HTTPEP: a HTTP Performance Enhancing Proxy for Satellite Systems

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[1],
Department of Electronics and Electrical Communications, Cairo University, Cairo, Egypt.
azahran@eecu.cu.edu.eg

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 end-user's web browsing 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 HTTP optimizations, which enable this acceleration. We show that, optimizing the transport layer to the communication mechanism of a HTTP PEP gives further benefits in terms of a reduction in bandwidth usage and computing resource utilization. Our performance evaluation shows an average reduction in Web page-load latency of up to 27%.

General Terms: Performance, Design, Human Factors

Keywords: HTTP acceleration, HTTP Performance Enhancing Proxy, High latency, Satellite.

1. INTRODUCTION

HTTP and TCP are the underlying protocols that enable web browsing. HTTP is a stateless, transactional protocol that governs content exchange between web clients and servers. TCP is responsible for flow control, congestion control, and reliable ordered transmission of packets. Additionally, TCP is a connection-oriented self-clocked transport protocol. Hence, the performance of HTTP/TCP is significantly affected by changes in the protocol stack and the link between the client and server. For example, the performance of HTTP/TCP is significantly affected when a satellite system is deployed in intermediaries in the network. The use of such IP satellite networks is gaining an increasing momentum due to their ability to deliver communication services to difficult to reach regions. Figure 1 shows a general architecture for such a satellite system including

—Satellite gateways (SGs), which are traffic aggregators: 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.

The end-user's web browsing quality of experience is adversely impacted by the long Round Trip Time (RTT) (˜600ms for GEOstationary Satellites), the high bit error rate (BER) and the limited bandwidth of the satellite system [Caini et al. 2007]. These properties of a satellite based Internet access system pose severe challenges for HTTP/TCP. For example, TCP does not have a means

[1]This work was completed while the author was at University College Cork

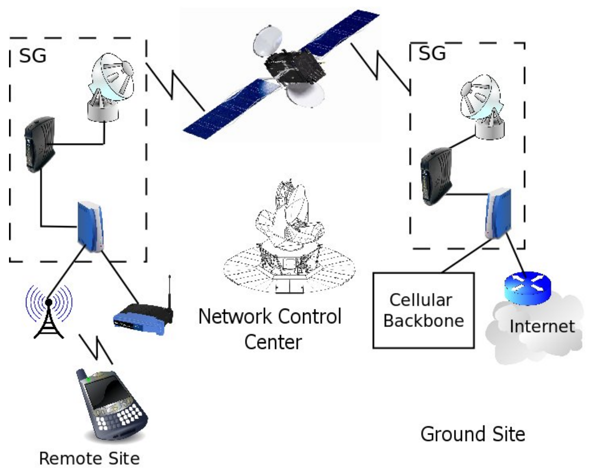

Figure. 1: Satellite System

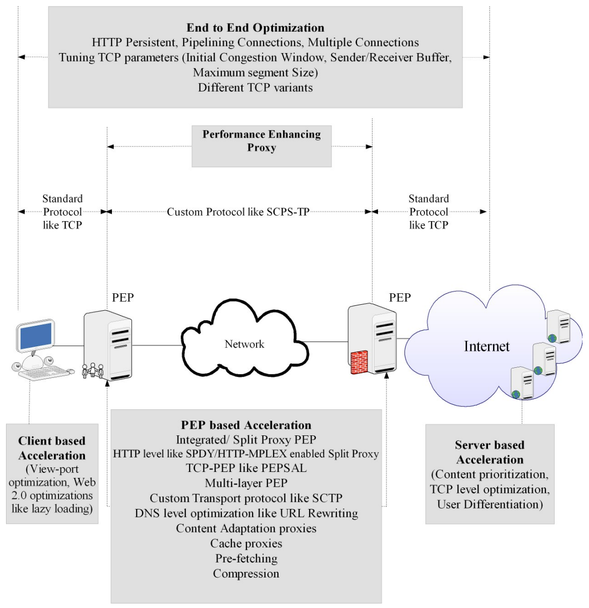
End to End Optimization
HTTP Persistent, Pipelining Connections, Multiple Connections
Tuning TCP parameters (Initial Congestion Window, Sender/Receiver Buffer, Maximum segment Size)
Different TCP variants
Performance Enhancing Proxy
Standard Protocol like TCP
Custom Protocol like SCPS-TP
Standard Protocol like TCP
PEP
PEP
Network
Internet
Client based Acceleration (View-port optimization, Web 2.0 optimizations like lazy loading)
PEP based Acceleration
Integrated/ Split Proxy PEP
HTTP level like SPDY/HTTP-MPLEX enabled Split Proxy
TCP-PEP like PEPSAL
Multi-layer PEP
Custom Transport protocol like SCTP
DNS level optimization like URL Rewriting
Content Adaptation proxies
Cache proxies
Pre-fetching
Compression
Server based Acceleration (Content prioritization, TCP level optimization, User Differentiation)

Figure. 2: HTTP PEP Operation

to distinguish between the high latency of satellite systems and network congestion. Further, today's 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 it deteriorates the end-user's browsing experience due to the sequential request-response nature of the protocol. This inherent characteristic of HTTP is accentuated when it comes to high latency links such as broadband satellite systems.

