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[3], 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 designed to perform over the wired-Internet, does not work well over GPRS. Instead, by limiting the number of browser connections, which also enables aggressive pipelining of requests, can give significant performance benefits. We show that by using a proxy located close to the wired-wireless boundary that implements performance enhancements at both the transport (TCP) and the application layer, can lead to substantial reduction in web download times over GPRS.

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), the means through which web documents are requested and delivered. HTTP 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. Wireless networks also suffer from a number of other discrepancies. Our link characterization measurements reveal that GPRS links have very high RTTs (>1000ms), fluctuating bandwidths, and occasional link outages. Thus TCP performance suffers in many 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.

A number of such pressing performance issues, aside from those discussed above needs to be solved to improve HTTP performance over GPRS. As we show later, large HTTP transfers can lead to excess queueing over the downlink. This can harm other existing or new flows, with potential to cause unfairness. Web browser behaviour also has a substantial effect on 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 web 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 a number of questions:

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

To answer these questions, we perform a number of experiments over Vodafone UK's GPRS infrastructure.[1] Through experiments conducted over our test bed, we show that HTTP can underperform for a number of reasons. Also, based on the research conducted earlier, we discuss limitations in TCP that can deleteriously impact HTTP performance. We identify potential performance problems in HTTP (including those in TCP) and find simple yet effective ways to overcome them. In particular, using a mobile proxy can reap significant performance benefits to web browsing over GPRS. We also show that by 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. Moreover, our technique optimizes the number of browser connections (and control overhead) and eliminates negative effects of normal TCP. 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.

In this paper, we do not evaluate optimizations schemes such as header compression, delta encoding or prefetching that can also give performance benefits to low-bandwidth GPRS users. We do, however, intend to do a quantitative evaluation of such schemes over GPRS in the near future.

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 presents a discussion on TCP fairness, browser performance over GPRS and pipelining incentives. In section V, we describe our scheme to improve web performance and demonstrate its effectiveness.

II. Characteristics of the GPRS Network

GPRS[1][2], 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

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

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[6]. 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.

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 the TCP's RTO measurement but also effects its self-clocking strategy. Sender side packet bursts can further impair RTT measurements.

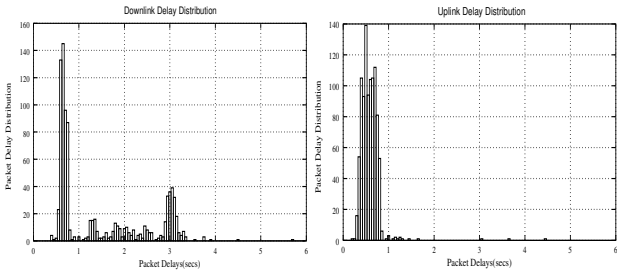

Fig. 1. Single packet time in flight delay distribution plots showing (a) downlink delay (b) uplink delay distribution. Measurements involved transfer of 1000 packets with random intervals more than 4s between successive packet transfers.

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, 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 relatively rare 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 [3].

TCP Start-up Performance:- Figure 2 (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 about 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.

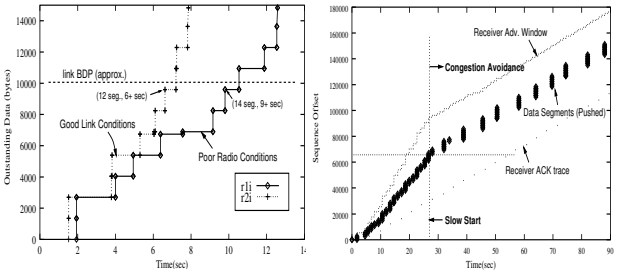

Fig. 2. Plot (a) shows that slow-start takes 6+ seconds before the congestion window is expanded sufficiently to enable the connection to utilise the full link bandwidth. (b) shows the characteristic exponential congestion window growth due to slow-start (SS).

A further point to note in figure 2(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.

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. 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

[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.

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 3 (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:

- **RTT Inflation**:- Higher queueing delays can severely degrade TCP performance[5]. A second TCP connection established over the same link is likely to have its initial connection request time-out [11].
- **Inflated Retransmit Timer Value**:- RTT inflation results in an inflated retransmit timer value that impacts TCP performance, for instance, in cases of multiple loss of the same packet[11].
- **Problems of Leftover (Stale) Data**:- For downlink channels, the data in the pipe may become obsolete when a user aborts a web download and abnormally terminates the connection. Draining leftover data from such a link may take on the order of several seconds.
- **Higher Recovery Time**:- Recovery from timeouts due to dupacks (or sacks) or coarse timeouts in TCP over a saturated GPRS link takes many seconds. This is depicted in figure 3(a) where the *drain time* is about 30s.

