

Performance of PEPs in Cellular Wireless Networks

Pablo Rodriguez Vitali Fridman*
Microsoft Research, Cambridge. UK

Abstract

Wireless networks all over the world are being upgraded to support 2.5 and 3G mobile data services. GPRS and UMTS networks in Europe, and CDMA 1xRTT and CDMA 2000 networks in the USA and Asia are currently being deployed and tested to provide wireless data services that enable ubiquitous mobile access to IP-based applications. In recent years a lot of attention has been paid to improving the physical and the MAC wireless layers. Thus, advances in coding mechanisms, FEC, and scheduling algorithms have been critical for the success of these technologies. However, despite the advances in layer-2 techniques and the momentum behind these networks, surprisingly little attention has been paid to evaluate application performance over cellular networks and how the different protocol stacks interact with the wireless bearer.

In this paper we identify the main performance problems experienced by transport-level, session-level and application-level protocols in Cellular wireless networks. We present our practical experience results, and investigate how a Wireless Performance Enhancing Proxy can improve the performance over these networks.

1 Introduction

GSM networks all over the world are being upgraded to support the General Packet Radio Service (GPRS). GPRS provides wireless data services that enable ubiquitous mobile access to IP-based applications. In addition to providing new services for today's mobile user, GPRS is important as a migration step toward third-generation (3G) networks. GPRS allows network operators to implement new IP-based mobile data applications, which will continue to be used and expanded for 3G services. In addition, GPRS provides operators with a testbed where they learn about the challenges and technical problems of deploying such a network.

Over the last several years important advances have been made in the lower layers of the wireless protocol stack. As a result, new modulation protocols, smarter schedulers, and optimized error correction techniques were introduced. Despite the advances in the wireless data link and MAC layers, little attention has been paid to evaluation of the performance of the higher layers of the protocol stack, e.g., TCP, HTTP, etc. Optimized wireless networking is one of the major hurdles that mobile computing must solve if it is to enable ubiquitous access to networking resources. However, current data networking protocols have been optimized primarily for wired networks and do not work well in wireless networks. GPRS wireless environments have very different char-

acteristics in terms of latency, jitter, and error rate as compared to wired networks. As a result applications experience unnecessary losses, bursty behavior, slow response times, and sudden losses of connectivity. This mismatch between traditional wireline protocols and the wireless bearer can significantly reduce the benefits provided by link layer protocols. Accordingly, traditional protocols have to be adjusted and tuned to this medium.

Some of the performance problems observed in GPRS networks have also been observed to some extent in other long-thin or long-fat networks (e.g., Satellite, WLANs, Metricon Ricochet). However, little in depth analysis has been done about the real underlying problems that impact data performance over GPRS. In this paper we investigate the performance of layers 4-7 of the protocol stack over GPRS networks. We first analyze the characteristics of a GPRS network and then study the performance issues of TCP, DNS and HTTP.

To overcome the problems posed by wireless links we introduce a **Wireless Performance Enhancing Proxy** (W-PEP) architecture that attempts to handle the underlying characteristics of GPRS wireless links. The W-PEP sits at the edge of the wireline network, facing the wireless link, and monitors, modifies, and shapes traffic going to the wireless interface to accommodate the wireless link peculiarities. W-PEP implements a number of optimizations at the transport, session and application layers that overall provide a significant improvement to the end user experience. It can also include caching, logging and billing subsystems. Some of these W-PEP proxies have been recently deployed in wireless commercial networks primarily to enhance remote Web access for business users using their laptops or PDAs. However, little data has been published about the real performance improvement provided by these W-PEPs. In this paper we present one of the first studies of the real benefits of these proxies in 2.5G GPRS networks.

The rest of the paper is organized as follows. The next section discusses related work. In Section 3 we present a brief overview of cellular wireless networks. Section 4 discusses the architecture of the Wireless Performance Enhancing Proxy. Section 5 describes the latency components of a typical Web transfer in a wireless link. Section 6 presents transport and session level optimizations. Section 7 discusses application level optimizations and Section 8 compares then with transport and session level optimizations. Finally, Section 9 briefly discusses the impact of different proxy deployment strategies and we conclude the paper in Section 10 with some future work discussion.

2 Related Work

Previous solutions to improve data delivery in wireless data networks have been proposed at the physical, link and MAC layers [1, 2, 3], at the transport layer (TCP optimizations) [4, 5, 6, 7, 8,

*pablo@microsoft.com, vitali@fridman.name

9] as well as at the application layer (data compression) [10, 11]. In the literature, physical/link/MAC layer enhancements have been proposed that aim to provide improved scheduling algorithms over wireless links to increase the total system throughput [12, 13], provide fairness or priorities between the different users, assure minimum transmission rates to each user [13, 14] and incorporate forward error correction on the link to reduce retransmissions. The scheduling algorithms aim to control the system or user throughput at the physical layer. For data applications, it is equally important to consider the data performance at higher layers in the protocol stack, especially at the transport (TCP) layer. It has been observed that due to lower-layer retransmissions, channel condition changes, handoff delays or priority scheduling the round trip time for TCP packets can abruptly increase and lead to delay spikes over wireless links [7]. These delay spikes may cause TCP timeouts, which triggers the congestion control mechanism in TCP, leading to a decreased congestion window size and consequently low throughput performance [7, 8, 9]. Delay jitter algorithms have been proposed to mitigate the effect of delay spikes. Techniques such as the ACK Regulator [7] has been proposed to monitor and control the flow of acknowledgment packets in the uplink channel and therefore regulate the traffic flow in the downlink channel of a wireless link. The motivation for these solutions is to avoid buffer overflow and the resulting congestion avoidance mechanism of TCP. Similarly [15] proposes to avoid slow-start and congestion avoidance all together by clamping the TCP window to a static estimate of the Bandwidth Delay product of the link. At the application layer several data compression techniques have been proposed [10, 11] to increase the effective throughput of wireless links. Examples include degrading the quality of an image, reducing the number of colors, compressing texts, etc.

Several proxy based protocols have been proposed to fix TCP problems over wireless links. Snoop [16] is a TCP aware link-layer scheme that ‘sniffs’ packets in the base station and buffers them. If duplicate acknowledgements are detected, incoming packets from the mobile host are retransmitted if they are present in a local cache. I-TCP [17] splits the TCP connection and uses TCP over the wireless link albeit with some modifications. W-TCP [18] proposes to split the connection at the base station. Other more recent papers suggest the use of a small proxy at the mobile host combined with a proxy on the wireline network to implement a new transport protocol that replaces TCP [19].

Despite the large amount of work in this area, most of the research in wireless performance enhancing proxies has focused on solving the impact of highly lossy wireless networks on TCP. However, as we will see in this paper, current 2.5G and 3G networks provide very strong link-level reliability mechanisms that hide losses from the higher protocol layers. This suggests the need for revisiting the use of performance enhancing proxies in next generation wireless networks. In this paper we consider the impact of Wireless Performance Enhancing Proxies in current 2.5G and 3G wireless networks. We consider the range of optimizations that can be implemented at the W-PEP at different layer of the stacks, e.g., transport, session, and application. We first discuss the existing problems and then we propose possible solutions that we validate by using data from real wireless deployments.

