Flow Aggregation for Enhanced TCP over Wide-Area Wireless

Rajiv Chakravorty, Sachin Katti, Jon Crowcroft, Ian Pratt
{firstname.lastname}@cl.cam.ac.uk
University of Cambridge Computer Laboratory,
JJ Thomson Avenue, Cambridge CB3 0FD, U.K.

Abstract— **Throughout the world, GSM cellular mobile networks are being 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 currently considerably worse.**

In this paper we examine the performance of TCP and HTTP over GPRS, and show how certain network characteristics interact badly with TCP to yield problems such as: link under-utilization for short-lived flows, excess queueing for long-lived flows, ACK compression, 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 transparently splits TCP connections 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 maximise performance of the wireless link while inter-working 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 conclude that installing such a proxy into GPRS network would be of significant benefit to users.

I. Introduction

World over, 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 IP-based applications and services a reality.

However, despite the momentum behind GPRS, surprisingly little has been done to evaluate 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 LANs 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.

Sachin Katti is a final year BTech student in the Department of Electrical Engineering at the Indian Institute of Technology (IIT), Mumbai. He interned at the University of Cambridge Computer Laboratory during summer 2002.

In this paper, we present our practical experiences using GPRS networks, and our attempts to improve their performance. In section II we briefly report on work to characterize GPRS link behaviour (see [30] 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 maximise link utilizaton, reduce latency, and improve fairness between flows.

Sections V and VI describe the experimental setup and present an evaluation of our proxy's performance. The paper goes on to discuss related work, and how flow aggregation could be added as an extension to existing TCP implementations.

II. GPRS Network Characterization

GPRS [1], [2], [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], [8], [9]. To gain clear insight into the characteristics of the GPRS link, we have conducted a series of link characterization experiments. A comprehensive report on GPRS link characterization is available in the form of a separate technical report [30]. Below, we enunciate some key findings:

High and Variable Latency:- GPRS link latency is very high, 600ms-3000ms for the downlink and 400ms-1300ms on the uplink. Round-trip latencies of 1000ms or more are typical. The delay distribution is shown in figure 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) [28], [29]. 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 (approx. 100-200ms) during data transfer. The latest release (GPRS 1999) does consider an extended TBF life-time; however, this optimization can lead to inefficient scheduling at the base station controller (BSC), with some improvement ($\sim$100ms) in overall RTTs [28].

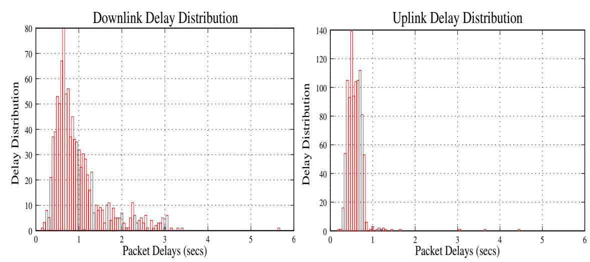

Fig. 1. Single packet time-in-flight delay distribution plots showing (a) downlink delay (b) uplink delay distribution. Measurements involved transfer of 1000 packets with random intervals $>$ 4s between successive packet transfers.

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 (3 downlink channels, 1 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 of 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 non-congestive 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 40s. Sudden signal quality degradation, prolonged fades and intra-zone 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 base station controllers (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 behaviour of mobile applications, such as web browsing, file download, reading email, news etc. We provide a more complete treatment on TCP problems over GPRS in [7].

TCP Start-up Performance:- Figure 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 link bandwidth delay product (BDP) is also marked, approximately 10KB. This 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. For a TCP connection to fully utilize the available 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 from the initial connection request (TCP SYN) to ramp the congestion window up to the link BDP. Hence, for transfers shorter than about 18KB, 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.