Many different technologies for accelerating HTTP/TCP have been developed, some of which have been specifically designed to address these issues with HTTP/TCP in satellite systems. Figure 2 details such HTTP acceleration techniques can be applied at the client, at the server or at intermediaries in the network itself. Client-side acceleration of HTTP includes techniques for issuing requests for content in parallel to the web server. While server-side HTTP acceleration includes focusing on high throughput for content delivery. Technologies/techniques such as cache proxies [Squid], pre-fetching of HTTP content, and persistent TCP connections are deployed in satellite systems to accelerate HTTP. A HTTP Performance Enhancing Proxy (PEP) [Broadband Internet Access] [Chakravorty et al. 2005], can employ all of these techniques to accelerate HTTP. A HTTP PEP can be deployed as shown in Figure 2 in a high latency segment of a network between the end-users and the web server. In this case, the PEP separates the satellite portion from the rest of the network and provides special treatment of HTTP for the satellite segment.

In this paper we build on our previous work [Davern et al. 2011] in which we contributed our HTTP PEP (HTTPEP) to the research community. HTTPEP improves the end-user's web browsing experience over a high-latency link by optimizing the sequential operation of the HTTP protocol, by the elimination of TCP and application layer signaling and by the application of compression techniques. HTTPEP appears as a HTTP proxy to a web browser on the end-user's machine, while the functionality of the proxy is in fact split between two the ground and remote nodes. The main contributions of this paper with respect to our previous work are,

—We give a detailed description and performance evaluation of our novel bundling algorithm which is used to transfer web content between the split nodes.

—We show that, optimizing the transport layer to the communication mechanism of a HTTP PEP gives further benefits in terms of a reduction in bandwidth usage and computing resource utilization at the ground and remote nodes.

The rest of the paper is organized as follows. Section 2 is dedicated to background and related work. In section 3, we present the system components and operating mechanisms. Section 4 presents our system performance evaluation. Finally, conclusions and future work are presented in section 5.

2. BACKGROUND AND RELATED WORK

The Internet has evolved extensively since its inception from text-only communications to much more demanding content rich applications like social networking, video streaming and graphic intensive web sites [Sadre and Haverkort 2008]. At the same time, web sites are incorporating richer methods of user interaction, complex scripting and presentation technologies [Cormode and Krishnamurthy 2008]. As a result, web browsing is becoming more bandwidth intensive operation. But, a significant percentage of web users are accessing Internet through resource limited satellite connections. So poor user perceived web browsing performance over a capacity limited congested link is a common phenomenon which can result into high level of user frustration and poor productivity. A recent study by Akamai Technologies [Technologies 2009] found that users expect a web page to load in two seconds or less, and forty percent of users will wait for no more than three seconds before leaving a site. Therefore, in order to expedite web browsing in today's Internet, especially for users who access via resource constrained links, extensive research [Caini et al. 2009] [Marchese et al. 2004] [Kruse et al. 2001a] [Mattson and Ghosh 2007] has been conducted and various approaches [Floyd et al. 1998][Rodriguez and Biersack 2002][Rodriguez and Fridman 2004] and products [XipLink 2011] [Riverbed 2011] [Altobridge 2011] have been proposed.

The rest of this section is organized as follows. Section 2.1 describes a characteristic of HTTP, which can be optimized to improve web page load times. Section 2.2 describes two characteristics of TCP, which become problematic given the particular operating conditions of satellite systems. In section 2.3 we present specific background for HTTPEP.

2.1 HTTP Sequential Operation

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*[2]. 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 [Mozilla].

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. HTTP-MPLEX [Mattson and Ghosh 2009], and GETLIST [Padmanabhan 1998] methods are client-side solutions that aggregate a number of requests into a single message. Server-side solutions include [Wills et al. 2001] 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++ [Swen 2001] framework which is designed to deliver an HTML document and its nested resources in one transaction. HTTPEP differs from these solutions as it requires no changes to the existing HTTP/TCP protocol stack.

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

2.2 TCP Session Establishment and Slow Start

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 [Kota and Marchese 2003]. Each TCP connection requires a three-way handshake (SYN-SYN.ACK-ACK) for session establishment before sending any data over a TCP connection. This behavior not only delays the data transmission but also introduces a data transmission overhead when opening multiple TCP connections. TCP depends on a reactive congestion control mechanism by which a sender should conservatively increase its transmission window during the slow start phase. This behavior leads to slowing down the TCP session and under-utilization of network resources [Allman et al. 1997].

These issues with TCP on wired networks are greatly accentuated when it comes to high latency links such as broadband satellite systems. 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. Thus, for GEO Satellites TCP session establishment, for the satellite portion of the link, takes approximately 0.9 secs. This issue can be mitigated by carrying the HTTP session over permanent TCP connections. During TCP slow start the amount of unacknowledged data is limited by the sender's congestion window and the bandwidth is not being utilized for data transmission while the sender is waiting for acknowledgments from the receiver. In GEO satellite systems this delay is typically 0.3 seconds and thus there are large gaps in data transmission. The long delay of this type of link reduces the TCP congestion window (CWND) growth rate, which is dependent on the link RTT. Whereas, TCP Hybla [Carlo and Firrincieli 2004] was specifically designed for high latency links and its congestion algorithm is not based on the RTT of the link, but rather on the RTT of a reference connection.