3 Overview of Cellular Networks

GPRS networks represent an evolution of GSM networks and are considered to be an intermediate step towards 3G networks, e.g., UMTS. That is why GPRS networks are given the name of 2.5G. While GSM networks are mostly a European phenomenon, similar efforts are under way in the USA and in Asia. For instance, CDMA 1xRTT is the equivalent of GPRS in the USA market. CDMA 1xRTT is supposed to evolve to the equivalent 3G standard, CDMA 1xEV-DO, at about the same time when GPRS is expected to evolve to UMTS. Despite the difference in the cellular wireless standards in different countries, all these networks share a similar set of problems. To better understand the problems posed by these networks we next show the result of a set of pings through a live commercial GPRS network in Europe. Similar results were obtained in other networks both in Europe and the USA.

```
Pinging www.microsoft.com [192.11.229.2] (32 bytes):
Reply from 192.11.229.2: bytes=32 time=885ms TTL=109
Reply from 192.11.229.2: bytes=32 time=908ms TTL=109
Reply from 192.11.229.2: bytes=32 time=820ms TTL=109
Reply from 192.11.229.2: bytes=32 time=4154ms TTL=109
Reply from 192.11.229.2: bytes=32 time=870ms TTL=109
Reply from 192.11.229.2: bytes=32 time=1090ms TTL=109
Reply from 192.11.229.2: bytes=32 time=795ms TTL=109
...
Ping statistics for 192.11.229.2:
Packets: Sent = 80, Received = 80, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
Minimum = 761ms, Maximum = 4154ms, Average = 1001ms
```

As we can see the average RTT delay measured by a ping request is equal to one second. Moreover, the variability in the delay is quite extreme, ranging from 761 msec to 4 sec. An important point to note is that there were *zero losses*. A deeper analysis using tcpdump also revealed *zero out of order* packets. The main reason for this is the strong link-layer reliability implemented by cellular wireless networks. Cellular wireless networks implement ARQ retransmissions, FEC, and other related techniques to ensure that packets are not lost or misordered by the air interface. As an example, link-layer GPRS protocols provide 10^{-9} packet loss rates and similar values for out-of-order packet reception. The fact that reliability is implemented at the link layer ensures that higher layers see practically no losses or packets out of order. However, the drawback is that the variability and value of the wireless RTTs increases drastically. This is an inherent problem of cellular wireless links that will not go away with the arrival of 3G wireless networks.

Another problem is the fact that these wireless networks have a very low throughput. For instance GPRS networks provide a throughput of 15-25 Kbps while CDMA 1xRTT provide a throughput of 50-70 kbps for HTTP traffic. The fact that the throughput is so low is due to a number of factors. First there is a direct relationship between the long and variable RTTs described before and the throughput. Second, there is an overhead associated with the fact that multiple protocol layers are used over the wireless interface. In a GPRS network the protocol stack usually consists of many layers, which add significant overhead: link layer → link layer reliable protocol → link layer mobility protocol → ppp → ip → tcp → application. Finally, there is a significant overhead due to the fact that information needs to be processed at multiple nodes in the carrier’s network, e.g., Base Sta-

tion, Base Station Controllers, SGSNs (Serving GPRS Support Node), GGSNs (Gateway GPRS Support Node), WAP Gateway, etc. All these factors play a significant role in limiting the maximum throughput that can be obtained through a wireless link.

These link layer problems require an in-depth study of their impact on the higher layer protocols. As we will see in this paper, a careful study of these problems and a set of intelligent optimizations techniques can overcome many of the problems that plague GPRS links.

4 Wireless PEP

Wireless Performance Enhancing Proxies (W-PEP) are required to minimize the big performance mismatch between terrestrial and wireless links. By splitting the connection between the terrestrial and the wireless side into two different connections, W-PEPs can significantly improve end-to-end protocol performance. Previous work in wireless performance enhancing proxies assumed high losses in the wireless links, however, 2.5 and 3G links provide reliable and orderly delivery by implementing link-layer reliability. As a result we consider it crucial to revisit the concept of W-PEPs, taking into account the peculiarities of 2.5 and 3G wireless networks.

The W-PEP can be located in multiple places in the network. The most natural place is to collocate the W-PEP with the GGSN since this the first IP node where all IP traffic is anchored from the Base Station System (BSS) (Figure 1). Other locations could also provide interesting benefits and a more distributed architecture (e.g., at the SGSN, at the Base Station). However, deploying the W-PEP at these locations would need substantial work since it requires accommodating a layer 3-7 proxy between layer-2 nodes. The W-PEP can be deployed using an explicit proxy configuration or a transparent proxy configuration plus a layer-4 switch. Later in the paper we will describe the benefits and drawbacks of each type of deployment. W-PEPs provide a number of enhancements that improve the overall end-to-end user experience. These optimizations cover transport layer optimizations, session level optimizations and application level optimizations.

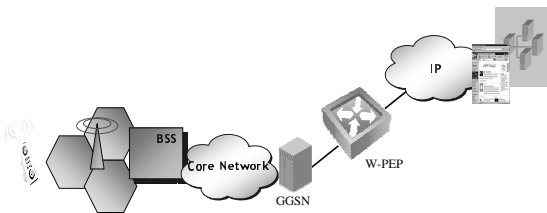


Figure 1: W-PEP network architecture

Figure 2 shows the different layers of optimizations that can be used to improve data performance across wireless links. At the bottom of the stack solutions have been proposed to improve the physical/link layer. These solutions are aimed at providing better scheduling algorithms over the wireless links to increase the total system throughput, provide fairness or priorities between the different users, assure minimum transmission rates to each user, and incorporate forward error correction techniques on the link to minimize retransmissions. These optimizations are usually implemented at base stations or base station controllers. However,

for data applications, it is equally important to consider the data performance at higher layers of the protocol stack, especially at the TCP, session, and the application layer.

TCP optimizations try to minimize the impact that the wireless link has on the TCP behavior. These optimizations are targeted to minimize the impact of long and highly variable delays, spurious timeouts, buffer overflows, temporary disconnections, etc. The session layer optimizations try to minimize the amount of DNS lookups through the air interface and ensure that TCP sessions are kept alive over multiple downloads. Similarly, application level optimizations include several data compression and transformation techniques to increase the effective throughput of the wireless link. Examples include degrading the quality of an image, reducing the number of colors, compressing text, etc. One important characteristic of W-PEP is that it does not require any modifications to the client or server stacks and performs all the optimizations transparently. This makes it very easy to deploy and avoids the hassle of having to provide and maintain new purpose build client software. However, we point out those optimizations that would benefit more from having a special client-software.

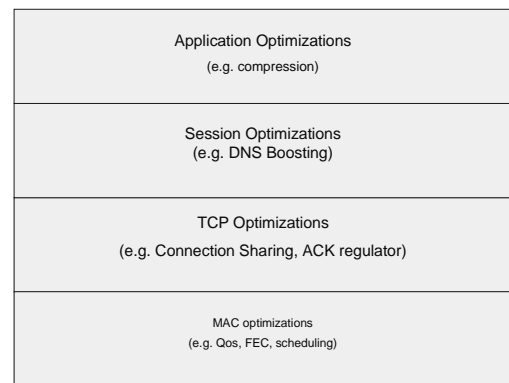


Figure 2: Protocol Stack Optimizations

In the rest of the paper we will show the performance benefits of a W-PEP implementation. This implementation runs on Solaris 8 and enables TCP and session optimizations, application-level optimizations, and caching. The W-PEP was tested on a GPRS cell where several cell parameters could be configured. The primary cell parameters available for configuration were the number of time-slots available for GPRS traffic, the signal to noise ratio, the amount of background traffic, and the duration of the periods with no signal. For most experiments we considered a typical average loaded GPRS cell with the following parameters: 18 dB of SNR, phone with four downlink slots and 2 uplink slots, background traffic of voice calls with 60 seconds of active time and 15 seconds of inter-arrival time exponentially distributed, 10 background users, and no data slots being reserved exclusively for GPRS traffic.

To remove dependency on the real Internet and its unpredictability, we downloaded and hosted all necessary objects on our own web server. We also hosted our own DNS server with all necessary records to reproduce the exact setup of the target Web pages.

