Using TCP Flow-Aggregation to Enhance Data Experience of Cellular Wireless Users

Rajiv Chakravorty, *Student Member, IEEE*, Sachin Katti, *Student Member, IEEE*, Ian Pratt, *Member, IEEE*, and Jon Crowcroft, *Fellow, IEEE*

***Abstract*—All over the world Global System for Mobile Communication (GSM) cellular mobile networks have been upgraded to support the "always-on" general packet radio service (GPRS). Despite the apparent availability of levels of bandwidth not dissimilar to that provided by conventional fixed-wire telephone modems, the user experience using GPRS is still considerably poor.**

In this paper, we examine the performance of protocols such as transmission control protocol (TCP) over GPRS, and show how certain network characteristics interact badly with TCP to yield problems such as: link underutilization for short-lived flows, excess queueing for long-lived flows, acknowledgment bunching, poor loss recovery, and gross unfairness between competing flows.

We present the design and implementation of a transparent TCP proxy that mitigates many of these problems without requiring any changes to the TCP implementations in either mobile or fixed-wire end systems. The proxy is interposed in the cellular provider's network, and splits TCP connections transparently into two halves—the wired and wireless sides. Connections destined for the same mobile host are treated as an *aggregate* due to their statistical dependence. We demonstrate packet scheduling and flow control algorithms that use information shared between the connections to maximize performance of the wireless link, while interworking with unmodified TCP peers. We also demonstrate how fairness between flows and response to loss is improved, and that queueing and, hence, network latency is reduced. We discuss how TCP enhancing proxies could be *transparently* deployed, and conclude that installing such a proxy into GPRS network would be of significant benefit to users.

***Index Terms*—General packet radio service (GPRS), proxy, third-generation (3G), transmission control protocol (TCP), universal mobile telecommunications system (UMTS), wireless.**

I. Introduction

WORLD over, global system for mobile communication (GSM) cellular networks are being upgraded to support the general packet radio service (GPRS). GPRS offers "always on" connectivity to mobile users, with wide-area coverage and data rates comparable to that of conventional fixed-line telephone modems. This holds the promise of making ubiquitous mobile access to Internet protocol (IP)-based applications and services a reality.

However, despite the momentum behind GPRS, surprisingly little has been done to evaluate transmission control protocol/Internet protocol (TCP/IP) performance over GPRS. There are some interesting simulation studies [1], [4], but we have found actual deployed network performance to be somewhat different.

Some of the performance issues observed with GPRS are shared with wireless local area networks (WLANs) like 802.11b, satellite systems, and other wide-area wireless schemes such as Metricom Ricochet and cellular digital packet data (CDPD). However, GPRS presents a particularly challenging environment for achieving good application performance.

In this paper, we present our practical experiences using GPRS networks, and our attempts to improve the performance seen by users. In Section II, we briefly report on work to characterize GPRS link behavior (see [35] for a fuller treatment). Section III identifies particular problems experienced by TCP running over GPRS, and examines how these are exacerbated by application-layer protocols such as HTTP.

In Section IV, we introduce the design of our TCP proxy, which is inserted into the network near the wired-wireless border and aims to transparently improve the performance of TCP flows running over the network without requiring modifications to either the wired or wireless end systems. In particular, we demonstrate how there are significant advantages to treating all TCP flows to a particular mobile host as an aggregate, taking advantage of the flows' statistical dependence to perform better scheduling and flow control in order to maximize link utilization, reduce latency, and improve fairness between flows.

Sections V and VI describe the experimental test setup and present an evaluation of our proxy's performance. Section VII examines how TCP proxies can be *transparently* deployed in GPRS network, along with some discussion on the impact of use of proxies on end-to-end protocol semantics. This paper goes on to discuss related work, and concludes with a brief outline about on-going research.

II. GPRS Network Characterization

GPRS [1], [3], like other wide-area wireless networks, exhibits many of the following characteristics: low and fluctuating bandwidth, high and variable latency, and occasional link "blackouts" [7], [9]. To gain clear insight into the characteristics of the GPRS link, we have conducted a series of link characterization experiments. These have been repeated under a wide range of conditions, using different models and manufacturer

Manuscript received February 2, 2004; revised January 20, 2005. This paper was presented at the IEEE INFOCOM 2003, San Francisco, CA.

R. Chakravorty is with the Department of Computer Sciences, University of Wisconsin, Madison, WI 53706 USA and the Computer Laboratory, University of Cambridge, Cambridge CB3 0FD, U.K. (e-mail: Rajiv@cs.wisc.edu; Rajiv.Chakravorty@cl.cam.ac.uk).

S. Katti is with the Massachusetts Institute of Technology (MIT), Cambridge, MA 02139 USA (e-mail: skatti@mit.edu).

I. Pratt and J. Crowcroft are with the Computer Laboratory, University of Cambridge, Cambridge CB3 0FD, U.K. (e-mail: Ian.Pratt@cl.cam.ac.uk; Jon.Crowcroft@cl.cam.ac.uk).

Digital Object Identifier 10.1109/JSAC.2005.845628

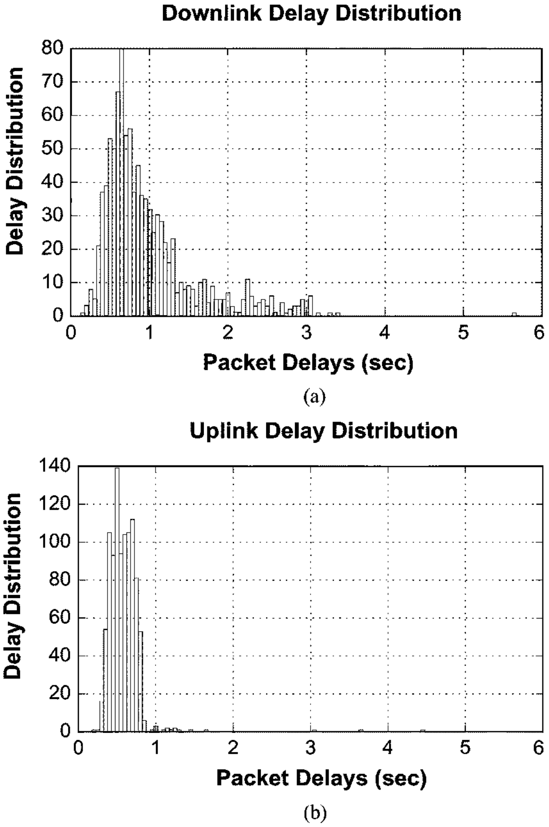


Fig. 1. Single packet time-in-flight delay distribution plots showing (a) downlink delay and (b) uplink delay distribution for 1000 packets of size 1024 bytes each.

of handsets, and different network operators located in several European countries. We have found no major performance differences between the network operators, and variation between different handsets of similar GPRS device class is minimal. *Note that details of how these tests were conducted (e.g., uplink and downlink latency measurements, radio conditions, tools used etc.) can be found in* [9]. *Reference* [35] *also gives a comprehensive description on GPRS link characterization in the form of a separate technical report*. Below, we enunciate some key findings.

High and Variable Latency: GPRS link latency is very high and variable: 600–3000 ms for the downlink and 400–1300 ms on the uplink. Round-trip latencies of 1000 ms or more are typical. The delay distribution is shown in Fig. 1. The link also has a strong tendency to "bunch" packets; the first packet in a burst is likely to be delayed and experience more jitter than following packets.

The additional latency for the first packet is typically due to the time taken to allocate the temporary block flow (TBF) [30], [34]. Since most current GPRS terminals allocate and release TBFs immediately (implementation based on GPRS 1997 release), protocols (such as TCP) that can transfer data (and ACK) packets spaced in time may end up creating many small TBFs that can each add some delay (approximately 100–200 ms) during data transfer.

The latest release (GPRS 1999) does consider an extended TBF lifetime; however, this optimization can lead to inefficient scheduling at the base station controller (BSC), with some improvement ($\sim$100 ms) in the overall RTTs). However, unlike relatively stable indoor 802.11b-based WLANs, wireless cellular networks have to cope with the harsh-outdoor mobile environments—GPRS typically requires use of sophisticated signal processing, interleaving, FEC/link-layer ARQ, etc. The net effect of this is that cellular links like GPRS typically suffer from high and variable RTTs, link outages, and burst losses, e.g., during deep fading or cell handovers. In [16], Chan and Ramjee show latencies of the order of 179 ms to over 1 s for code-division multiple-access (CDMA)-based 3G1X networks.

Fluctuating Bandwidth: We observe that signal quality leads to significant (often sudden) variations in perceivable bandwidth by the receiver. Sudden signal quality fluctuations (good or bad) commensurately impact GPRS link performance. Using a "$3 + 1$" GPRS phone such as the Ericsson T39 (three downlink channels, one uplink), we observed a maximum raw downlink throughput of about 4.15 kB/s, and an uplink throughput of 1.4 kB/s. Using a "$4 + 1$" phone, the Motorola T280, we measured an improved maximum bandwidth, to 5.5 kB/s in the downlink direction. Conducting these tests at various times of the day and at different locations revealed no evidence of network (channel) contention occurring. This is perhaps to be expected due to the currently small number of GPRS users and the generous time slot provisioning employed by most operators.

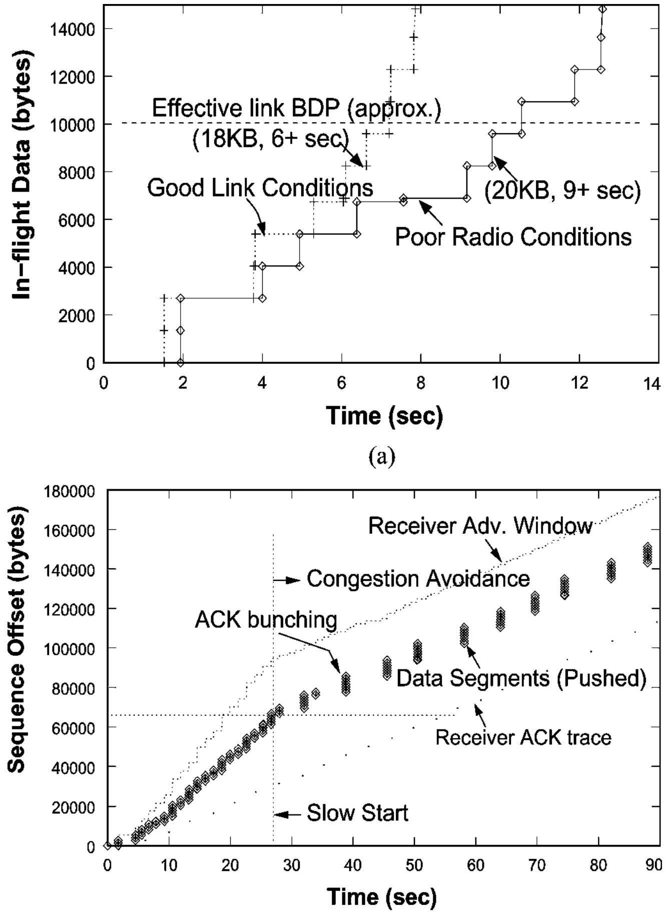
Packet Loss: The radio link control (RLC) layer in GPRS uses an automatic repeat request (ARQ) scheme that works aggressively to recover from link layer losses. Thus, higher level protocols (like IP) rarely experience noncongestive losses.