Different mechanisms [Allman et al. 1999; Adami et al. 2001; Chai et al. 2007; Giambene et al. 2006; Caini et al. 2009] are proposed to improve TCP performance in satellite systems. One particular mechanism is the use of TCP performance enhancing proxies (PEPs) [Border et al. 2001]. 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, referring to figure 2, the two PEPs need to communicate the application data using TCP or some custom protocol. Thus, the HTTP session is broken by the PEP.

To overcome the slow start data transmission 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. Firstly, 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. Secondly, 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.

2.3 HTTPEP Design Characteristics - Background

HTTPEP will typically be deployed according to the splitting approach [Border et al. 2001], where the satellite portion is separated from the rest of the network. The satellite system architecture will be that of figure 1 where the ground and remote gateway nodes can provide quality of service differentiation and control all the traffic over the satellite. In this case, there is an opportunity to provide cross-layer improvements to, for example, the transport layer for multimedia applications [Cruickshank et al. 2009]. With regard to HTTPEP, an integrated HTTP/TCP solution is possible where, for example, the TCP slow start algorithm is specifically optimized according to the dynamic requirements of HTTPEP.

We assume that IP services are available at the remote side of the satellite link [Altobridge 2011]. For example, local SGSN and GGSN nodes could be deployed at the remote side gateway,

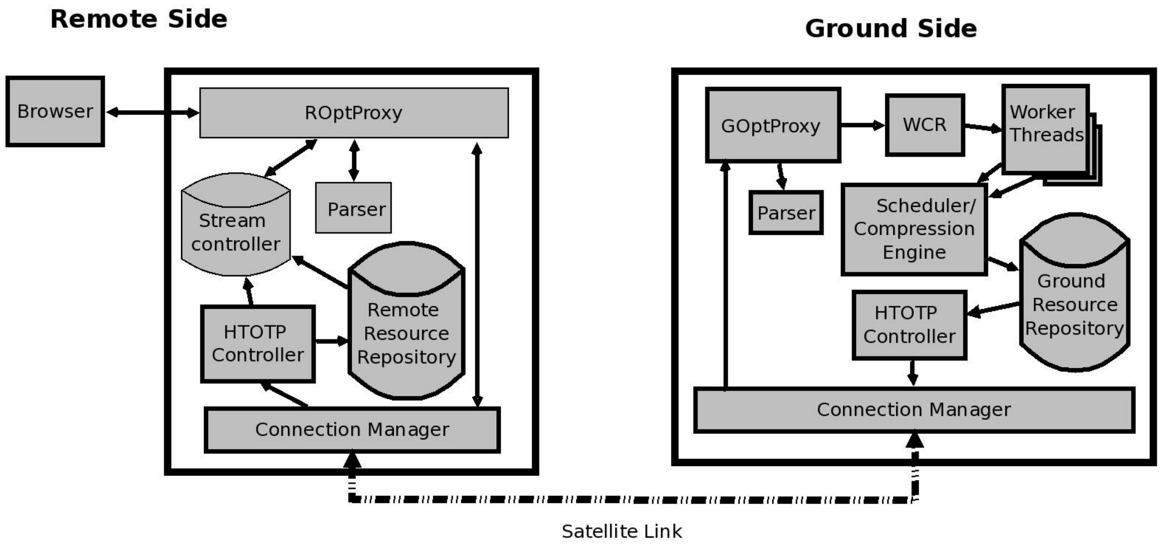

Figure. 3: HTTPEP System Components

which would provide access to the Gi interface at remote gateway. HTTPS traffic cannot be optimized because our algorithm requires direct access to the HTML source. Session integrity is ensured because all end-to-end HTTP communication between the client and server is maintained. Our algorithm optimizes the mode of this communication.

When the HTTPEP solution is not in place, then HTTP traffic passes directly from the remote to the ground nodes without any intervention or optimizations.

When HTTPEP is in place then it mitigates the effects of HTTP/TCP operation on web page load delay over high latency links through our novel bundling mechanism and an optimal use of TCP over a split architecture as described in the following section.

3. SYSTEM OPERATION

HTTPEP consists of the system components as detailed in figure 3 . HTTPEP has a “split HTTP proxy” architecture by which the functionality of a HTTP PEP is split 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 system splits 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.

The rest of this section is organized as follows. Section 3.1 gives an overview of how the system serves HTTP GET requests from the browser. In section 3.2, we describe the components of the system. In section 3.3, we describe the particular optimizations that the system performs to reduce the end-user’s web page load time. In section 3.4 we present motivations to modify the transport layer to suit the operation of HTTPEP.

3.1 Operational Overview

Referring to figure 4, to serve a 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. GOptProxy passes the request to the WCR (Web Content Retrieval) module.

