run, and Reporting. In section III-A, we describe how an *Evaluation* object is defined and in section III-B we describe the main performance metrics.

A. Defining an Evaluation in HTTP-AE

To perform an evaluation of a particular HTTP acceleration technology the evaluator initially creates an *Evaluation* object using the HTTP-AE front end GUI. The evaluator can configure various *Evaluation* object attributes e.g. clear the browser cache between runs. A number of *User Agent* objects are then created, which are associated with the *Evaluation* object. Each *User Agent* controls a browser to request a set of URLs in sequence. The *User Agent* archives performance metrics such as page load latency for each of the URLs in this set. These metrics can be compared and contrasted using HTTP-AE's reporting facility. The evaluator can configure various attributes for each *User Agent* for example the number of times a particular URL should be visited. The page load time for a particular URL is typically dependent on many factors which may be outside the control or interest of the evaluator, for example the page load is dependent on web server load and on network usage. Therefore, HTTP-AE allows the evaluator to access a particular URL a number of times to reach defined targets in terms of the confidence interval of the results. Since a *User Agent* is a logical entity, the evaluator should provide the user's physical machine, associated interface, and login credentials on this machine. Note, that if more than one user is sharing the same interface on the same machine, source network address translation (NAT) can be used to distinguish the traffic associated with each user. This is attained by using an *iptables* [25] rule, to associate each user's traffic, identified by its user ID, and destined to port 80 to a separate IP using the SNAT target option. This *iptables* rule can be used where a set of users share the same machine but are associated with different interfaces.

The evaluator creates a set of *Scenario* objects, which are associated with the *Evaluation* object. These *Scenario* objects control the conditions of a particular evaluation. For example, a Linux script can be associated with the Scenario object, which distinguishes the user traffic based on port number. Figure 3 depicts a logical view of one of these scenarios namely the *HTTP PEP* scenario. In this case, the system under evaluation is a HTTP PEP. When this scenario is executed, each associated *User Agent* generates a set of results including

TCP level traces which are archived in association with the *User Agent*. HTTP-AE has a reporting facility which can be used to compare and contrast the results between scenarios. For example, the *HTTP PEP Scenario* in figure 2, refers to a network with a deployed HTTP PEP and the *Firefox alone Scenario* refers to the same system without the HTTP PEP.

B. HTTP-AE Metrics

Generally, the web browsing Quality of Experience (QoE) is application dependent. Additionally, it depends on the system under evaluation. However, there exist typical objective metrics that would enable any evaluator to evaluate and compare the performance of different system designs. In HTTP-AE, we consider different latency components including

- Base page latency, which corresponds to the delay between issuing the first HTTP GET request to the time at which the HTTP response is received. Typically, the user starts to see textual content at this moment and the client starts parsing the page for fetching embedded resources to complete the page download. These delay components represent a good indicator for different network functionality. For example, it gives an indication to the efficiency of reactive routing protocols in AdHoc networks. In satellite systems, many PEPs introduce optimization techniques to reduce the delay resulting from the sequential operation of TCP and HTTP. Hence, base page latency can differentiate between different system designs.
- View-port loading delay, which refers the time to load all the objects in the user's current viewport. Typically, prioritizing the content of the user view port would improve the user QoE. One way to prioritize the viewport is to embed a java-script in the page to determine the objects within the user viewport on the client side. Note that such script may not only be embedded by the server but also by agents or proxies in the system.
- Total page load latency, which corresponds to the total time required to download the base page and all of its embedded objects starting from issuing the request in the browser.

These delay components are logged based on the interaction between the browser and the user agent. For example, we control Firefox using Mozrepl [26] and employ the latter to obtain and log different latency-based metrics. Clearly, these metrics can be affected by any change in the different system components including the configuration and design of web client, web server, and intermediate network. In addition to different latency metrics, HTTP-AE automatically monitors the network interfaces associated with each user and logs different traces using *tcpdump*. These traces are analyzed using *capinfos*, a terminal-based tool that is part of *Wireshark* [27]. *Capinfos* enable the obtaining of different statistics from the collected traces including the user bandwidth, the traffic volume crossing the interface, the data sent and received (bytes and packets).

HTTP-AE considers the user bandwidth share as an important metric that identifies the system capability for differentiating different user classes and services. For example, end-users with different subscription plans on a mobile network could be allocated with different levels of QoS including differentiation of bandwidth. This metric can be used to evaluate the effect of service and user differentiation on the end-users. Similarly, the user bandwidth share can be used to determine the fair distribution of resources among peer users.