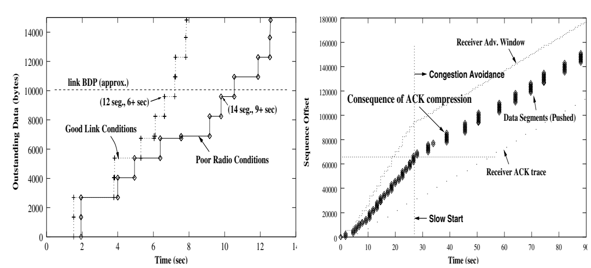

Fig. 2. Plot (a) shows that slow-start takes over 6 seconds to expand the congestion window sufficiently to enable the connection to utilise the full link bandwidth. (b) shows the characteristic exponential congestion window growth due to slow-start (SS).

ACK Compression:- 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 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 when the mobile terminal has data to send and receive concurrently. ACK bunching or compression [15] not only skews upwards the TCP's RTO measurement but also affects its self-clocking strategy. Sender side packet bursts can

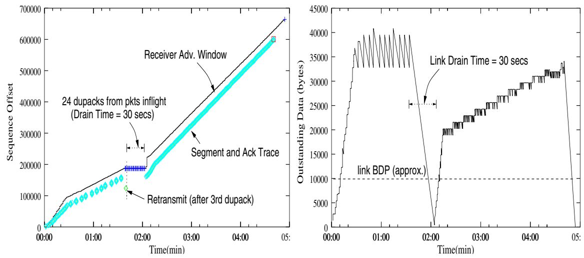

Fig. 3. Case of timeout due to a dupack(sack). Plot (a) shows the sender sequence trace and (b) shows corresponding outstanding data.

further impair RTT measurements.

Excess Queuing:- 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 figure 8). However, we found that the existing GPRS infrastructure offers substantial buffering: UDP burst tests indicate that over 120KB of buffering 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 64KB unless *window scaling* is explicitly enabled. Even so, this still amounts to several times the BDP of unnecessary buffering, leading to grossly inflated RTTs due to queueing delay. Figure 3 (b) shows a TCP connection in such a state, where there is 40KB of outstanding data leading to a measured RTT of tens of seconds. Excess queueing exacerbates other issues:

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 time-out [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 figure 3(a) where the *drain time* is about 30s.

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). 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 emphasises 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.

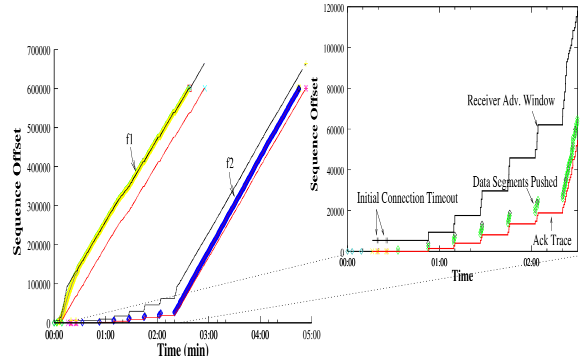

Fig. 4. Close-up of time sequence plots for two concurrent file transfers over GPRS, where `f2` was initiated 10 seconds after `f1`.

Fairness between flows:- Excess queueing can lead to gross unfairness between competing flows. Figure 4 shows a file transfer (f2) initiated 10 seconds 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 seconds) and bear the full brunt of excess queueing delays due to f1. Flow f2 continues to badly underperform until f1 terminates. Flow fairness turns out to be an important issue for web browsing performance, since most browsers open multiple concurrent HTTP connections [19]. The implicit favouring of long-lived flows often has the effect of delaying the "important" objects that the browser needs to be able to start displaying the partially downloaded page, leading to decreased user perception of performance.

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 the proxy were 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 characterised by occasional link stalls during hand-offs and the occurrence of burst losses. A loss detection and recovery scheme tailored for this environment might avoid unnecessary backing-off or packet retransmission.

Effective Flow Control Mechanism – We wished to avoid the problems with excessive buffering at the wired-wireless gateway, and have the proxy take responsibility for effective buffer management using ‘smart’ techniques.

