HTTP acceleration over high latency links

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

***Abstract*—Broadband satellites enable Internet access for remote communities and niche markets such as ships and airplanes. However, their inherent characteristics, such as long delays and limited resources, significantly degrade the user quality-of-experience. In this paper, we propose a novel HTTP Performance Enhancing Proxy (PEP) that accelerates web-browsing and improves the utilization of satellite resources. We describe TCP and HTTP optimizations, which enable this acceleration. Our performance evaluation shows an average reduction in Web page-load latency of up to 27%.**

***Index Terms*—HTTP acceleration, HTTP Performance Enhancing Proxy, High latency, Satellite.**

I. Introduction

The use of IP satellite networks is gaining an increasing momentum due to their ability to deliver communication services to difficult to reach regions or to nation-wide areas. Figure 1 shows a general architecture for such a satellite system including

- Satellite gateways (SGs), which are traffic aggregators sitting at the edge of network and usually serving more than one user, e.g. residents of a specific remote community. A remote and ground gateway are shown, the remote gateway acts as an aggregation point for remote community traffic and the ground gateway provides an aggregation point for services hosted in the core network.
- Satellite, whose main role is to forward the traffic across attached satellite terminals, i.e. gateways.
- Network Control Center (NCC) is a logical entity that controls the satellite network. NCC performs different functions including synchronizing terminals, resource management, alarm management, security management, performance management, billing and accounting.

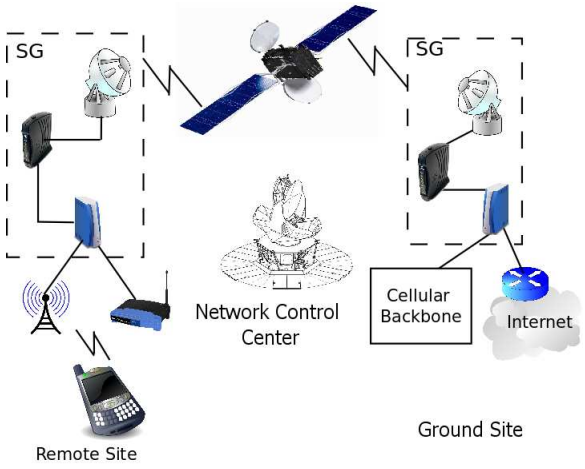

Figure 1. Satellite System

In this architecture, SGs usually represent a bottleneck, where for example web browsing at a client machine, suffers from long page load time due to the long RTT (~600ms for Geostationary Satellites) and the limited bandwidth of the link [1]. In such an Internet access system, techniques such as cache proxies [2], pre-fetching of Hypertext Transfer protocol (HTTP) content, and persistent TCP connections are used to improve the end-user's web browsing experience. HTTP PEPs [3], can employ all three of these techniques to accelerate HTTP over satellite.

In this paper, we present 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. A novel bundling mechanism is used to transfer web content between the split nodes. Overall, HTTPEP gives an average of 27% reduction in Web page load latency.

The rest of the paper is organized as follows. Section II is dedicated to background and related work. In Section

III, we present the system components and operating mechanisms. Section IV shows our system performance evaluation. Finally, conclusions and future works are presented in section V.

II. Background and Related Work

The Hypertext Transfer protocol (HTTP) is a stateless, transactional protocol that governs content exchange between web clients and servers. HTTP features a sequential operation that delays the retrieval time of embedded web *resources*[1]. For example, when a client on the end-user's machine issues an 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.

Web browsers do not typically aggregate their GETs but request the required resources one at a time. However, a browser may pipeline its GET requests. In this case, multiple requests can be sent before any responses are received. Pipelining reduces load on the network as several GET requests can be combined into one TCP packet. But not all web servers or HTTP proxies support it. For this reason, most of the major browsers have pipelining disabled by default [4].