Link Outages: Link outages are common while moving at speed or, obviously, when passing through tunnels or other radio obstructions. Nevertheless, we have also noticed outages during stationary conditions. The observed outage interval will typically vary between 5 and 40 s [9]. Sudden signal quality degradation, prolonged fades, and intrazone handovers (cell reselections) 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 burst losses.

Occasionally, we also observed downlink transfers to stop altogether during transfers. We believe this to be a specific case of link-reset, where a mobile terminal would stall and stop listening to its TBF. We believe this to be due to inconsistent timer implementations within mobile terminals and BSCs. Recovering from such cases required the PPP session to be terminated and restarted.

III. TCP Performance Over GPRS

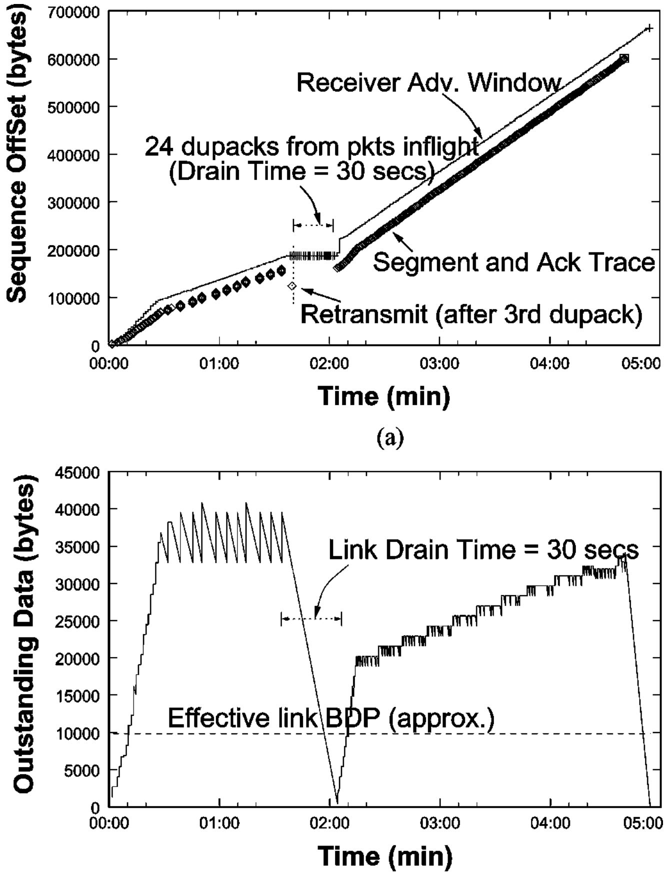
In this section, we discuss TCP performance problems over GPRS. In particular, we concentrate on connections where the majority of data is being shipped in the downlink direction, as this corresponds to the prevalent behavior of mobile applications, such as web browsing, file download, reading e-mail, news, etc. We provide a more complete treatment on such TCP problems over GPRS in [7].



(a)

(b)

Fig. 2. (a) Shows that slow-start takes over 6 s to expand the congestion window sufficiently to enable the connection to utilize the full link bandwidth. (b) Shows the characteristic exponential congestion window growth due to slow-start (SS). Maximum segment size (MSS) was set at 1400 bytes.



(a)

(b)

Fig. 3. Case of timeout due to DupAcks. (a) Shows the sender sequence trace. (b) Shows corresponding outstanding data. MSS was set at 1400 bytes.

TCP Startup Performance: Fig. 2(a) shows a close up of the first few seconds of a connection, displayed alongside another connection under slightly worse radio conditions. An estimate of the *effective* link bandwidth delay product (BDP) is also marked, approximately 10 kB. The actual value of BDP based on bandwidth and RTT of GPRS downlink may be somewhat less, around 5 kB to qualify it as a long-thin link [31]. However, experiments over GPRS reveal (we shall discuss this further in Section VI-C) that this effective BDP of 10 kB for a "3 + 1" phone is approximately correct, under both good and bad (fluctuating) radio conditions, as although the link bandwidth drops under poor conditions the RTT tends to rise. For a TCP connection to fully utilize the available effective link bandwidth, its congestion window must be equal or exceed the effective BDP of the link. We can observe that in the case of good radio conditions, it takes about 6 s from the initial connection request (TCP SYN) to ramp the congestion window up to the effective link BDP. Hence, for transfers shorter than about 18 kB, TCP fails to even exploit the meagre bandwidth that GPRS makes available to it. Since many HTTP objects are smaller than this size, the effect on web browsing performance can be substantial.

ACK Compression: A further point to note in Fig. 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 in response to arriving packets. The "bunching" on the uplink is due to the GPRS link layer (see, [9]). This effect is not uncommon, and appears to be an unfortunate interaction that can occur even when the mobile terminal has data to send and receive concurrently. ACK bunching or compression [17] not only skews upwards the TCP's RTO measurement but also affects its self-clocking strategy. Sender side packet bursts can further impair RTT measurements.

Excess Queueing: Due to its low bandwidth, the GPRS link is almost always the bottleneck of any TCP connection, hence, packets destined for the downlink get queued at the gateway onto the wireless network (known as the CGSN node in GPRS terminology, see Fig. 8). However, we found that the existing GPRS infrastructure offers substantial buffering: user datagram protocol (UDP) burst tests indicate that over 120 kB of buffering (at IP level) is available in the downlink direction. For long-lived sessions, 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 implementations is limited to 64 kB 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. Fig. 3(b)

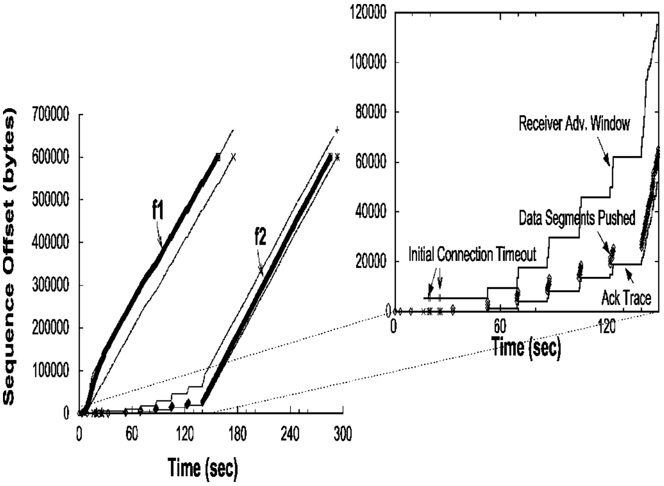


Fig. 4. Sequence plots for two concurrent file transfers over GPRS, where flow f2 (shown in close-up) was initiated 10 s after f1. Because of the excess data buffering caused by f1, flow f2 fails to initiate sufficiently until f1 terminates. MSS in this case was set at 1400 bytes.

shows a TCP connection in such a state, where there is 40 kB of outstanding data leading to a measured RTT of tens of seconds.

Excess queueing causes other problems.

- **RTT inflation**: Higher queueing delays can severely degrade TCP performance [10]. A second TCP connection established over the same link is likely to have its initial connection request timeout [5].
- **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 [5].
- **Problems of leftover (stale) data**: For downlink channels, the queued data may become obsolete when a user aborts a web download and abnormally terminates the connection. Draining leftover data from such a link may take many 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 Fig. 3(a), where the *drain time* is about 30 s.

Excess data buffering also plague other cellular data networks. Experiments conducted over CDMA-based 3G1X networks reveal ten times more buffering than the bandwidth-delay product of the third–generation (3G) downlink [16]. Schemes like TCP *ACK regular* have been devised, which uses per-TCP-flow queue to regulate ACKs to the TCP source—this scheme can be used to mitigate the impact of excess network buffering in 3G [16]. In Section IV, we shall discuss a receiver window based adaptation scheme for GPRS to overcome this problem.

TCP Loss Recovery Over GPRS: Fig. 3(a) and (b) depicts TCP's performance during recovery due to reception of a DupAck (in this case, a SACK). Note the long time it takes TCP to recover from the loss, on account of the excess quantity of outstanding data. Fortunately, use of SACKs ensures that packets transferred during the recovery period are not discarded, and the effect on throughput is minimal. This emphasizes the importance of SACKs in the GPRS environment. In this particular instance, the link condition happened to improve significantly just after the packet loss, resulting in higher available bandwidth during the recovery phase.

Fairness Between Flows: Excess queueing can lead to gross unfairness between competing flows. Fig. 4 shows a file transfer (f2) initiated 10 s after transfer (f1). When TCP transfer (f2) is initiated, 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 s) 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 [21]. The implicit favoring 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.

IV. Improving TCP Performance With an Interposed Proxy

A. Design Objectives and Motivation

Having identified the causes of poor TCP performance over GPRS, we set out to determine whether the situation could be improved. Our fundamental constraint was that we wanted to improve performance without requiring any changes to be made to the network protocol stacks of either the fixed or mobile TCP end systems. Experience shows that the vast majority of schemes that require such changes are doomed to never see widespread deployment.

Instead, we focused on what could be achieved through the use of a proxy located close to the wired-wireless network boundary, able to see all traffic heading to and from the mobile host. A "split TCP" [6] approach was adopted, whereby the proxy transparently divides the connection into two halves: one from the fixed host to the proxy and the other from the proxy to the mobile host. The TCP stacks on both the mobile and wired-facing sides of the proxy can be modified as necessary, providing they can still communicate with conventional implementations. We can exploit the fact that the packet-input code in all TCP stacks we have encountered will accept and process all validly formed TCP packets it receives without concern as to whether the transmitter is actually operating the congestion control algorithms mandated in the various TCP RFCs.