5 Latency Components

To have a better understanding of the different factors that determine the overall latency to download a document, we have dis-

aggregated the total download time of a document into various components. To do this we use tcpdump traces. We try to determine the impact of: a) the time to resolve DNS names; b) the time to establish TCP connections; c) the idle times between the end of the reception of an embedded object and the beginning of the reception of the next one; and d) the transmission time of the data payload. We do not explicitly consider the processing time of the server since we assume that in a wireless link, the server processing time is negligible compared to the other latency components. Figure 3 shows the GPRS latency components for the top-level page of the 10 most popular web sites. These Top 10 pages are described in table 6. From this Figure we see that time to deliver the payload accounts for most of the latency (about 65%). This is an expected result since the throughput of the GPRS link is quite low. The idle RTTs in between GET requests for embedded objects account for the second largest latency component, especially in pages with many objects. These idle times represent a significant overhead since the RTT through wireless links can be quite high. HTTP headers also account for a large overhead since many objects in Web pages tend to be quite small and the Header sizes could potentially be on the same order as the size of the object itself (see Section 7.1).

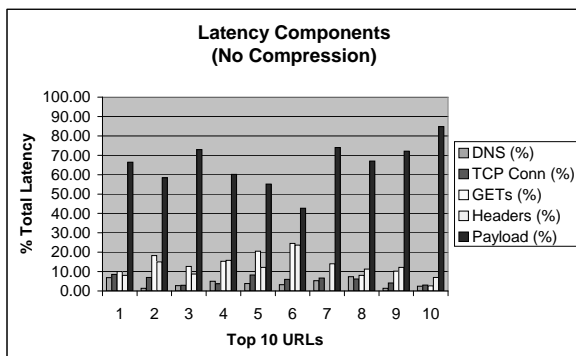


Figure 3: Latency Distribution (No compression)

At the transport and session layer, DNS queries and TCP connection setups account for the smallest portion of the overhead. Most of these overheads are barely noticeable in terrestrial links with quite small RTTs. However, in GPRS links where RTTs are in the order of seconds, these overheads can significantly affect the end-user experience. In the next sections we describe in more detail these overheads and identify solutions for some of these problems.

6 Transport/Session Optimizations

TCP faces several problems in GPRS links. Given the Radio Link Protocol (RLP) link-layer retransmissions, TCP sees very large and variable delays. On top of this, the throughput of the wireless link is very small. The two possible options to deal with these problems are to tune/alter the TCP/IP networking stack at an intermediate node only (i.e., W-PEP), or to change/replace the networking stack at the mobile host and intermediate node. The latter approach may be based on a completely new transport protocol (e.g., based on UDP) that handles the particularities of wireless links and replaces TCP. This new transport protocol should provide reliability, congestion control, flow control, and fairness.

However, TCP has proven to be a very flexible and robust protocol that can be adapted to many types of networks. Creating a completely new transport protocol may not prove to be the most efficient solution since one may end up re-implementing most of the TCP features. Therefore, in this paper we propose solutions that do not replace TCP but instead optimize it for cellular wireless links. These type of solutions can be implemented in an intermediate proxy such as the W-PEP and do not require any modifications to TCP/IP stack on either end of connection (servers or mobile clients).

Before describing in detail the problems and solutions for TCP performance in wireless links, we would like to have a rough estimate of how much improvement we should expect from TCP optimizations. From Figure 3 we can estimate the best case scenario for TCP and session-level optimizations. By eliminating TCP setup, and removing DNS lookups, the best improvement possible is about 18%. As we will see later, this number goes up to 30% once the content is compressed since the relative impact of TCP setup and DNS queries increases (See Figure 8). Next we will consider how to deal with specific TCP problems in more detail.

6.1 TCP Tuning

To better understand the problems of the TCP performance in wireless links we run the following experiment. We first calculate the FTP throughput obtained when downloading a large file. Then we compare it with the throughput obtained when downloading a Web page of the same size using HTTP (the page had 10 embedded objects). The results can be seen in table 1.

Table 1 shows that the throughput provided by an FTP transfer of a large file is quite close to the maximum TCP throughput than can be achieved through the wireless link. The main reason for this is that FTP uses a single connection and therefore TCP connection setup overhead is very small. This result also shows that the long-term throughput provided by the tested Solaris TCP implementation is very good. We tested other OS stacks (e.g., Linux) and the performance was slightly worse, however, it became comparable when tuning several important TCP parameters in the manner that we will describe in this section.

When we compare the FTP throughput of a large file with the HTTP throughput of a size-equivalent Web page, we see that the HTTP throughput drastically drops from 37 – 39 Kbps to 21 – 28 Kbps. This is due to the large number of TCP connections opened by the browser and the DNS lookups needed to resolve the domain names of embedded objects. In addition to the TCP and DNS problems, HTTP also suffers from the fact that there are multiple idle RTTs in between object requests since pipelining is not enabled. We will later discuss in more depth the impact of pipelining.

Table 1: TCP Throughput

	Throughput (Kbps)
Maximum Airlink GPRS Rate	53.6
Maximum TCP Throughput Rate	43-45
FTP Rate	37-39
HTTP Rate	20-28

As discussed earlier in the paper, we will use a standard TCP/IP stack and tune it to optimize TCP performance in GPRS net-

works. Different stacks require different levels of tuning, however, the optimal combination of TCP parameters should be the quite similar for different OS implementations. In order to tune a given TCP stack we consider critical parameters that could potentially impact the TCP wireless performance. These parameters include the maximum transfer unit size, the slow start window, and several timers that determined the retransmission timeouts calculation:

- a) A careful selection of the MTU allows for efficient link utilization. Selecting a small MTU increases the chance of a successful transmission through a lossy link; however, it also increases the overhead of header to data. Selecting a large MTU, on the other hand, increases the probability of error and the delay to transmit one segment. However, wireless links such as GPRS and CDMA provide low-level techniques that strongly reduce the probability of loss. Having large MTU has several benefits, such as a smaller ratio of header overhead to data and a rapid increase of TCP's congestion window (TCP's congestion window increases in units of segments). As a result, a large MTU (*tcp_mss_def* about 1460 bytes) may prove to be quite beneficial to optimize TCP in GPRS networks.
- b) Traditional slow start, with an initial window of one segment, is too slow over wireless networks. Slow start is particularly harmful for short data transmissions [20], which is commonly the case for web traffic. Increasing the initial slow start window (*tcp_slow_start*) to three to four segments may improve the transfer time significantly and therefore it is highly recommended as a possible TCP optimization.
- c) An correct estimation of a connection timeout (RTO) is very important since it may lead to unnecessary segment retransmissions and a sharp decrease in TCP throughput. To estimate RTO, the sender uses periodic measurements of the RTT between the client and the server. However, it takes several RTTs before the sender can efficiently estimate the actual delay of the connection. In between, many packets can be retransmitted inefficiently. One way to minimize spurious retransmission due to incorrect RTT estimation is to set the initial values of the RTT closer to its real values in the wireless link. Thus, parameters such as *tcp_rtx_interval_initial* and *tcp_rtx_min* can play an important role in optimizing TCP performance.

To have a better understanding of the impact of a wrong estimation of the retransmission timer, in Figure 4 we show a TCP trace for a page download from a standard Web server which parameters have not been tuned for GPRS links. We focus on the number of retransmissions that were received at the client (marked by an *R*). The results show that at the beginning of the download the Web server has a completely wrong (too short) estimation of the RTT, thus, it generates many unnecessary retransmissions. As the download progresses the retransmissions disappears since the RTT estimation improves. However, it takes about 30 seconds for the Web server to properly estimate the parameters of the wireless connection and stop sending unnecessary retransmissions. This overhead can be catastrophic for short Web transfers. We obtained similar results with other pages and found that in many situations the number of retransmissions accounted for up to 50% of the packets. As we will see later, W-PEP solves this problem.

