Client-Side Framework for Automated Evaluation of Mechanisms to Improve HTTP Performance

Paul Davern, Noor Nashid, Cormac J Sreenan,
Mobile and Internet Systems Laboratory, Department of Computer Science,
University College Cork, Cork, Ireland.
p.davern@cs.ucc.ie
Ahmed Zahran,
Department of Electronics and Electrical Communications,
Cairo University, Cairo, Egypt.
azahran@eecu.cu.edu.eg

***Abstract*—The proliferation of sophisticated web technologies requires efficient tools to evaluate the performance of HTTP traffic under various conditions. In this paper, we present HTTP-Automated Evaluation (HTTP-AE) as a *multi-user* client-side framework for evaluating HTTP performance. The framework can be used to evaluate mechanisms, which improve HTTP performance. We present several case studies in which HTTP-AE is used to evaluate three HTTP acceleration mechanisms deployed in an emulated satellite system. These case studies show that the framework can be used to test different design aspects that may affect HTTP performance. Hence, by using the proposed framework, one can determine the advantages and limitations of different network design configurations.**

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

I. Introduction

The complexity and richness of web content is driving researchers to develop more and more sophisticated techniques to optimize HTTP. A browser such as Firefox uses a diversity of such techniques to deliver web content more efficiently to the end-user. The development of such a HTTP optimization technique goes hand in hand with its evaluation in the real world. In this paper, we propose that there is a need in the research community for a generic real-world HTTP evaluation framework. There are many partial solutions available, but most researchers, spend much time developing their own HTTP evaluation tools to suit their own needs.

Web page load time is a primary metric for any such client-side evaluation of HTTP. However, web page load time is a metric, which is dependent on many factors e.g. the complexity of web content. But, given a controlled testing environment, for example, where the network conditions are fixed, and where a known web session is executed then the performance of HTTP will be directly related to web page load time. In this situation a mechanism to improve HTTP performance can be evaluated with respect to a reference evaluation. For example a browser with pipelining enabled can be evaluated with respect to the same browser without pipelining. In our previous paper [1] we presented HTTP-AE as a framework to automate the evaluation of HTTP performance of a particular HTTP acceleration technology. In this paper we present a re-design of HTTP-AE, which makes it more generic and enables the evaluation of mechanisms, which improve HTTP performance in a multi-user environment.

The optimization of HTTP is specifically focused around HTTP issues. For example, HTTP features a sequential operation that delays the retrieval time of embedded web *resources*[1]. When a client on the end-user's machine issues a HTTP GET request for a particular web page, the web server replies with the *base* HTML page containing references for other *nested* resources required by the client to display the page to the end-user. These resources are requested through additional HTTP GET requests over possibly new TCP connections opened by the client to the web server.

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 mechanisms for accelerating HTTP/TCP have been developed to improve end-user web browsing quality of experience for satellite systems. We present case studies in which HTTP-AE is used to evaluate HTTP acceleration mechanisms 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

[1]In this context, a web resource refers to for example an icon, an image, a CSS page, or other similar content.

doi:10.4304/jnw.7.11.1749-1759

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. In Section III, we present the HTTP-AE framework and particular use cases of the framework. In Section IV we present background to our case studies and the satellite emulation test bed. In Section V we present the case studies of using the framework. Finally, conclusions are presented in Section VI.

II. Background and related work

In the literature, the performance of HTTP has been evaluated using different approaches including analysis, simulation, emulation, and experimental testbeds. Heidemann [2] 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 [3] [4] employ empirical models that are developed using network traffic to simulate HTTP behaviour. For example, PackMime-HTTP [4] is a well-known HTTP traffic generator and is used in several simulation-based studies. However, one difficultly with such models and tools is that they may not be able to cope with the speed of evolution of web content and design. Thus, evaluating web browsing in real-world environments is expected to provide more accurate results.

Real HTTP testing includes both emulated and real environments. In emulated environments, e.g. [5] [6] specific tools, such as Dummynet [7], IKR Emulation Library [8] 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 real tests for evaluating the performance of web browsing in real environments. For example, Chakravorty *et al.* [9], [10], [11] evaluate web browsing performance in GPRS using a commercial test bed under different scenarios in the presence of a network optimization proxy.

In [12] 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. This work proposes 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 evaluation.

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

Currently, there exist several techniques and tools that can evaluate the performance of HTTP from the perspective of end-user quality of experience:

- For example, Firebug [13] gives client-side statistical information about web page load time. It is basically a JavaScript debugger for Firefox and other browsers [14]. It allows users to perform standard debugging actions. It also provides a mechanism to analyse the CSS and DOM trees of a particular web site [15]. But there is no built-in way to log this information. At the same time, it does not provide a method of modifying the conditions of the test or automating a series of tests.
- YSlow [16] and Page Speed [17] test web application performance. By employing good web design techniques the performance of end-users web browsing can be improved significantly. Tools like YSlow, perform comparisons of a particular web page against a predefined set of such good design rules, and then suggest different ways to optimize the web page design.
- Selenium is a test tool that allows a web page tester to write automated user interface tests for web applications [18]. It enables the testing of a particular web application with different browsers. Thus, Selenium is mainly a functional and acceptance testing tool for web applications.
- JMeter [19] and httperf [20] were designed to perform load testing on a web server. These tools can simulate the behaviour of multiple users from a single machine. 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 do not mimic the complete behaviour of a web browser. 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.
- Philipp *et al.* [21] have presented an automated web navigation tool for Internet Explorer. This tool takes a URL list from an input file and automatically browses through these web pages. This tool does not provide a method for modifying network conditions and gathering performance metrics for varying conditions.
- Ramakrishnan *et al.* [22] have proposed a technique for assessing the end-user's web browsing quality of experience by performing web page instrumentation. This approach leverages scripting and event notification mechanisms in HTML and uses scripting languages to measure and collect client-perceived response times. The code is attached to a web page which after being downloaded to the web browser, measures the download time of individual embedded

objects and sends back page load time to the web server. Cherkasova *et al.* [23] give details of other such tools. This browser-instrumentation technique is attractive in that it accurately measures client-perceived response time to gather detailed response times for all objects embedded within a web page. At the same time, it can log such user interaction. Unfortunately, this approach needs instrumentation at both client and server side.

