WWW Performance over GPRS

Rajiv Chakravorty and Ian Pratt
{rajiv.chakravorty, ian.pratt}@cl.cam.ac.uk
University of Cambridge Computer Laboratory,
JJ Thomson Avenue, Cambridge CB3 0FD, U.K.

***Abstract*— In this paper, we present investigative results of HTTP performance over GPRS (General Packet Radio Service). Following on from an earlier study of GPRS[2], in which we uncovered a number of performance problems with TCP (e.g. sub-optimal start-up performance, excess queueing, spurious timeouts etc.), we discuss how and to what extent these limitations can impact HTTP. We also examine some other issues specifically linked to HTTP performance over GPRS.**

Our experimental results show that aggressive behaviour on the part of web browsers optimized for the wired-Internet does not work well over GPRS. Instead, limiting the number of browser TCP connections, which enables aggressive pipelining of requests, can give significant performance benefits. We show that using a proxy located close to the wired-wireless boundary that implements performance enhancements at both the transport (TCP) and application layer can lead to substantial reduction in web page download times over GPRS.

***Index Terms*—GPRS, wireless, WWW, HTTP, persistent, pipelining**

I. Introduction and Background

The World Wide Web (WWW) is currently responsible for a significant fraction of Internet traffic. Key to its operation is the HyperText Transfer Protocol (HTTP), which uses TCP (Transmission control protocol) - The Internet's *de facto* reliable transport protocol, designed to detect congestion and avoid overload.

Unfortunately, TCP performance is known to degrade over wireless links where losses are mostly non-congestive, predominantly due to external environmental factors such as fading, interference etc. Our link characterization measurements reveal that GPRS links have very high RTTs (>1000ms), fluctuating bandwidths, and occasional link outages. Thus TCP performance suffers in several ways:

- A sluggish slow-start that takes many seconds (due to high RTTs) for the window to ramp-up and allow full link utilization,
- Excess queueing over the downlink can result in gross unfairness to other TCP flows, and a high probability of timeouts during initial connection request,
- Spurious TCP timeouts due to occasional link 'stalls' and,
- Slow recovery (many seconds) after timeouts.

Web browser behaviour also has a substantial effect on the page download times over GPRS. In an effort to improve response times on wired-Internet links, client browsers open many concurrent TCP connections. We show that such a behaviour on the part of the clients may result in saturation of the downlink buffers, and an increased control overhead that can negatively impact page download times over GPRS. We attempt to answer questions like:

- *How fair is TCP over GPRS? How does it impact Web performace?*
- *How does browser behaviour influence page download times?*
- *What is the quantitative benefit achievable when requests are pipelined?*

This paper has been abridged to fit the conference requirements of 5 pages. Download the full version from: http://www.cl.cam.ac.uk/∼rc277/gprs.html

To answer these questions, we perform a number of experiments over Vodafone UK's GPRS infrastructure.[1] Through experiments conducted over our GPRS test bed, we demonstrate that HTTP can underperform for a number of reasons. We go on to show that using a mobile proxy can reap significant performance benefits to web browsing over GPRS. Avoiding slow start and simultaneously limiting excess TCP data over the downlink (using a TCP congestion window clamping technique in the proxy), combined with even a moderate support from a web browser for request pipelining, can result in at least 15-20% reduction in mean download times. Our approach improves web performance is two ways - (a) optimizing web client (browser) performance and (b) tuning HTTP and TCP in order to improve performance over GPRS.

The paper is structured as follows: The next section will briefly discuss the characteristics of the GPRS network. Section III describes TCP problems over GPRS. Section IV gives the experimental set-up while section V describes our scheme to improve web performance and demonstrate its effectiveness. We end the paper with our conclusions and plans for further work.

II. Measurements from GPRS Network

A. GPRS Link Characterization

GPRS [1], like other wide-area wireless networks, exhibits many of the following characteristics: low bandwidth, high and variable latency, ack compression, link blackouts and rapid bandwidth fluctuations over time. To gain a clear insight into the characteristics of the GPRS link, we conducted a series of link characterization experiments. All the experiments were conducted using a Motorola T260 GPRS (3+1) (3 downlink, 1 uplink channels) phone over Vodafone UK's GPRS network. A comprehensive report on GPRS link characterization is available in the form of a seperate technical report [4]. We enunciate some key findings from the experiments:

High and Variable Latency:- We have observed typically high latencies over GPRS, as high as 600ms-3000ms for the downlink and 400ms-1300ms on the uplink. Round-trip latencies are 1000ms or more.