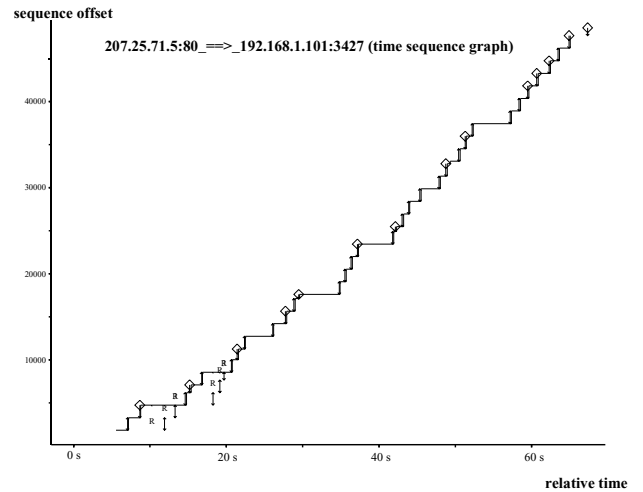


Figure 4: TCP Behavior (W-PEP disabled).

6.1.1 TCP Parameter Selection

To determine the optimal TCP parameter mixture for GPRS links we run an experiment where a 50 KB file is downloaded multiple times with a different combination of the parameters described in the previous section (*tcp_mss_def*, *tcp_slow_start*, *tcp_rtx_interval*, and *tcp_rtx_min*). We used all possible combinations of these parameters with the following input values: *tcp_mss_def* = (Default, 1500, 1400, 1200) Bytes, *tcp_slow_start_initial*, *tcp_slow_start_idle* = (Default, 4, 3, 2) segments, *tcp_rtx_interval_initial* = (Default, 7000, 5000, 3000) msec, and *tcp_rtx_min* = (400, 3000, 5000) msec, where DEFAULT are the default TCP settings of the Solaris TCP stack. The default TCP settings for Solaris 8.2 are: MSS = 536, Slow start initial = 4, Slow start after idle = 4, Retransmission initial = 3000, Retransmission min = 400. We repeated each download 20 times, averaged the results, and compared them with the results obtained using the default TCP parameter setup. Due to space limitations we are not showing here the complete set of results for all combinations.

The measurements showed that most combinations provided little benefit versus the default configuration, however, some of them provided an improvement of up to 17% improvement over the default configuration. The optimal tuning configuration consisted on: MSS = 1400, Slow start initial = 4, Slow start after idle = 4, Retransmission initial = 7000 ms, Retransmission min = 3000ms. These results indicates that large MTUs, large values of the slow start window, and values of the initial and minimum RTT that are closer to those in the wireless links, can significantly improved the overall performance of TCP. In addition to reducing latency, having a better tuned TCP stack also improves bandwidth usage since it decreases the number of retransmissions. To illustrate this point, we now repeat a similar experiment than the one presented in Figure 4 using W-PEP with a tuned TCP stack. The results are presented in Figure 5. We can note that with W-PEP there are no retransmissions, even at the beginning of the connection. We repeated the same experiment with many other Web sites and we always saw a significant reduction in the number of retransmissions. Without W-PEP, the number of retransmissions for certain Web sites sometimes accounted for half of the data delivered, therefore, significantly reducing the available bandwidth and download rates. Using W-PEP, on the other hand, the number of retransmissions due to spurious timeouts was negligible.

As a result, a well-tuned TCP stack can drastically improve the wireless link efficiency.

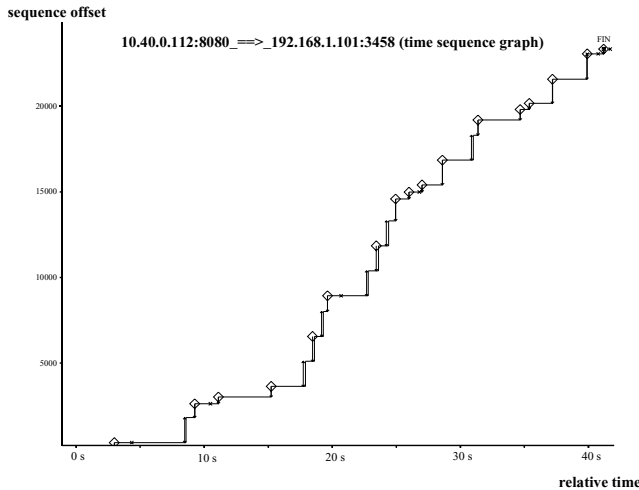


Figure 5: TCP Behavior (W-PEP enabled).

6.2 TCP Connection Sharing

In standard TCP/IP implementations each new TCP connection independently estimates the connection parameters to match an end to end bandwidth and round trip time of the network. The two critical connection parameters that require estimation are RTT and congestion window size. For each connection the estimation process normally starts with the same default values, set in the stack configuration. This estimation process takes several RTTs to converge. In wireless links with very large and highly variable RTTs, estimation process converges very slowly. For short connections typical for Web access, it is common to have most of the traffic flow over connections with sub optimal parameters, therefore underutilizing the available capacity. It can be observed however that RTT and congestion parameters for connections to the same mobile host are likely to have very similar values, primarily determined by device and network capabilities (e.g., number of uplink and downlink time slots), and by current network load and conditions (e.g., current link error rate).

To minimize the amount of time required for TCP to converge to the right connection parameters, we instrumented W-PEP TCP stack to cache RTT and congestion window parameters of currently running or recently expired connections, and to reuse them as a starting values for the new connections *to the same mobile host*. A second advantage of this approach is that connection parameters are effectively estimated over considerably longer period of time, spanning several connections to the same mobile host. This provides for much better estimates of link parameters. We set a timeout before the cached values expire to 2 minutes. If there are no new connections to the same mobile host over this period of time, the cached values are cleared. We found this value to be a good heuristic timeout.

Whenever a new connection is initiated, W-PEP re-uses previously cached connection parameters. The new connection immediately starts sending data at the same rate at which a previous connection had been sending just before it finished. Therefore, we virtually eliminate the overhead of the slow start phase of most connections, the convergence process happens much faster, and estimated connection parameters are more precise.

One drawback of this approach is that at the beginning of a given

connection W-PEP may produce bursts of data that are higher than it would otherwise be the case; however, current GPRS networks provide large buffers that can easily absorb it.

To evaluate the performance of this TCP connection state sharing technique we downloaded the Top 10 pages with and without this feature enabled, and averaged the results over multiple rounds. We used HTTP1.1 persistent connections. The results showed an average improvement of 12% over all pages. This is a significant improvement that can clearly have an effect on end-user experience, and it does not require any modification of the client or server TCP stacks.

6.3 Number of TCP connections

One of the main factors that impacts the performance of TCP through GPRS networks is the number connections opened and closed. Each time a new connection is opened, there is a corresponding connection setup overhead which includes several idle RTTs before TCP is able to fully utilize the available link capacity. Next we study how standard browsers such as Netscape or IE use/reuse TCP connections and what their impact is on GPRS networks. To study this, we considered a page with 25 embedded objects and 142 KB of data that was downloaded by the browser. While the download was happening we collected TCP traces using tcpdump to study the dynamics of TCP. The traces were captured on a Windows 2000 laptop. The address of the laptop was 192.168.1.101. Version of IE used was 5.50. The IE was set to use an explicit HTTP proxy at the address 10.40.0.112 port 8080. "Use HTTP1.1 through proxy connection" (in IE Options) was turned on. The collected traces cover the period time between the initial page request and when all its embedded objects are received in full.