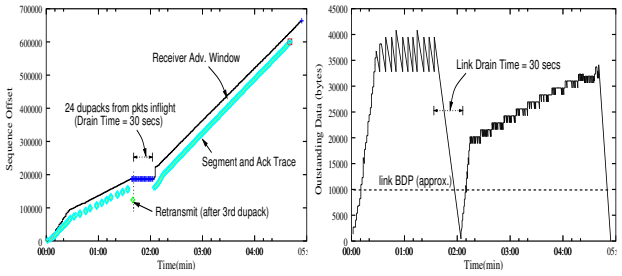

Fig. 3. Case of timeout due to a dupack(sack). Plot (a) shows the sender sequence trace and plot (b) shows corresponding outstanding data.

TCP loss recovery over GPRS:- Figure 3(a)-(b) depicts TCP's performance during recovery due to reception of a dupack (in this case a SACK). The point to note here is the large amount of 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 around the time of 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.

IV. TCP Fairness Evaluation over GPRS

To analyse TCP fairness, we emulate a common web browser behaviour by opening two HTTP (TCP) connections and investigate its performance. A simple case of only two TCP connections is examined, though current web browsers exacerbate the problem by opening multiple TCP connections[4]. We consider two long connections (600KB transfers). With no easy way to uncover temporal dependencies for connection start up times, we consider two distinct possibilities - (1) both connections are initiated at nearly the same time or (2) when one is initiated after the other achieved steady state. We have noticed that web browsers often start making requests even before the *root* resource (often a HTML file) is fully available. Also, other than its significance to connection start-up times, this experiment essentially investigates the impact of a long flow vis-a-vis other flows over GPRS.

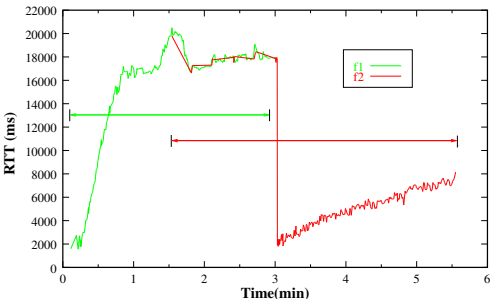

Fig. 4. Plots shows sender perceived RTTs for two (`f1` and `f2`) 600KB file transfers. Transfer (`f2`) was initiated after the first (`f1`) achieved steady state.

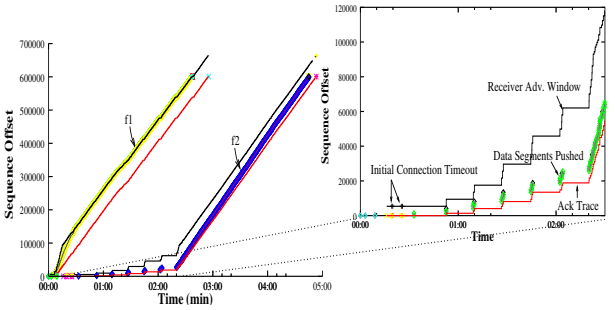

Fig. 5. Time sequence plots for two simultaneous file transfers over GPRS. Close-up plot shows `f2` for the duration of `f1` started only after `f1` reached steady state.

Figure 5 shows a file transfer (f2) initiated after the first transfer (f1) has achieved a steady state. This is shown in figure 4 where packets of f1 saturate the downlink leading to excess queuing delays (15-20s). When a second TCP transfer (f2) is initiated, it struggles to get going, in fact it times out twice on initial connection (SYN), before being able to send data. Even while this happens, 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 (similar to f1 packets, shown in figure 4) and bear the full brunt of excess queuing delays due to f1. Flow f2 continues to underperform due to excess RTTs for the complete duration of the life-time of f1.

A second case is shown in figure 6, where two flows begin at nearly the same time. In this case, both flows utilize the downlink queue proportionately, resulting in near fairness to both the transfers. This motivates us to another set of questions:

- How often can we expect large responses? Large responses here would mean responses that are of multiple order of the BDP of the GPRS downlink.
- How do web servers schedule responses?

Insight on the former can be gained by considering the heavy-tailed distributions in web server workloads, where majority of

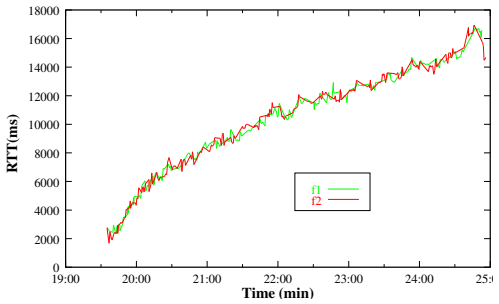

Fig. 6. Sender RTTs for two (`f1` and `f2`) 600KB file transfer over GPRS. Both transfers (`f1` and `f2`) were initiated almost at the same time.

the connections are small, but most server load is due to few large connections. It has been shown through relatively recent empirical measurements by F. D. Smith et. al. [13] that at least 15% of HTTP responses have a size larger than 10KB. This brings us closer to a particular problem known as *head of line* (HOL) blocking[22]. HOL blocking problems in HTTP can adversely impact user-perceived latency, when a slow response holds up all subsequent responses. This has largely to do with the way web servers (or proxies) perform response scheduling. Traditionally, web servers rely on the operating system to schedule responses, where emphasis is typically on fairness and on response times. This gives a simple first-come-first-serve response order over incoming requests. Hence, a large response over GPRS will easily the saturate GPRS downlink resulting in unfairness to other exisiting flows or new flows.