Link Outages:- Link outages are common while moving at speed or, obviously, when passing through tunnels. Nevetheless, we have also noticed outages during stationary conditions. The observed outage interval will typically vary between 5 and 40s. Sudden signal quality degradation, prolonged fades and intra-zone handovers can lead to such link blackouts. When link outages are of small duration, packets are justly delayed and are lost only in few cases. In contrast, when outages are of higher duration there tend to be more loses.

Varying Bandwidths:- We observe that signal quality leads to significant (often sudden) variations in bandwidth perceivable by the receiver. Sudden signal quality fluctuations (good or bad) commensurately impacts GPRS link performance. Using a 3+1 GPRS phone,

[1] We made similar but less thorough performance measurements on GPRS networks throughout Europe, and found performance to be very similar.

we observed a maximum raw downlink throughput of about 4.15 KB/s and an uplink throughput of 1.4 KB/s.

Packet Loss:- As mentioned earlier, GPRS uses an aggressive RLC-ARQ technique whereby the link-layer tries hard to recover from any radio losses. This refects in a very stable link condition where higher layer protocols perceive few non-congestive losses.

III. Informal Description of TCP Problems over GPRS

In this section, we discuss TCP performance problems over GPRS. The observations made here relate to the downlink, as it is most important for activities like web browsing. We provide a more complete treatment on TCP problems over GPRS in [2].

TCP Start-up Performance:- Figure 1 (a) shows a close up of the first few seconds of a connection, alongside another connection under slightly worse radio conditions. An estimate of the link bandwidth delay product (BDP) is also marked, approximately 10KB[2]. For a TCP connection to fully utilize the link bandwidth, its congestion window must be equal or exceed the BDP of the link. We can observe that in the case of good radio conditions, it takes over 6 seconds to ramp the congestion window up to a value of link BDP from when the intial connect request (TCP's SYN) was made. Hence, for transfers shorter than about 10KB, TCP fails to exploit even the meagre bandwidth that GPRS makes available to it. Since many HTTP objects are around (or smaller) than this size, the effect on web browsing performance can be dire.

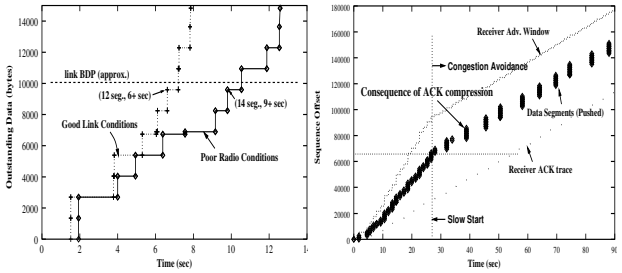

Fig. 1. Plot (a) shows that slow-start takes 6+ seconds before the congestion window is expanded sufficiently to enable the connection to utilise the full link bandwidth. (b) shows the characteristic exponential congestion window growth due to slow-start (SS).

A further point to note in figure 1(b) is that the sender releases packets in bursts in response to groups of four ACKs arriving in quick succession. Receiver-side traces show that the ACKs are generated in a 'smooth' fashion, hence it is surmised that the compression occurs as a result of the GPRS uplink (since the wired network is well provisioned). This effect is not uncommon, and appears to be an unfortunate interaction that can occur when the mobile terminal has data to send and receive concurrently.

ACK Compression:- During data transfers, ACKs from the mobile router to the mobile host can get closely spaced, resulting in a phenomemon well known as *ACK compression*. ACK compresssion in GPRS has its genesis in the link layer retransmission mechanism. The radio link control (RLC) layer in GPRS uses an automatic repeat request (ARQ) scheme that works aggressively to recover from link layer losses. As packets have to be delivered in order, the RLC waits before link level retransmissions are successful, and then hands over the packets to the higher layer. This results in ACK bunching that not only skews upwards TCP's RTO measurement but also effects its self-clocking strategy. Sender side packet bursts can further impair RTT measurements.