Concentrating on the critical mobile downlink direction, the goals for such a proxy are as follows.

- **Improved utilization**: Failing to utilize the available bandwidth on "long-thin" links when there is data to send is clearly wasteful.
- **Fairness**: A "fair"' allocation of bandwidth to competing flows regardless of their age or RTT.
- **Error detection and recovery**: GPRS link performance is characterized by occasional link stalls during handoffs and the occurrence of burst losses. A loss detection and recovery scheme tailored for this environment might avoid unnecessary backing-off or packet retransmission.

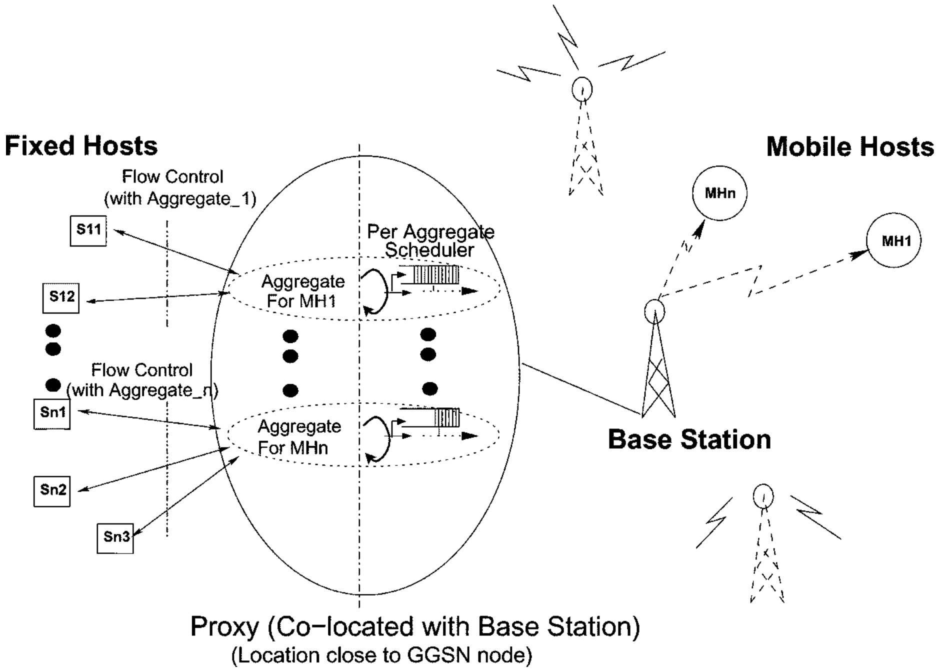

Fixed Hosts
Flow Control (with Aggregate_1)
S11
S12
Flow Control (with Aggregate_n)
Sn1
Sn2
Sn3
Per Aggregate Scheduler
Aggregate For MH1
Aggregate For MHn
Mobile Hosts
MHn
MH1
Base Station
Proxy (Co−located with Base Station)
(Location close to GGSN node)

Fig. 5. Proxy-based TCP flow aggregation scheme.

- **Effective flow control mechanism**: The proxy should avoid the problems of excessive data buffering at the wired-wireless gateway, and also take responsibility for effective buffer management using “smart” techniques.

We now describe the key concepts behind TCP flow aggregation.

B. TCP Flow Aggregation Mechanism

In conventional TCP implementations, every connection is independent, and separate state information (such as `srtt`, `cwnd`, `ssthresh`, etc.) is kept for each. However, since all TCP connections to a mobile host are statistically dependent (they all share the same wireless communication link), certain TCP state information might best be shared between flows to the same mobile host. On the wireless-facing side, our proxy treats flows to the same mobile host as a single *aggregate*. The scheme is depicted in Fig. 5.

Past research has made use of this concept for wired networks. In [13], Balakrishnan *et al.* show that flows can learn from each other and share information about the congestion state along the network path, which they term as *shared state learning*. Sharing state information across different TCP flows that use the same wireless communication channel to a mobile device such as in GPRS can exploit this statistical dependence. Our proxy shares state information including a single congestion window and RTT estimates across all TCP connections within the aggregate. Sharing state information enables all the connections in an aggregate to have better, reliable, and more recent knowledge of the wireless link. We, therefore, take all the state information and group it together into one structure that we call an aggregate control block (ACB). All individual TCP connections reference this structure as part of their local state. Details of this structure are given in Fig. 6.

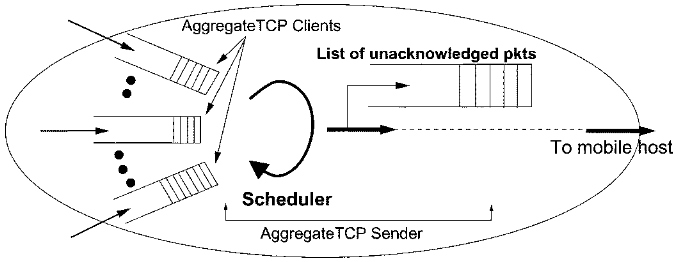


Fig. 6. Sample logical aggregate in the proxy for a given mobile host.

The wired-facing side of our proxy is known as the aggregate TCP (ATCP) client, while the mobile-facing side is called the ATCP sender. The ATCP client receives packets into small per-connection queues, that feed into a scheduler operating on behalf of the whole aggregate. Note that a single congestion window for the whole aggregate is maintained, and whenever the level of unacknowledged data on the wireless link drops one maximum segment size (MSS) below the size of the current congestion window the scheduler selects a connection with queued data from which a further segment will be sent. While making the selection the scheduler respects the mobile host’s receive window for each of the individual flows. Once transmitted, segments are kept in a queue of unacknowledged data until ACK’ed by the mobile host. The ATCP sender can perform retransmissions from this queue in the event of loss being signalled by the mobile host, or from the expiry of the aggregate’s retransmission timer.

The ATCP client employs “early ACKing”—acknowledging most packets it receives from hosts as soon as they are accepted into the per-flow queues, before the destination end system receives them. Notice that the practical effect this has on TCP’s end-to-end semantics is *mitigated* by never using early acknowledgment for FINs. Since the per-connection queues contain reassembled data, the proxy sometimes coalesce multiple small packets from the sender into a single segment by the time it is sent over the wireless link. This helps to reduce protocol header overhead.

The proxy can employ different connection scheduling strategies depending on the nature of the incoming traffic. Presently, we use a combination of priority-based and ticket-based stride scheduling [11] to select which connection to transmit from. This enables us to give strict priority to interactive flows (such as Telnet) while, for example, sharing out the remaining bandwidth in a fixed ratio between WWW and FTP flows.

Within this framework, the proxy optimizes GPRS link performance using three key mechanisms described in the following sections.

C. ATCP Sender Congestion Window Strategy

A major cause of poor performance with TCP over GPRS is link under utilization during the first few seconds of a connection due to the pessimistic nature of the slow start algorithm.

Slow start is an appropriate mechanism for the Internet in general, but within the proxy, information is available with which we can make better informed decisions as to congestion window size.

The ATCP Sender uses a fixed size congestion window (`cwnd`), shared across all connections in the aggregate. The size is fixed to a relatively static estimate of the effective 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 ensures 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 (from whatever connection the scheduler has selected) 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 utilized 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 around 20% larger than the effective link BDP. 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 proxy needs to ensure that any initial packet burst does not overrun CGSN buffers. Since the effective BDP of current GPRS links is small ($\approx$10 kB), this is not a significant problem at this time. Future GPRS (even EDGE or 3G) devices supporting more downlink channels may require the proxy to employ traffic shaping to smooth the initial burst of packets to a conservative estimate of the link bandwidth. The error detection and recovery mechanisms used by the ATCP sender are discussed in Section IV-E.

TABLE I
FLOW CONTROL ALGORITHM

```
Process_packet()
if ($Wireless\_Link\_Timeout$)
    set $w_i(t_k) = 0$
    send ack for packet
    return
elseif ($Connection\_Start\_Phase$)
    update $s_i(t_k)$, $s_a(t_k)$
    if($s_i(t_k) \geq s_a(t_k)$)
        set $Connection\_Start\_Phase = 0$
        goto normal
    endif
    set $w_i(t_k) = w_i(t_{k-1}) + C$
    send ack for packet
    return
endif
update $s_i(t_k)$, $Queue\_Occupancy$
set $fq = a * Queue\_Occupancy - b$
normal:
set $w_i(t_k) = w_i(t_{k-1}) + (c - fq * s_i(t_k)) * (t_k - t_{k-1})$
send ack for packet
return
```

D. ATCP Client Flow Control Scheme

When the proxy "early ACKs" a packet from a client it is committing buffer space that can not be released until the packet is successfully delivered to the mobile host. Hence, the proxy must be careful how much data it accepts on each connection if it is to avoid being swamped. Fortunately, the proxy can control the amount of this data it accepts through the receive window it advertises to hosts.

The proxy must also try to ensure that sufficient data from connections is buffered so that the link is not left to go idle unnecessarily (for example, it may need to buffer more data from senders with long RTTs—perhaps other mobile hosts), but also limit the total amount of buffer space committed to each mobile host. Furthermore, we wish to control the window advertised to the sending host in as smooth a fashion as possible. Hence, we use zero window advertisements only as a last resort, for example, during an extended wireless link stall.

Previous research studies have investigated similar receiver advertised window adaptation schemes. In [19], Kalampoukas *et al.* present explicit window adaptation (EWA), which controls the end-to-end receiver advertised window size in TCP to correspond to the delay-bandwidth product of the link. In this scheme, active TCP connections are allowed to adapt automatically to the traffic load, the buffer size and bandwidth-delay product of the network without maintaining any per-connection state. Likewise, Andrew *et al.* [20] propose a TCP flow-control scheme that takes feedback available from an access router to set the TCP receiver advertised window. Analysis in [20] show that the scheme can achieve a higher utilization for TCP flows while still preserving system stability. Following a similar approach, we build a simple flow control scheme, by utilizing feedback available from the aggregate about the wireless link performance.