- A completely different approach is undertaken by [24] [25] [23] and [26] to find the client-perceived page load time. Jianbin *et al.* [24] have presented a tool which can monitor the client-perceived end-to-end response time for secure web services. This tool resides near the monitored server and collects traffic passively in and out of that server. Then it applies a size-based analysis method on HTTP requests to infer the client-perceived page-view response time. Thus, these techniques can be beneficial to measure end-to-end performance of web services from the server side.
- Google's Chromium Benchmarking Extension [27] was designed to evaluate SPDY. This tool is only applicable to Google's Chrome. Further functionality, such as the ability to archive results, is required to compare and contrast two different HTTP acceleration technologies with the Chromium Benchmarking Extension tool.

HTTP-AE is a multi-user client-based framework for automating the performance evaluation of HTTP/TCP. During a web browsing session an end-user will interact with a set of diverse web applications each of which is addressed by a URL. Thus, a client in HTTP-AE is defined according to a set of URLs that are to be requested. HTTP-AE issues a HTTP GET request for a particular *base* HTML page of a given URL. The *base* page is parsed and any *nested* resources are subsequently requested.

The multi-user aspect of the framework, represents a means for determining, for example, the relative HTTP performance of each end-user, in combination with other users and a particular HTTP acceleration technology. Further, by operating at the client side, HTTP-AE allows mechanisms which improve HTTP performance to be evaluated independently and concurrently. For example, HTTP-AE can evaluate the effect on end-user quality of experience when a HTTP Performance Enhancing Proxy (HTTP PEP) alone or this HTTP-PEP in combination with a TCP-PEP has been deployed into the network [28].

III. Use cases of the HTTP-AE Framework

Figure 1, shows the three main classes of mechanisms for improving HTTP performance, that can be evaluated with HTTP-AE. That is mechanisms that are positioned at the client such as browser specific HTTP optimizations; mechanisms that are positioned in the intermediaries of the network for example caching mechanisms that

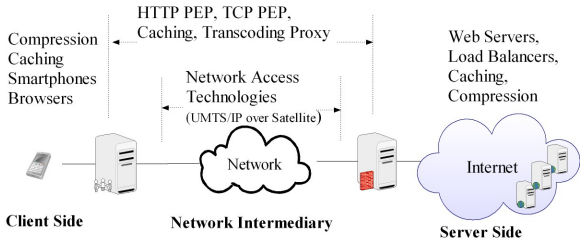

Fig. 1. mechanisms for improving HTTP performance

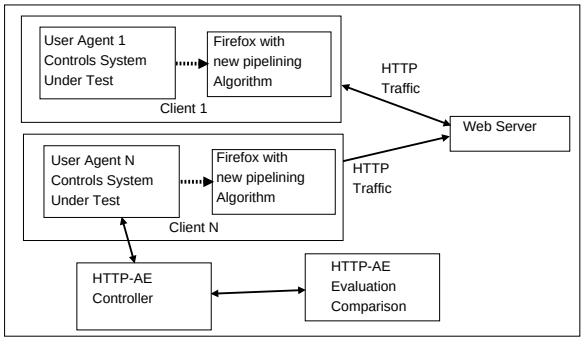

Fig. 2. Browser with new pipelining algorithm

improve HTTP performance over network access technologies such as satellite systems; and mechanisms that are positioned at the server such as load balancers.

A. Using the HTTP-AE Framework

Let us suppose that a researcher wishes to evaluate a new pipelining algorithm for Firefox with HTTP-AE. In this case, the evaluation with HTTP-AE works by comparing/contrasting the new pipelining algorithm with respect to a reference evaluation. For example, the evaluator may evaluate the performance of a known session of web browsing when the pipelining algorithm is in place with respect to the corresponding browsing session when the algorithm is not in place. Figure 2, gives a logical view of the evaluation of the new pipelining algorithm for Firefox. In this case, the evaluator configures a set of *Clients* on the network, where each *Client* consists of the *User Agent* HTTP-AE component and Firefox. Each of these *Clients* will simulate the browsing action of a real end-user on the network. To achieve this the *User Agent* requests a set of URLs in sequence using Firefox's XPCOM [29], a cross platform component object model which can be controlled using JavaScript.

Since a *Client* is a logical entity, the evaluator should provide the user's physical machine, associated interface, and login credentials on this machine using the HTTP-AE front end. The *HTTP-AE Controller* communicates with each of the *User Agents* in the system to collect statistics such as the page load time. In this case, the page load time is also measured using XPCOM. Note, that if more than one *Client* is to share the same interface on the same machine, source network address translation (NAT) can be used to distinguish the traffic associated with each *Client*.