Excess Queuing:- Due to its low bandwidth, the GPRS link is almost always the bottleneck, and packets destined for the downlink get queued at the CGSN Node (the wired-wireless 'router'). However, we found that the existing GPRS infrastructure offers substantial buffering: initial UDP burst tests indicate over 120KB of buffering is available in the downlink direction. Therefore for a long session, TCP's congestion control algorithm could fill the entire router buffer before incurring packet loss and reducing its window. Typically, however, the window is not allowed to become quite so excessive due to the receiver's flow control window, which in most TCP implementation is limited to 64KB unless *window scaling* is explicitly enabled. Even so, this still amounts to several times the BDP of unnecessary buffering, leading to grossly inflated RTTs due to queueing delay. Figure 2 (b) shows a TCP connection in such a state, where there is 40KB of outstanding data leading to a measured RTT of around 30 seconds. Excess queueing exacerbates other issues: it inflates RTT and retransmit timer values, results in higher drain times for leftover (stale) data, and slows recovery from timeouts.

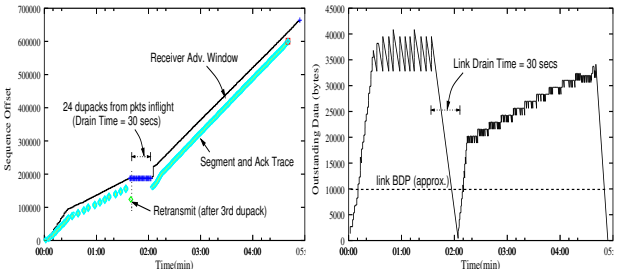

Fig. 2. Case of timeout due to a dupack(sack). Plot (a) shows the sender sequence trace and plot (b) shows corresponding outstanding data.

TCP loss recovery over GPRS:- Figure 2(a)-(b) depicts TCP's performance during recovery due to reception of a dupack (in this case a SACK). Note the long time (30 seconds) it takes TCP to recover from the loss, on account of the excess quantity of outstanding data. Also of note is the link condition, which improved significantly after the packet loss, resulting in higher available bandwidth. Fortunately, use of SACKs ensures that packets transferred during the recovery period are not discarded, and the effect on throughput is minimal. This emphasises the importance of SACKs in the GPRS environment.

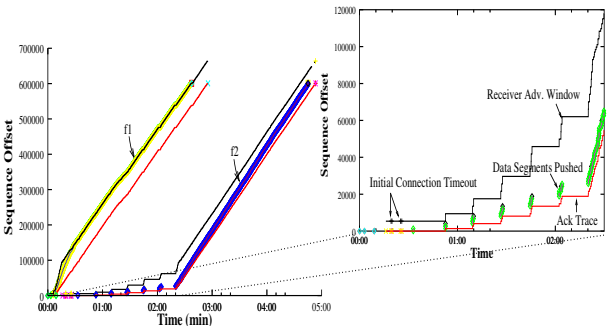

Fig. 3. Close-up of time sequence plots for two concurrent file transfers over GPRS, where `f2` was initiated 10 seconds after `f1`.

Fairness between flows:- Excess queueing can lead to gross unfairness between competing flows. Figure 3 shows a file transfer (f2) initiated 10 seconds after transfer (f1). When TCP transfer (f2) is initiated,

[2]The estimate is approximately correct under both good and bad radio conditions, as although the link bandwidth drops under poor conditions the RTT tends to rise.

it struggles to get going. In fact it times out twice on initial connection setup (SYN) before being able to send data. Even after establishing the connection, the few initial data packets of f2 are queued at the CGSN node behind a large number of f1 packets. As a result, packets of f2 perceive very high RTTs (16-20 seconds) and bear the full brunt of excess queueing delays due to f1. Flow f2 continues to badly underperform until f1 terminates. Flow fairness turns out to be an important issue for web browsing performance, since most browsers open multiple concurrent HTTP connections [6]. The implicit favouring of long-lived flows often has the effect of delaying the "important" objects that the browser needs to be able to start displaying the partially downloaded page, leading to decreased user perception of performance.

A. Experimental Test Bed Setup

As shown in figure 4, we used a laptop connected to a Motorola T260 '3+1' GPRS phone (3 downlink, 1 uplink channels) through a serial PPP (point-to-point) link to act as a GPRS mobile terminal. Vodafone UK's GPRS network was used as the infrastructure.

The base stations (BSs) are linked to the SGSN (Serving GPRS Support Node) which is then connected to a GGSN (Gateway GPRS Support node). Both SGSN and GGSN node is co-located in a CGSN (Combined GPRS Support Node) in the current Vodafone configuration. A well provisioned virtual private network (VPN) connects the lab network to that of the Vodafone's backbone via an IPSec tunnel over the public Internet. A RADIUS server is used to authenticate mobile terminals and assign IP addresses.