(3) WCR fetches the base page associated with the URL from the origin server and returns

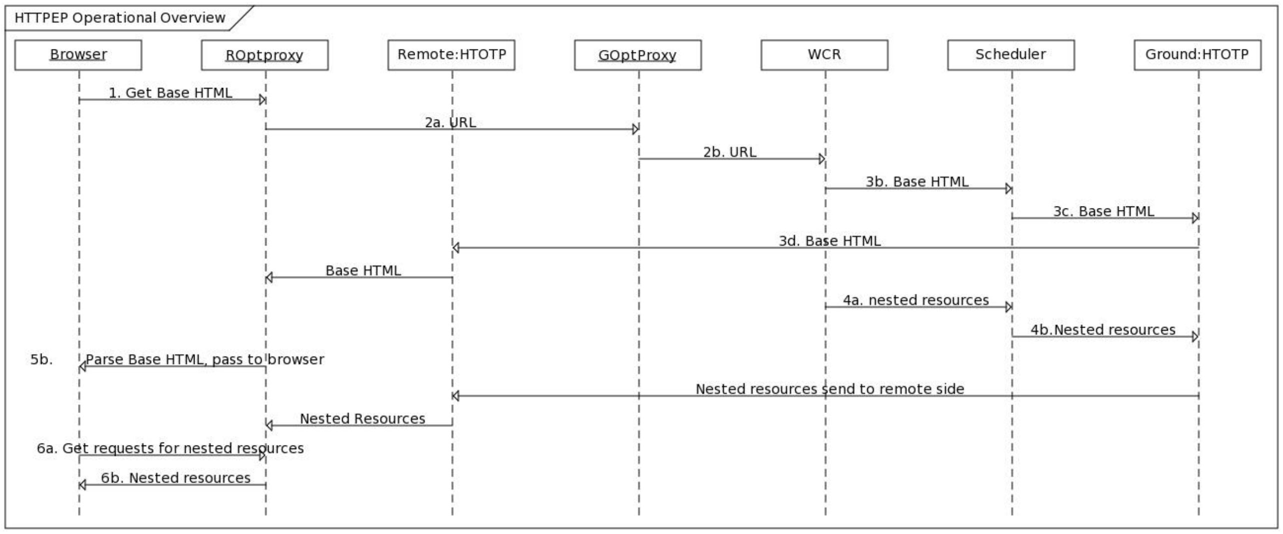

Figure. 4: Operational Steps

this back to the Scheduler/Compression Engine. The scheduler places the base-HTML into Ground Repository. The base-HTML is subsequently transferred to the remote side by the Hypertext Object Transfer Protocol (HTOTP) controller.

(4) GOptProxy parses the base page and finds the set of web resources that are embedded in that page. WCR then invokes a set of worker threads to fetch these nested resources. After fetching a resource, the worker thread passes the resource to Scheduler/Compression Engine, which places it into the Ground Resource Repository for the HTOTP controller. The associated HTTP headers for these resources are also compressed. Session information such as cookies are preserved in this process.

(5) 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.

(6) 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.

3.2 Component Description

The Connection Manager is responsible for maintaining and monitoring the TCP connections that HTTPEP uses between the ground and remote sides. At startup, the Connection Manager on both the ground and remote sides establish a set of persistent TCP connections between them. This is only done during the setup phase. These connections are maintained throughout the system operation and are used for all communication between the proxies. Resources are streamed into these connections from the ground to the remote sides. We use these persistent TCP connections for three reasons. Firstly, the browser opens multiple connections to the web server and issues GET requests for resources in parallel over these connections. Thus, the natural operation that the browser expects of HTTP is that resources arrive concurrently. To emulate this behaviour of HTTP as closely as possible we multiplex the requested resources concurrently into the multiple TCP connections. The resources could also be multiplexed into one TCP connection or transferred over other transport layer protocols such as SCTP [Natarajan et al. 2008]. Secondly, the use of multiple connections is a typical solution to overcome network underutilisation during

TCP slow start [Dukkipati et al. 2010]. Thirdly, we use persistent TCP connections to minimize the number of TCP sessions that need to be setup over the satellite link [Kruse et al. 2001b].

ROptProxy intercepts HTTP GET requests for a particular URL from the end-user's browser. ROptProxy maintains a table of mappings from the URLs to the corresponding socket connection to the browser.

HTOTP is a custom protocol which allows multiple HTTP responses to be multiplexed into one or multiple TCP connections. There is a HTOTP Controller module on the ground and remote sides. The HTOTP controller on the remote side is invoked by the connection manager when a HTOTP control packet arrives at the remote side. The HTOPT controller interprets the control packet and interacts with the Stream Controller to return the associated Web resource to the browser. When the scheduler needs to transfer a Web resource to the remote side, it places the resource into one of a set of queues, which are maintained by the Ground Resource Repository. These queues are used to separate the traffic so that, each TCP connection, carries a similar amount of data. The scheduler places resources in these queues in a round-robin fashion. The HTOTP controller on the ground side has a separate thread of execution for each of these queues. A given thread monitors a particular queue in the Ground Resource Repository and transfers resources in this queue to the remote side over an associated TCP connection.