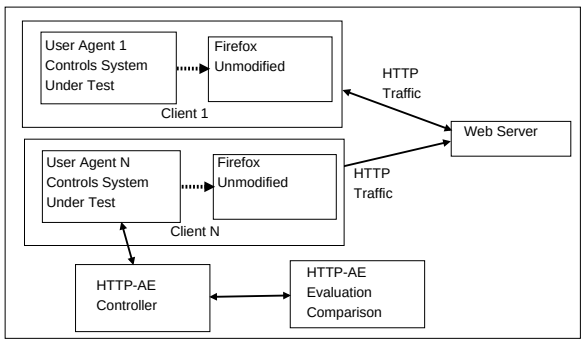

Fig. 3. Browser without new pipelining algorithm

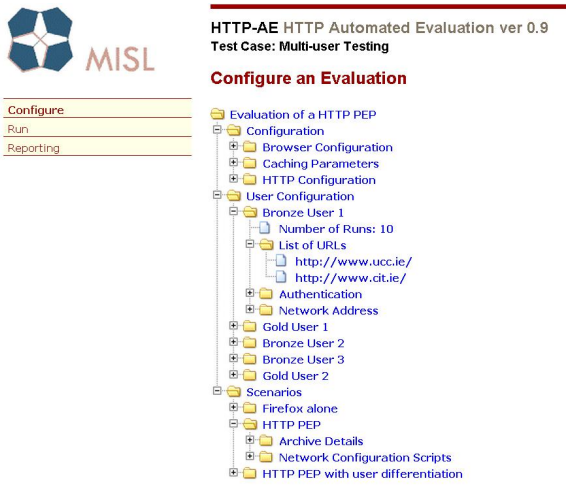

Fig. 4. Configure an Evaluation Object

This is attained by using an *iptables* [30] rule, to associate each simulated end 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.

Figure 3, gives a logical view of the evaluation of the browser alone i.e. without the new pipelining algorithm. The results from this reference evaluation can be contrasted, using *Evaluation Comparison* component of HTTP-AE, with the corresponding results collected when the pipelining algorithm is in place.

B. Defining an Evaluation in HTTP-AE

Figure 4, shows a HTTP-AE front end 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 V-C and V-D to evaluate a HTTP PEP.

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

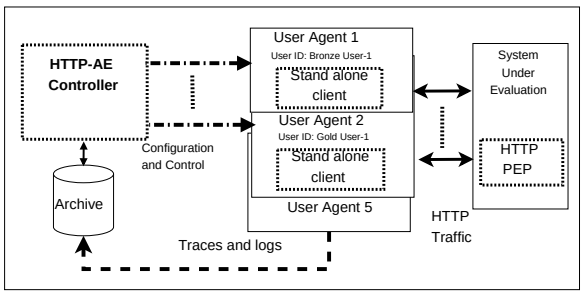

Fig. 5. HTTP PEP scenario logical view

To perform an evaluation of a particular mechanism which improves HTTP performance 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. An *Evaluation* has a set of *Scenarios* and a set of *User Agents* associated with it.

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 4 shows three *Scenarios* namely *Firefox alone*, *HTTP PEP* and *HTTP PEP with user differentiation*. In our example in section III-A *Firefox With Pipelining* could be one scenario, while *Firefox Alone* could be another scenario.

The *User Agent* archives performance metrics such as page load latency for each of the URLs in this set with the HTTP-AE *Controller*. These metrics can be compared and contrasted using HTTP-AE's *Evaluation Comparison* component. This component is selectable under the reporting section of the user interface. 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 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.

Figure 5, depicts a logical view of the evaluation of a HTTP PEP, which is the subject of the case study in Section IV of the paper. This is the second scenario depicted in figure 4 i.e. the *HTTP PEP* scenario. When this scenario is executed, each associated *User Agent* accesses a set of URLs in sequence.

Each *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 4, refers to a network with a deployed HTTP PEP and the *Firefox alone* scenario refers to the same system

without the HTTP PEP, while the *HTTP PEP with user differentiation* scenario refers to the same system with the HTTP PEP and Linux based user differentiation in place.

There are two types of *User Agent*: the stand alone client and the browser extension. The stand alone client is a fully functional HTTP client, which does not render the web content to the end user. This client can be deployed to any Java based system. The browser extension is designed to enable the evaluation of a browser or browser technologies. For example, the Firefox browser extension has XPCOM hooks into Firefox to control it. In each case the *User Agent* executes a browsing session for 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.

C. 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 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* [31]. *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 Study Background and Testbed

Broadband satellite is becoming more and more pervasive to deliver Internet services to remote communities [32]. However, issues with HTTP/TCP on wired networks are greatly accentuated when it comes to high-latency links such as broadband satellite systems. Hence, many different technologies for accelerating HTTP/TCP have been developed to improve end-user web browsing quality of experience over satellite systems. Figure 6 indicates where some of technologies are deployed into the network.

Technologies/techniques such as cache proxies [33], 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) [34] can employ all three of these techniques. An HTTP PEP can be deployed as shown in Figure 6 in a high latency segment of a network between the end-users and the web server. TCP PEPs [28] 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.

A. Evaluation testbed

Figure 7 shows our emulated satellite testbed used in our evaluation. The testbed consists of two PCs with Intel Core 2 Duo CPU E4700 2.60 GHz, running Fedora 10.