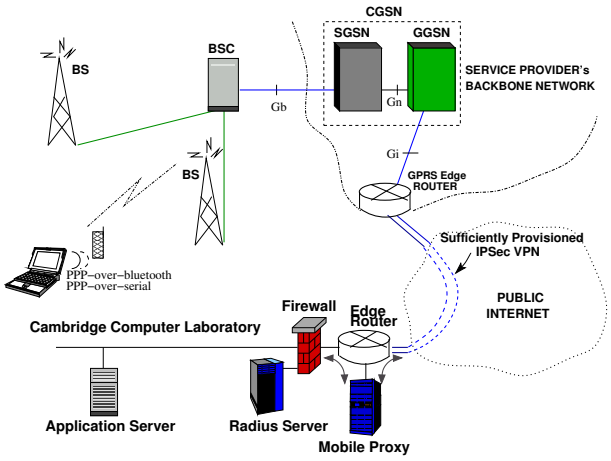

Fig. 4. Experimental Test Bed set-up

We have configured routeing within the lab such that all traffic going to and from mobile clients will pass through a Linux-based software router. This enables us to perform traffic monitoring as well as providing a location to run a *mobile proxy* that aims improve web performance over GPRS. The test bed also consist of another machine: a web server, which is located in the same network close to the proxy. The mobile proxy runs a modified version of the squid v2.4 [8] caching proxy, while the web server runs Apache 1.3.22 [9]. As shown in the setup of figure 4, a web client (browser) in the laptop uses the GPRS phone and connects to the mobile proxy, which then forwards the request to the co-located web server. The squid proxy has been modified to enable it to accept and respond to pipelined requests.

IV. Improving Web Performance over GPRS

An important goal with pipelining is to minimize the total number of connections over GPRS. In this section, we show that request pipelining combined with an interposed performance enhancing mobile proxy can improve web download performance over GPRS.

The proxy operates at both the transport and application layers:

A. Transport Level Enhancement (TL-E) using TCP cwnd clamping

A major cause of poor performance with TCP over GPRS is link under utilization during the first 10 seconds of a connection due to the pessimistic nature of the slow start algorithm. Slow start is certainly an appropriate mechanism for the Internet in general, but for the GPRS link, the proxy can make a better informed decision as to the congestion window size. Hence, our proxy makes use of a fixed size congestion window (`cwnd`), for all connections passing through the proxy. The size is fixed to a relatively static estimate of the Bandwidth Delay Product (BDP) of the link. Thus, slow start is eliminated, and further unnecessary growth of the congestion window beyond the BDP is avoided. We call this TCP `cwnd` clamping.

The underlying GPRS network is ensuring that bandwidth is shared fairly amongst different users (or according to some other QoS policy), and hence there is no need for TCP to be trying to do the same based on less accurate information. Ideally, the CGSN could provide feedback to the proxy about current radio conditions and time slot contention, enabling it to adjust the 'fixed' size congestion window, but in practice this is currently unnecessary.

Once the mobile proxy is successful in sending C_{clamp} amount of data it goes into a self-clocking state in which it clocks out one segment each time its receives an ACK for an equivalent amount of data from the receiver. With an ideal value of C_{clamp}, the link should never be under utilised if there is data to send, and there should only ever be minimal queueing at the CGSN gateway. Typically, the ideal C_{clamp} ends up being slightly higher than the value calculated by multiplying the maximum link bandwidth by the typical link RTT. This excess is required due to link jitter, use of delayed ACKs by the TCP receiver in the mobile host, and ACK compression occurring due to the link layer.

While starting with a fixed value of `cwnd`, the mobile proxy needs to ensure that any initial packet burst does not overrun CGSN buffers. Since the BDP of current GPRS links is small (≈10KB), this is not a significant problem at this time. For future GPRS devices supporting more downlink channels (e.g. 8+1), the proxy may need to use traffic shaping to smooth the initial burst of packets to a conservative estimate of the link bandwidth.

In the case of a packet loss, we preserve the `cwnd` value, clocking out further packets when ACKs are received. RTO triggered retransmissions operate in the normal manner. Our transport level enhancement offers several benefits that include faster startup for short flows, reduced queuing at the proxies and quick recovery from losses.