Web server scheduling schemes have interesting implications to reducing mean download times. Mark Crovella et. al. [8] show that a shortest connection first (scf) approach by a web server can improve mean response times by a factor of 4-5. For persistent connections, an analogous scheme called shortest response processing time (srpt) [9] scheduling can be advantageous. Both schemes claim meagre penalty for long sessions. Unfortunately, neither is strictly applicable when response sizes are not known in advance e.g. when responses are generated dynamically. With ever increasing popularity of dynamic web content, it is questionable if **scf** and **srpt** can offer substantial benefits to dynamic responses. Hence further work is required to extend such schemes and improve response times for downloads that involve dynamic data. Response re-ordering can also become complicated when requests have strict resource dependencies and it may not be possible to have their responses reordered[22].

A. Why Browser Performance matters?

The inherent nature of TCP's congestion control algorithm implies that N persistent connections will be N times more aggressive as compared to a single TCP connection. By opening more persistent connections browsers not only improve response times, but can also avoid head-of-line (HOL) blocking problem. Therefore browsers often open more concurrent connections with a server (or proxy). For example, Netscape seems to use a maximum (non-frame based) of 6 connections per web server while Internet Explorer uses 2[4]. Such browser action results in high pay-off over limited bandwidth-delay product links such as GPRS. Multiple connections keep the link busy resulting in efficient link utilization.

However, there remains potential drawbacks in using multiple connections over low banwidth GPRS. First, there is a control overhead associated with higher numbers of connections. Second, transaction (3-way TCP handshake) cost for a given connection is high, and more connections can increase response times. However, the main problem is that it can take only a few RTTs for multiple concurrent connections to exceed the GPRS CGSN router downlink BDP value. The exponential nature of the slow-start phase combined with packets from multiple flows leads to excess queuing over the downlink. As a result, any subsequent new TCP connection will have a high chance of timing out during its initial connection request. Worse, this new connection will endure very high RTTs, which can cause it to severely underperform, with an additional probability of spurious timeouts.

B. Pipelining Incentives over GPRS

Persistent connections allow multiple requests to be issued on the same TCP connection. However, a new request can only be issued after receiving a complete response from the server. HTTP/1.1 *pipelined* connections allow multiple requests to be kept outstanding before a response is received. Previous studies have demonstrated substantial improvement of page download times using pipelined connections [14][15]. However, none of the earlier literatures appear to quantify benefits associated with the degree of pipelining i.e. pipelining effectiveness. If we assume that a client would pipeline its request as aggressively as possible over a given connection, then how do we measure pipelining effectiveness? We introduce a new term *Pipeline Factor* (PF). PF typically represents pipelining aggressiveness of a web-client. A high PF indicates that a client is able to keep more requests outstanding during its connection lifetime.

PF for a single connection is defined here as:

$$PF = \frac{\sum_{z=1}^{n} \beta_z(t)}{n} \quad (1)$$

where n corresponds to the total number of requests scheduled during a connection's lifetime. Here $\beta_z(t)$ is the *pipeline index* calculated separately for each request as the total number of requests (including the request 'z') minus the responses received before request 'z' was made. The maximum possible PF for a connection is (PF_{max}=$\frac{\sum_{z=1}^{n} z}{n}$), which happens when all the requests are pipelined before a response can be received. PF can achieve a minimum value of 1, when a connection is effectively persistent with no pipelining[3]. When PF is 1, a single request is always outstanding during the connection lifetime. Notice equation 1 gives greater weight to connections that can keep more requests outstanding. Having more requests outstanding (high PF) gives the server (or proxy) more opportunity to keep the downlink busy and achieve a high link utilization. A link with a high RTT benefits from employing a high PF factor.

Figure 7 shows a sample PF calculation for a pipelined connection. As shown, the pipeline index for the the 3^{rd} and the 4^{th} request is the same for both: 2 each. The index value for the 4^{th}

[3]we refer to 1 request pipeline as persistent

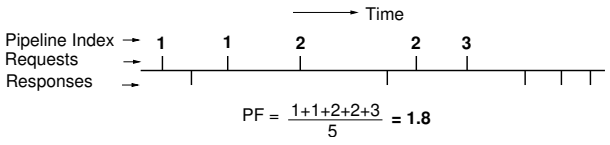

Fig. 7. PF calculation of a pipelined connection

request is 2 because its index gain is negated[4] by the reception of two responses. The PF value comes out to about 1.8 from a maximum possible PF of 3 for this connection.