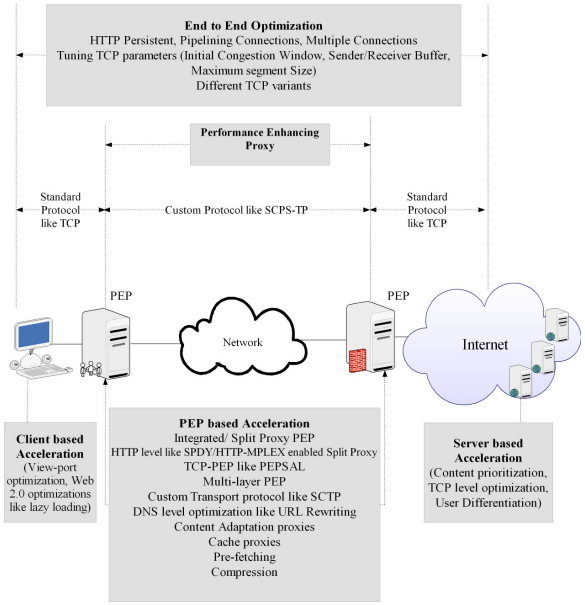

Fig. 6. HTTP Acceleration Techniques

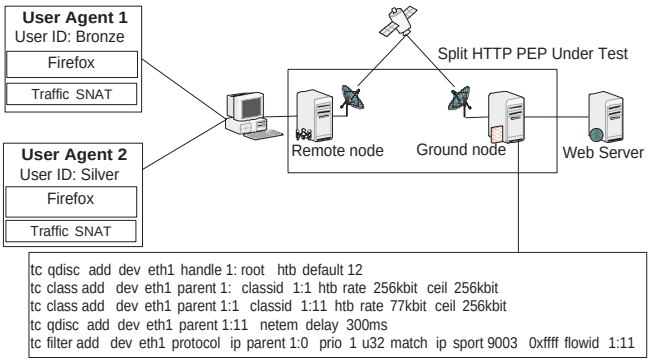

Fig. 7. Testbed

The two PCs, one emulating the remote node and the other emulating the ground node, are connected using a direct 1Gb/s Ethernet connection over which a Satellite link behavior is emulated using *traffic control (tc)* commands in Linux. That is, the data rate is limited to 64Kb/s, 128Kb/s, 256Kb/s, 512Kb/s, and 1 Mb/s. 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 shown results, the URLs point to a set of pages containing the same text and a variable number of embedded images from Caltech 101 image data-set [35]. The number of images in the set of pages varies from 1 to 30 per page. The images are all JPEGs and have an average size of 15K. 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 behavior of network and server access.

Figure 7 also shows the experimental configuration for user differentiation. Each User Agent is associated with a Linux 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%.

V. Case Studies

In these case studies, 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. In Section V-A, we evaluate the effectiveness of using multiple connections to retrieve HTTP content over satellite. In Section V-B, we study the effect of using different TCP slow-start congestion control algorithms on HTTP performance. In Section V-C, we present an evaluation of our HTTP PEP [36]. In Section V-D, we present a modification to our HTTP PEP to support user differentiation and we evaluate its performance with multiple web browsing users.

A. Multiple TCP connections

During TCP slow-start, the amount of unacknowledged data is limited by the sender's congestion window. Thus, the bandwidth is under-utilized 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 opening simultaneous connections for HTTP acceleration in 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. Hence, the transmission of small segments, *i.e.* tinygrams [37], is reduced.

- 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 [5].

Now we shall examine the use of multiple connections in a modern web browser and its performance impact while operating in a resource-limited environment. For this study, we have chosen to evaluate Firefox's use of multiple connections from the perspective of end-user QoS and data transmission overhead for an emulated satellite system. In the following experiments, Firefox has been studied for pipelined connections for varied bandwidth.

Figure 8-12 plots the page load delay as seen by the client versus the number of embedded resources per page for different number of connections and for different link capacities varying from 64kbps to 1Mbps. The figures also show one connection results in a significant increase in page load delay when compared with three and more connections. Note that Firefox does not pipeline the first request for embedded objects and so there is a significant delay before it requests the second and subsequent objects. Figure 8 and 10 show that there is little difference between page load time for three to eight TCP persistent connections for low bandwidth links.

Generally when more TCP connections are opened, during the slow-start phase, the aggregated initial Congestion Window (*cwnd*) of multiple connections outperform fewer TCP connections, but for resource-constrained pipes very few packets can be transmitted. For example, a 128 kb/s link with 600 ms RTT cannot transmit more than 10 X 1500 byte packets per second. A single TCP sender's initial *cwnd* allows it to transmit three maximum segment size (MSS) bytes of data. As a result, opening more connections do not make much difference for low-bandwidth links. Figures 11, 12 [2] show that as the bandwidth increases, the browser's use of multiple connections improves in terms of the page load performance. The improvement becomes less noticeable as the number of objects on the page increases. However, using more than three connections, the data transmission overhead becomes significant as shown in Figure 13. This overhead is due to the small TCP segments and extra signaling load.

[2]Note the altered Y-axis range across Figures 8, 9, 10, 11, and 12 in order to increase visibility of delay differences

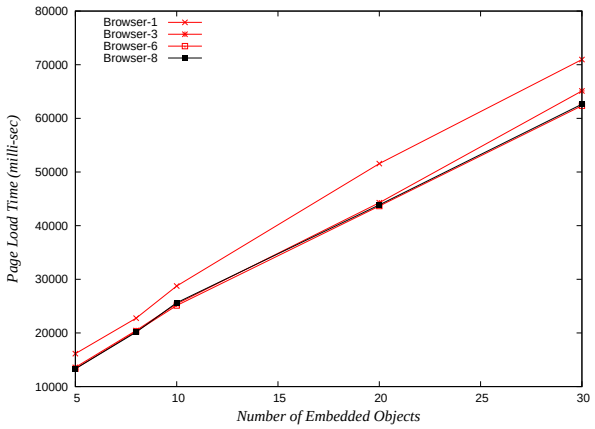

Fig. 8. Performance of Firefox using 1 to 8 pipelined connections in 64 Kb/s and 600ms RTT link