B. Application Level Enhancements (AL-E)

Apart from its traditional functionality of acting as a data caching proxy, our application level enhancement offers a minor modification to the current squid caching proxy. The modification allows squid to accept HTTP/1.1 pipelined connections from pipelined capable web clients (browsers). Other application level optimization schemes (e.g. delta compression, prefetching schemes etc.) can be used to further improve web performance over GPRS.

C. Quantifying the Benefits of our scheme

To evaluate the performance benefits of our scheme we perform experimental downloads over GPRS using both static and dynamic web content.

For the static web content, we composed a number of objects from a number of web-sites. We justify use of such a test web site purely for reasons of simplicity, which we actually shown to assist in evaluating pipelining effectiveness [3]. Simple web sites typically have a

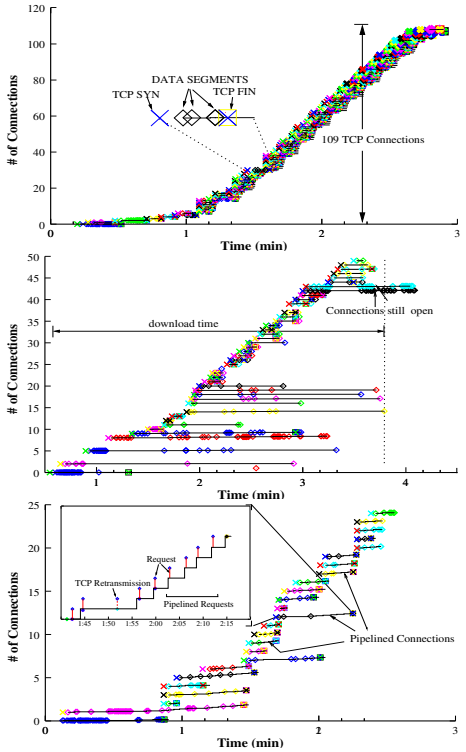

Fig. 5. Mozilla Connnection Timelines for our CNN web-site, using (top-bottom) (a) non-persistent connection mode, (b) persistent connection mode and (c) pipelined connection mode.

base HTML documents along with many embedded or *inlined* objects (gif's, CSS, scripts etc). We have observed that popular news sites (e.g CNN, BBC) can easily have an index page (index.html) of about 40-50KB with over 50 embedded objects. To offer better control over content presentation, these web sites also make use of cascading style sheets (CSS) and scripts. Hence is our static web site, we collectively synthesize about 45 gifs and jpegs images of various sizes from popular news sites such CNN and BBC. The file and object size distribution for our test web-site is shown in table I.

Resource Type	Size Range (approx.)	# of files
index.html	40K	1
jpegs/gifs	200B-2KB	20
gifs	2KB-5KB	20
gifs	5KB-10KB	4
gifs	>10KB	1

TABLE I
COMPOSITION OF OUR STATIC REFERENCE TEST WEB-SITE

For dynamic web-content, we pick a popular news web-site, CNN `www.cnn.com`[3]. To obviate the ill-effects of fast changing web-content in news web-sites such as CNN, we make a local copy of part of the site in a locally provisioned web server. Moreover, having a web server close to the border of wireline-wireless network removes 'noise' from our measurements by avoiding network performance effects of the Internet.

1) Browser Selection: To compare download performance of our scheme, we chose mozilla [10]. The latest release from mozilla 5.2 (developer build version) also supports pipelining. However, after analysing browser traces we found few instances where it actually pipelined requests. In fact, we observed that connections in which requests were pipelined would start in a persistent mode and later switch to pipelining. In persistent connection mode, which is the default setting, it seems that mozilla can make use of a maximum of 6 simultaneous connections to an intermediate proxy. While in the non-persistent mode, it can use a maximum of 8 parallel connections via a proxy.

2) Difficulty in Measuring Browser Download Times: Measuring download times with a browser is not as trivial as it at first appears: Many browsers keep connections open even after the complete web-site is downloaded. While this makes little difference to a user surfing for some information, it impedes accurate measurements of web site download times. To overcome this problem, we make use of *browser timelines*. Browser timelines are plots that indicate connection timelines made by a browser i.e the number of connections, connection start and end points (if a end point exists), number of requests made, and data received on each request-reponse exchange. We have written a script (*timeline* [7]) that uses `tcpdump` and `tcptrace` information to plot browser connection timelines.