Likewise, we introduce another term *pipelining efficiency* (η). Since browsers might pipeline requests on more than one connection - the pipelining efficiency in this case will be determined by how effectively requests are pipelined across all connections. The overall pipelining efficiency for 'm' connections can be given by:

$$\eta = \frac{\sum_{i=1}^{m} PF_i}{\sum_{i=1}^{m} PF_i^{max}} = \frac{\sum_{i=1}^{m} PF_i}{\sum_{i=1}^{m} \frac{\sum_{z=1}^{n} z}{n}} \quad (2)$$

It is typically not possible for such browsers that support pipelining to achieve 100% pipelining efficiency (η), as it would mean they would then have to be cognizant about all the requests over given connections. Browsers typically need atleast one response (for non-frame based static web-pages) to parse and make subsequent requests for other *inlined* objects. For web sites offering dynamic content, strict resource dependencies can limit number of pipelined requests that eventually lowers PF over a given connection. The easiest case is for a web-site with static content, where after receiving the first response, all other requests can be easily pipelined. To show how PF relates to web download performance, we perform further experiments over GPRS. The results presented here consider only one TCP connection used to issue pipeline requests with different PF values. We later generalize the results for any number of connections.

The majority of current web browsers have yet to offer any support for pipelining. Notable exceptions are mozilla [20] and opera [21] where support for pipelining exists, but the option needs to be explicitly enabled by the user. We believe software designers implementing pipelining in web browsers should aim to maintain a high PF[5].

B.1 Experimental Test Bed Setup

Our experimental set-up was that used during the link characterization measurements. As shown in figure 8, we used a laptop connected to a Motorola T260 GPRS phone (3+1) (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.

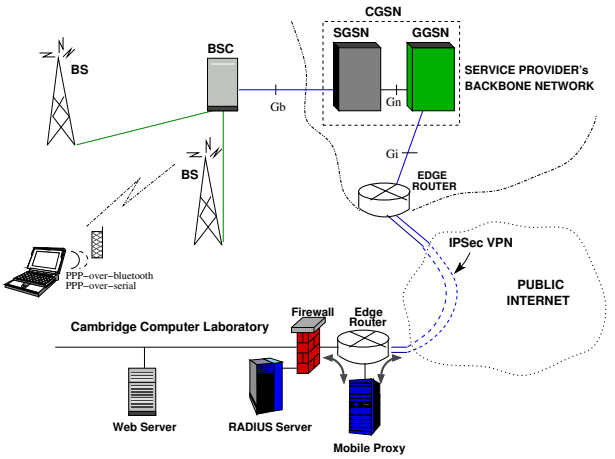

Fig. 8. Experimental Test Bed set-up

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 an other 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 [17] caching proxy, while the web server runs Apache 1.3.22[18]. As shown in the setup of figure 8, 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.

B.2 Test Web Site

For the test web-site, we composed a number of objects from other web-sites. We justify use of such a test web site purely for reasons of simplicity, which we show later, assists in evaluating pipelining effectiveness. Simple web sites typically have a 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.

In our test web-site, we avoid using dynamic content, however, we do recognize the importance of dynamic content and address this case later. For the static test web-page, we have collectively synthesized 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.

[4] obtained as: 4 (requests) - 2 (response) = 2

[5] we show later that mozilla starts with persistent mode and then switches to pipelined mode. Moreover, it seems to pipeline request over few connections, giving a poor pipelining efficiency.

Resource Type	Size Range	# 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

B.3 Evaluating Pipelining Effectiveness over GPRS with PFs

In this section we show how pipelining effectiveness (using PFs) can influence web download times. We intend to quantify the improvement a client can achieve by aggressively pipelining requests on a single connection over GPRS. To demonstrate this, we wrote a program that emulates a web browser and which makes pipelined HTTP GET requests over a single connection to retrive objects used in our test web-site. Using this client program, we perform a number of experimental downloads with different values of PF i.e. with different number of outstanding requests. Our test web-site has a total of 46 objects (including index.html), which bounds the PF value to a maximum of 23.

We have arranged the test web site such that all the inlined objects in the web-page are arranged in the *increasing order* of their sizes. We also record the download times (using `tcpdump`[24]) of our test web-site over GPRS, first with a simple persistent connection (PF=1, η=$\frac{1}{23}$) and later pipelining 3 (PF=1.97, η=$\frac{1.97}{23}$), 5 (PF=2.95, η=$\frac{2.95}{23}$), 7(PF=4.08, η=$\frac{4.08}{23}$) and 9 (PF=4.95, η=$\frac{4.95}{23}$) outstanding requests respectively, receiving all the responses, and so on in a *send-receive* fashion before all the web objects could be retrieved. We believe a client browser may or may-not have such *send-receive* behaviour, depending upon if certain pipelined requests have strict resource dependencies.