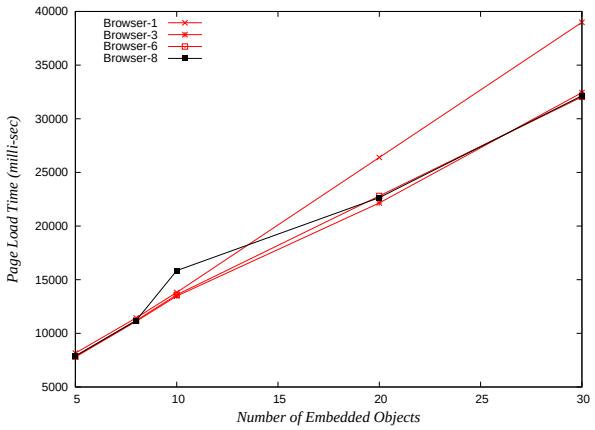

Fig. 9. Performance of Firefox using 1 to 8 pipelined connections in 128 Kb/s and 600ms RTT link

TCP offers a single sequential, in-order bytestream data delivery scheme which results in head-of-line (HOL) blocking and worsens web response time. TCP also suffers from slow-start problems. Modern browsers use multiple TCP connections to overcome these scenarios. Although it is expected that multiple TCP connections may improve HTTP throughput, experimental results show that multiple TCP senders can degrade HTTP performance for resource-constrained links. These experimental results motivate the use of fewer connections for current web browsers while working over high-latency links with low bandwidth.

B. TCP variants

Congestion management is the focus of this subsection. One server side optimization for HTTP is to increase the initial TCP congestion window. This increase allows the server to push more content initially during TCP slow-

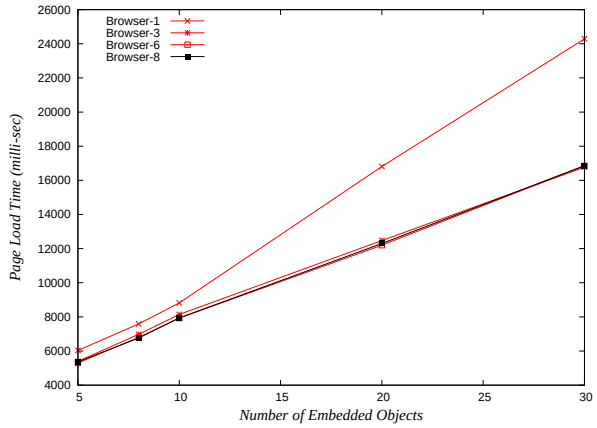

Fig. 10. Performance of Firefox using 1 to 8 pipelined connections in 256 Kb/s and 600ms RTT link

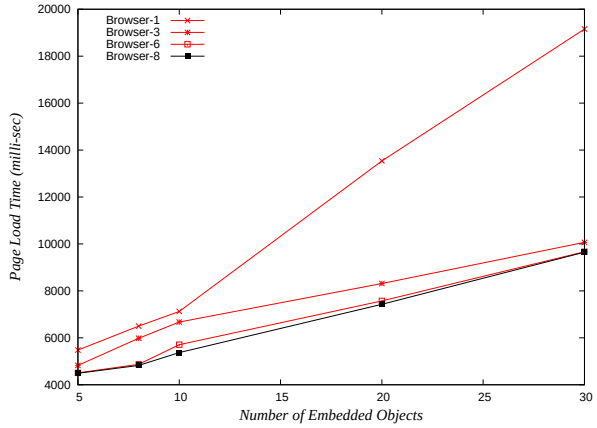

Fig. 11. Performance of Firefox using 1 to 8 pipelined connections in 512 Kb/s and 600ms RTT link

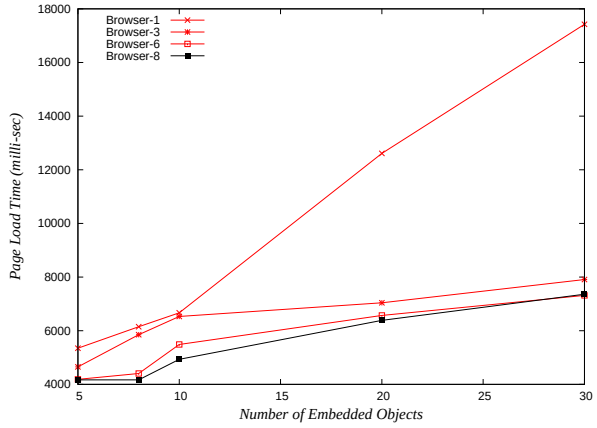

Fig. 12. Performance of Firefox using 1 to 8 pipelined connections in 1 Mb/s and 600ms RTT link