Figure 5 shows the sample browser timelines for CNN web-site using mozilla. The connection timelines are shown for download of CNN web-site using browser's non-persistent connection mode (figure 5(a)), persistent connection mode (figure 5(b)) and pipelined connection mode (figure 5(c)). The figure (b)-(c) show steps in connection timelines indicating requests sent over an existing connection to be reused in persistent and pipelined connection mode for multiple request-response exchanges. As shown in figure 5(b), some browser connections are kept open even after the complete download[4]. For all such cases, we simply measured the download time with respect to connections that finished successfully prior to the one's that were kept open. This is shown in figure 5(b) with a dotted vertical line. Also, shown in figure 5(c) is the close-up of a connection in which requests were pipelined. The connection starts with a persistent mode and then switches to make use of pipelining. It is evident here that mozilla pipelined requests only occasionally, giving a relatively poor pipelining efficiency.

3) Measurements: We performed experimental downloads with mozilla using three different modes: non-persistent mode, persistent connection mode and pipelined connection mode. All experiments were conducted for the static test web site as well as for our locally available CNN web site offering dynamic content. For CNN, we found out that almost all the data was cacheable except for a few objects for which the requests had to be forwarded to our web server.

We averaged download times from 20 sucessful runs and plot mean value of the download time and corresponding standard deviation. Figure 6 shows that mean download time using mozilla for our test web-site. With non-persistent connections, we found only a small advantage in using the proposed transport level enhancement (TL-E). It seems that for non-persistent connections, the overall gains made by eliminating slow-start are negated by the overhead due to large number of connections. Nevertheless, there is still some modest improvement in download times when using TL-E with persistent connections. The average improvement in download times when switching from non-persistent to persistent mode and using TCP *cwnd clamping* was more than 10%. As far as the number of connections are concerned (see figure 7) the reduction was more than 80%. For pipelined connections, as evident from figure 6, we find that using TL-E can reduce mean download time by about 20%.

However, browser traces indicate only few intances of connections over which requests were pipelined. We believe that since the pipelining efficiency of the browser was low (as indicated earlier from browser timelines in figure 5(c)), only meagre benefit could be ex-

[3]CNN Timestamped: 22^{nd} April, Updated: 0910 GMT

[4]Persistent connections are kept open for re-use, but are typically timed out if idle for 60 seconds [6]. In some cases they are kept 'live' by scripts periodically refetching objects e.g. news tickers.

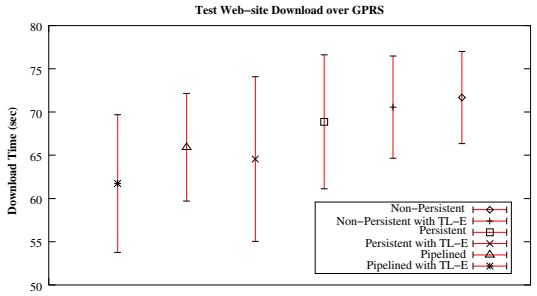

Fig. 6. Test Web Page download time over GPRS using Mozilla

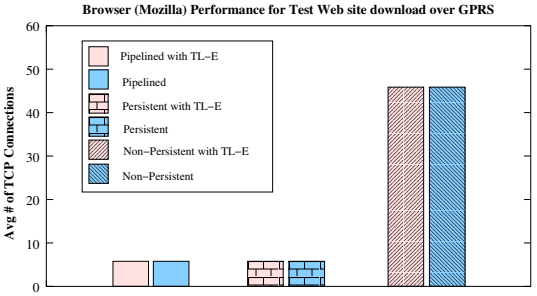

Fig. 7. Avg. Connections made to download the Test web site using Mozilla

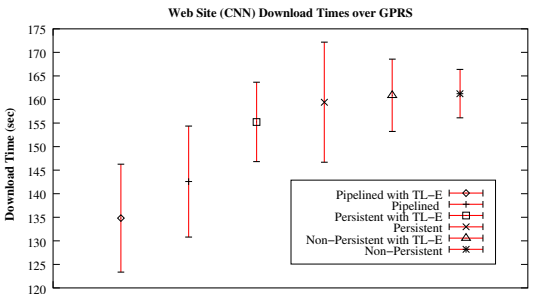

Fig. 8. CNN download time over GPRS using Mozilla.