The Stream Controller module maintains blocking access to the Web resources for each GET request from ROptProxy as follows:

—If a particular Web resource is available in the Remote Resource Repository then the resource is streamed directly into the browser's corresponding socket connection.

—If the resource is on the way from the ground side then the browser's socket connection is blocked on the read end of a Linux pipe until it arrives. When the resource starts to arrive the HTOTP controller streams the resource directly through the write end of the associated Linux pipe to the corresponding browser connection.

—If a resource arrives to the HTOTP controller and the browser has not yet opened a socket connection for this resource then the resource is stored temporarily into the Remote Resource Repository.

3.3 Web Page Load Time Optimizations

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

—The expensive three-way handshake over the satellite link is eliminated because initial GET requests are sent over the permanent TCP connections. It is only at the ground side that the TCP connections are opened to the web server.

—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.

—The compression assists in reducing the amount of traffic crossing the satellite link.

3.4 TCP Optimizations for HTTPEP

The HTOTP controller currently uses multiple TCP connections to push web content to the remote side. However, we are motivated to use less TCP connections to reduce the data overhead associated with using multiple TCP connections. But, the reason we use multiple TCP connections is to overcome the data throughput issues associated with TCP slow start. Therefore, we require an alternate mechanism to overcome slow start than using multiple TCP connections. A second motivation for using less connections is that most of the nested web content will be available at the ground side at the same time. This means that we will be pushing a lot of web content in one block to the remote side. By minimizing the number of concurrent connections during this process we will get a more controlled growth of the congestion window.

One solution would be to increase the initial TCP congestion window, which would allow the the ground side to push more content initially during TCP slow start over say one connection. 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 ground side, gives a more generic solution than manipulating the initial congestion window directly. In section 4.2, we give a detailed analysis of the transport layer optimizations that are used to reduce the overhead associated with using multiple TCP connections.

4. 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 by default 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 requests crossing the emulated satellite link. The number of GET requests is a function of the number of resources per page.

—Web page load delay, 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 [Mozilla], a cross platform component object model which can be controlled using JavaScript.

The testbed clients are PCs running FireFox, which is controlled through 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 [Group 2010]. 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.

The rest of this section is organized as follows. Section 4.1 gives a performance evaluation of HTTPEP in comparison to a browser. In section 4.2, we evaluate the effectiveness of using multiple TCP connections and different TCP congestion control algorithms in HTTPEP to push HTTP content over the satellite link.

4.1 Performance of HTTPEP in comparison with a browser

Figure 5 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

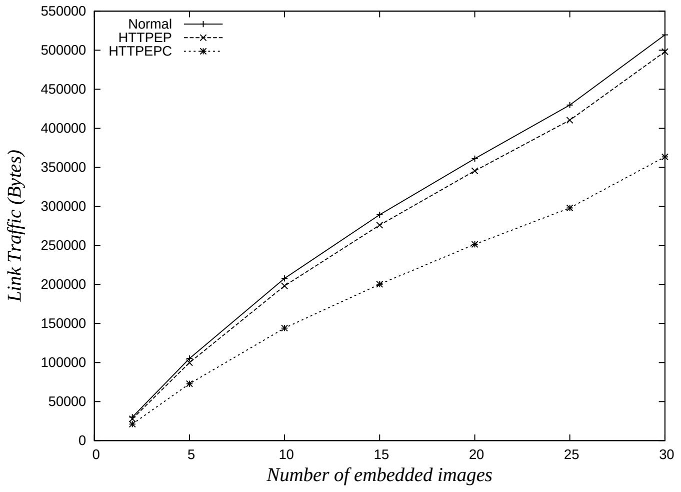

Figure. 5: Traffic crossing the link

between the browser and the web-server without an intermediate proxy), HTTPEP without compression (HTTPEP)[3], HTTPEP with compression of resources (HTTPEPC).

The figure shows a saving in terms of number of bytes crossing the link 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 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 [GraphicsMagick] library. Our investigation has shown that improvements are proportional to the size of the embedded resources.

Figure 6 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 7 plots the page load time for the HTTPEP and normal operation versus the satellite link capacity. In this figure we are interested in evaluating our algorithm in relation to the browser alone. Hence compression is not included. 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 is constrained to 256Kbps the