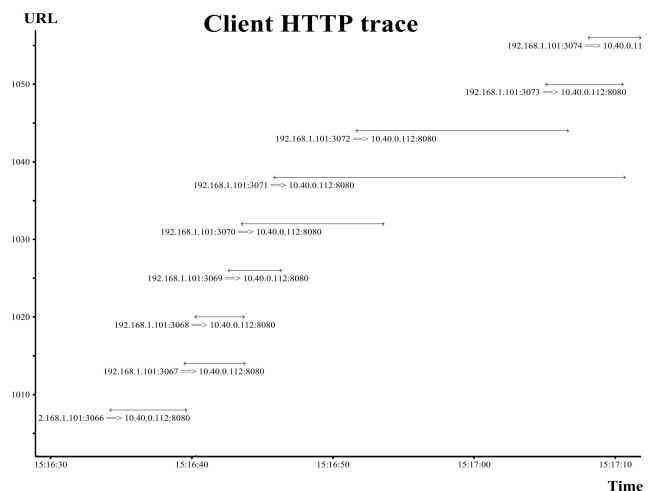


Figure 6: TCP Connection Behavior

Figure 6 shows a time graph of the TCP connections opened and closed by the browser. From this figure we see that the total number of connections used to retrieve the page by the browser is equal to 10 connections (the first connection coincide with the beginning of X axis on the graph, and can only be seen with appropriate magnification of this corner). All connections were closed by the browser and not by the proxy or the server. The number of connections used is very large. According to HTTP1.1 specifications, browsers are supposed to use only two connections to a proxy. In the same Figure we can also note that IE never uses more than 2 connections in parallel, however, it routinely opens

new connections, just to close an "older" connection in lieu of newly opened one.

Table 2: Distribution of TCP connections

	Connections	Objects/Connection
	1	1
	1	15
	1	7
	1	2
	6	1
Total	10	25

Table 2 shows a breakdown of number of URLs served per connection. From this table we see that the bulk of objects is retrieved using 2 or 3 connections, however the rest of the connections are only used to retrieve 1 or 2 objects per connection. In fact, in this experiment, 6 out of 10 connections were used to retrieve only 1 object. We have conducted the same experiment with other Web sites and we observed the same behavior.

To better understand this behavior we analyzed the source code of TCP connection scheduler of Netscape browser. Next we describe the algorithm used when a new object request is received. We assume that the browser is explicitly connected to the proxy, thus, all connections opened to the proxy can be potentially reused to retrieve any object from any Web site.

```

if (num_conn_open < max_conn){
    if (idle_conn & !conn_expired()){
        reuse_conn();
    }
    elseif{
        open_new_conn();
        close_expired_conn();
    }
}
elseif{
    open_new_conn();
    close_oldest_idle_conn();
}

```

The main motivation behind this algorithm is the following. Browsers open multiple connections in parallel to fetch embedded objects as fast as possible. The browser first opens a TCP connection to fetch the home page (i.e., index.html file). If embedded objects need to be fetched, the browser will try to re-use any existing connections that have finished previous downloads (i.e., idle connections) and that have not yet expired (as determined by the keep-alive timeout). If no connections are idle, the browser will open new connections until it reaches the maximum number of connections allowed. For an HTTP 1.1 browser in explicit proxy mode this limit is usually two. Once the browser has reached its maximum number of connections, if a new object request arrives, the browser still creates a new connection. At this point, the browser is exceeding its maximum connection allowance. To balance the number of connections, the browser closes the first connection that becomes idle or any connection that has been active for more than the maximum allowed delay.

This is an aggressive behavior that prevents browsers from getting slowed down by connections that are stalled. In a wireless

network, however, connections take a long time to complete. The browser frequently thinks that the connection is stalled and opens a new connection to try to get the download going as fast as possible. At the same time it closes other pending active connections that according to the browser's algorithm are not progressing at a fast enough rate. This can lead to a behavior where many connections are frequently opened and closed and a single object can take a very long time to complete download since it never gets to progress. This is counter productive in wireless networks since opening and closing many connections creates and extremely high overhead that significantly increases the download times. One way to overcome these problems is to ensure that browsers for wireless networks try to keep a constant number of connections that they re-use them as much as possible, and use connection timers that are in accordance with the delays/throughputs of wireless links. These modifications require slight changes or re-configuration of the client browsers.

Even though the number of connections should be kept constant as much as possible, the number of connections should not be very small to prevent stalled connections from delaying the overall download. In the next experiment we try provide some intuition of why using a very small number of connections may not be very efficient. To this end, we calculate the performance of a file download using one single connection vs. four connections. We consider a large 1 MB file. When using four connections, the file is divided into four equal pieces and all of them are fetched in parallel. The results show that the throughput obtained with four connections is about 11% higher than with a single connection. One reason for this higher throughput is that multiple asynchronous connections can better utilize the wireless link capacity since some connections are able to grab additional bandwidth while others are idle or slow. Another reason why having four connections instead of one reduces the download time is because it minimizes the impact of the idle times in-between requests for embedded objects. The natural way to remove these idle times is through the use of pipelining. However, some browsers do not support it yet, and therefore, parallelizing requests over several connections overcomes the drawbacks of not having pipelining support. We will study further the impact of pipelining later in the paper.

6.4 Temporal Block Flow Release

Wireless Base Station Controllers (BSC) and SGSNs allocate wireless link resources to a particular mobile host only for a certain period of time. After a mobile host is idle for more than a preconfigured period of time, the wireless link resources are released. The logical resource in question is called TBF (temporary block flow) and we will refer to this behavior as TBF release. This improves the utilization of a given GPRS channel since the period of channel inactivity is limited. However, when the mobile host comes back and starts requesting data again it goes through the acquisition of a new TBF and associated time slot. Acquiring a new TBF and time slot is an expensive process that adds an initial latency before data transfer can be initiated. In the next experiment we try to determine the TBF release timeout, i.e., the idle period of time before the mobile host's GPRS channel is released. To this end we periodically ping the W-PEP from the mobile host with increasing pinging intervals. The following table shows that there is a jump in the value of RTT when the time between pings increases from 5 seconds to 6 seconds. This indicates a TBF release timeout of around 6 seconds. This empirical

experiment to determine the TBF value was later confirmed by a number of GPRS vendors.

Table 3: Ping times for different inter-ping intervals

Ping Interval (sec)	Avg. Ping Time (msec)
1	1667
2	1726
3	1664
4	1778
5	1726
6	2110
7	2551
8	2345
9	2123
10	2314

A TBF equal to 6 sec can create very frequent wireless channel releases and acquisitions, especially given that user think times in between page downloads are frequently higher than 6 seconds. As a result when the think time between requests is higher than the TBF value, the mobile host suffers an extra delay to acquire a new GPRS channel before any data can be transmitted. In order to determine the impact of the TBF release in a page download, we consider the following experiment. We downloaded the Top 10 pages repeatedly, with a new page request happening 10 seconds after the end of the previous page download finished. We then repeated the same experiment while having a background ping from the W-PEP to the mobile host every 5 seconds. This background ping happens before the TBF is released, thus, the mobile host gets to keep the channel and does not need to re-acquire on each page. Only one ping request is required per mobile host to keep the wireless GPRS channel active. The results show that keeping the GPRS channel and not releasing it in between page requests provides a 15% latency improvement. In a real implementation this background ping should stop after the mobile host inactivity period goes over a given threshold, e.g., 20 sec, to avoid flooding the wireless link with unnecessary packets and to release the GPRS channel.

6.5 Session-level overheads: DNS

DNS is a session-level protocol that is used to resolve the server names associated with all objects in a Web page. However, DNS queries can have a significant impact in the GPRS performance. For example, we observed that *www.britannica.com* has 14 different domain names used by objects embedded on its home page, or *www.cnn.com* has 6 different embedded domain names. Performing DNS queries through the wireless interface can drastically increase the overall download time of this and similar pages. The way of overcoming DNS delays is by caching DNS responses and re-using them for a certain Time-to-live without having to re-contact the DNS server each time a given domain is accessed. However, popular domains names are frequently served by content distribution networks, which set very small TTLs in their DNS responses. When TTL response is very small, the browser has to repeat the same DNS query over and over again to resolve a given domain name. Performing repeated DNS queries through terrestrial links may not have a significant performance impact; however, in wireless links this can be a source of major overhead.