The pseudocode for the flow control scheme is given in Table I. Note that the algorithm used during connection startup (denoted by *Connection_Start_Phase*) is different from that used during the later phase. During the connection start-phase, each received packet is ACKed as early as possible and the advertised window is calculated as follows. Consider $s_a(t_k)$ to be current average sending rate of the aggregate per connection and $s_i(t_k)$ the running average of the sending rates over the wireless link for connection i. The proxy calculates $s_i(t_k)$ as a sliding window averaging function given by

$$s_i(t_k) = \frac{1}{\alpha} \sum_{j=k-\alpha}^{k} \frac{1}{t_j - t_{j-1}}$$

where α is the window over which the average is calculated. By controlling the value of α, we achieve the desired "smoothing" estimate for the sending rate. For GPRS links, this averaging window is typically small.

We consider the case of a single (first) TCP connection in an aggregate. To start with, we can use an initial value of advertised window of $W_{\text{init}} = \text{RTT}_{\text{init}} \times S_{\text{init}} \times \beta$, where $(\text{RTT}_{\text{init}} \times S_{\text{init}})$ corresponds to the effective bandwidth-delay product of the GPRS downlink. A fixed GPRS-specific RTT_{init} value can be used initially, and then later estimated (using appropriate estimation filters [24]). Similarly, S_{init}—the initial sending rate is also initially estimated. β is the *overprovision* factor, which gives control over an aggregate's buffer target set-point. In our experiments, we over-provision the aggregate buffer by more than 20% of the effective link BDP, which ensures that there is always sufficient data to avoid any underutilization of the GPRS link.

However, when more new TCP connections join the aggregate (as typically happens in web sessions), any new connection "i" will likely satisfy the condition

$$s_i(t_k) < s_a(t_k)$$

and the advertised window will be set at

$$w_i(t_k) = w_i(t_{k-1}) + C$$

where C is a function of MSS, and is calculated as $\max(\text{MSS}, (W_{\text{init}}/n))$, where n is number of connections in the aggregate at time t_k. For the first advertised window value for any connection, $w_i(t_{k-1}) = 0$ and, hence, $w_i(t_k) = C$. This ensures that new and short-lived TCP flows are never discriminated against during the initial startup phase by bringing the connection up to the average sending rate of the aggregate so that it achieves parity with other already "established" connections. For those TCP flows that eventually have a sending rate equal to or exceeding the average sending rate over the GPRS link, the flow control algorithm described below takes over.

The steady-state flow control algorithm advertises a window size for connection "i" calculated using

$$w_i(t_k) = w_i(t_{k-1}) + [\mu_c - f(q(t_k))s_i(t_k)](t_k - t_{k-1}) \quad (1)$$

where t_k is the instant when the last packet was sent over the wireless link for the ith connection. μ_c is the constant rate at which the window increases and $q(t_k)$ is the overall queue occupancy of the aggregate buffer (shown as *Queue_Occupancy*) at time instant t_k.

Also, the cost function $f(q(t_k))$ is calculated as

$$f(q(t_k)) = aq(t_k) - b \quad (2)$$

where a and b are parameters which control the steady state queue size. It can be shown through analysis that equilibrium queue size $q_f \rightarrow (b + c)/(a)$ (see, [20]). The scheme works as follows: whenever a packet for a particular aggregate is received, the overall queue occupancy $q(t_k)$ for that aggregate is sampled. The scheme then dynamically computes the advertised window size (using (1)) such that as more connections join, overall aggregate queue occupancy $(q(t_k))$ starts to gradually increase, while average sending rate $(s_i(t_k))$ for that connection starts to go down (recall that connections are fairly scheduled in the aggregate). However, at some point, overall queue occupancy in the aggregate starts to dominate, and as a consequence, lower values of window size are advertised. Thus, (1) allows advertised window w_i to grow at a constant rate μ_c, whereas the overall aggregate queue occupancy terms $(q(t_k))$ and sending rate $(s_i(t_k))$ for a connection in the aggregate reduce it. The whole process achieves equilibrium (for all connections) in the steady state with a constant window size that balances queue occupancy with a constant growth rate. This in turn ensures availability of packets to send without ever congesting the proxy.

The scheme is slightly modified to make use of the feedback from the wireless link to a particular mobile host from the aggregate. So in the case of link stalls, a timeout at the wireless side (shown as *Wireless_Link_Timeout*) would result in the advertised window w_i being set to zero. Upon recovery, we restore the previously advertised window size. Normal flow control takes over after recovery and brings the system state to equilibrium. This flow control scheme works effectively to control the proxy's buffer utilization, and although a little elaborate for that required by current GPRS networks should scale to much higher bandwidths, and can be easily applied to other split TCP applications.

E. ATCP Sender Error Detection and Recovery

Packet losses over a wireless link like GPRS can usually occur due to two reasons: 1) bursty *radio losses* that persist for longer than the link-layer is prepared to keep retransmitting a packet and 2) during cell reselections due to cell update procedure (or even routing area update in GSM) that can lead to a "link-stall" condition from few to many seconds [9]. In both cases, consecutive packets in a window are frequently lost. TCP detects these losses through duplicate ACKs, or timeouts in the extreme case. Not knowing the "nature" of the loss, it reacts by invoking congestion control measures such as *fast retransmit* or *slow start*. However, since the link frequently returns to a healthy state after the loss, unnecessary backoff is often employed resulting in underutilization.

The split TCP aggregate approach of our proxy affords us the opportunity of improved detection of the nature of wireless losses, and hence make a better job of recovery. The aim is to recover aggressively from transient losses, keeping the link at full utilization, but be careful during extensive stalls or blackouts not to trigger unnecessary retransmission.

Fortunately, TCP-SACK enables the receiver to inform the sender of packets which it has received *out of order*. The sender

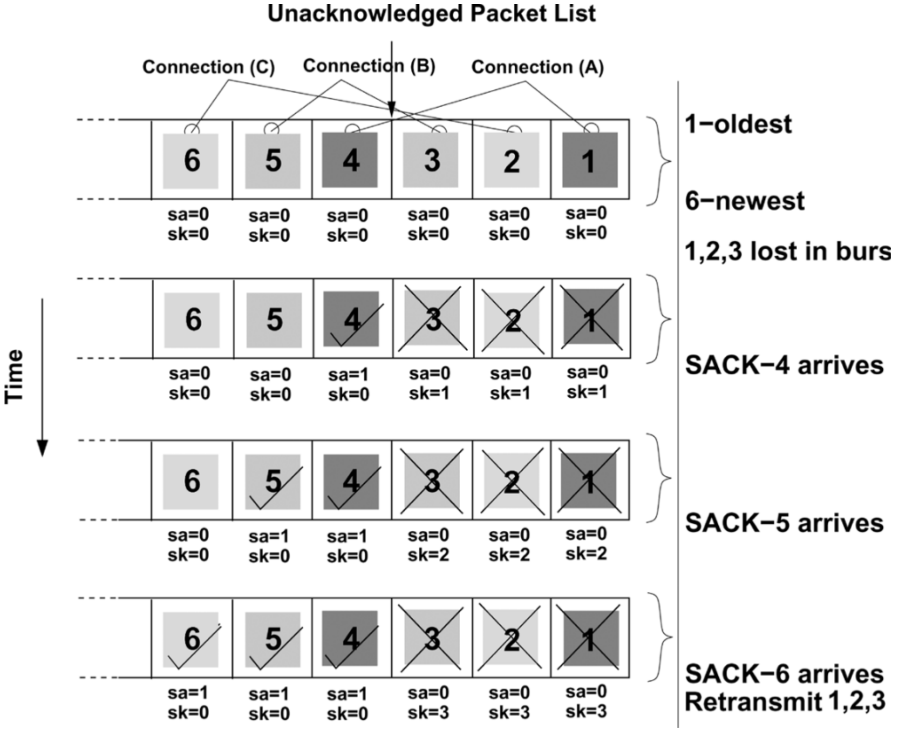


Fig. 7. Skipack scheme for error recovery (per-mobile as seen from the proxy).

can then selectively retransmit in order to fill in the gaps. Recently, empirical study involving large-scale passive analysis for TCP traces over different GPRS networks has shown that use of TCP-SACK leads to much higher per-TCP connection utilization for all-size flow types [15]. When this SACK feature is used along with the flow aggregation concept, it gives us an elegant mechanism to discover sequence gaps across the entire gamut of packets sent by the aggregate TCP sender, and thereby improves our ability to determine the nature of the loss. This improved diagnosis coupled with a recovery strategy fine tuned for the wireless link results in good utilization. The scheme is described further below, and Fig. 7 depicts a snapshot of it in action.

Normal ACK: The ACB maintains a list of unacknowledged packets. Associated with each packet in the list is a field called *skipack* (shown as sk in Fig. 7). Whenever an ACK is received for a packet in the list it is removed. The *skipack* variable of packets which were sent before the acknowledged packet are then incremented. We do this based on the observation that due to the nature of the GPRS link-layer, packet reordering does not occur. We assume packets belonging to the same connection are processed by the mobile host in typical FIFO order. Thus, an ACK received for a newer packet implies loss or corruption of older packets in the same connection.

Bursty Error Period: When there is a bursty error period on the wireless link, multiple packets will be lost. This will result in the generation of duplicate ACKs with SACK information. Packets which are SACKed are marked so that they are not retransmitted later by setting $\mathrm{sacked} = 1$ (shown as sa in Fig. 7). UnSACKed packets $(\mathrm{sa} = 0)$ sent before the packet which was SACKed have their *skipack* counterincremented. The key point is that this is done in the proxy not for the packets of just that particular connection, but for the whole list of unacknowledged packets for the aggregate. Thus, when a DupAck with SACK is received for a particular connection the *skipack* counter for the packets of all connections which were sent before that packet and have not been SACKed are incremented. This is justified since sending order is maintained during reception of ACKs.

However, care must be taken since TCP connections to a mobile host are independent, so ACKs of packets for connections sent later to the host might arrive before packets that were sent earlier. This is not unusual as connections may be employing delayed ACKs. However, if further such "early ACKs" are received, it is increasingly indicative of a transient loss necessitating recovery. Our recovery strategy retransmits aggressively during a transient loss: We wait for the *skipack* counter to reach three, then retransmit all packets having this value in the list. Note that a higher *skipack* retransmit counter value would make the recovery scheme less agile, while lowering its value would result in redundant retransmissions. We have conducted preliminary investigation of this scheme for a reasonable value of *skipack* retransmit counter. For this, we used about 200 min of packet traces[1] involving file downloads over three different cellular GPRS networks in two different conditions: 1) driving at variable speed (10–40 km/hr) and 2) while stationary at different locations from the base station. Initial investigation by analyzing these traces for packet loss patterns for the skipack scheme show that, empirically, a value of 3 for the restransmit counter suits the characteristics of current GPRS networks and terminal devices. Results from the analysis will be available in a separate technical report. We intend to further refine this scheme based on TCP packet-loss patterns from the evaluation of large-scale GPRS traffic traces we are collecting.