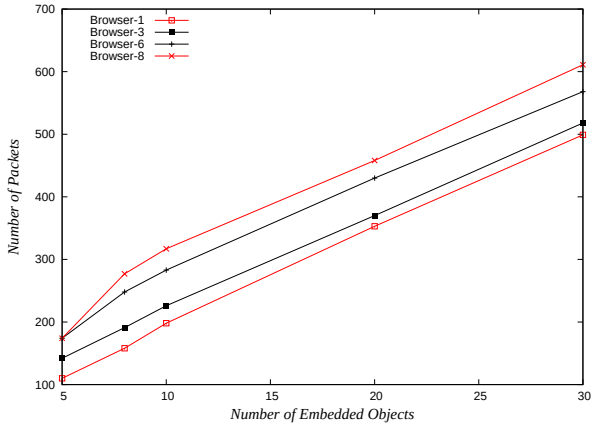

Fig. 13. Packet overhead of using 1/3/6/8 pipelined connections in 256 Kb/s and 600ms RTT link

start. However it is difficult to determine an appropriate value for the initial congestion window independent of the congestion algorithm utilized. Many 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 using different TCP variants Hybla, BIC, Vegas and Westwood at the Ground node [38]. Each of the variants adopts a different congestion control algorithm during slow start. Figures 14-18 show that Hybla, which is designed for high latency links, has the best performance while Vegas results in the worst performance of the four evaluated TCP variants. TCP Vegas is greatly impeded by the long RTT [39]. The number of nested resources does not appear to suit one congestion algorithm over another. These results with regard to TCP variants confirm the work done by other researchers [39]. We conclude from these results that using a TCP Hybla connection at the ground node will give slightly better performance in terms of web page load time for the end-users.

C. HTTP PEP

In our previous paper [36], we presented a 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.

In this Section, we evaluate the HTTP Performance Enhancing Proxy (HTTPEP). Experimental results are