[3]The HTTPEP results include HTTP header compression

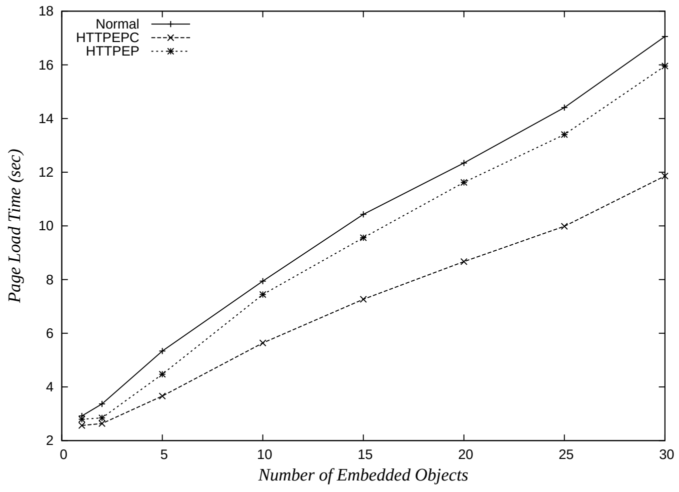

Figure. 6: Average Page time delay

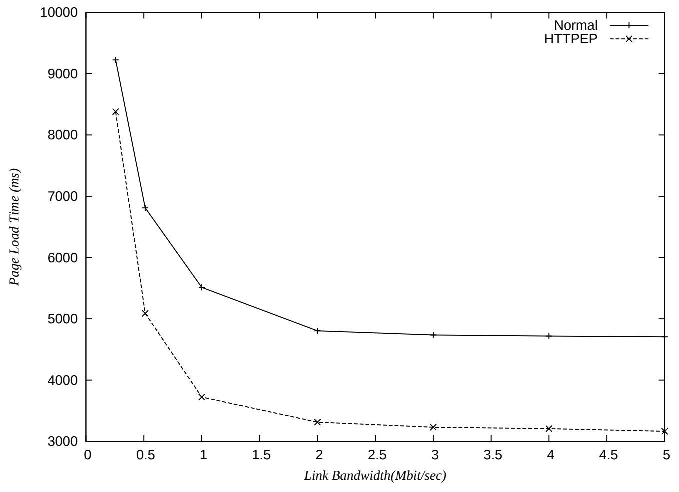

Figure. 7: Effect of varying the bandwidth

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.

4.2 Optimization of the transport layer for HTTPEP

We used eight TCP connections for HTTPEP between the ground and remote nodes for the evaluation presented in section 4.1. In this section, we present a performance evaluation of various transport layer configurations for our particular algorithms. Hence, we only include results for HTTPEP without any compression. We are motivated to reduce the bandwidth overhead, the

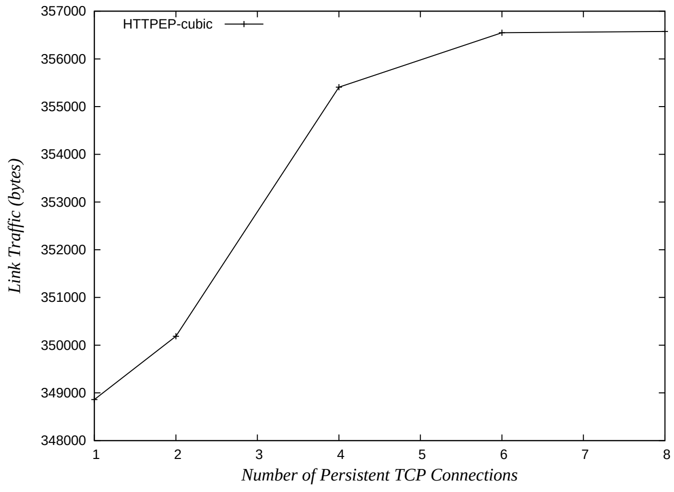

Figure. 8: Bandwidth usage for varying number of TCP connections

end-user's web page load delay, and the number of TCP connections that we use at the computing limited remote node. Because we use persistent TCP connections, there will be no reduction in data overhead associated with TCP session startup and close down when we move from say eight to one TCP connections.

For these tests, a set of web pages are built according to the statistics about the size, number of embedded objects on the world wide web [Ramachandran and Jain 2011]. The web page size was set to 320 KB with forty embedded resources.

Figure 8 shows the data overhead associated with using a varying number of TCP connections for HTTPEP. The TCP configuration, used for this test is the standard Linux TCP cubic with the initial congestion window set to three. There is 10KB less data transferred over the satellite link for one TCP connection in comparison to using eight connections. This reduction in bandwidth usage is due to a decrease in the number of ACKs sent by the receiver. Also, there will normally be a TCP segment transmitted at the end of a data transmission session, which is less than the maximum segment size. When eight TCP connections are used, there will be eight such smaller segments transmitted, whereas, when one TCP connection is used there is only one such TCP segment transmitted.

In figure 9, we plot the end-user's page load time when the initial congestion window is set to 10 (CWND-10 TCP cubic) at the ground node versus the Linux standard configuration where the CWND is set to 3 (CWND-3 TCP cubic). The figure shows that modifying the initial congestion window gives a large improvement in page load time for a small number of TCP connections. As the number of connections increase the improvement degrades rapidly. Thus, we conclude that by using one TCP connection with a high initial congestion window gives a comparable performance to using eight connections. However, it is difficult to determine an appropriate value for the initial congestion window independent of the congestion algorithm utilized. Thus, the performance of HTTPEP was evaluated when different TCP variants Hybla, BIC, Vegas and Westwood are used at the ground node, so as to give a more generic solution than manipulating the congestion window directly. Each of the variants uses a different congestion control algorithm during slow start. Figure 10 shows that Hybla gives the best performance for low bandwidth links while Vegas gives the worst performance of the four TCP variants that we evaluated. These results with regard to TCP variants over satellite, confirm the work done by other researchers -