[1] These TCP traces (including visual trace close-ups) are available: http://www.cl.cam.ac.uk/users/rc277/traces.html

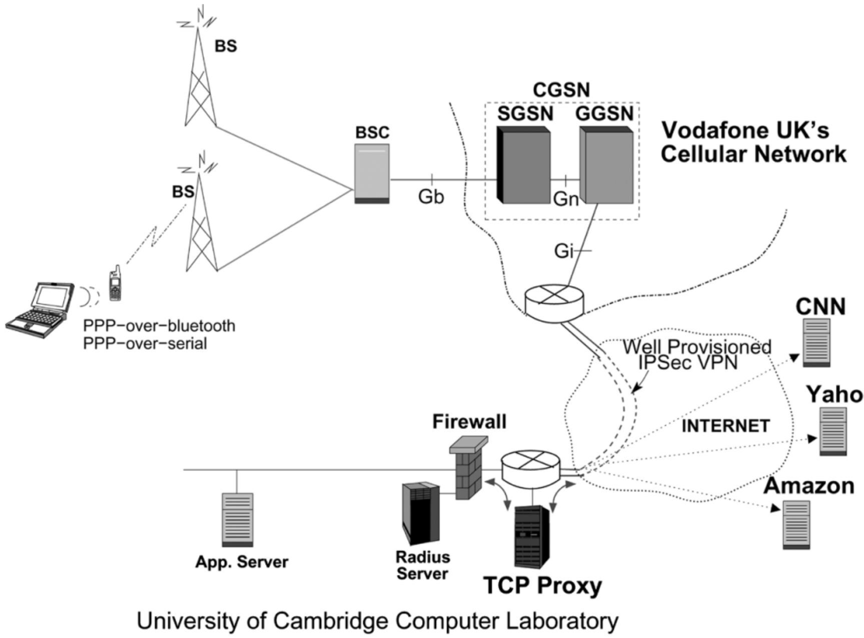


Fig. 8. Experimental testbed setup.

Timeouts: Bursty error periods tend to be short compared to RTTs. As described above, if the *skipack* counter reaches three, we can recover from a loss without resorting to an expensive timeout. Hence, by keeping the timeout value relatively conservative, we can avoid timeouts after bursty error periods. Thus, timeouts only occur during extensive blackouts or link stalls. It would be highly wasteful to keep transmitting during a link blackout (e.g., due to deep fading or during cell-reselection) since a large proportion of the packets will likely be lost. Hence, after a timeout the cwnd is reduced to one. A single packet is transmitted until an ACK is received. Once an ACK is received the cwnd is set back to the original size since the reception of the ACK implies restoration of the link. Hence, after the first timeout, the timeout value is made aggressive by setting it to $1.5 \times \mathrm{RTT}$. After a timeout, the single packet being transmitted is effectively a "probe" packet, and we should resume normal operation as soon as the link recovers. Hence, instead of exponentially backing off, a more aggressive timeout value (linear backoff) is used to enable quick detection of link recovery.

Recovery Strategy: In normal TCP, three duplicate ACKs trigger fast retransmit. On our link, reception of duplicate ACKs signifies that the link is currently in an operational state. Application of fast retransmit would result in unnecessary backoff and, hence, underutilization. Hence, we maintain cwnd at the same value—packets whose *skipack* counter has reached three are simply retransmitted.

V. Experimental Test Setup

Our experimental test bed for evaluating the transparent TCP proxy[2] is shown in Fig. 8. The mobile terminal was connected to the GPRS network via a Motorola T260 GPRS phone (three downlink channels, one uplink). Tests were performed with different mobile terminals, using Linux 2.4, Windows 2000, and WinCE 3.0. Vodafone U.K.'s GPRS network was used as the infrastructure.

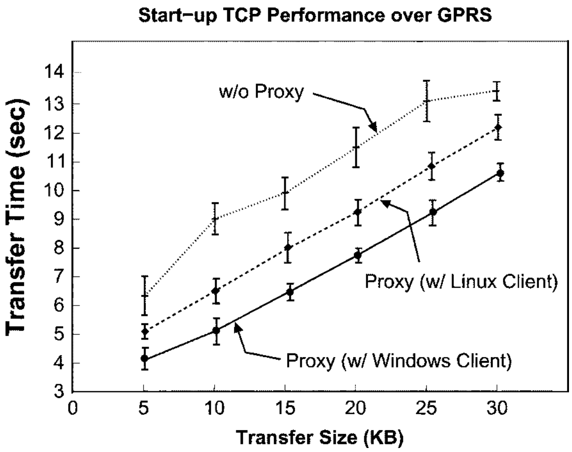
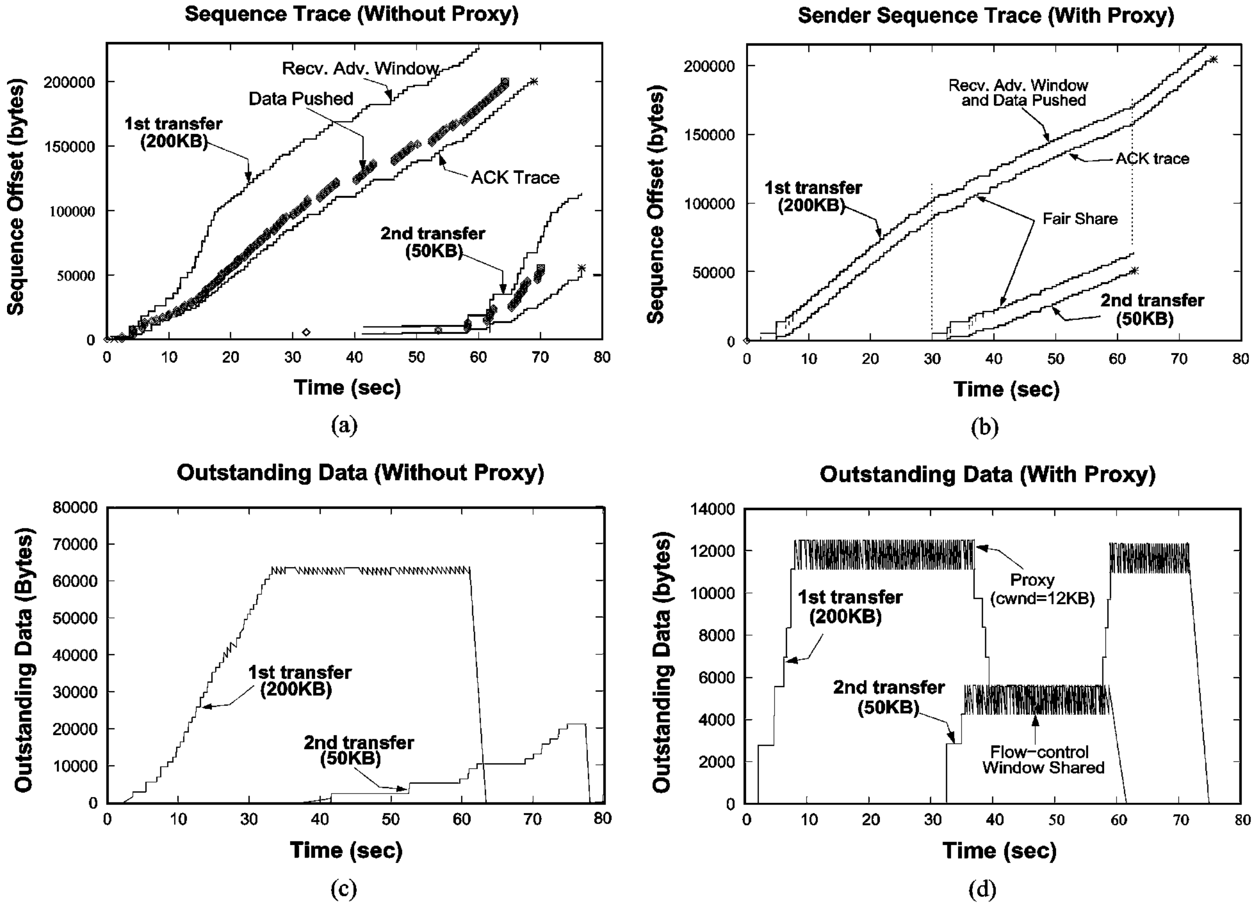


Fig. 9. Results of the download transfers conducted over the GPRS network. Plot shows the transfer times for different transfer sizes with and without the TCP enhancing proxy. The error bars correspond to the standard deviation. Each transfer test was repeated 25 times for a given size. The MSS was set at 1400 bytes in this experiment.

In this setup, base stations are linked to the serving GPRS support node (SGSN), which is connected to a gateway GPRS support node (GGSN). Both the SGSN and GGSN nodes are co-located in a combined GPRS support node (CGSN) in the current Vodafone configuration [36].

Since we were unable to install equipment next to the CGSN, we made use of a well provisioned IPSec VPN tunnel to route all traffic via the Computer Laboratory. The proxy was then located at the end of the tunnel, with routing configured so that all packets flowing to and from the mobile host are passed to it for processing. Packets arriving at the proxy that are to/from hosts in the mobile host address range are passed to the user-space proxy daemon by means of Linux's netfilter [37] module.

[2] The GPL'ed proxy source code is available: http://www.cl.cam.ac.uk/users/rc277/soft.html



(a)

(b)

(c)

(d)

Fig. 10. (a) and (b) Shows the TCP sender side sequence and ack-sequence number time plot for 200 kB transfer, with a second connection of 50 kB initiated after 30 s, without using the proxy. (c) and (d) Shows the outstanding data plot of each connection. Shows sequence trace and the outstanding data for similar transfers but using our proxy. MSS was set at 1400 bytes.

In the test described, a number of different remote "fixed" hosts were used, some located in the Laboratory, some elsewhere on the public Internet, and others were in fact other mobile hosts.

VI. Experimental Results

In this section, we discuss results from experiments conducted over the GPRS testbed using our proxy. We evaluate how our transparent proxy achieves its goals of: faster flow startup; higher downlink utilization; improved fairness (and controllable priority); reduced queueing delays; and better loss recovery.

A. Higher Downlink Utilization

To quantify the benefits of avoiding slow-start for short TCP sessions, we performed a series of short (5–30 kB) downloads from a test server that reflect web session behavior. Each transfer for a given size was repeated 25 times, with traces recorded using `tcpdump` and later analyzed.