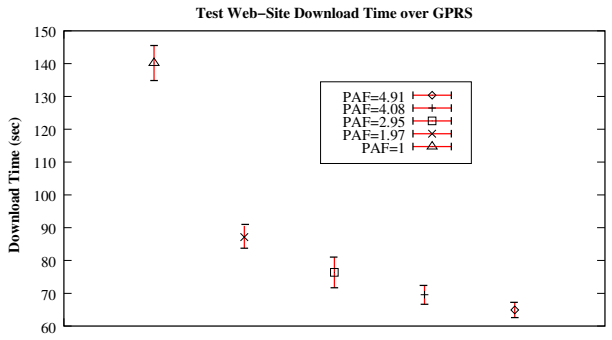

Fig. 9. Test web-page download times over GPRS. Increasing the Pipelining Factor(PF) drastically reduces the overall test web-site download times.

Corresponding to each PF, we averaged the download times for 20 successful runs. All the runs were conducted with the reponse objects in the proxy cache's main memory (after cache warm-up). Figure 9 shows the mean download times and its standard deviation for our web-site over GPRS using a single pipelined connection. One can observe from figure 9 that increasing the pipelined factor (PF) improves overall response times. It is evident from figure 9 that increasing PF from 1 (persistent) to 1.97 (3 outstanding requests) can improve download times by about 35%. Elevating PF to 4.95 (with 9 outstanding request) can result in a further reduction of response times by about 50%. We verified whether increasing PF any further could reduce download times: We found that increasing PF to 23 (with all 45 requests outstanding) only resulted in a marginal improvement the download times. Note here that a single TCP connection uses slow-start and so the start-up performance suffers. In the next section, we propose a TCP enhancement that overcomes startup performance bottleneck by avoiding *slow-start* mechanism.

Despite the reduction in download achieved through aggressive use of pipelining, we have still observed that the downlink is sometimes underutilized for certain periods of the connection. This is because our client program makes pipelined requests for all the small responses first, and then for the big ones. So, when PF is kept low (less number of oustanding requests) and their corresponding response sizes initially small (responses are arranged in their increasing order of sizes), the proxy is unable to fully utilize the downlink while its still expecting some more requests. Due to the idempotent nature of HTTP/1.1, responses re-ordering at the proxy is not possible and so our client program inadvertently coerces the squid proxy to perform a shortest response size scheduling over all the responses.

The downlink also remains underutilized whenever PF is low or when a client has to wait to receive its responses to pipelined requests. To circumvent this problem, browsers can make use of some additional connections while still allowing the proxy to re-order responses to the pipelined request. Smartly re-ordering the response order at the proxy can possibly improve link utilization and also avoid the HOL blocking problem. This can be shown with application-level re-ordering[6] of inlined objects. In this case, we have randomly arranged inlined objects (randomly mixing inlined objects) in our test web site and performed the download tests. Figure 10 shows the improvement - a simple application level re-ordering gives modest improvements in web download times.

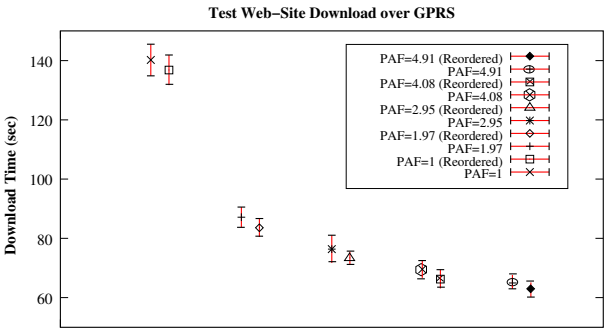

Fig. 10. Downloading Test Web-Site with application level re-ordering. An intelligent re-ordering by the server (or proxy) can further improve download times.

Existing web clients make use of multiple connections when

[6]While we currently suffice to show the benefits with an application level re-ordering, a smart re-ordering of responses by a proxy can also be implemented as an extention to the squid proxy. We plan to modify squid for intelligent response re-ordering in the future.

requesting multiple objects. So, resultant PF will be dependent upon how effectively requests are pipelined across all connections. A high degree of connection parrallelism helps to utilize the downlink better, but it also lowers the performance benefits achieved with an increased control and transactional overhead associated with each extra connection.

V. Improving Web Performance over GPRS

We have shown that request pipelining in browsers can improve web performance over GPRS. The other important goal with pipelining is to minimize the total number of connections over GPRS. Not reusing exiting connections and instead opening redundant ones can needlessly lower PF and reduce pipelining efficiency(η). 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 bottleneck with TCP over GPRS is its sub-optimal performance at connection start-up due to the slow-start mechanism. Hence we can infer that a substantial benefit can be offered to usually short and bursty web sessions by avoiding slow-start and instead, making use of the full capacity of the downlink. We propose a modification to TCP to be used over GPRS by the mobile proxy, which we refer to as TCP *cwnd clamping*. The proxy transparently splits TCP connections [10] into two legs: the 'wired' section and the 'wireless' section. Over the wireless section, the proxy uses a modified TCP sender (with TCP *cwnd clamping*) that uses a fixed size congestion window, the size picked to be the current estimate of the BDP of the link. Thus, slow start is eliminated, and further unnecessary growth of the congestion window is avoided. As a further optimisation we re-write the receiver window advertised in ACKs heading back to the sender to control the amount of data it causes to be queued at the proxy. A big advantage using such a modified proxy is that no changes are required to existing TCP implementation of mobile or fixed hosts. A similar modification will not be required to the mobile host, since web transfers are primarily downlink for which wireless links such as the asymmetric nature of such wireless links.