To this end, it is worth noting that the proposed framework is applicable not only to end-user clients machines but also can be used in the middle of the network to evaluate the performance of intermediate nodes such as PEPs. In the latter case, other relevant network performance metrics such as signaling load can be obtained by filtering the collected traces. In the following section, we present several case studies in which we employ HTTP-AE to evaluate web browsing performance in satellite based networks.

IV. Case Studies

In the case studies we present here we use HTTP-AE to evaluate several HTTP acceleration techniques in a high latency network such as a Satellite system. Each of the case studies is independent. The rest of this section is organized as follows. Section IV-A describes the emulated satellite test bed we used for the case studies. In section IV-B, we evaluate the effectiveness of using multiple connections to retrieve HTTP content over satellite. In section IV-C, we study the effect of using different TCP slow start congestion control algorithms on HTTP performance. In section IV-D, we present an evaluation of our HTTP PEP. In section IV-E, we present a modification to our HTTP PEP to support user differentiation and we evaluate its performance with multiple web browsing users.

A. Evaluation Test Bed

Figure 4 shows the emulated satellite test bed we used for the case studies. The test bed consists of two PCs with Intel Core 2 Duo CPU E4700 2.60 GHz, running Fedora 10. The two PCs, one emulating the remote node and the other emulating the ground node, are connected using a direct 1Gbps Ethernet connection over which a Satellite link behavior is emulated using *tc* commands in Linux. That is, the data rate is limited to 64Kbps, 128Kbps and 256Kbps and a 300ms one way delay is introduced in each direction. A client PC was connected to the remote node running HTTP-AE.

For the results that are shown, the URLs point to a set of pages that contain the same text and a number of embedded images from Caltech 101 image dataset [28]. The number of images in the set of pages varies from 1 to 30. The images are all JPEGs and have an average size of 10K. These pages are served from a web-server co-located in the ground node. In the results, each point corresponds to the average of 10 runs; i.e., each page is browsed ten times and the performance metrics is the average of these ten experiments to reduce the impact of random timing behaviour of network and server access. All results were determined to fall within a confidence interval of 95%.

Figure 4 also shows the experimental configuration for user differentiation. Each User Agent is associated with a Linux

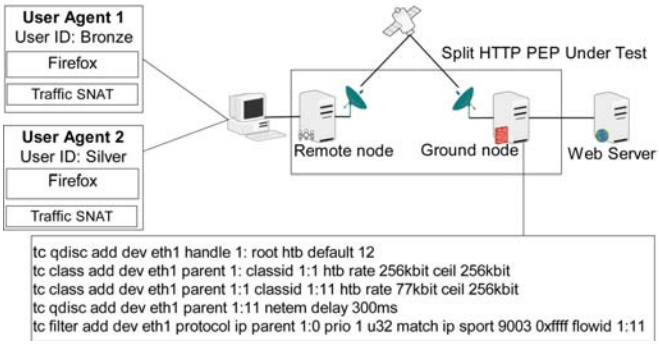

Figure 4. Test bed

user. Each emulated user's traffic was source NATed so that the traffic for a given user was assigned to a separate IP address. User differentiation at the ground node was achieved using the commands (for the silver user) as detailed in the figure. Gold users are assigned 50% of the bandwidth, silver 30% and bronze 20%.

B. Multiple TCP connections

During TCP slow start the amount of unacknowledged data is limited by the sender's congestion window. Thus, the bandwidth is not being utilized for data transmission while the sender is waiting for acknowledgments. To overcome this limitation Web browsers typically open up multiple connections to a Web server.