Fig. 9 plots the transfer times with and without the mobile proxy. The times shown include the full TCP connection establishment and termination overhead, which constitutes a significant fraction of the overall time for shorter transfers. Even using a large MSS of 1400 bytes, the proxy can be seen to yield clear performance benefits through better link utilization. If smaller MSSs are used, the speedup is more marked, due to the greater number of RTTs that slow-start takes to achieve full link utilization.

Note that when using a Linux 2.4 client, the performance improvement was not better than using Windows 2000 or WinCE clients. This is due to Linux offering an initial receive window of just 5392 bytes. When used with a normal TCP sender, Linux expands the window sufficiently quickly for it to never be the limiting factor. However, since we skip slow start, there is no time for the window to grow and it, thus, limits the quantity of data we can initially inject into the network and, hence, we do not quite achieve our goal of full link utilization. We considered sending further data "optimistically," but rejected the idea as distasteful, and also a potential source of compatibility problems.

Even with the Linux 2.4 receiver, the proxy provides significant performance gains for short-lived flows as are prevalent with HTTP/1.0 transfers. This benefit is maintained and even enhanced when using HTTP/1.1 persistent TCP connections [8]. When using persistent connections it is normally the case that the server has to let the TCP connection go idle between object transfers since "pipelining" is rarely supported. Normally, this results in the congestion window being set back to its initial two segment value. The proxy avoids this, and the resultant benefit is more pronounced due to the lack of connection establishment and termination phases.

B. Achieving Fairness and Flow Control Between Connections

We configured the proxy to distribute equal tickets to incoming flows and, hence, demonstrate that fairness can be achieved between multiple connections. In the following experiments, we initiated one transfer, then started a second about 30 s later.

Without the proxy, Fig. 10(a) and (b) shows the second flow taking a long time to connect, and then suffering very poor performance; it makes little progress until the first flow terminates at about 60 s. During this time, the amount of outstanding data for the first transfer remains the same; however, the second struggles to get going and is unable to initiate sufficiently until the first one terminates.

With the proxy, the second short flow of 50 kB connects quickly in presence of the first, then receives a fair share of the bandwidth. Fig. 10(c) and (d) shows the bandwidth of the first flow being halved during the duration of the second flow. The figure shows how our proxy advertisers updated value of the receiver window, to bring down the amount of outstanding data for the first flow to about half of its initial value (set at slightly more than 20% of effective downlink BDP). Furthermore, our proxy also quickly brings the second flow to its full-utilization. Thus, the proxy works as expected, enabling interactive applications to make progress in the face of background bulk transfers such as FTP.

C. Reduced Queueing

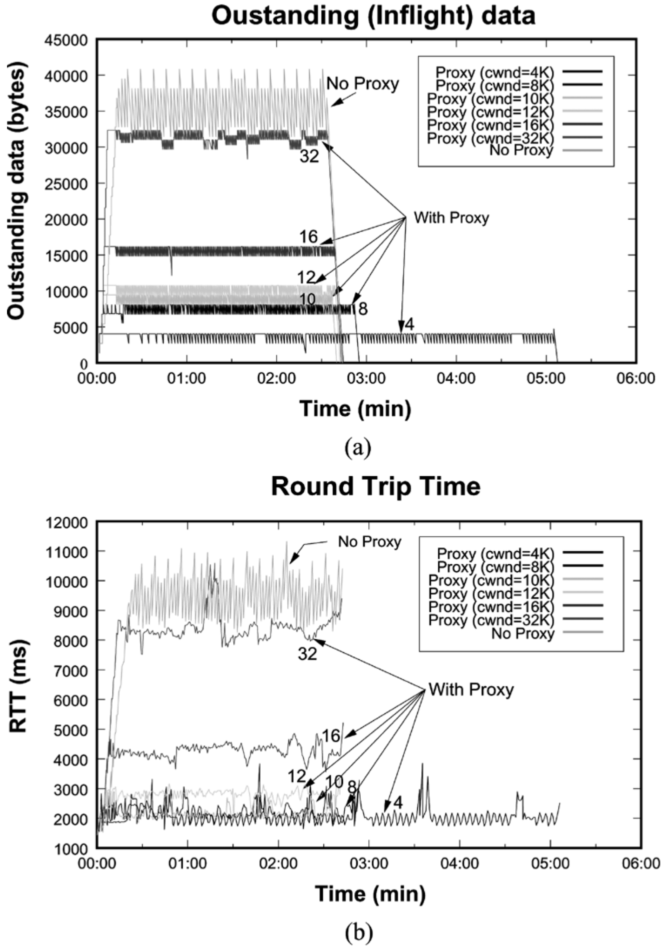
In these tests, we demonstrate how the proxy can achieve reduced queueing (and, hence, RTT), while maintaining high throughput. A 600 kB file is downloaded with and without the proxy. To demonstrate the effect the congestion window has on performance, the experiment has been repeated with the proxy configured such that it uses a static window of various nominated sizes. The experiments were performed under ideal radio conditions so as to minimize chances of packet loss.

Fig. 11(a) shows that under these conditions the transfer takes 155 s with any window size greater than equal to 10 kB. Below this size, the link is underutilized and throughput drops. If the window size is increased beyond 10 kB the level of queueing increases, approaching that of the case without the proxy for a window size of 32 kB. Fig. 11(b) shows how these queues translate into elevated RTT.

D. Faster Recovery

In Fig. 12, we show the implementation's response to a simple loss scenario. Around 90 s into a file transfer, the link stalls due to a cell handoff. A single retransmission occurs, and then the link recovers swiftly. This should be compared with the similar scenario without the proxy previously shown in Fig. 3—recovery is now significantly quicker due to the aggressive (linear) backoff mechanism.

In the presence of multiple flows, applying TCP-SACKs to the entire aggregate can further improve response to loss. To quantify the benefits from the scheme, we are collecting `tcpdump` traces of our user community using the proxy. Work to provide a thorough real-world evaluation of the scheme from these traces is on-going.



(a)

(b)

Fig. 11. (a) Outstanding (in-flight) TCP data and (b) sender perceived RTTs during a 600 kB file transfer. Queueing delay can be reduced by clamping the congestion window (`cwnd`) without effecting throughput. However, a `cwnd` of less than 10 kB leads to underutilization, validating use of *effective* BDP. The numbers highlighted in the plots give different sizes of `cwnd` (in kilobytes) used with the proxy. MSS was set at 1400 bytes.

VII. Issues and Discussion

A. Transparent Proxying in GPRS Networks

Transparent proxies are commonplace in today's Internet [33]. Application-level transparent proxies are frequently employed by Internet service providers (ISPs) to interpose WWW caches and redirect streaming media requests to the closest replicated content. Adopting a similar approach, GPRS (and 3G) network operators could deploy TCP enhancing proxies in their networks to benefit mobile users.

Using transparent proxying, mobile users require no software or configuration changes. Furthermore, layer-4 switching can be used to distribute load from users amongst a set of proxies.

Since TCP enhancing proxies do not require fine-grained wireless channel monitoring, there is flexibility as to their placement in the network. Fig. 13 shows two possibilities: 1) close to the gateway router of the cellular network provider, where

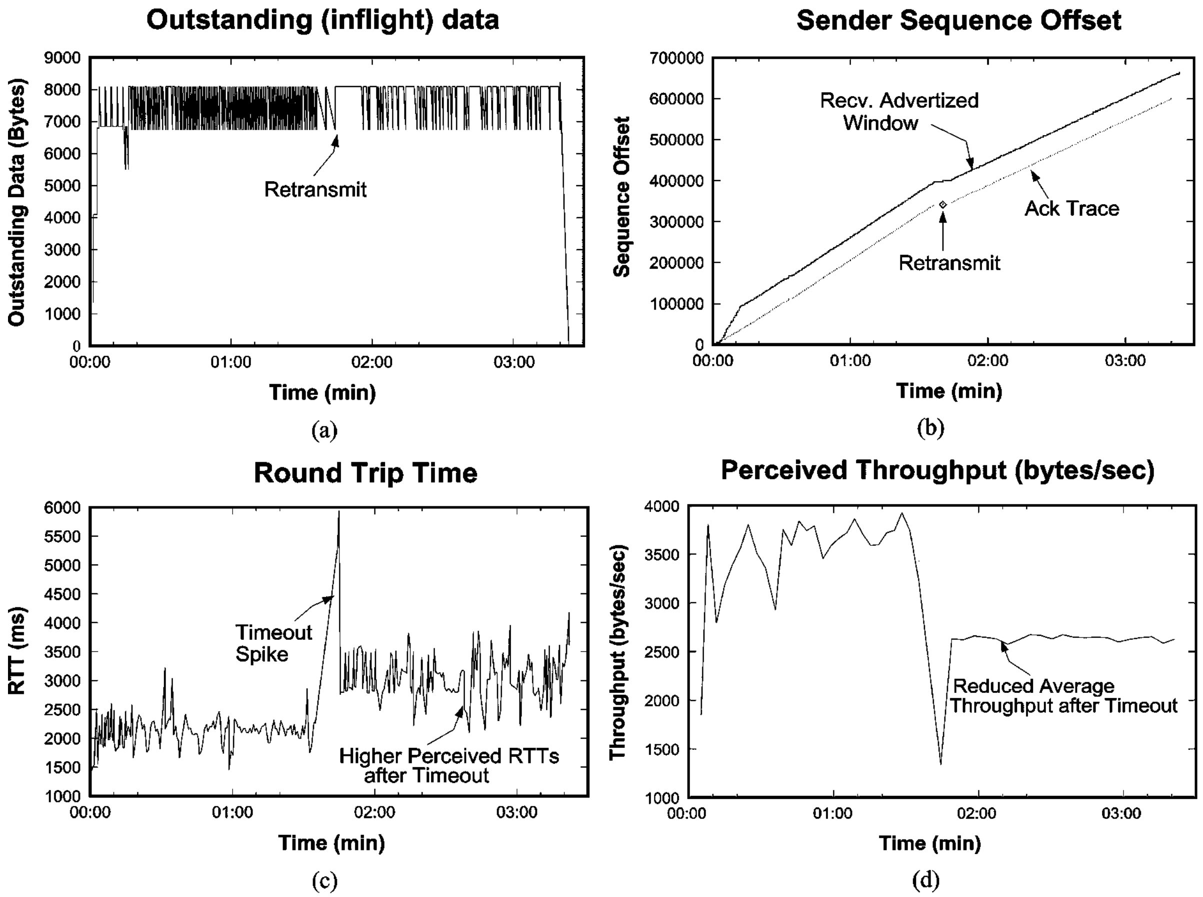


Fig. 12. Swift recovery from a TCP timeout during a 600 kB file transfer, while using the proxy. Plots showing (a) Outstanding (in-flight) data. (b) Sender sequence trace. (c) RTT plot with TCP timeout spike. (d) Receiver perceived throughput. MSS was set at 1400 bytes.