In order to estimate the impact of DNS queries we conducted the following experiment. We download the main CNN page with all its embedded objects. First we download it ensuring that all DNS lookups required to resolve the embedded domain names happen through the GPRS link. Then we repeat the same experiment with all DNS lookups being satisfied from the local DNS cache, thus avoiding the GPRS link. To make sure that in the first experiment DNS lookups were not satisfied from a local client DNS cache, the DNS Time To Live key value was set to 0 in the Windows registry, and IE was restarted after every run (IE keeps its own DNS cache). The results obtained show that avoiding DNS lookups over the wireless interface reduces the response time by 16%. This is a significant time reduction that requires special attention. There are several ways to fix this problem by having the proxy do the lookups over a terrestrial link in a transparent way. However, due to its lengthy considerations we prefer to make this the subject of another paper [21].

To summarize the results presented in this section, a well-tuned wireless TCP stack can provide significant reduction in the user perceived latency. Such a wireless TCP stack should avoid slow-start as much as possible by sharing connection information with other connections, having a large initial window and a large MTU. Moreover, a well-tuned TCP stack should avoid spurious timeouts through a better estimation of the wireless link RTTs and should avoid frequent GPRS channel releases in the middle of a transaction. The number of TCP connections used can also have a large impact in the overall download time. Having a large number of TCP connections that are not properly re-used can seriously harm the end-user performance, while very few connections may not be able to gather the entire available wireless throughput. Similarly, too few connections may not be able to offset the overhead caused by idle RTTs when pipelining is not available. Finally, session-level overheads such as DNS lookups can account for a large portion of the overall latency and require intelligent mechanisms to push such overhead to the wireline network.

7 Application Level Optimizations

Application level optimizations are intended to minimize the overhead of the application level protocol (e.g., HTTP) and to minimize the time to delivery the payload. Figure 3 showed that payload transmission time accounts for the major portion of the delay since the bandwidth of the GPRS link is quite low. The idle time in between object requests (GETs) as well as the HTTP headers also account for a large portion of the total delay, however, its significance depends on the number of objects in a given page. Pages with a large number of small objects have a higher number of idle RTTs and higher proportion of HTTP headers to content, while pages with a low number of large objects barely experience these kinds of overhead.

Given that the time to deliver the payload accounts for most of the transmission time, minimizing the amount of data delivered will significantly reduce the total download time. To this end we have instrumented the W-PEP to intercept all Web requests and process the responses before passing them to the mobile host. Once W-PEP downloads the Web document requested, it performs the following actions: a) *transform* the page format into a suitable page format readable in the mobile host. If the mobile host is a laptop then there is no page transformation; b) *lossless content compression*, e.g., compresses text/html files; c) *lossy compres-*

sion of images. Regarding lossy compression, W-PEP has several adjustable levels of compression which can be configured by the W-PEP administrator, the content provider, or the end-user. The number of parameters that can be adjusted include: number of colors in a GIF, level of quality degradation in JPEGs, and whether animated GIFs should be converted into static ones or not.

7.1 Compression Results

Next, we present the compression factors attained by W-PEP on different types of Web pages. We focus on optimizing the down-link channel since it carries most of the data in WEB applications. We assume that the page is already pre-formatted to fit the device screen, and therefore, we do not consider the impact of content transformation. Instead, we measure the compression achieved by W-PEP through lossy or lossless compression. For lossless compression, W-PEP used gzip on all possible content-types. Some servers already support gzipping of text/html content, however, we found that most pages were uncompressed, and therefore, W-PEP had to compress them. Recent versions of most browsers support the reception of compressed content, which is uncompressed on the fly. For lossy compression we selected 16 colors for GIFs, no animated GIFs (in banners), and a level of JPEG quality degradation that was barely perceived by the human eye. W-PEP caches all transformed content, thus, it only needs to compress it or transcode it once and then multiple requests for the same object can be served from the cache, substantially increasing scalability.

We first consider three different Synthetic Pages that were created to represent three typical pages. We consider a text-only page, a mixed page with text and images, and a heavily loaded image page. The page-compression results obtained by W-PEP are presented in table 4.

Table 4: Compression Factors. Synthetic Pages.

Type	Objects	Size	Compression
Text Only	1	15 KB	x5.6
Mixed	4	12 KB	x3.1
Large Images	10	560 KB	x7.5

From these results we see that the amount of data coming out of the W-PEP, and traversing the wireless link, is much lower than the amount of data coming into the W-PEP from the Internet (about 3 to 7 times less). The page with the large images as well as the page with the large portions of text can be highly compressed since images and text usually have a large number of redundant information. However, the mixed page with multiple smaller objects is less compressible since compression algorithms do not tend to work well on small chunks of information (such as small GIF images).

We repeated the same experiment using the Top 100 pages to understand how W-PEP would behave in a real scenario. The results obtained are presented in Figure 7. The average compression factor for the Top 100 pages is 2.83, however, some specific pages with large portions of text and highly compressible images can be compressed by a factor close to 6

Table 5 presents the compression factor for each content type individually as well as the percentage of a page corresponding to each type. We can see that GIFs as well as HTML content amount

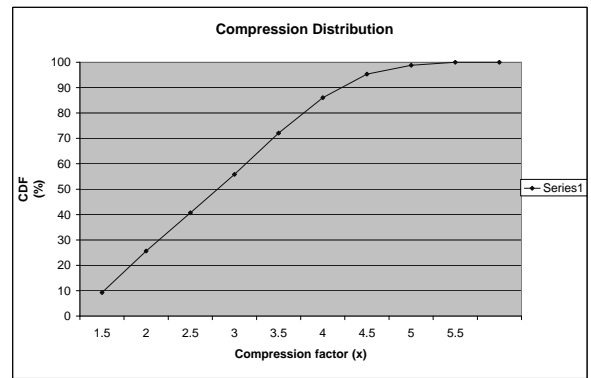


Figure 7: Page Compression Distribution (100 Top Pages)

for a large portion of all files and can be highly compressed, which helps achieving a high overall compression factor.

Table 5: Compression Factors by Content Type.

Content Type	% of Content	Compression Factor
Octetstream	0.45	x2.3
Xjavascript	4.50	x2.73
Xpotplus	0.60	x2.5
Xshockwaveflash	0.68	x3.1
GIF	77.33	x2.44
JPEG	4.80	x1.93
PNG	0.38	x1.92
CSS	0.23	x4.94
HTML	8.41	x3.84
Text/PLA	0.60	x3.45

Given the amount of overhead caused by HTTP headers (Figure 3), we now try to estimate how much benefit would be obtained by compressing the response HTTP headers. For pages with many small objects, HTTP headers can account for a large overhead and header compression would certainly be beneficial. Current servers do not support compression of HTTP headers. Despite this, we consider the compression factor that could be obtained when gzipping the response HTTP headers at the W-PEP and unzipping them at the mobile host. Note that this would require some special decompressing client software since current browsers do not support decompressing HTTP response headers. The average data compression factor obtained when compressing HTTP response headers is a 15% higher than without compressing them, which is quite significant.

7.2 Acceleration Results

Given the compression factors calculated in the previous section we would like to determine how this data compression factors translate into a latency reduction experienced by the end user. To this end we calculated the compression factors and the latency reduction seen by the end user for the Top 10 Web pages (see Table 6). The results of this calculations are shown in Table 7. The latency reduction is calculated as the ratio between the total page download time with W-PEP and without W-PEP. We can see that the average latency reduction for these pages is 2.12, and the average compression factor is 3.18. The average ratio between latency/compression is 0.64, thus, the compression fac-

tor for most pages is higher than the latency reduction achieved. This indicates that there is not a perfect direct translation between compressing data and reducing latency, though, the correlation is quite high. This is caused by other overheads that are not affected by compressing data (e.g., idle RTTs, TCP connection setup, DNS lookups, etc.) and that still accounts for a significant portion of the total download latency as we will see later.