There are quality of service issues associated with this HTTP acceleration technique when it comes to satellite systems:

- A Web browser that opens multiple TCP connections, requires more processing resources on the ground and remote gateway nodes than the same browser that opens say one connection. This problem becomes more exacerbated when several users are browsing at the same time.
- Each connection that a browser opens requires 7 additional packets to establish the session and to tear it down. Thus, there is a non negligible data transmission overhead, associated with connection setup and tear down, for several users each opening multiple connections. Also TCP will combine data more readily into less packets, when an application uses say one connection as against several.
- There are less packets transmitted, if the browser were to pipeline its requests, as against the case where it uses multiple connections to achieve the same result. For example, if the browser were to open 3 TCP connections to retrieve 3 Web resources then, 3 packets are required to issue the GET requests. If these 3 GET requests were to be pipelined then they could be combined into 1 packet. Also, the 3 responses arrive in one continuous stream, thus TCP is able to combine packets at the sender more readily.
- The delay in opening a TCP connection is 1.5 Round Trip Times (RTT) - here we assume that there is no data in the ACKs. The browser opens its first connection to the Web server to retrieve the base HTML. If the base page has a small number of nested resources, then, it may be more efficient for the browser, to use the connection it has already opened to retrieve those resources, rather than opening more connections to achieve the same result. If the browser were to pipeline all the requests, for the nested resources over the first connection then, more efficiency could be achieved. However, Firefox does not pipeline the initial GET requests for nested resources and only pipelines its requests gradually. This may be Firefox's mechanism to determine whether or not the web server supports pipelining [4].

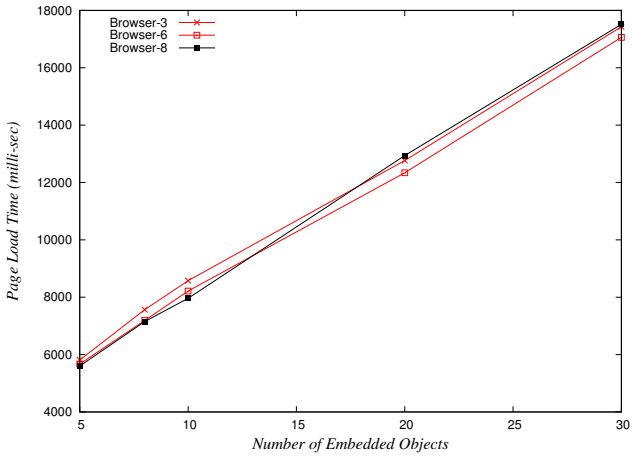

Figure 5. Firefox using 3 to 8 TCP connections for 256Kbit link

In this study we evaluate the Firefox's use of multiple connections from the perspective of end-user QoS and data transmission overhead for the emulated satellite system.

Figure 5 shows that there is little difference between page load time for 3 to 8 TCP connections for a 256Kbit link, while the data transmission overhead is significant - see figure 6. This data transmission overhead is due to the extra signaling load and increased packet fragmentation. Figure 7 shows that there is no advantage in opening more pipelined connections in a narrow link. Firefox with one connection shows a significant increase in page load delay when compared with 3-8 connections. However, Firefox does not pipeline the first request for an embedded object and so there is a significant delay before it requests the second and subsequent objects. Figure 7 shows that for up to 10 images there is no difference in terms of page load delay when the browser opens up multiple connections over a 256Kbit link. In fact, for small number of images, it is more efficient in terms of page load delay to open only one connection. Figures 8,9 show that as the bandwidth increases, the browser's use of multiple connections gives an improvement in the page load performance. The improvement becomes less noticeable as the number of objects on the page increases.

Figure 10 shows, that as expected there is little difference in the bandwidth share given to a number of users all browsing with Firefox.