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 (since they share the same wireless 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 figure 5. Hari Balakrishnan *et al.* [13] show that flows can learn from each other and share information about the congestion state along the network path, which they call *shared state learning*. They use it as a basis for their *congestion manager*.

On similar lines, our proxy can share state 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 figure 6.

The wired-facing side of the proxy is known as the AggregateTCP (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. A single congestion window for the whole aggregate is maintained, and whenever the level of unacknowledged data on the wireless link drops one MSS (Maximum Segment Size) below the size of the current congestion window the scheduler selects a connection with queued data from which a further segment will be sent. Of course,

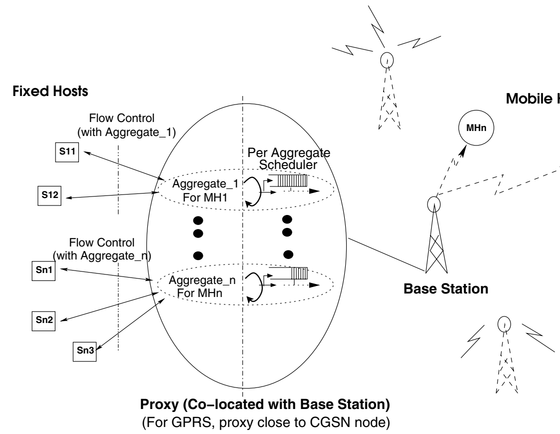

Fig. 5. Proxy based TCP flow aggregation scheme

while making the selection the scheduler must respect 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. The practical effect this has on TCP’s end-to-end semantics is mitigated by never using early acknowledgement for FINs. Since the per-connection queues contain re-assembled data, the proxy can sometimes coalesce multiple small packets from the sender into a single segment by the time it is sent over the wireless link. This can help to reduce protocol header overhead.

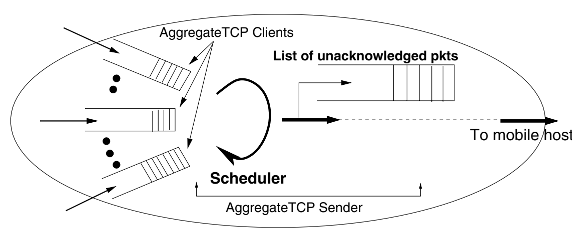

Fig. 6. Sample logical aggregate in the proxy for a given MH

Our 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 Bandwidth Delay Product (BDP) of the link. Thus, slow start is eliminated, and further unnecessary growth of the congestion window beyond the BDP is avoided. We call this TCP `cwnd` clamping.

The underlying GPRS network 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 utilised if there is data to send, and there should only ever be minimal queueing at the CGSN gateway. Typically, the ideal C_{clamp} ends up being around 30% larger than the value calculated by multiplying the maximum link bandwidth by the typical link RTT. This excess is required due to link jitter, use of delayed ACKs by the TCP receiver in the mobile host, and ACK compression occurring due to the link layer.

While starting with a fixed value of `cwnd`, the mobile proxy needs to ensure that any initial packet burst does not overrun CGSN buffers. Since the BDP of current GPRS links is small ($\approx$10KB), this is not a significant problem at this time. Future GPRS 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.

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, it can control the amount of data it accepts through the receive window it advertises to hosts.

The proxy must 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. Zero window advertisements should only be used as a last resort, for example, during an extended wireless link stall.

Some studies in the past have investigated receiver advertized window adaptation schemes. In [17], L. Kalampoukas *et al.* present Explicit Window Adaptation (EWA), which controls the end-to-end receiver advertized 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, L.L.H. Andrew *et al.* [18] propose a TCP flow-control scheme that takes feedback available from an access router to set the TCP receiver window. The scheme claims a higher utilization for TCP flows while still preserving system stability. Following a similar approach, we build on this flow control scheme, utilising feedback available from the aggregate about the wireless link performance.