Table 6: Top 10 Web sites.

Name	Objects	Domains	Size (KB)
1-Altavista	6.00	3	25.4
2-Chek	14.00	1	37.7
3-CNN	36.00	8	185
4-Excite	16.00	5	55.4
5-Fortunecity	36.00	7	142
6-Go	16.00	2	33.7
7-Google	2.00	1	10.3
8-Lycos	5.00	3	30.3
9-MS	16.00	2	87
10-Yahoo	4.00	2	37.4

Table 7: Top 10 Web sites. Compression vs. Acceleration

Name	Compression	Speedup
1-Altavista	2.15	1.51
2-Chek	2.77	1.67
3-CNN	2.72	2.8
4-Excite	3.46	2.43
5-Fortunecity	3.46	1.97
6-Go	3.66	1.72
7-Google	3.96	1.76
8-Lycos	4.09	1.76
9-MS	3.95	2.37
10-Yahoo	3.53	3.18

To better understand how different factors affect the overall latency after compression has been applied, we re-calculated the results of Figure 3. The new latency distribution with compressed data is presented in Figure 8. From this Figure we see that payload's transmission time has been reduced substantially and accounts for a much smaller portion (about 40%) of the total latency. This causes other latency factors such as idle RTTs, or HTTP headers to account for a much higher portion of the delay, sometimes equal or more than the actual payload delivery.

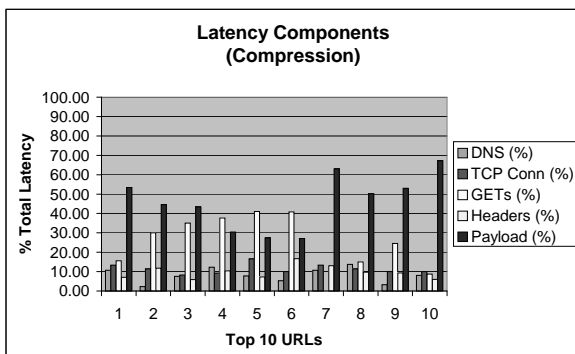


Figure 8: Latency Distribution (Compression).

7.3 Impact of Pipelining

We will now consider the impact of pipelining. Pipelining is required to avoid the idle RTTs that occur when browsers wait until an embedded object is fully received before requesting the next object. This overhead can be quite important for GPRS links since the RTT delays through a GPRS link are usually over one second. One way to avoid these idle RTTs is by requesting a given embedded object before the previous one has been fully downloaded. This is known as *request pipelining* and its benefits have been well studied in the past on wireline and satellite networks. To estimate the benefits associated with pipelining in a cellular network, we compare the latency obtained when downloading the Top 10 Web pages with and without request pipelining. The results obtained are presented in table 8. This table shows that the speedup factor obtained when pipelining is turned on is 3.06, which is a 42% improvement versus not having pipelining (table 7). Similar results were obtained in [22].

Table 8: Top 10 Web sites. Pipelining Improvement

Name	Speedup (Pipelining)
1-Altavista	1.87
2-Chek	2.42
3-CNN	4.42
4-Excite	4.13
5-Fortunecity	3.57
6-Go	2.96
7-Google	2.00
8-Lycos	2.36
9-MS	3.25
10-Yahoo	3.59

Assuming that pipelining was turn on, we now attempt to determine whether the relationship between compression and latency reduction improved. To this end we compare the latency reduction obtained with pipelining vs. the compression factors. (We do not show the individual results for each page.) The results obtained show a new average ratio between latency reduction and compression of 0.90 versus 0.64 without request pipelining. This indicates that when request pipelining is employed, gains achieved by data compression translate almost fully into a latency reduction factor.

Despite the advantages of pipelining, especially in wireless networks, most servers still do not support it. Browsers, on the other hand, frequently support request pipelining. In order to enable request pipelining through the wireless link, W-PEP supports request pipelining. Pipelining is not always supported in the wireline connection to the origin servers. But having W-PEP implement pipelining over the wireless link provides most of the benefit since the wireless link dominates the end-to-end latency.

8 Comparison

In this section we try to determine the relative impact of the different optimizations presented in this paper. We compare transport-level optimizations with application-level optimizations. To do this, we measured the latency of downloading certain Web objects when W-PEP performs only transport-level optimizations, and when W-PEP performs both transport-level and application-level optimizations. The application-level opti-

mizations included pipelining and compression of Web objects (lossless and lossy). The transport-level optimizations considered were TCP tuning, TCP connection sharing, TBF release avoidance, use of two persistent TCP connections, and no DNS lookups.

We first compared both sets of optimizations for a large object, a 400 KB image, and then we repeated the same experiment with a Web page. In Figure 9 we show the speedup obtained when downloading this large image for the cases of no W-PEP, W-PEP with transport-level optimizations only, and W-PEP with transport-level and application-level optimizations. From this figure we see that relative benefits obtained from optimizing long-term TCP behavior are quite small (about 6%). Compressing the large image, on the other hand, provides a compression factor over 800% with very small visual perception impact. It is clear from this example that application-level optimizations are much more important than transport-level optimizations when considering a large single object that is highly compressible.

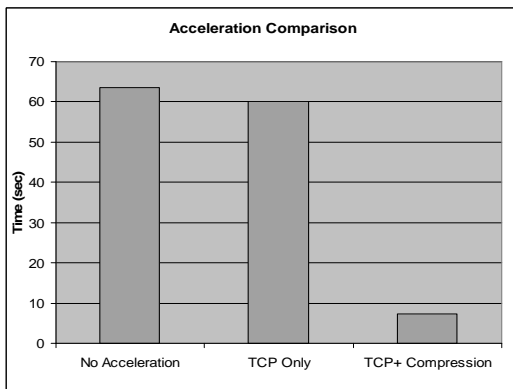


Figure 9: Comparison between application-level and transport-level optimizations. 400 KB image.

In Figure 10 we repeated the same experiments for a typical Web page. For this purpose we use a 100 KB page with 8 embedded objects, and 3 different embedded domain names. Given that this page has multiple objects and it is hosted in several different domain names, the overhead of DNS lookups and TCP connection setup is much higher. Using the previously mentioned transport-level optimizations, the speedup obtained equals 30% instead 6% obtained with a single large object. Transport-level optimizations provide better results in this case since the complexity of the page is higher, with many objects hosted in multiple domains, which is the case of many popular Web sites. Regarding application-level optimizations, we note that they do not work that well with the tested Web page since the objects were quite small and not very compressible. The actual speedup obtained when compressing text and images on the tested Web page was close to 250%, which is a much lower number than the 800% obtained previously with a single large image. Still this factor is much higher number than the 30% provided by the transport-level optimizations alone. Therefore, application-level optimizations amount for a larger portion of the overall latency reduction, although their relative impact highly depends on the type of pages and their complexity.