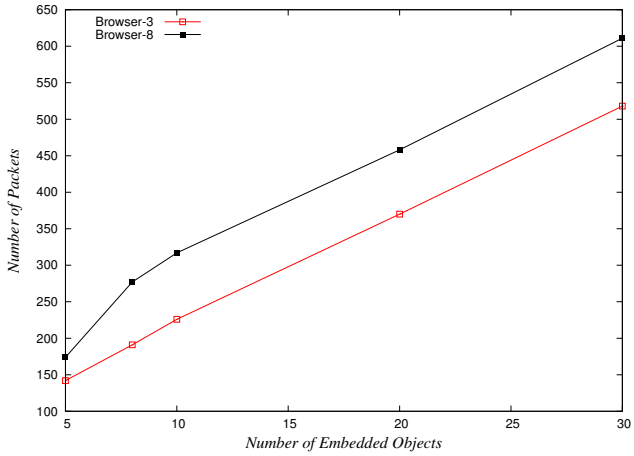

Figure 6. Packet overhead 3 versus 8 connections

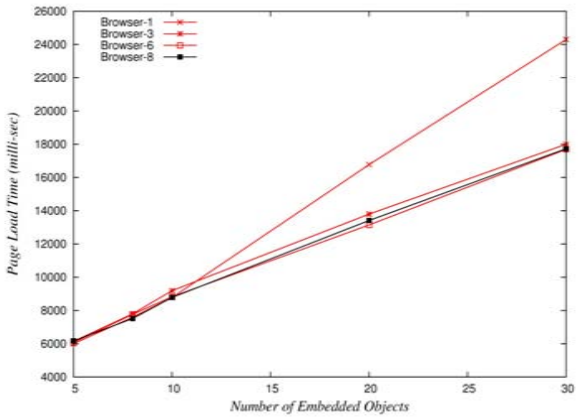

Figure 7. Pipelining 1 to 8 connections 256Kbit

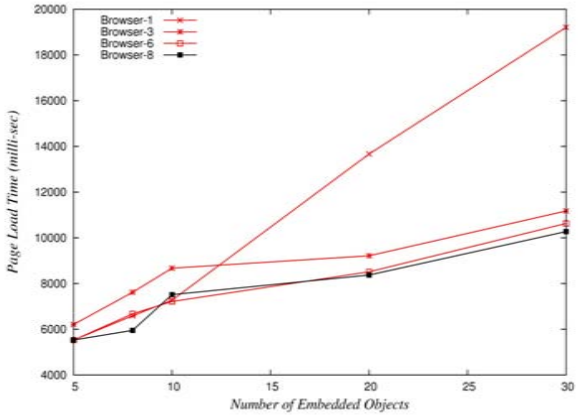

Figure 8. Pipelining 1 to 8 connections 512Kbit

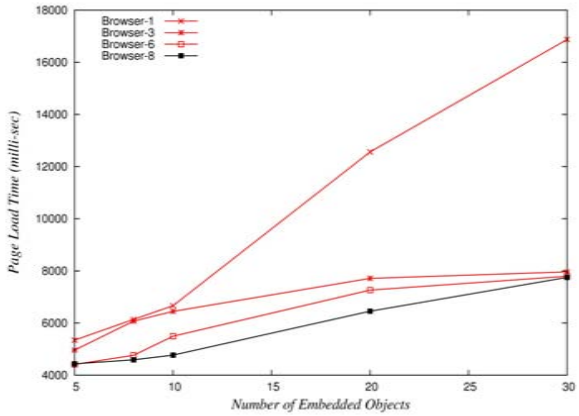

Figure 9. Pipelining 1 to 8 connections 1Mbit

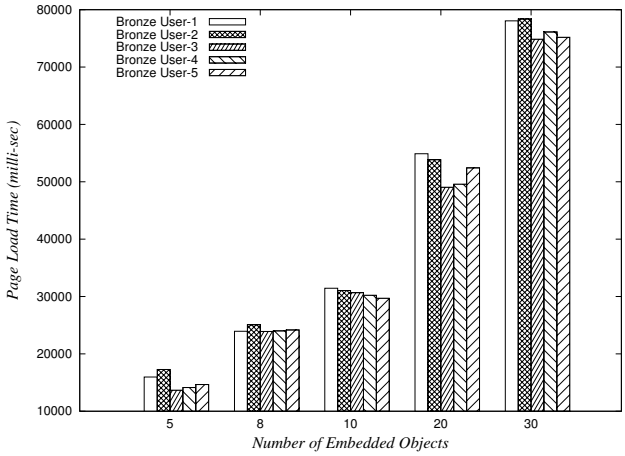

Figure 10. Multi-user default Firefox configuration

C. TCP variants

One server side optimization for HTTP, is to increase the initial TCP congestion window, which allows the server to push more content initially during TCP slow start. However it is difficult to determine an appropriate value for the initial congestion window independent of the congestion algorithm utilized. Various TCP variants have been developed to address TCP slow start issues. The use of such standards at the server, gives a more generic solution than manipulating the initial congestion window directly.

In this study, the web browsing performance is evaluated when different TCP variants Hypla, BIC, Vegas and Westwood are used at the Ground node. Each of the variants uses a different congestion control algorithm during slow start. Figure 11 shows that Hypla gives the best performance for low bandwidth links while Vegas gives the worst performance of the four TCP variants that we evaluated. TCP Vegas is greatly impeded by the long RTT - see [29] for more details. The number of nested resources does not appear to suit one congestion algorithm over another.