There are several approaches specified to retrieve an aggregated set of resources, commonly known as *bundles*, from a web server. These approaches can be generally classified as client-side and server-side solutions. MGET [5], GETALL [6], and GETLIST [7] methods are client-side solutions that aggregate a number of requests into a single message. Server-side solutions include [8] in which the server decides when and how to construct bundles of frequently accessed resources. As clients cannot control the resources that are contained in a bundle, so clients must decide whether to request the bundles or resources individually. Similar philosophy has been adopted in the Web++ [9] framework which is designed to deliver an HTML document and its nested resources in one transaction.

There are several Transmission Control Protocol (TCP) issues, which contribute to web page load latency over satellite links [10]. For example there are issues related to the connection setup and slow start. It is well known that TCP employs a three-way handshake (SYN-SYN-ACK) to establish TCP connections and to tear them down. The process of opening such connections over a high latency link introduces a long delay into web page loading time[11].

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

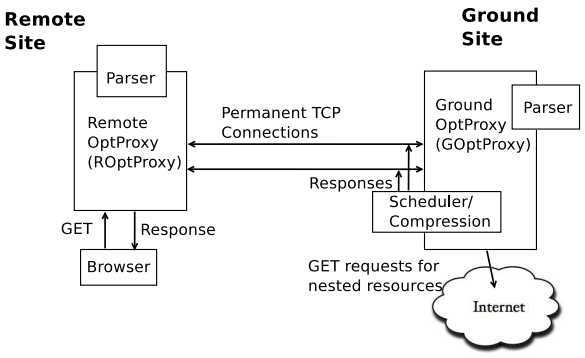

Figure 2. System Components

When a TCP connection is first opened, the TCP sender enters the slow start phase. In this phase TCP builds up its congestion window as it transmits data. TCP takes a long time to exit this phase when it is operating over a high latency link and during this period the link will be underutilized [12]. Once TCP exits the slow start phase it is able to utilize the bandwidth more efficiently.

Different mechanisms [12], [13], [14], [15], [16] are proposed to improve TCP performance in satellite systems. One particular mechanism is the use of TCP performance enhancing proxies (PEPs) [17]. TCP PEPs adopt different techniques to enhance TCP performance over high latency links including ACK spacing, local ACK, local transmission, header compression, payload compression and priority-based multiplexing. However, breaking the end-to-end semantics in PEPs has different implications including security and reliability. Typically, using PEPs conflicts with the need for network level security or application end-to-end acknowledgment. Future work will address such issues.

HTTPEP mitigates the effects of HTTP/TCP operation on web page load delay over high latency links through a bundling mechanism over a split architecture as described in the following section.

III. System Operation

Figure 2 presents the system components of HTTPEP. HTTPEP has a "split HTTP proxy" architecture by which the functionality is distributed between two HTTP proxy nodes implemented at both ends of the satellite link. The

remote proxy (ROptProxy) appears as an HTTP proxy to the end user web browser at the remote site. The ground proxy (GOptProxy) acts as a typical web client to web servers. The two proxies split the HTTP protocol at the remote side proxy, re-issues it at the ground proxy and reconstructs it again at the remote proxy. The two proxies communicate the web content using our novel bundling mechanism.

At startup, ROptProxy and GOptProxy establish a fixed number TCP connections between them. This is only done during the setup phase. These pipes are maintained throughout the system operation and are used for all communication between the proxies.

In order to serve an HTTP web page request, the system operates as follows:

1) When a new web page is required at a client, an initial GET request is sent by the browser for this page (termed the base page).
2) ROptProxy intercepts this GET request and forwards the URL to GOptProxy over one of the fixed TCP connections.
3) GOptProxy fetches the base page associated with the URL from the web server and passes it to a Scheduler/Compression Engine. The base HTML is scheduled for transmission to the remote site over the persistent connections.
4) GOptProxy parses the Base HTML and retrieves the associated web resources that are embedded in that page.
5) These resources are stored temporally at the ground site. They are then compressed and scheduled for transfer to the remote site. The associated HTTP headers for these resources are also compressed. Session information such as cookies are preserved in this process.
6) When the base HTML is received at the remote side, ROptProxy parses it to determine which resources will be on the way from the ground. Then it passes the base HTML to the client.
7) When the client makes subsequent requests for the associated resources of the base HTML, ROptProxy determines if the resources are on the way from the ground side. If so, the request from the client is delayed until the resource arrives. Otherwise, this request is considered to be a new base page request and the system executes the steps outlined above.