Attempting to apply our scheme to the Internet as a whole would certainly be disastrous; slow start and congestion avoidance normally serve essential roles. However, in the GPRS case congestion avoidance is largely redundant. It is possible for the proxy to maintain state about all of the TCP connections heading to a particular mobile terminal and share the BDPs worth of buffering out amongst the connections appropriately. The underlying GPRS network is ensuring that bandwidth is shared fairly amongst 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 rapidly adjust its fixed size congestion window, but in practise this is currently unnecessary.

TCP *cwnd clamping* avoids slow start and offers full use of the link bandwidth during connection startup. It uses a clamped value (C_{clamp}) of congestion window (`cwnd`) and maintains it for the full duration of the connection. 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 from the receiver. This approach maintains the amount of outstanding data to an *optimistic* value of the link BDP, neither overrunning the link, nor under utilizing it.

The `cwnd` remains clamped even during times of poor link performance i.e. during handoff's, interference or fading. While starting with a fixed value of `cwnd`, the mobile proxy needs to ensure that any initial packet burst does not overrun link buffers. Since the bandwidth-delay product (BDP) of current GPRS links is small (e.g. $\approx$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 absence of any queuing or packet loss, the window size necessary to keep a link busy without any idle times should correspond to the BDP of the link. A reasonable value for the clamp window value i.e. C_{clamp} can be made from the maximum values of GPRS link delay (D_{link}) and bandwidth (B_{link}) i.e. D_{max} and B_{max}. These values can be obtained through appropriate link characterization measurements. In such a case, the clamp value (C_{clamp}) for the congestion window will be given by $C_{clamp} = D_{max} \times B_{max}$. This value should avoid the link going idle and hence suffering from under-utilization.

If the estimate of the BDP is good, packets arriving at CGSN router will experience minimal queueing before being sent over the air. As radio conditions vary the amount of queueing may increase, but will be bounded, and is likely to be significantly less than the excesses of normal TCP.

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 offer the following benefits:

- **Faster Startup**:- It avoids slow-start and instead makes full use of the downlink capacity. This improves start-up performance of a short connections and reduces overall transfer times.
- **Reduced Queuing Delays**:- Excessive queuing is reduced by limiting TCP data over the link. As a consequence, RTT inflation and its impact on retransmit timer values are also minimized.
- **Quick Recovery from Losses**:- TCP *cwnd clamping* reduces drain time during losses leading to quick TCP recovery. By limiting data over the link, spurious retransmission cycles due to sudden delay fluctuations can be avoided. This also reconciles with other negative effects such as stale (or leftover) TCP data due to abnormal disconnections.

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 applica-

tion level optimization schemes can be used to further improve web performance over GPRS. We are currently exploring a number of such optimization schemes (e.g. delta compression[23], prefetching schemes [4] etc.) for use over GPRS in the future.

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 static web content we make use of the same test web-site that we used earlier. For the dynamic webpage, we pick a popular news web-site, CNN `www.cnn.com`[7]. 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.

C.1 Browser Selection

To compare download performance of our scheme, we chose mozilla[20]. 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 first 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.

C.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*[3]) that uses `tcpdump` and `tcptrace` information to plot browser connection timelines.

Figure 11 shows the sample browser timelines for CNN website using mozilla. The connection timelines are shown for download of CNN web-site using browser's non-persistent connection mode (figure 11(a)), persistent connection mode (figure 11(b)) and pipelined connection mode(figure 11(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 11(b), some browser connections are kept open even after the complete download[8]. For all