Pseudocode for the 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 $s_i(t_k)$, $sa(t_k)$
    if($s_i(t_k) \geq sa(t_k)$)
        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

The pseudo code for the flow control scheme is given. Notice 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. We calculate $s_i(t_k)$ 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, the averaging window can be small.

We consider the case of a single (first) TCP connection in an aggregate. To start with, we can use an initial value of advertized window of W_{init} = RTT_{init} × S_{init} × β, where ($RTT_{init} \times S_{init}$) corresponds to the bandwidth-delay product of the GPRS downlink. A fixed GPRS-specific RTT_{init} value can be used initially and later estimated (using appropriate estimation filters [22]), while S_{init} – the initial sending rate can also be intially estimated. β is the *over-provision* factor, and gives control over an aggregate's buffer target set-point. In experiments, we have over-provisioned each aggregate buffer by 50% of the link BDP, which has ensured that there is always sufficient data to avoid under-utilization of the GPRS link.

When more new 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 maximum segment size (MSS), and is calculated as $max(MSS, \frac{W_{init}}{n})$, where n is number of connections in the aggregate at time t_k. For the first advertized window value for any connection, $w_i(t_{k-1}) = 0$, hence $w_i(t_k) = C$. This ensures that new and small connections are not discriminated against during the initial start 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 i^{th} 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 in the *Pseudocode* 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 approaches (b+c)/a (*i.e.* $q_f \rightarrow \frac{b+c}{a}$) [18]. 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 equation 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 equation 1 allows advertized 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.

As also shown in the *pseudocode*, the scheme is modified to make use of 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* in *pseudocode*) would result in the advertised window w_i being set to zero. Upon recovery, we restore the previously advertized 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 routing area update) that can lead to a 'link-stall' condition from few to many seconds [28]. 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 back-off is often employed resulting in under-utilization.

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.

TCP-SACK enables the receiver to inform the sender of packets which it has received *out of order*. The sender can then selectively retransmit in order to fill in the gaps. When this 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 figure 7 depicts a snapshot of it in action.