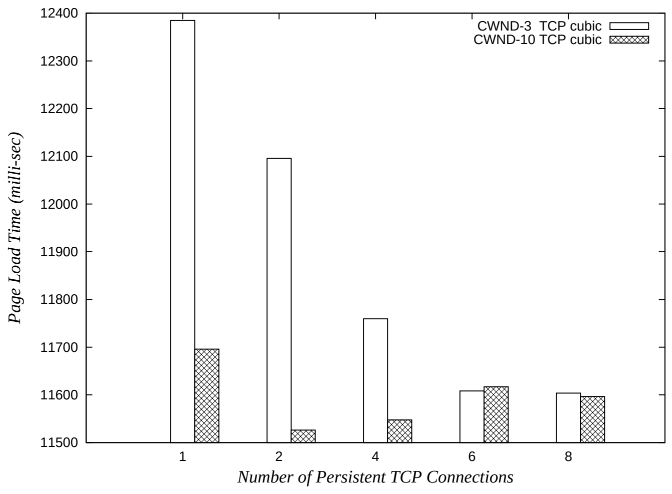

Figure. 9: Performance with initial CWND change

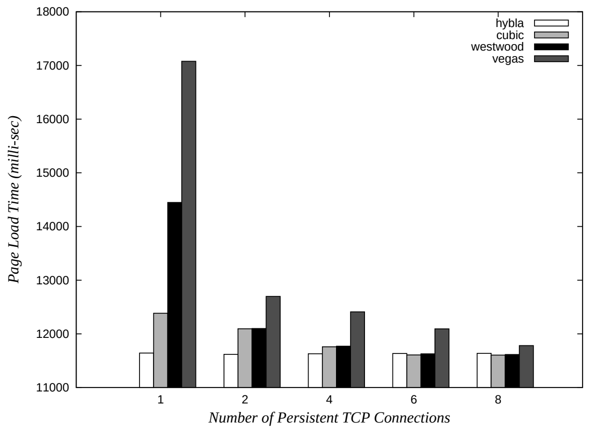

Figure. 10: Performance of TCP variants

see [Caini et al. 2009] for more details.

We conclude from these results, that the use of one TCP Hybla connection at the ground node, will give comparable performance in terms of web page load time for the end-users, as using eight TCP connections. The use of one TCP connection will reduce the computing resources required at the remote node, which is normally a resource limited node. Also, there will be a reduction in bandwidth usage when one TCP connection is used as against using the Linux standard eight TCP cubic connections.

5. CONCLUSIONS AND FUTURE WORK

A novel HTTP PEP (HTTPEP) is described 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 show that, optimizing the transport layer to the communication mechanism of a HTTP PEP gives further benefits in terms of a reduction in bandwidth usage and computing resource utilization.

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

REFERENCES

Adami, D., Marchese, M., and Ronga, L. 2001. TCP/IP-based multimedia applications and services over satellite links: experience from an ASI/CNIT project. *IEEE Personal Commun. 8,* 3, 20–27.

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

Allman, M., Hayes, C., Kruse, H., and Osterman, S. 1997. TCP performance over satellite links. In *5th International Conference on Telecommunication Systems.*

Altobridge. 2011. Altobridge data-at-the-edge. http://www.altobridge.com/products/altobridge-data-at-the-edge

Border, J., Kojo, M., Griner, J., Montenegro, G., and Shelby, Z. 2001. RFC3135: Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations.

Broadband Internet Access. mGuard. http://www.broadband-internet-access.com/.

Caini, C., Firrincieli, R., and Lacamera, D. 2009. Comparative Performance Evaluation of TCP Variants on Satellite Environments. *Communications, 2009. ICC '09. IEEE International Conference on*, 1 –5.

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

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

Carlo and Firrincieli, R. 2004. TCP Hybla: a TCP Enhancment for heterogeneous networks. *International Journal of Satellite Communications and Networking 22.*

Chai, W. K., Karaliopoulos, M., and Pavlou, G. 2007. 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*. Springer-Verlag, Berlin, Heidelberg, 17–29.

Chakravorty, R., Clark, A., and Pratt, I. 2005. Optimizing web delivery over wireless links: design, implementation, and experiences. *Selected Areas in Communications, IEEE Journal on 23,* 2, 402 – 416.

Cormode, G. and Krishnamurthy, E. 2008. Key differences between web1.0 and web2.0.

Cruickshank, H., Giambene, G., Mort, R. J., and Berioli, M. 2009. BSM Integrated PEP with Cross-Layer Improvements. *International Workshop on Satellite and Space Communications (IWSSC 2009).*

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

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

Floyd, R., Housel, B., and Tait, C. 1998. Mobile Web access using eNetwork Web Express. *Personal Communications, IEEE 5,* 5, 47 –52.

Giambene, P. C. G., Bartolini, D., Luglio, M., and Roseti, C. 2006. Dynamic resource allocation based on a TCP-MAC cross-layer approach for DVB-RCS satellite networks. *International Journal of Satellite Communications and Networking 24,* 5, 367–385.

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

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

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

Kruse, H., Allman, M., Griner, J., and Tran, D. 2001a. Experimentation and modelling of HTTP over satellite channels. *International Journal of Satellite Communications 19,* 1, 51–68.