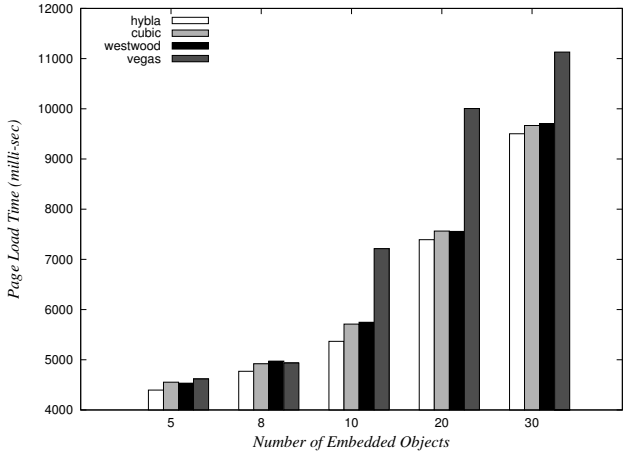

Figure 11. Performance of TCP variants

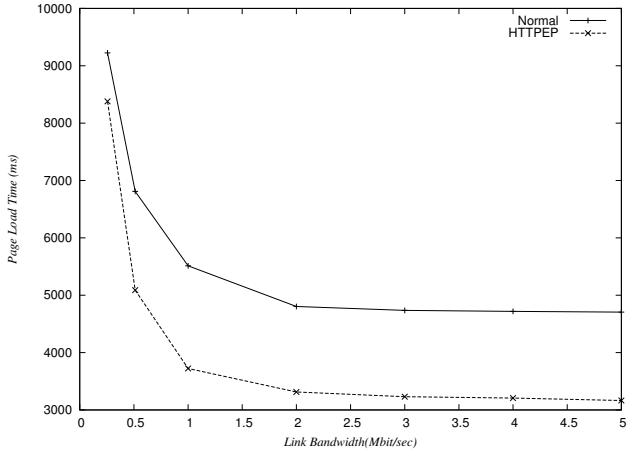

Figure 13. Effect of varying the satellite link capacity

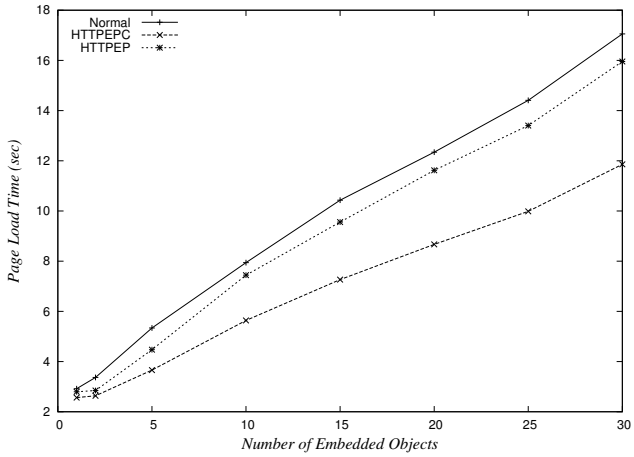

Figure 12. Average page load delay

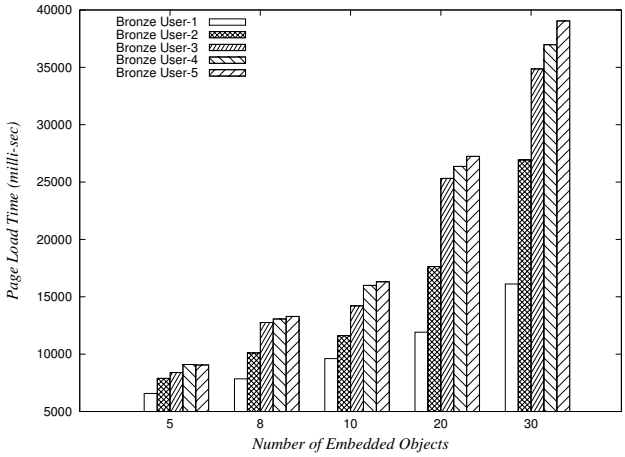

Figure 14. Multi-user over HTTPEP with no user differentiation

D. HTTP PEP

In our previous paper [30], we presented an HTTP PEP (HTTPEP), which improves the user's web browsing experience over a high-latency link. HTTPEP has a HTTP split proxy architecture between the ground and the remote sites.