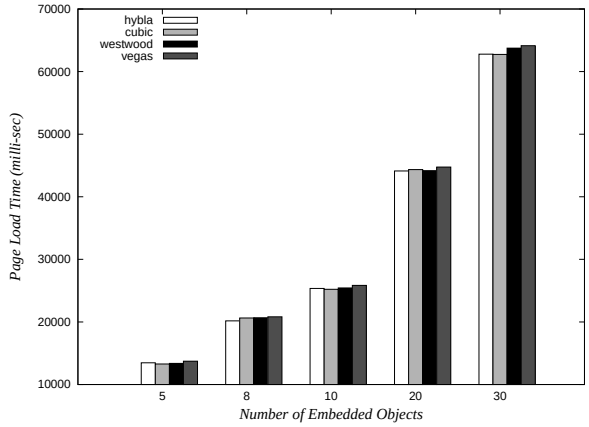

Fig. 14. Performance of TCP variants in 64 kb/s and 600ms RTT link

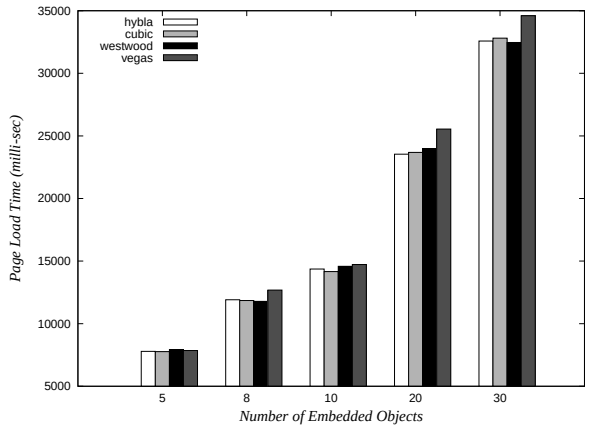

Fig. 15. Performance of TCP variants in 128 kb/s and 600ms RTT link

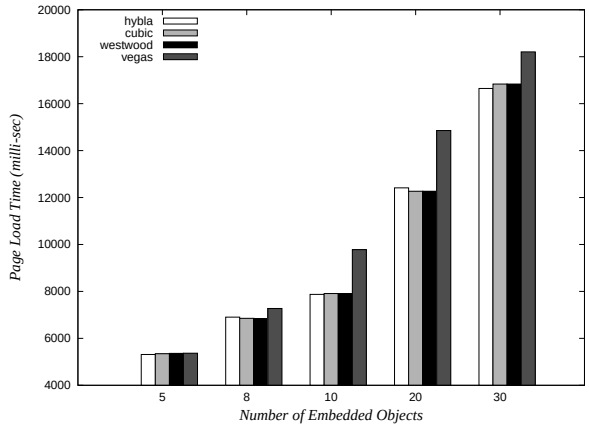

Fig. 16. Performance of TCP variants in 256 kb/s and 600ms RTT link

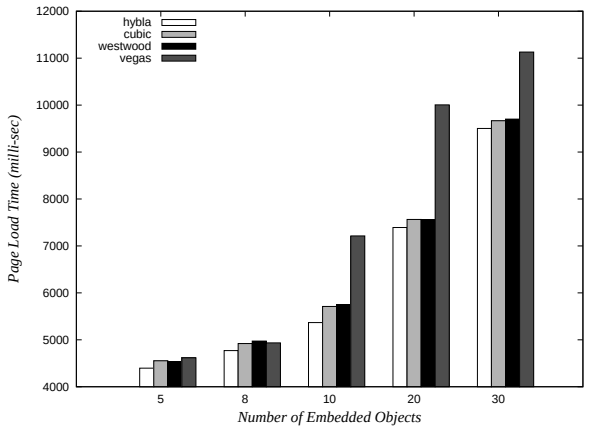

Fig. 17. Performance of TCP variants in 512 kb/s and 600ms RTT link

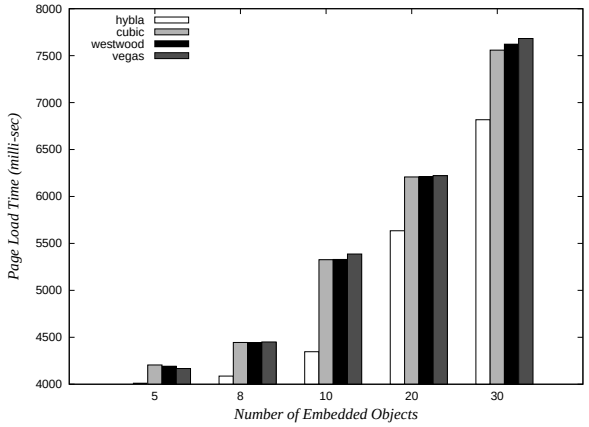

Fig. 18. Performance of TCP variants in 1 Mb/s and 600ms RTT link

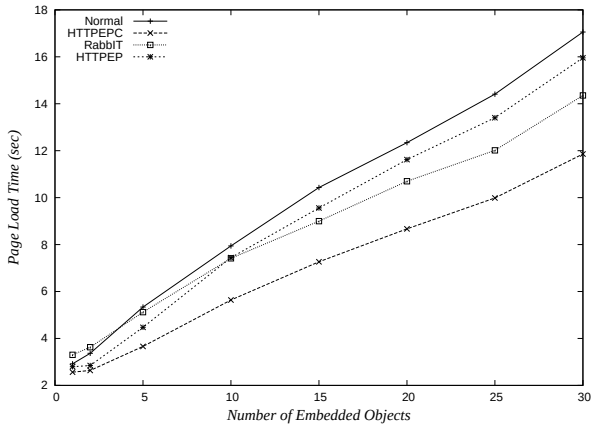

Fig. 19. Average page load delay for 256 Kb/s and 600ms RTT link

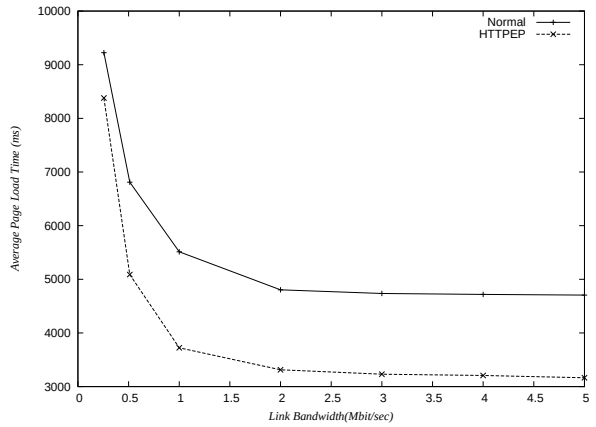

Fig. 20. Effect of varying the satellite link capacity with 600ms RTT

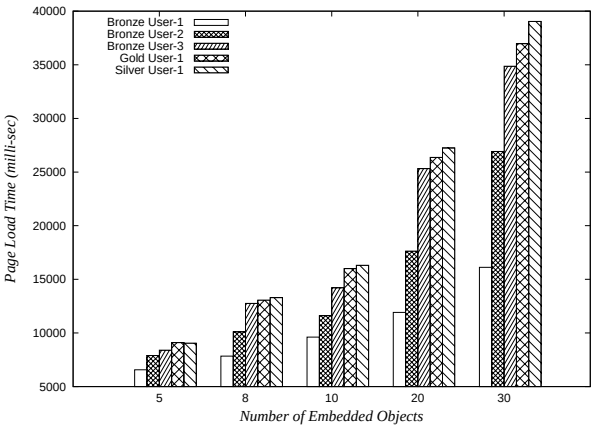

Fig. 21. Multi-user over HTTPEP with no user differentiation

compared with the standard HTTP and a open-source transcoding proxy, RabbIT [40]. RabbIT is an application layer performance enhancing proxy that speeds up web browsing over slow links. It can employ different techniques like text and image compression, caching, removal of background images. With the default configuration RabbIT uses an aggressive compression ratio. For the following set of experiments RabbIT was configured to use the same compression factor as the HTTPEP for a fair comparison.

Figure 19 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). The 10% average saving in page load time is due to our HTTP and TCP protocol optimisations.

Figure 20 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.

D. User differentiation

Figure 21 shows the page load time for a number of users with no user differentiation using HTTPEP with 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 solve this issue, we need to re-design the algorithm that HTTPEP uses to service incoming requests.

We modified HTTPEP's scheduler so that traffic destined for a particular user class is scheduled to an appropriate TCP connection. In this experiment, we have 3 permanent TCP connections between the ground and

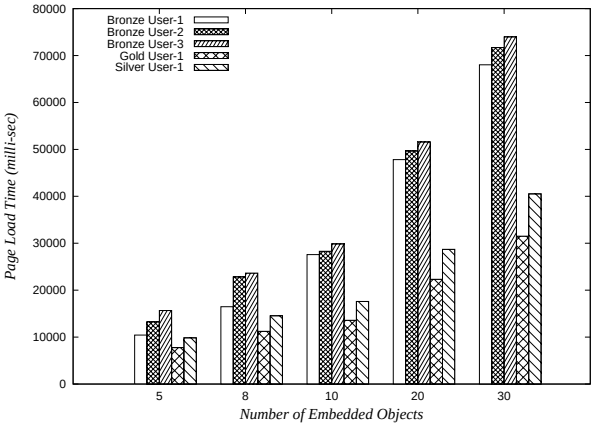

Fig. 22. Multi-user over HTTPEP with user differentiation

remote sides and 50% of the bandwidth allocated to the first connection, 30% to the second connection, and 20% to the third connection. Figure 22 shows the page load time experienced by the different classes of users (gold, silver, bronze) in the new system. The figure shows that the gold and silver users are getting the appropriate service differentiation while the first bronze user on the system is still getting better service than the other bronze users. Since users belonging to the same class should enjoy same quality of service, these results indicate the need for a fair scheduling technique.