Resources are streamed into multiple TCP persistent connections from the ground to the remote sides. When scheduler needs to transfer a resource it chooses the next free TCP connection in a round-robin fashion. Thus the resources will be transmitted to the remote site in parallel. The resources could also be multiplexed into one TCP connection or transferred over other transport layer protocols such as SCTP[18]. If the client is not waiting for a particular resource then ROptProxy stores it temporarily at the remote site and it is returned to the client when it is required.

The HTTPEP split architecture optimizes the web page retrieval in several different ways:

- The expensive three-way handshake over the satellite link is eliminated as the TCP connections to the web server are made at the ground site.
- The initial GET for a particular base HTML causes a bundle of GET requests for its nested resources to be issued at the ground site. The associated resources are then transferred from the ground to the remote site. The subsequent GET requests for nested resources within the base HTML by the client do not cross the high latency link but are served locally. This mechanism overcomes the sequential operation of HTTP.
- The persistent connections from the ground to the remote sites allow TCP to build up its congestion window and so the effects of the slow start could be mitigated. In the presence of a TCP PEP there would not be such an issue with the TCP slow start.
- The compression assists in reducing the amount of traffic crossing the satellite link.

These improvements are reflected in the next section.

IV. Performance Evaluation

The evaluation took place on a test bed consisting of two PCs with Intel Core 2 Duo CPU E4700 2.60 GHz, running Fedora 10, one running ROptProxy and the other running GOptProxy. The ground and remote nodes, are connected using a direct 1Gbps Ethernet connection over which the satellite link behavior is emulated using *tc* commands in Linux. That is, the data rate is limited to 256Kbps and a 300ms one way delay is introduced in each direction. Several PCs are connected to the remote node each running a web browser.

In our performance evaluation, we consider different performance metrics including:

- Traffic volume crossing the link determined by gathering statistics on the interfaces connected to the emulated satellite link on both ends using tcpdump

traces. The most relevant statistics include information about sent data (bytes and packets). The statistics are obtained using capinfos, a terminal-based tool that is part of Wireshark. Note that tcpdump is used for the sake of automating the testing and result generation.

- HTTP signaling load represented by the number of GET request crossing the emulated satellite link. The number of GET requests is a function of the number of resources per page.
- End-to-end delay performance measured as the delay from the instance of sending the first GET request to the time at which the entire page, including all embedded resources, is received. The page load time is measured using FireFox's XPCOM [19], a cross platform component object model which can be controlled using JavaScript.

The testbed clients are PCs running FireFox, which is controlled through the XPCOM. At the beginning of each experiment the FireFox cache is cleared.

The testing process is driven by a C program that reads a list of URLs from a file. The program drives FireFox to navigate through this set of URLs automatically. 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 [20]. 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.

Figure 3 plots the traffic crossing the satellite link in bytes versus the number of embedded resources for different operating modes including normal operation (i.e. a direct connection between the browser and the web-server without an intermediate proxy), HTTPEP without compression (HTTPEP)[2], HTTPEP with compression of resources (HTTPEPC).

The figure shows a marginal saving in terms of number of bytes crossing the bent pipe for the HTTPEP solution in comparison to normal operational mode. This byte reduction is due to several factors. First the elimination of TCP signaling for connection establishment and release. Second the compression of the HTTP headers. Third, the reduction in application layer signaling in that embedded

[2]The HTTPEP results include HTTP header compression

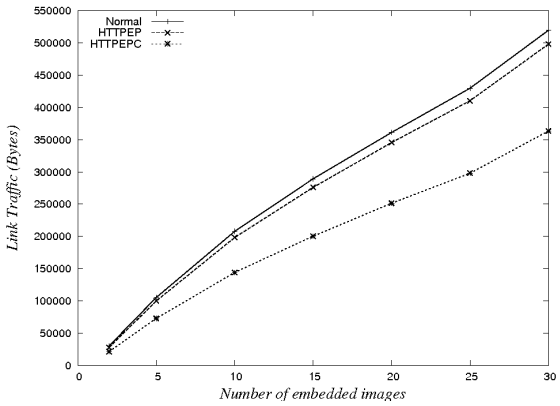

Figure 3. Traffic crossing the link in bytes

resources are not explicitly requested by the remote site but are automatically transmitted.