9 W-PEP Deployment Strategies

Transparent proxy deployments account for more than 90% of all ISP cache deployments [23]. The main reasons why system

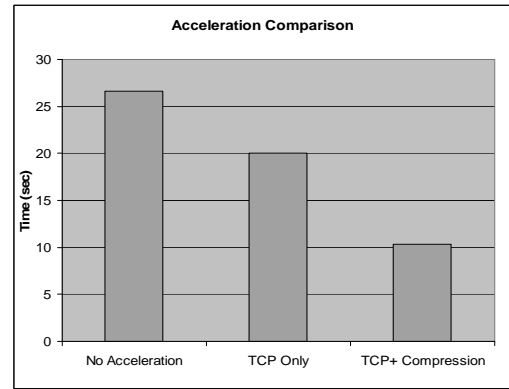


Figure 10: Comparison between application-level and transport-level optimizations. 100 KB page. 8 embedded objects. 3 different domain names.

administrators prefer to have a transparent proxy rather than an explicit proxy is because explicit proxies require configuration of the client's browser/OS. Despite the repeated attempts at standardization, there is no single proxy discovery protocol that is widely supported by caching vendors and browsers. In addition to the lack of client configuration required by transparent proxies, system administrators also prefer transparent proxies to explicit proxies for security reasons. First, administrators are hesitant to expose the proxy's IP address to the public, which is a prerequisite to make explicit proxy configurations work. Instead carriers prefer to use transparent proxies since they can be hidden from both the servers and the clients. Second, administrators frequently want to enforce client requests to travel through a proxy where they implement content blocking or different classes of service. With explicit proxy, clients could modify the browsers configuration to try to bypass the proxy and obtain access to preferred or blocked services.

As a result, wireless carriers prefer transparent W-PEP deployments rather than explicit ones. However, as we will see in this section, transparent proxy deployments in wireless networks have a serious performance drawback with respect to explicit proxy deployments. The reasons for this are that with explicit proxy deployments browsers do not perform any DNS lookup over the air interface. Moreover, browsers only keep a fixed number of connections open to the proxy that they reuse to fetch objects from multiple servers. With transparent proxy deployments, on the other hand, browsers act as if they were talking to the origin servers directly. Browsers perform DNS queries and open multiple connections over the wireless interface for each new domain name visited. While this overhead may go unnoticed in wireline networks, in wireless networks with large and variable delays these factors can have a significant impact.

To be able to quantify the performance impact of having a transparent proxy deployment vs. an explicit proxy deployment we performed the following experiment. We took the Top 10 pages and downloaded each multiple times averaging the results over an explicit and a transparent proxy configuration. The results showed that an explicit proxy configuration provides a 1.38 average latency reduction with respect to a transparent configuration.

These results show that deploying a W-PEP in a transparent mode causes a significant penalty to the end-user experience that should not go unnoticed by wireless carriers. One interesting challenge is being able to use a transparent W-PEP deployment to enjoy client-free configuration and advance control and security fea-

tures, and at the same time emulate the performance obtained with an explicit proxy configuration so that the end user does not suffer a reduction in user experience. Some of these techniques can be found in [21].

10 Conclusions

In this paper we have identified the main problems experienced by Web applications in 2.5G and 3G wireless networks. We have considered the problems at each layer of protocol stack separately, e.g., transport, session, and application. Given the peculiarities of these wireless networks with very large and highly variable latencies and low throughputs, we introduced a Wireless Performance Enhancing Proxy (W-PEP). The W-PEP does not require any modification to the sender or the client's stack, and it can be deployed completely transparently at the border between the Wireline and the wireless network. We showed that the suite of optimizations implemented at the W-PEP can provide significant latency reduction to the end user, thus, fostering the utilization of such links to access Web content.

As possible future directions of this work we are considering use of W-PEP to handle temporary loss of connectivity, (e.g., when users go inside a tunnel) or how it can be used to minimize the impact when the mobile user switches access networks, e.g., GPRS to WiFi. Finally, other possible areas where W-PEP could improve performance are dynamic content acceleration, wireless gaming or streaming.

References

- [1] P. Bhagwat, P. Bhattacharya, A. Krishna, and S.K. Tripathi, "Enhancing Throughput over Wireless LANs using Channel State Dependent Packet Scheduling," in *Proc. of IEEE Infocom*, Mar. 1996.
- [2] 3rd Generation Partnership Project 2, "Data Service Option for Spread Spectrum Systems: Radio Link Protocol Type 3," http://www.3gpp2.org/Public.html/specs/C.S0017-0-2.10_v2.0.pdf, Aug. 2000, 3GPP2 C.S0017-0-2.10 v2.0.
- [3] 3rd Generation Partnership Project, "Technical Specification Group Radio Access Network: RLC Protocol Specification," June 2001, 3GPP TS 25.322 V3.7.0.
- [4] H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz, "Improving TCP/IP Performance over Wireless Networks," in *Proc. of ACM Mobicom*, Nov. 1995.
- [5] G. Holland and N.H. Vaidya, "Analysis of TCP Performance over Mobile Ad Hoc Networks," in *Proc. of ACM Mobicom*, Aug. 1999.
- [6] R. Ludwig, B. Rathonyi, A. Konrad, K. Oden, and A. Joseph, "Multi-Layer Tracing of TCP over a Reliable Wireless Link," in *Proc. of ACM SIGMETRICS*, Mar. 1999.
- [7] M.C. Chan and R. Ramjee, "TCP/IP Performance over 3G Wireless Links with Rate and Delay Variation," in *Proc. of ACM Mobicom*, Sept. 2002.
- [8] K. Brown and S. Singh, "M-TCP: TCP for Mobile Cellular Networks," *ACM Computer Communications Review*, vol. 27, no. 5, 1997.
- [9] P. Sinha, N. Venkitaraman, R. Shivakumar, and V. Bharghavan, "WTCP: A Reliable Transport Protocol for Wireless Wide-Area Networks," *Wireless Networks*, vol. 8, no. 2-3, 2002.
- [10] Forelle Systems Inc., "The Venturi Server," http://www.fourelle.com/pdfs/Venturi_V2.1_Brochure.pdf.
- [11] Bytemobile Inc., "The Macara Optimization Service Node," <http://www.bytemobile.com/html/products.html>.
- [12] 3rd Generation Partnership Project 2, "Inter-Operability Specification (IOS) for High rate Packet Data (HRPD) Access Network Interfaces," http://www.3gpp2.org/Public.html/specs/A.S0007-0_v2.0.pdf, Nov. 2001, 3GPP2 A.S0007-0 v2.0.
- [13] P. Bender, P. Black, M. Grob, R. Padovani, N. Sindhusayana, and A. Viterbi, "CDMA/HDR: A Bandwidth-Efficient High-Speed Wireless Data Service for Nomadic Users," *IEEE Communications Magazine*, July 2000.
- [14] 3rd Generation Partnership Project 2, "CDMA2000 High Rate Packet Data Air Interface Specification," http://www.3gpp2.org/Public.html/specs/C.S0024-0_v4.0.pdf, Oct. 2002, 3GPP2 C.S0024-0 v4.0.
- [15] R. Chakravorty, S. Katti, J. Crowcroft, and I. Pratt, "Flow aggregation for enhanced tcp over wide-area wireless," in *IEEE INFOCOM*, 2003.
- [16] R.Katz H.Balakrishnan and S. Seshan, "Improving TCP/IP performance over Wireless Networks," in *Proceedings of ACM MOBICOM*, 1995.
- [17] Ajay Bakre and B. R. Badrinath, "Indirect TCP for mobile hosts," in *icdcs*, May 1995.
- [18] K. Ratnam and I. Matta, "W-TCP: An Efficient Transmission Control Protocol for Networks with Wireless Links," in *In Proceedings of Third IEEE Symposium on Computer and Communications*, 1998.
- [19] R. Chakravorty, A. Clark, and I. Pratt, "GPRSWeb: Optimizing the Web for GPRS Links," in *ACM/USENIX First International Conference on Mobile Systems, Applications and Services*, 2003.
- [20] V. Padmanabhan, *Addressing the challenges of web data transport*, Ph.D. thesis.
- [21] P. Rodriguez, Sarit Mukherjee, and Sampath Rangarajan, "Session level techniques for improving web browsing performance on wireless links," in *Bell-Labs Technical Report*, 2003.
- [22] R. Chakravorty and I. Pratt, "WWW Performance over GPRS," in *IEEE MWCN*, 2002.
- [23] Private Communication, "With Inktomi and Cisco," 2002.