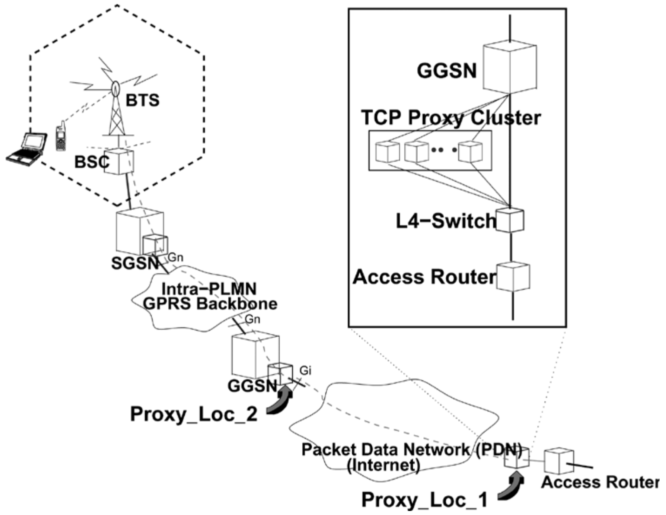


Fig. 13. Transparent TCP proxying in a GPRS network.

traffic density is high (in Fig. 13 shown as `TCP Proxy(1)`) or 2) near to the wired-wireless boundary (i.e., GPRS GGSN node or close), where traffic from a number of mobile hosts is aggregated (shown as `TCP Proxy(2)`).

The close-up plot in Fig. 13 shows one such example using a layer-4 switch to redirect TCP connections to a TCP enhancing proxy cluster. By using such a scheme, issues such as scalability, availability, and reliability can be addressed. Each proxy in the cluster can handle TCP traffic for a set of GPRS mobile clients.

B. Proxies and End-to-End Layering Semantics

On issue to be considered when deploying a proxy such as we propose is the effect on end-to-end semantics and its consequences on overall reliability and security. Using a "split TCP" [6] scheme, the proxy *transparently* divides the connection into two legs: one from the fixed host to the proxy, and the other from the proxy to the mobile host.

We attempt to mitigate the effect of transparent proxying on the overall end-to-end semantics of the TCP connections by never acknowledging ("early-ACKing") the last FIN of each individual TCP connection, hence ensuring the mobile host has all outstanding data before the "close" system call returns at the sender. We know of no applications that rely on stronger semantics than this. Architectural and policy issues relating to transparent proxies are discussed in the Internet Architecture Board's (IAB) Open Pluggable Edge Services document RFC 3238 [32].

VIII. Related Work

The academic literature contains a plethora of solutions for elevating performance over wireless links. Berkeley's SNOOP [14], delayed DupAcks scheme [28], M-TCP [25], I-TCP [22], Freeze-TCP [27], FDA [29], W-TCP [18], WTCP [26], and TCP

TABLE II
WIRELESS TCP SOLUTIONS

TCP Solutions Features	SNOOP[14]	M-TCP[17] I-TCP[3]	Freeze-TCP[37]	FDA[16]	WTCP[28]	W-TCP[18]	TCP Westwood [8]	**Proxy Flow Aggregation**
Change/Modify end TCP	×	×	√	×	√	×	√	×
Avoid Excess Queuing	×	×	×	√	√	×	√	√
Fast Start-up	×	×	×	×	√	×	×	√
Ensure Flow Fairness	×	×	×	×	×	×	×	√
Handle small 'link-stalls'	×	√	×	×	√	√	√	√
Handle Disconnections	×	√	√	×	×	√	×	√
Connection Semantics	√	×	√	√	√	√	√	**Mitigates**

Westwood [23] are some of the more important ones. Careful examination of existing schemes suggests four broadly different approaches: link-layer-based schemes (both TCP aware and unaware) (e.g., [14] and [28]), split connection based approaches (e.g., I-TCP [22] and W-TCP [18]) and early warning based approaches (e.g., Freeze-TCP [27]), and finally those necessitating end system changes (e.g., WTCP [26], Freeze-TCP [27], and TCP Westwood [23]).

Snoop [14] is a TCP aware link-layer scheme that "sniffs" packets in the base station and buffers them. If DupAcks are detected, incoming packets from the mobile host are retransmitted if they are present in a local cache. On the wired side, DupAcks are suppressed from the sender, thus avoiding unnecessary fast retransmissions and the consequent invocation of congestion control mechanisms. End-to-end semantics are preserved. However, when bursty errors are frequent, the wired sender is not completely shielded, leading to conflicting behavior with TCP's retransmission mechanisms.

The Snoop protocol scheme was originally designed for WLANs rather than "long-thin" wide-area wireless links. As such, it does not address the problems of excess queueing at base stations or proxies. Hu and Yeung developed FDA [29], which uses a Snoop-like strategy, but uses a novel flow-control scheme which goes some way to prevent excess queueing. Further, Snoop's link-layer retransmissions and suppression of ACKs can inflate the TCP sender's RTT estimates and, hence, its timeout value, adversely affecting TCPs ability to detect error free conditions.

Delayed DupAcks [28] is a TCP unaware link-layer scheme: it provides lower link-layer support for data retransmissions and a receiver or mobile host side TCP modification that enables the receiver to suppress DupAck's for some interval d when packet(s) are lost over radio. However, the interval d is difficult to determine as it depends on frequency of losses over the wireless medium. The base station is not TCP aware, thus, a receiver cannot determine whether a loss is due to radio errors or congestion, which can force the sender to timeout in such situations. Further, DupAck bursts can also aggravate congestion over wire-line links that have high BDP.

In [16], Chan and Ramjee propose *ACK Regulator* for improving end-to-end TCP performance over CDMA 2000-1X-based 3G wireless links. The scheme requires no changes to end-host systems. However, 3G links offer much higher channel variations than GSM-based GPRS due because of the processing delays and rate variations involved in the channel-based scheduling schemes used [16]. Schemes like *ACK regulator* can be used to handle the impact of such high rate and delay variations, which also mitigates the extent of excess data buffering in 3G.

The second broad approach is to split the TCP connection into two sections. This allows wireless losses to be completely shielded from the wired ones. I-TCP [22] uses TCP over the wireless link albeit with some modifications. Since TCP is not tuned to the wireless link, it often leads to timeouts eventually causing stalls on the wired side. Due to the timeouts, valuable transmission time and bandwidth is also wasted. I-TCP could also run short of buffer space during periods of extended timeouts due to the lack of an appropriate flow control scheme.

M-TCP [25] is similar to I-TCP except it better preserves end-to-end semantics. M-TCP uses a simple zero window ACK scheme to throttle transmission of data from the wired sender. This leads to stop-start-stop bursty traffic on the wired connection, and the lack of buffering in the proxy can lead to link underutilization for want of packets to send. Holding back ACKs also affects sender's RTT estimates, affecting TCP's ability to recover from nonwireless related packet losses.

Ratnam and Matta propose W-TCP [18], which also splits the connection at the base station. However, it acknowledges a packet to the sender only after receiving an acknowledgment from the mobile host. W-TCP changes the timestamp field in the packet to account for the time spent idling at the base station. Retransmission characteristics have been adjusted to be aggressive on the wireless side so that the link is not underutilized.

WTCP [26] is an end-to-end scheme which primarily uses interpacket separation as the metric for rate control at the receiver. Congestion related loss detection is also provided as a backup mechanism. A drawback in WTCP is that it entails changes at both the wired sender and the mobile host. Even if a receiver side change can be envisaged, widespread adoption by wired senders seems unlikely.

TCP Westwood [23] is a scheme that improves TCP performance under random and sporadic losses, by desisting from overly shrinking the congestion window on packet loss. It does so by simultaneously estimating end-to-end bandwidth available to TCP, and uses it as a feedback measure to control the congestion window. However, it requires modification to TCP at the end-system.

The third broad approach identified covers schemes that use various kinds of early warning signals. Freeze-TCP uses zero window probes (ZWPs) like M-TCP, but is proactive since the mobile host detects signal degradation and sends a zero window warning probe. The warning period, i.e., the time before which actual degradation occurs should be sufficient for the ZWP to reach the sender so that it can freeze its window. The warning period is estimated on the basis of RTT values. One pitfall is the reliability of this calculation and Freeze-TCP's inability to deal with sudden random losses. Furthermore, Freeze-TCP requires end-system changes.

IX. Conclusion and Ongoing Research

In this paper, we have proposed a flow aggregation scheme to enhance data performance transparently of TCP flows over cellular wireless links such as GPRS. The proposed split TCP scheme was implemented and evaluated on a GPRS network testbed. We summarize the key features of this proxy.

- *Fast startup*: The proxy avoids slow start during connection startup, thus quickly bringing the link up to full utilization, to the particular benefit of short-lived flows.
- *Flow fairness*: The proxy explicitly schedules packets in the aggregate to fairly allocate bandwidth between flows and control the amount of data outstanding over the wireless link.
- *Agile recovery*: Error recovery over GPRS is significantly improved due to the extension of the SACK mechanism to cover the entire aggregate. As a result, using the proxy losses are detected faster and recovery is done without unnecessary backoff.
- *Smart buffer management*: Within the proxy, the flow control algorithm manages proxy buffer space to ensure sufficient data is buffered to keep the wireless link fully utilized, but tries to adjust flows smoothly, and limits the buffer space committed to each mobile host.

In ongoing research, we are recording full `tcpdump` packet traces of the GPRS traffic generated by our user community. We plan to perform empirical analysis of this long-term trace to quantify the real-world performance benefits provided by the proxy for the applications in use by our users.

We are currently considering ways of improving our wireless link BDP estimation algorithm. Although not particularly critical for GPRS where the BDP stays roughly constant even under changing radio conditions and even during cell reselections, we feel that a more dynamic scheme may be required for EDGE and UMTS (3G) systems. Calculating the BDP of the aggregate in a similar manner to that employed by TCP Vegas or Westwood seems a promising approach.