Typically, the number of GET requests is function of the number of resources per page. Assuming a page has n resources, then the number of GET requests is (n+1), in which the "1" corresponds to the initial GET request. HTTPEP eliminates this application layer signaling. However, the solution obtains the base HTML page through an explicit URL request passed from ROptProxy to GOptProxy.

Additional significant savings in bandwidth usage is attained by compressing the resources before they are transferred to the remote site. For the results shown here the compression quality factor of the JPEG images is reduced to 50 using the GraphicsMagick [21] library. Our investigation has shown that improvements are proportional to the size of the embedded resources.

Figure 4 plots the page load delay as measured at the client machine versus the number of embedded resources per page. The figure shows an average page load delay reduction is 27% using compression in comparison to an average saving of 10% without compression.

The page load delay saving of 10% is due to two factors. First the elimination of the latency in TCP signaling for connection establishment. Second, by using the bundling mechanism the resources are made available at the remote site more quickly than with the browser alone.

Figure 5 plots the page load time for the HTTPEP and normal operation versus the satellite link capacity. Each point in the figure is an average over all the test pages. The figure shows a significant improvement of the HTTPEP solution over the browser alone as the satellite link capacity increases. When the bandwidth

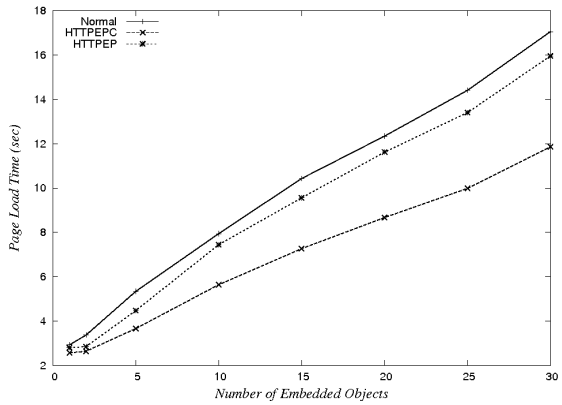

Figure 4. Average page load delay

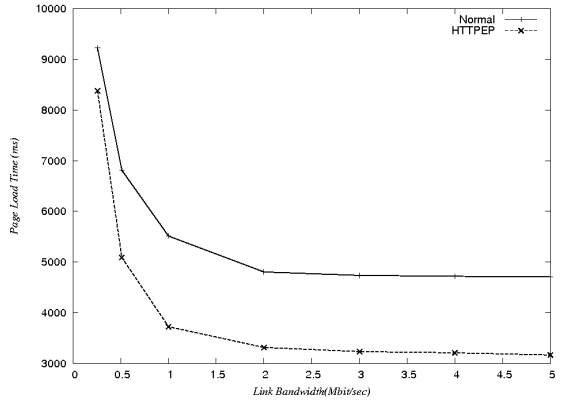

Figure 5. Effect of varing the satellite link capacity

is constrained to 256Kbps the average improvement of HTTPEP over the browser alone is 10%, whereas, when the bandwidth is set to 500Kbps there is an average improvement of 27%. The average improvement levels out at 32% when the link capacity reaches 2Mbps.

V. Conclusions and Future Work

A novel HTTP PEP (HTTPEP) is introduced in this paper. The performance evaluation shows that several gains can be attained by deploying HTTPEP in a satellite based Internet access system. There is a reduction in the amount of traffic crossing the satellite link and a significant reduction in the page load time up to 27% on average under the aforementioned operating scenario i.e. using a link constrained to 256 Kbps with an RTT of 600ms. We are looking to integrate new caching techniques into the split architecture to further improve the web browsing experience in satellite systems.