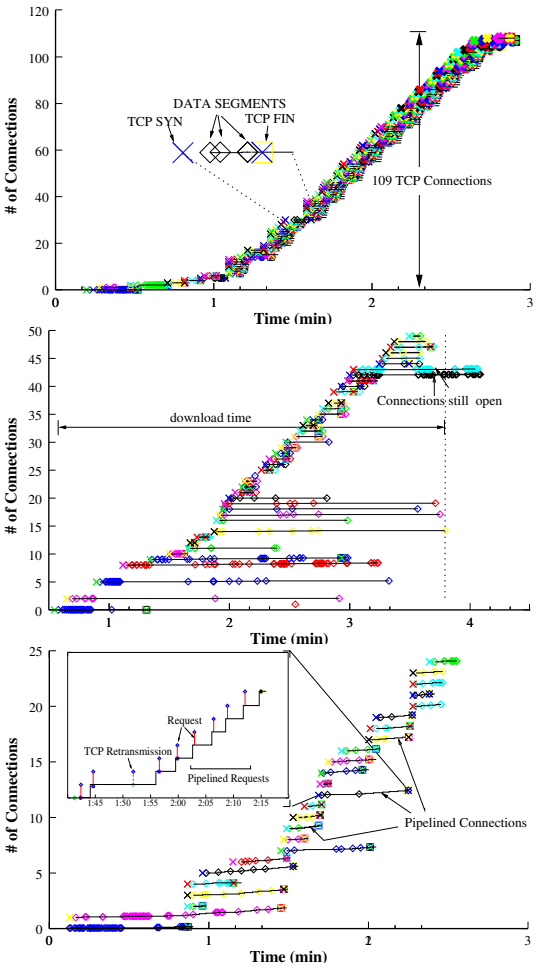

Fig. 11. Mozilla Connnection Timelines for our CNN web-site, using (top-bottom) (a) non-persistent connection mode, (b) persistent connection mode and (c) pipelined connection mode.

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 11(b) with a dotted vertical line. Also, shown in figure 11(c) is the close-up of a connection in which requests were pipelined. The connection starts with a persistent mode and then switches to pipeline requests over the given connection. It is evident here that mozilla pipelined requests only over selected connections out of the whole connection pool, giving a relatively poor pipelining efficiency.

C.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.

As again, we averaged download times from 20 sucessful runs

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

[8]persistent connections are kept open for re-use. Browsers like netscape idle out persistent connections, while Internet Explorer uses some form of timeout (usually 60 seconds)[4]. We suspect and in this case, it seems to be the handiwork of scripts that sometimes keep connections open.

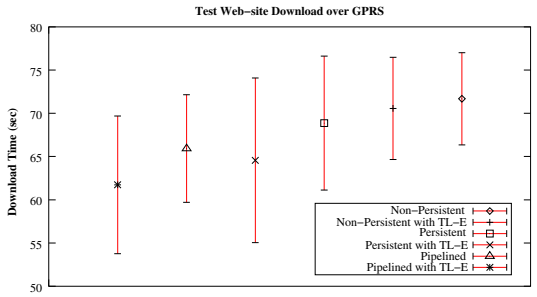

Fig. 12. Test Web Page download time over GPRS using Mozilla

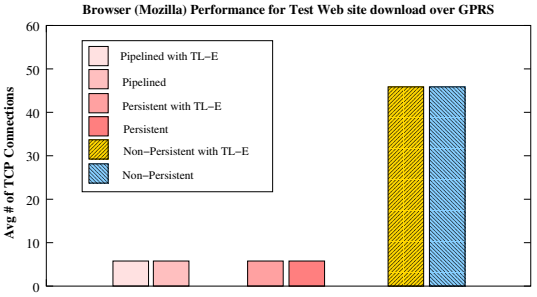

Fig. 13. Avg. Connections made to download the Test web site using Mozilla

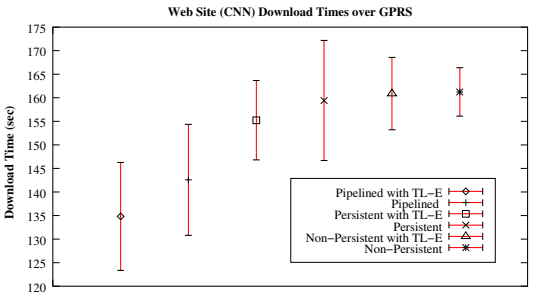

Fig. 14. CNN download time over GPRS using Mozilla.

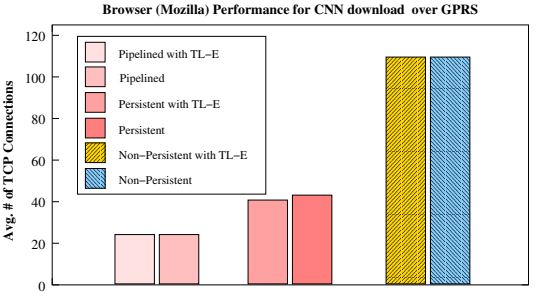

Fig. 15. Avg. Connections made to download CNN using Mozilla

and plot mean value of download time and corresponding standard deviation. Figure 12 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, overall gains made by eliminating slow-start is negated by the overhead due to large number of connections. Nevertheless, there is still some modest improvement in downoad 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 13) the reduction was more than 80%. For pipelined connections, as evident from figure 12, 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 11(c)), only meagre benefit could be extracted 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 15), 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 14 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 of the web browser when using the pipelined mode with TL-E. The average connection numbers from 109 in non-persistent mode, reduced to an average of 24 (see figure 15), an overall reduction of more than 75% in the total number of connections when using pipelined connections.

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 could be pipelined. Obviously, this will reduce the PF factor for the connection and lower overall pipelining efficiency. This need not be essentially true for web-sites that are wholly static, but more probable for web-sites that generate dynamic data.

Further, we find that a reduction in the number of connections is advantageous not only due to low-bandwidth nature of the GPRS links but also because it mitigates the overall control and associated transactional (3-way TCP handshake) cost 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.