The current work using the proxy has focussed on the downlink direction as we perceive this as being the most critical direction for performance of most applications. Besides the downlink, we are also considering how the split TCP approach could be used to improve the uplink, but the options without modifying the mobile host are rather more limited. Fortunately, vanilla uplink performance does not exhibit many of the gross problems posed by the downlink.

Acknowledgment

The authors wish to thank Vodafone Group R&D, Sun Microsystems, Inc., and BenchMark Capital for supporting this work. Thanks also go to T. Harris for comments on an earlier version of this paper.

References

[1] G. Brasche and B. Walke, "Concepts, services, and protocols of the new GSM phase 2+ general packet radio service," *IEEE Commun. Mag.*, pp. 94–107, Aug. 1997.

[2] B. Walke, *Mobile Radio Networks, Networking, and Protocols*, 2nd ed. New York: Wiley, 2001.

[3] C. Bettssetter, H. Vogel, and J. Eberspacher, "GSM phase 2+ general packet radio service GPRS: Architecture, protocols, and air interface," *IEEE Commun. Surveys*, vol. 2, no. 3, 3rd Quarter 1999.

[4] M. Meyer, "TCP performance over GPRS," in *Proc. IEEE WCNC*, 1999, pp. 1248–1252.

[5] R. Ludwig, B. Rathonyi, A. Konrad, K. Oden, and A. Joseph, "Multilayer tracing of TCP over a reliable wireless link," in *Proc. ACM SIGMETRICS*, 1999, pp. 144–154.

[6] O. Spatscheck, J. Hansen, J. Hartman, and L. Peterson, "Optimizing TCP forwarder performance," *IEEE/ACM Trans. Netw.*, vol. 8, no. 2, pp. 146–157, Apr. 2000.

[7] R. Chakravorty, J. Cartwright, and I. Pratt, "Practical experience with TCP over GPRS," in *Proc. IEEE GLOBECOM*, Nov. 2002, pp. 1678–1682.

[8] R. Chakravorty and I. Pratt, "WWW performance over GPRS," in *Proc. IEEE Int. Conf. Mobile Wireless Commun. Netw.*, Sep. 2002, pp. 527–531.

[9] ——, "Performance issues with general packet radio service," in *J. Commun. Netw. (JCN) (Special Issue on Evolving from 3G Deployment to 4G Definition)*, vol. 4, Dec. 2002, pp. 266–281.

[10] D. Dutta and Y. Zhang, "An active proxy based architecture for TCP in heterogeneous variable bandwidth networks," in *Proc. IEEE GLOBECOM*, 2001, pp. 2316–2320.

[11] C. Waldspurger and W. Weihl, "Stride scheduling: Deterministic proportional-share resource management," Massachusetts Inst. Technol., Cambridge, MA, Tech. Rep.—MIT/LCS/TM-528.

[12] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," in *IEEE/ACM Trans. Netw.*, vol. 5, Dec. 1997, pp. 756–769.

[13] H. Balakrishnan, H. Rahul, and S. Seshan, "An integrated congestion management architecture for Internet hosts," in *Proc. ACM SIGCOMM*, 1999, pp. 175–187.

[14] H. Balakrishnan, R. Katz, and S. Seshan, "Improving TCP/IP performance over wireless networks," in *Proc. ACM MobiCom*, 1995, pp. 2–11.

[15] P. Benko, G. Malicsko, and A. Veres, "A large-scale, passive analysis of end-to-end TCP performance over GPRS," in *Proc. IEEE INFOCOM*, Mar. 2004, pp. 1882–1892.

[16] M. C. Chan and R. Ramjee, "TCP/IP performance over 3G wireless links with rate and delay variation," in *Proc. ACM MobiCom*, 2002, pp. 71–78.

[17] L. Zhang, S. Shenker, and D. Clark, "Observations on the dynamics of a congestion control algorithm: The effects of two-way traffic," in *Proc. ACM SIGCOMM*, 1991, pp. 30–39.

[18] K. Ratnam and I. Matta, "W-TCP: An efficient transmission control protocol for networks with wireless links," in *Proc. 3rd IEEE Symp. Comput. Commun.*, 1998, pp. 74–78.

[19] L. Kalampoukas, A. Varma, and K. K. Ramakrishnan, "Explicit window adaptation: A method to enhance TCP performance," in *Proc. INFOCOM*, 1998, pp. 242–251.

[20] L. L. H. Andrew, S. Hanly, and R. Mukthar. Analysis of a flow control scheme for rate adjustment by managing flows. presented at 4th Asian Control Conf. [Online]. Available: http://www.ee.mu.oz.au/staff/lha/LAicp.html

[21] Z. Wang and P. Cao. Persistent connection behavior of popular browsers. [Online]. Available: http://www.cs.wisc.edu/cao/papers/persistent-connection.html

[22] A. Bakre and B. R. Badrinath, "I-TCP: Indirect TCP for mobile hosts," in *Proc. 15th IEEE Conf. Distrib. Comput. Syst.*, May 1995, pp. 136–143.

[23] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth estimation for enhanced transport over wireless links," in *Proc. ACM MobiCom*, 2001, pp. 287–297.

[24] M. Kim and B. D. Noble, "Mobile network estimation," in *Proc. ACM MobiCom*, 2001, pp. 298–309.

[25] K. Brown and S. Singh, "M-TCP: TCP for mobile cellular networks," in *Proc. ACM SIGCOMM Comput. Commun. Rev. (CCR)*, vol. 27, 1997, pp. 19–43.

[26] P. Sinha, N. Venkitaraman, R. Sivakumar, and V. Bhargavan, "WTCP: A reliable transport protocol for wireless wide-area networks," in *Proc. ACM MobiCom*, 1999, pp. 231–241.

[27] T. Go, J. Moronski, D. S. Phatak, and V. Gupta, "Freeze-TCP: A true end-to-end enhancement mechanism for mobile environments," in *Proc. IEEE INFOCOM*, 2000, pp. 1537–1545.

[28] V. Bakshi, P. Krishna, N. H. Vaidya, and D. K. Pradhan, "Improving performance of TCP over wireless networks," in *Proc. 17th IEEE Conf. Distrib. Comput. Syst.*, May 1997, pp. 365–373.

[29] J.-H. Hu and K. L. Yeung, "FDA: A novel base station flow control scheme for TCP over heterogeneous networks," in *Proc. IEEE INFOCOM*, 2001, pp. 142–151.

[30] A. Gurtov, M. Passoja, O. Aalto, and M. Raitola, "Multi-layer protocol tracing in a GPRS network," in *Proc. IEEE Veh. Technol. Conf.*, Sep. 2002, pp. 1612–1616.

[31] H. Inamura, G. Montenegro, R. Ludwig, A. Gurtov, and F. Khafizov, "TCP over second (2.5G) and third (3G) generation wireless networks," Request for Comments, RFC 3481 (BCP 71), Feb. 2003.

[32] S. Floyd and L. Diagle, "IAB architectural and policy considerations for open pluggable edge services," Request for Comments, RFC 3238, Jan. 2002.

[33] G. Barish and K. Obraczka, "World wide web caching: Trends and techniques," *IEEE Commun. Mag.*, ser. Internet Technology Series, pp. 178–184, May 2000.

[34] P. Stuckmann, N. Ehlers, and B. Wouters, "GPRS traffic performance measurements," in *Proc. IEEE Veh. Technol. Conf.*, Sep. 2002, pp. 1289–1293.

[35] J. Cartwright. GPRS link characterization. [Online]. Available: http://www.cl.cam. ac.uk/users/rc277/linkchar.html

[36] *An Introduction to the Vodafone GPRS Environment and Supported Services*, Dec. 2000. Issue 1.1/1200.

[37] The Linux NetFilter Homepage. [Online]. Available: http://www.netfilter.org

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

Rajiv Chakravorty (S'98) He received the B.E. degree from Nagpur University, Nagpur, India, in 1997 and the M.Tech. degree from the Indian Institute of Technology (IIT), Delhi, in 1999. He is working towards the Ph.D. degree at the Computer Laboratory, University of Cambridge, Cambridge, U.K.

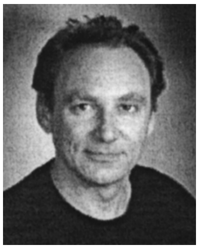
Since January 2005, he has been a Research Scholar in the Department of Computer Sciences, University of Wisconsin, Madison. He has worked with Philips Research, ASA Laboratories, Eindhoven, The Netherlands. He also pursued research at ComNets, RWTH-Aachen, Germany. His current research interests include mobile and wireless systems, and networking.

Mr. Chakravorty is a recipient of DAAD Scholarship Award from Germany, and the Sun Microsystems Scholarship and Hughes Hall CommonWealth Scholarship from Cambridge University.

Sachin Katti (S'02) received the B.Tech. degree from the Electrical Engineering Department, Indian Institute of Technology (IIT), Bombay, in 2003. He is currently working towards the Ph.D. degree in computer science at the Massachusetts Institute of Technology (MIT), Cambridge, where he does research in network security, routing, and network measurements.

Ian Pratt (M'03) received the Ph.D. in computer science from University of Cambridge, Cambridge, U.K.

He was elected a Fellow of King's College, Cambridge, U.K., in 1996. He is a Senior Lecturer at the Computer Laboratory, University of Cambridge, Cambridge, U.K. He is a leader of the Systems Research Group, where he has been architect of a number of influential projects, including the Desk Area Network workstation, the Cambridge Open Mobile System, the Xen Virtual Machine Monitor, and the XenoServer infrastructure for global computing. His research interests cover a broad range of systems topics, including computer architecture, operating system design, mobile systems, and networking.



Jon Crowcroft (SM'95–F'04) is the Marconi Professor of Networked Systems in the Computer Laboratory, University of Cambridge, Cambridge, U.K. Prior to that, he was a Professor of Networked Systems in the Computer Science Department, University College London (UCL). He has published five books—the latest is *Linux TCP/IP Implementation* (New York: Wiley, 2001).

Dr. Crowcroft is a Fellow of the Association for Computing Machinery (ACM), a Fellow of the British Computer Society, a Fellow of the IEE, and a Fellow of the Royal Academy of Engineering. He was a member of the IAB; was General Chair for the ACM SIGCOMM from 1995 to 1999. He is on the Editorial Team for COMNET, and on the Program Committee for the ACM SIGCOMM and the IEEE INFOCOM.