HTTPEP transforms a GET request for a base web page from the end-user so that the nested web resources in that base page are streamed to the remote site. The resources are streamed into multiple TCP persistent connections from the ground to the remote sides. A scheduler on the ground side chooses a particular TCP connection in a round-robin fashion. Thus the resources will be transmitted to the remote site concurrently.

Figure 12 shows an average page load delay reduction of 27% using compression (HTTPEPC) in comparison to an average saving of 10% without compression over the normal operation (i.e. a direct connection between the browser and the web-server without an intermediate proxy).

Figure 13 plots the page load time for HTTPEP and normal operation versus the satellite link capacity. When the bandwidth is constrained to 256Kbps the average improvement of HTTPEP over the browser alone is 10%, as the bandwidth increases so does the average improvement, which levels at 32% when the link capacity reaches 2Mbps.

E. User differentiation

Figure 14 shows the page load time for a number of users with no user differentiation using HTTPEP. In this case there are eight TCP connections maintained between the remote and ground nodes. The figure clearly shows that the first user on the system is getting better service than subsequent users. To provide a solution to this issue, we need to re-design the algorithm that HTTPEP uses to service incoming requests, so that network resources are evenly distributed to the associated response handling. As the number of images increases so the effect becomes more pronounced.

We modified HTTPEP's scheduler so that traffic destined for a particular user class was scheduled to an appropriate TCP connection. In this case, we have 3 permanent TCP connections between the ground and remote sides and 50% of the bandwidth allocated to connection 1, 30% to connection 2, and 20% to connection 3. Figure 15 shows the page load time experienced by the different classes of users (gold, silver, bronze) on the system. The figure shows that the first bronze user on the system is getting better service than the other bronze users, while the gold and silver users are getting the

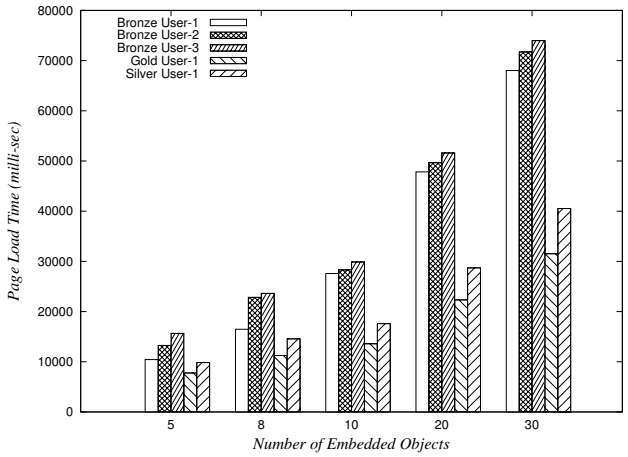

Figure 15. Multi-user over HTTPEP with user differentiation

appropriate service differentiation. However, all the bronze users should be getting an equal service. Thus, the future work in re-designing HTTPEP will have to take this issue into account.

V. Conclusion

A framework (HTTP-AE) for the evaluation of HTTP performance is introduced in this paper. We explain how HTTP-AE can be used to evaluate the performance of HTTP and HTTP acceleration technologies in multi-user systems. We present case studies in which HTTP-AE is used to evaluate three HTTP acceleration technologies deployed in a emulated satellite system. We show that the performance of typical web browsers can be further enhanced to adapt to the network conditions of satellite systems. We also show that HTTP acceleration technologies, which break the end to end dynamics of HTTP can provision network resources to some users to the detriment of others. The evaluation of our HTTP PEP showed that future work is required, to spread network resources evenly over the servicing of HTTP GET requests.

Acknowledgments: This work is supported by Enterprise Ireland Grant Number IP/2009/0026 and by Altobridge. The authors thank the anonymous reviewers for their comments and suggested changes that helped to improve the paper.

References

[1] J. Heidemann, K. Obraczka, and J. Touch, "Modeling the performance of HTTP over several transport protocols," *IEEE/ACM Trans. Netw.*, vol. 5, pp. 616–630, October 1997.

[2] B. A. Mah, "An Empirical Model of HTTP Network Traffic," *IEEE Computer and Communications Societies, Annual Joint Conference of the*, vol. 0, p. 592, 1997.

[3] J. Cao, W. Cleveland, Y. Gao, K. Jeffay, F. Smith, and M. Weigle, "Stochastic models for generating synthetic HTTP source traffic," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 3, 2004, pp. 1546 –1557.