Kruse, H., Allman, M., Griner, J., and Tran, D. 2001b. Experimentation and modelling of HTTP over satellite channels. *International Journal of Satellite Communications 19,* 1, 51–68.

Marchese, M., Rossi, M., and Morabito, G. 2004. PETRA: performance enhancing transport architecture for Satellite communications. *Selected Areas in Communications, IEEE Journal on 22,* 2, 320 – 332.

Mattson, R. and Ghosh, S. 2007. A performance analysis of page retrieval with HTTP-MPLEX on asymmetric links. In *Telecommunications and Malaysia International Conference on Communications, 2007. ICT-MICC 2007. IEEE International Conference on.* 466 –471.

Mattson, R. L. R. and Ghosh, S. 2009. Http-mplex: An enhanced hypertext transfer protocol and its performance evaluation. *J. Network and Computer Applications 32,* 4, 925–939.

Mozilla. What is HTTP Pipelining. http://www.mozilla.org/projects/netlib/http/pipelining-faq.html.

Mozilla. Xpcom. https://developer.mozilla.org/en/XPCOM.

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

Padmanabhan, V. N. 1998. Addressing the Challenges of Web Data Transport, Ph.D thesis, University of California at Berkeley.

Ramachandran, S. and Jain, A. 2011. Web page stats: size and number of resources. http://code.google.com/speed/articles/web-metrics.html.

Riverbed. 2011. Extreme Savings: Cutting Costs with Riverbed. http://www.riverbed.com/us/index.php.

Rodriguez, P. and Biersack, E. W. 2002. Bringing the web to the network edge: large caches and satellite distribution. *Mob. Netw. Appl. 7*, 67–78.

Rodriguez, P. and Fridman, V. 2004. Web content caching and distribution. Kluwer Academic Publishers, Norwell, MA, USA, Chapter Performance of peps in cellular wireless networks, 19–38.

Sadre, R. and Haverkort, B. R. 2008. Changes in the web from 2000 to 2007. In *Proceedings of the 19th IFIP/IEEE international workshop on Distributed Systems: Operations and Management: Managing Large-Scale Service Deployment.* DSOM '08. Springer-Verlag, Berlin, Heidelberg, 136–148.

Squid. http://www.squid-cache.org.

Swen, B. In Proceedings of WebNet, 2001. Speeding Up the Web Using the Web++ Framework.

Technologies, A. 2009. Akamai reveals 2 seconds as the new threshold of acceptability for ecommerce web page response times. http://www.akamai.com/html/about/press/releases/2009/press_091409.html.

Wills, C. E., Mikhailov, M., and Shang, H. 2001. 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*. ACM, New York, NY, USA, 257–265.

XipLink. 2011. Xiplink hub optimizer. http://www.xiplink.com/.

Paul Davern is a post-doctoral researcher in the Mobile & Internet Systems Laboratory (MISL) at University College Cork (UCC) in Ireland. He previously worked as a lecturer in Computer Science at Cork Institute of Technology (CIT). His current research interest is the optimization of Internet protocols. He has a Ph.D. in Computer Science from Dublin City University.

Noor Nashid is a Software Engineer at the IBM Tivoli Software Group, Cork,Ireland. He is expecting to obtain his Masters by Research degree from the Department of Computer Science, University College Cork (UCC) by November, 2011. He has previously worked as a System Engineer with Service Network Operations (SNO) and System Automation (SA) group in Grameenphone, Telenor, Bangladesh. He has received his B.Sc. in Computer Science and Engineering from the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology (BUET) in 2007. His research interests include content-centric networking, ubiquitous computing, transport and routing protocols for Internet and wireless networks, security and trust management, power and performance aware efficient resource management in cloud computing, and methods and tools for reliable software technology.

Cormac J. Sreenan is Professor of Computer Science at University College Cork (UCC) in Ireland. Prior to joining UCC in 1999 he was on the research staff at AT&T Labs - Research, Florham Park, NJ, and at Bell Labs, Murray Hill, NJ. At UCC he directs the Mobile & Internet Systems Laboratory (MISL), which is a group of over 12 research staff and students with research activity in multimedia and wireless networking and systems. Prof. Sreenan is currently on the Editorial Boards of ACM/Springer Multimedia Systems Journal, IEEE Transactions on Mobile Computing, and ACM Transactions on Sensor Networks. In the past he has served as Guest Editor for Communications of the ACM, IEEE Journal on Selected Areas in Communications and IEEE Wireless Communications Magazine. He has a Ph.D. in Computer Science from Cambridge University and is a Fellow of the British Computer Society.

Ahmed H. Zahran is an assistant professor at the Electronics and Electrical Communications department, Faculty of Engineering, Cairo University. He previously worked as a post-doctoral researcher with the MISL group at the Computer Science Department, University College Cork, Ireland. He obtained his PhD at the Department of Electrical and Computer Engineering, University of Toronto in 2007. He received his BSc and MSc in Electrical Engineering from Electronics and Electrical Communication Department at Faculty of Engineering, Cairo University in 2000 and 2002 respectively. His research interests span different topics in wireless mobile networking such as network architecture, mobility and resource management, and modeling and performance evaluation.