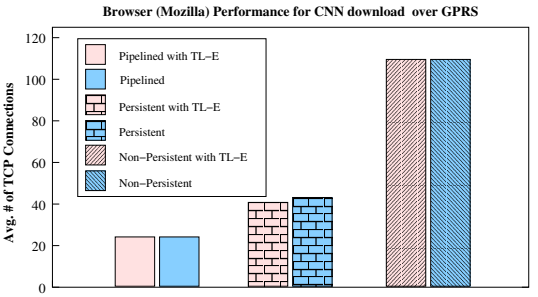

Fig. 9. Avg. Connections made to download CNN using Mozilla

tracted using pipelined connections.

On the contrary, observations valid for the CNN web site are somewhat different from those above. The use of TL-E with non-persistent connections show very little performance gains. The same is true for persistent connections that show only slight performance improvement in mean download times when compared to non-persistent connections. We believe that since the number of connections made is relatively high for the browser even with persistent mode (an average of more than 40 connections, see figure 9), the gain offered by eliminating slow-start is again negated by a high connection overhead.

However, using the browser's pipelined mode with CNN lowers the number of connections utilized and also indicates (using browser traces) a somewhat better pipelining efficiency when compared to the web site having static web content. Figure 8 shows that an improvement with pipelined mode was more evident when using the transport level enhancement (TL-E). We found an average reduction of more than 15% in mean download times when using the pipelined mode with TL-E. The average number of TCP connections in non-persistent mode was 109, reducing to an average of 24 when using pipelined connections, an overall reduction of more than 75% (see figure 9).

An average 15-20% improvement in mean download times for both static and dynamic content over GPRS with pipelined connections is encouraging, taking into account the low pipelining efficiency in mozilla. We believe that further benefit can be acheived if browsers pipeline their requests more aggresively.

While we would like browsers to achieve a high pipelining efficiency, there will be times when it cannot: data dependencies can be strict and hence responses to particular requests cannot be re-ordered. In such cases, browser clients will have to wait for a response before further requests can be pipelined. Obviously, this will reduce the overall pipelining efficiency, at least for web sites containing dynamically generated content.

Further, we find that a reduction in the number of connections is advantageous not only due to the low-bandwidth nature of the GPRS links but also because it mitigates the overall control and transactional (3-way TCP handshake) cost associated with each additional connection. We infer that aggressive pipelining of requests over browser connections reduces overall connection (control and transactional) overhead, and combined with the transport level enhancement (TL-E) can result in substantial improvement in web download times.

V. Conclusion and Future Work

In this paper, we have experimentally evaluated web-performance over a GPRS test bed. We have shown that pipelining over high RTT links such as GPRS can result in significant performance improvement in web page download times. We reported initial evaluations of our transport level enhancement (TCP *cwnd clamping* technique) and application level enhancement (pipelining) in our mobile proxy. Finally, we have shown that using a performance enhanced mobile proxy along with a browser capable of aggressive pipelining can reduce web download times over GPRS.

References

[1] G. Brasche and B. Walke, "Concepts, Services and Protocols of the New GSM Phase 2+ General Packet Radio Service", *IEEE Communications Magazine*, August 1997.
[2] R. Chakravorty, J. Cartwright, I. Pratt, "Practical Experience With TCP over GPRS", in IEEE GLOBECOM 2002, Taipei, Taiwan
source: http://www.cl.cam.ac.uk/users/rc277/gprs.html
[3] R. Chakravorty and I. Pratt, "WWW performance over GPRS" (complete paper), source: http://www.cl.cam.ac.uk/users/rc277/gprs.html
[4] J. Cartwright, "GPRS Link Characterization",
http://www.cl.cam.ac.uk/users/rc277/linkchar.html
[5] "An Introduction to the Vodafone GPRS Environment and Supported Services", Issue 1.1/1200, December 2000, Vodafone Ltd., 2000.
[6] Z. Wang and P. Cao, "Persistent Connection Behaviour of Popular Browsers", http://www.cs.wisc.edu/cao/papers/persistent-connection.html
[7] timeline (http://www.cl.cam.ac.uk/∼rc277/gprs.html,
tcpdump(http://www.tcpdump.org), tcptrace(http://www.tcptrace.org),
ttcp+(http://www.cl.cam.ac.uk/Research/SRG/netos/netx/)
[8] The squid proxy cache Homepage, http://www.squid-cache.org
[9] The Apache Software Foundation, http://www.apache.org
[10] Mozilla web browser, http://www.mozilla.org