[4] P. Natarajan, P. D. Amer, and R. Stewart, "Multistreamed web transport for developing regions," *NSDR '08: Proceedings of the second ACM SIGCOMM workshop on Networked systems for developing regions*, pp. 43–48, 2008.

[5] M. C. Necker, M. Scharf, and A. Weber, "Performance of Different Proxy Concepts in UMTS Networks," vol. 3427, pp. 36–51, 2005.

[6] "Dummynet," www.iet.unipi.it/ luigi/dummynet/.

[7] R. Chakravorty, A. Clark, and I. Pratt, "Optimizing Web delivery over wireless links: design, implementation, and experiences," *Selected Areas in Communications, IEEE Journal on*, vol. 23, no. 2, pp. 402 – 416, 2005.

[8] R. Chakravorty, S. Banerjee, P. Rodriguez, J. Chesterfield, and I. Pratt, "Performance optimizations for wireless wide-area networks: comparative study and experimental evaluation," *Proceedings of the 10th annual international conference on Mobile computing and networking*, pp. 159–173, 2004.

[9] R. Chakravorty, J. Chesterfield, P. Rodriguez, and S. Banerjee, "Measurement Approaches to Evaluate Performance Optimizations for Wide-Area Wireless Networks," *Passive and Active Network Measurement*, pp. 257–266, 2004.

[10] L. L. H. Andrew, C. Marcondes, S. Floyd, L. Dunn, R. Guillier, W. Gang, L. Eggert, S. Ha, and I. Rhee, "Towards a common TCP evaluation suite," 2008. [Online]. Available: http://hdl.handle.net/1959.3/44431

[11] "HTTP-MPLEX: An enhanced hypertext transfer protocol and its performance evaluation," *Journal of Network and Computer Applications*, vol. 32, no. 4, pp. 925 – 939, 2009.

[12] "Spdy," 2010, https://sites.google.com/a/chromium.org/dev/spdy.

[13] Carlo and R. Firrincieli, "TCP Hybla: a TCP enhancement for heterogeneous networks," *International Journal of Satellite Communications and Networking*, vol. 22, 2004.

[14] Altobridge, http://www.altobridge.com/.

[15] Squid, http://www.squid-cache.org.

[16] "Mguard," http://www.broadband-internet-access.com/.

[17] C. Caini, R. Firrincieli, and D. Lacamera, "PEPsal: A Performance Enhancing Proxy for TCP Satellite Connections ," *Aerospace and Electronic Systems Magazine, IEEE*, vol. 22, no. 8, pp. B–9 –B–16, 2007.

[18] "Firebug," http://getfirebug.com/.

[19] Yahoo, "Yslow," http://developer.yahoo.com/yslow/.

[20] "Page speed," 2010, http://code.google.com/speed/page-speed/.

[21] Apache, "Jmeter," http://jakarta.apache.org/jmeter/.

[22] HP, "httperf," http://www.hpl.hp.com/research/linux/httperf/.

[23] "Chromium benchmarking extension," https://sites.google.com/a/chromium.org/dev/chrome-benchmarking-extension.

[24] N. Dukkipati, T. Refice, Y. Cheng, J. Chu, T. Herbert, A. Agarwal, A. Jain, and N. Sutin, "An argument for increasing TCP's initial congestion window," *SIGCOMM Comput. Commun. Rev.*, vol. 40, pp. 26–33, June 2010.

[25] "iptables," http://www.frozentux.net/documents/iptables-tutorial/.

[26] "mozrepl," https://github.com/bard/mozrepl/wiki.

[27] C. Schroder, *Linux Networking Cookbook*. O'Reilly Media, Inc., 2007.

[28] C. C. V. Group, "Images," 2010, http://www.vision.caltech.edu/html-files/archive.html.

[29] C. Caini, R. Firrincieli, D. Lacamera, T. de Cola, M. Marchese, C. Marcondes, M. Y. Sanadidi, and M. Gerla, "Analysis of TCP live experiments on a real GEO satellite testbed," *Perform. Eval.*, vol. 66, pp. 287–300, June 2009.

[30] P. Davern, N. Nashid, A. Zahran, and C. J. Sreenan, "HTTP Acceleration over High Latency Links," in *New Technologies, Mobility and Security (NTMS), 2011 4th IFIP International Conference on*, 2011, pp. 1 –5.