VI. Conclusion and Future Work

In this paper, we have experimentally evaluated web-performance over GPRS. We have shown that use of normal TCP can result in a number of performance problems over GPRS. Apart from its sub-optimal performance at connection start-up, a large response over a given HTTP (TCP) connection can lead to a saturation of the downlink buffer (at the CGSN), resulting in severe unfairness to other flows. We discussed how intelligent re-ordering by a smart proxy might help to eliminate such unfairness and improve download performance over GPRS, also simultaneously reducing the chances of head of line blocking. We showed that pipelining over high RTT links such as GPRS can result in significant performance improvement of web download times. We derived a performance metric (PF) for measuring pipelining effectiveness, and showed that a browser can benefit by aiming for a high PF. We also 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 by using a performance enhanced mobile proxy along with a browser capable of aggressive pipelining over its connections can reduce web download times over GPRS.

In the future, we intend to evaluate several other optimization techniques such as the use of delta encoding and deterministic prefetching that can also benefit web performance over GPRS. We also wish to devise an appropriate scheme to enable intelligent response re-ordering in our mobile proxy.

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] C. Bettssetter, H. Vogel, J. Eberspacher, "GSM Phase 2+ General Packet Radio Service GPRS: Architecture, Protocols, and Air Interface", *IEEE Communication surveys* Third Quater 1999, Vol.2 no.3.

[3] Rajiv Chakravorty, Joel Cartwright, Ian Pratt, "Practical Experience With TCP over GPRS", submitted to IEEE GLOBECOM 2002, Taipei, Taiwan source: http://www.cl.cam.ac.uk/users/rc277/pub_new.html

[4] Li Fan, Pei Cao, Wei Lin and Quinn Jocobson "Web Prefetching Between Low-Bandwidth Clients and Proxies: Potential and Performance", In Proceedings of ACM SIGMETRICS 1999.

[5] D. Dutta and Y. Zhang, "An Active Proxy Based Architecture for TCP in Heterogeneous Variable Bandwidth Networks", IEEE GLOBECOM, November 2001.

[6] Joel Cartwright, "GPRS Link Characterization", http://www.fitz.cam.ac.uk/∼jjc36/gprs/linkchar.html

[7] "An Introduction to the Vodafone GPRS Environment and Supported Services", Issue 1.1/1200, December 2000, Vodafone Ltd., 2000.

[8] M. E. Crovella, R. Frangioso, and M. Harchol-Balter, "Connection Scheduling in Web Servers", In Proceedings of the 1999 USENIX Symposium on Internet Technologies and Systems (USITS '99), Boulder, Colorado, October 1999.

[9] Nikhil Bansal and Mor Harchol-Balter, "Analysis of SRPT Scheduling: Investigating Unfairness", In Proceedings of ACM SIGMETRICS 2001.

[10] Oliver Spatscheck et. al., "Optimizing TCP Forwarder Performance", IEEE/ACM Transactions on Networking, Vol. 8, No. 2., April 2000

[11] Reiner Ludwig et. al, "Multi-Layer Tracing of TCP over a Reliable Wireless Link", In Proceedings of ACM SIGMETRICS 1999.

[12] Hari Balakrishnan et. al., "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links", IEEE/ACM Transactions on Networking, Vol. 5, No.6, December 1997.

[13] F Donelson Smith et. al., "What TCP/IP Protocol Headers can tell us about the Web", In Proceedings of ACM SIGMETRICS 1999.

[14] Venkata N. Padmanabhan and Jeffrey C. Mogul, "Improving HTTP Latency", Computer Networks and ISDN Systems,28:25-35, 1995

[15] H. Nielsen, J. Gettys, A Baird-Smith, E. Prud'hommeaux, H. Lie and C. Lilley, "Network Performance Effects of HTTP/1.1 CSS1, and PNG", In Procedings of SIGCOMM 1997, Cannes, France, Sept. 1997

[16] The Linux NetFilter Homepage, http://www.netfilter.org

[17] The squid proxy cache Homepage, http://www.squid-cache.org

[18] The Apache Software Foundation, http://www.apache.org

[19] The **wget** utility, http://www.wget.org

[20] Mozilla web browser, http://www.mozilla.org

[21] Opera web browser, http://www.opera.com

[22] Jeffrey C. Mogul, "Support for out-of-order responses in HTTP", Internet Draft, Network Working Group, 6 April 2001.

[23] Jeffrey C. Mogul, Fred Douglis, Anja Feldmann, and Balachander Krishnamurthy. "Potential benefits of delta encoding and data compression for HTTP", In Proceedings of SIGCOMM 1997, pages 181-194. ACM SIGCOMM, Cannes, France, September, 1997.
[4] Zhe Wang and Pei Cao, "Persistent Connection Behaviour of Popular Browsers",
http://www.cs.wisc.edu/ cao/papers/persistent-connection.html

[24] tcpdump(http://www.tcpdump.org), tcptrace(http://www.tcptrace.org), ttcp+(http://www.cl.cam.ac.uk/Research/SRG/netos/netx/)