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

References

[1] C. Caini, R. Firrincieli, M. Marchese, T. de Cola, M. Luglio, C. Roseti, N. Celandroni, and F. Potortì, "Transport layer protocols and architectures for satellite networks," *International Journal of Satellite Communications and Networking*, vol. 25, pp. 1–26, Jan.–Feb. 2007.

[2] "Squid," 2010. Retrieved Sept 16, 2011, from http://www.squid-cache.org.

[3] "Mguard." Retrieved September 16, 2010, from http://www.broadband-internet-access.com/.

[4] "What is http pipelining." Retrieved October 9, 2010, from http://www.mozilla.org/projects/netlib/http/pipelining-faq.html.

[5] J. Franks. Retrieved September 29, 2010, from http://ftp.ics.uci.edu/pub/ietf/http/hypermail/1994q4/0260.html.

[6] V. N. Padmanabhan and J. C. Mogul, "Improving http latency," *Comput. Netw. ISDN Syst.*, vol. 28, no. 1-2, pp. 25–35, 1995.

[7] V. N. Padmanabhan, "Addressing the challenges of web data transport, ph.d thesis, university of california at berkeley," 1998.

[8] C. E. Wills, M. Mikhailov, and H. Shang, "N for the price of 1: bundling web objects for more efficient content delivery," in *WWW '01: Proceedings of the 10th international conference on World Wide Web*, (New York, NY, USA), pp. 257–265, ACM, 2001.

[9] B. S. (sun Bin, "Speeding up the web using the web++ framework," In Proceedings of WebNet, 2001.

[10] S. Kota and M. Marchese, "Quality of service for satellite IP networks: a survey," *International Journal of Satellite Communications and Networking*, vol. 21, pp. 303–349, 2003.

[11] M. Allman, C. Hayes, H. Kruse, and S. Osterman, "Tcp performance over satellite links," in *5th International Conference on Telecommunication Systems*, 1997.

[12] M. Allman, D. Glover, and L. Sanchez, "RFC2488: Enhancing TCP Over Satellite Channels using Standard Mechanisms," 1999.

[13] D. Adami, M. Marchese, and L. Ronga, "TCP/IP-based multimedia applications and services over satellite links: experience from an ASI/CNIT project," *IEEE Personal Commun.*, vol. 8, no. 3, pp. 20–27, 2001.

[14] W. K. Chai, M. Karaliopoulos, and G. Pavlou, "Providing Relative Service Differentiation to TCP Flows over Split-TCP Geostationary Bandwidth on Demand Satellite Networks," in *WWIC '07: Proceedings of the 5th international conference on Wired/Wireless Internet Communications*, (Berlin, Heidelberg), pp. 17–29, Springer-Verlag, 2007.

[15] P. C. G. Giambene, D. Bartolini, M. Luglio, and C. Roseti, "Dynamic resource allocation based on a TCP-MAC cross-layer approach for DVB-RCS satellite networks," *International Journal of Satellite Communications and Networking*, vol. 24, no. 5, pp. 367–385, 2006.

[16] C. Caini, R. Firrincieli, and D. Lacamera, "Comparative performance evaluation of tcp variants on satellite environments," *Communications, 2009. ICC '09. IEEE International Conference on*, pp. 1 –5, jun. 2009.

[17] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby, "Rfc3135: Performance enhancing proxies intended to mitigate link-related degradations," 2001.

[18] 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.

[19] Mozilla, "Xpcom." Retrieved: October 3, 2010, https://developer.mozilla.org/en/XPCOM.

[20] C. C. V. Group, "Images," 2010. Retrieved: September 16, 2010, from http://www.vision.caltech.edu/html-files/archive.html.

[21] GraphicsMagick, "Graphicsmagick image processing system." Retrieved: October 3, 2010, http://www.graphicsmagick.org/.