VI. Conclusion

In this paper, we present HTTP-AE as a *multi-user* client-side framework for the automated evaluation of mechanisms to improve HTTP performance. Additionally, we present several case studies in which HTTP-AE is used to evaluate three different HTTP acceleration technologies deployed in an emulated satellite system.

These case studies show that the framework can be used to test different design aspects of HTTP acceleration mechanisms. The case studies also show that by using the proposed framework, one can pinpoint the limitations of a given HTTP acceleration mechanism. We used HTTP-AE to uncover a resource allocation issue with our HTTP acceleration proxy. The proxy was not sharing resources fairly over the active users on the system. This knowledge lead to an improved design of our HTTP acceleration proxy.

Acknowledgments: This work is supported by Enterprise Ireland Grant Number IP/2009/0026 and by Altobridge.

REFERENCES

[1] P. Davern, N. Nashid, A. Zahran, and C. J. Sreenan, "Towards an Automated Client-Side Framework for Evaluating HTTP/TCP Performance," *Proc. of International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, June 2011.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] M. C. Necker, M. Scharf, and A. Weber, "Performance of TCP and HTTP Proxies in UMTS Networks," *European Wireless 2005 - 11th European Wireless Conference 2005 - Next Generation wireless and Mobile Communications and Services*, 2005.

[7] "Dummynet," Retrieved March, 2012, from, www.iet.unipi.it/ luigi/dummynet/.

[8] M. C. Necker, M. Scharf, and A. Weber, "Performance of different proxy concepts in umts networks," in *Wireless Systems and Mobility in Next Generation Internet*, ser. Lecture Notes in Computer Science, G. Kotsis and O. Spaniol, Eds. Springer Berlin / Heidelberg, 2005, vol. 3427, pp. 36–51.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] "Firebug," Retrieved March, 2012, from, http://getfirebug.com/.

[14] R. Lerner, "At the forge: Firebug," *Linux J.*, vol. 2007, pp. 8–, May 2007.

[15] C. Luthra and D. Mittal, *Firebug 1.5: Editing, Debugging, and Monitoring Web Pages*, 1st ed. Packt Publishing, 2010.

[16] "Yslow," Retrieved March, 2012, from, http://developer.yahoo.com/yslow/.

[17] "Page speed," Retrieved March, 2012, from, http://code.google.com/speed/page-speed/.

[18] A. Sirotkin, "Web application testing with selenium," *Linux J.*, vol. 2010, April 2010.

[19] "Jmeter," Retrieved March, 2012, from, http://jakarta.apache.org/jmeter/.

[20] "httperf," Retrieved March, 2012, from, http://www.hpl.hp.com/research/linux/httperf/.

[21] P. Svoboda, F. Ricciato, W. Keim, and M. Rupp, "Measured web performance in gprs, edge, umts and hsdpa with and without caching," in *World of Wireless, Mobile and Multimedia Networks, 2007. WoWMoM 2007. IEEE International Symposium on a*, june 2007, pp. 1 –6.

[22] R. Rajamony and M. Elnozahy, "Measuring client-perceived response times on the WWW," in *Proceedings of the 3rd conference on USENIX Symposium on Internet Technologies and Systems - Volume 3*, ser. USITS'01. Berkeley, CA, USA: USENIX Association, 2001, pp. 16–16.

[23] L. Cherkasova, Y. Fu, W. Tang, and A. Vahdat, "Measuring and characterizing end-to-end Internet service performance," *ACM Trans. Internet Technol.*, vol. 3, pp. 347–391, November 2003. [Online]. Available: http://doi.acm.org/10.1145/945846.945849

[24] J. Wei and C.-Z. Xu, "sMonitor: a non-intrusive client-perceived end-to-end performance monitor of secured internet services," in *Proceedings of the annual conference on USENIX '06 Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2006, pp. 21–21.

[25] D. Olshefski and J. Nieh, "Understanding the management of client perceived response time," *SIGMETRICS Perform. Eval. Rev.*, vol. 34, pp. 240–251, June 2006.

[26] Z. Li, M. Zhang, Z. Zhu, Y. Chen, A. Greenberg, and Y.-M. Wang, "WebProphet: automating performance prediction for web services," in *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, ser. NSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 10–10.

[27] "Chromium benchmarking extension," Retrieved March, 2012, from, https://sites.google.com/a/chromium.org/dev/chrome-benchmarking-extension.

[28] 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.

[29] "Xpcom," Retrieved March, 2012, from, https://developer.mozilla.org/en/XPCOM.

[30] "iptables," Retrieved March, 2012, from, http://www.frozentux.net/documents/iptables-tutorial/.

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

[32] Altobridge, Retrieved March, 2012, from, http://www.altobridge.com/.

[33] Squid, Retrieved March, 2012, from, http://www.squid-cache.org.

[34] "Mguard," Retrieved March, 2012, from, http://www.broadband-internet-access.com/.

[35] C. C. V. Group, "Images," Retrieved March, 2012, from, http://www.vision.caltech.edu/html-files/archive.html.

[36] 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.

[37] G. Minshall, Y. Saito, J. C. Mogul, and B. Verghese, "Application performance pitfalls and TCP's Nagle algorithm," *SIGMETRICS Perform. Eval. Rev.*, vol. 27, pp. 36–44, March 2000.

[38] G. Fairhurst, A. Sathiaseelan, H. Cruickshank, and C. Baudoin, "Transport challenges facing a next-generation hybrid satellite Internet," *International Journal of Satellite Communications and Networking*, vol. 29, no. 3, pp. 249–268, 2011.

[39] 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.

[40] "RabbIT ," Retrieved March, 2012, from, http://rabbit-proxy.sourceforge.net/.