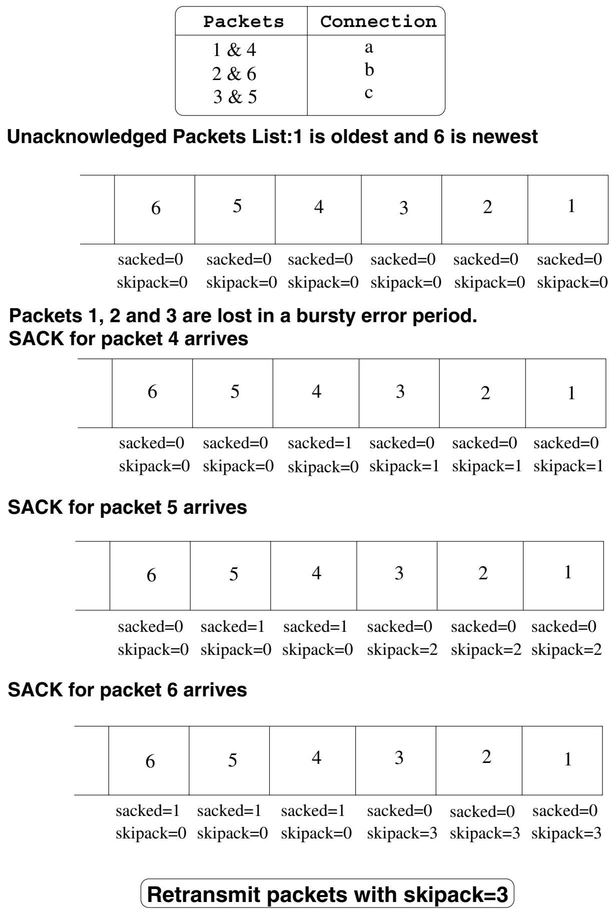

Fig. 7. An example showing skipack scheme for error recovery

Normal Ack – The ACB maintains a list of unacknowledged packets. Associated with each packet in the list is a field called $skipack$. 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 re-ordering does not occur. We assume packets belonging to the same connection are processed by the mobile host in 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 $sacked = 1$. UnSACKed packets ($sacked = 0$) sent before the packet which was SACKed have their $skipack$ counter incremented. The key point is that this is done 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 3 then retransmit all packets having this value in the list. A higher $skipack$ retransmit counter value would make the recovery scheme less agile, while lowering its value would result in redundant retransmissions. Empirically, a value of 3 seems to work well for the characteristics of GPRS.

Timeouts – Bursty error periods tend to be short compared to round trip times. As described above, if the $skipack$ counter reaches 3 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 (for 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 $1.5 \times RTT$. Thus, 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, 3 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 back-off and hence under-utilisation. Hence, we maintain `cwnd` at the same value – packets whose $skipack$ counter has reached 3 are simply retransmitted.

V. Experimental Test Set-up

Our experimental test bed for evaluating the transparent proxy is shown in figure 8. The mobile terminal was connected to the GPRS network via a Motorola T260 GPRS phone (3 downlink channels, 1 uplink). Tests were performed with different mobile terminals, using Linux 2.4, Windows 2000 and WinCE 3.0. Vodafone UK's GPRS network was used as the infrastructure.

Base stations are linked to the SGSN (Serving GPRS Support Node) which is connected to a GGSN (Gateway GPRS Support node). Both the SGSN and GGSN nodes are co-located in a CGSN (Combined GPRS Support Node) in the current Vodafone configuration [31].

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

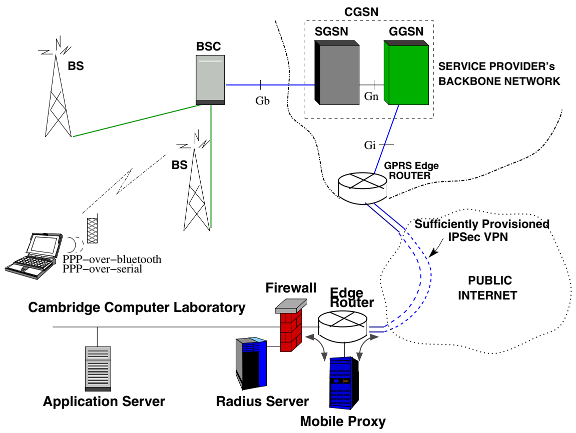

Fig. 8. Experimental Test Bed Set-up

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 [32] module.

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

VI. Experimental Results

In this section, we discuss some preliminary 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 down-link utilization; improved fairness (and controllable priority); reduced queuing 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 (5KB-30KB) downloads from a test server that reflect web session behaviour. Each transfer for a given size was repeated 25 times, with traces recorded using `tcpdump` and later analysed.

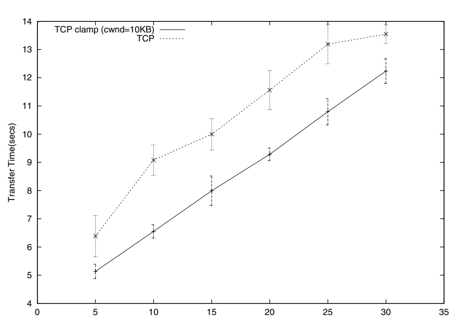

Fig. 9. Results of the `ttcp` download transfers conducted over GPRS network. Plot shows the transfer times for different transfer sizes with and without the proxy. The error bars correspond to the standard deviation. Each transfer test was repeated 25 times for a given size.

Figure 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. However, the clamped congestion window (C_{clamp} around 10KB in this case) can be seen to still yield clear performance benefits.

The results reported above were performed using a mobile host running Linux 2.4, where we noted that the performance gain was *less* than with other platforms. 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 not 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 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.

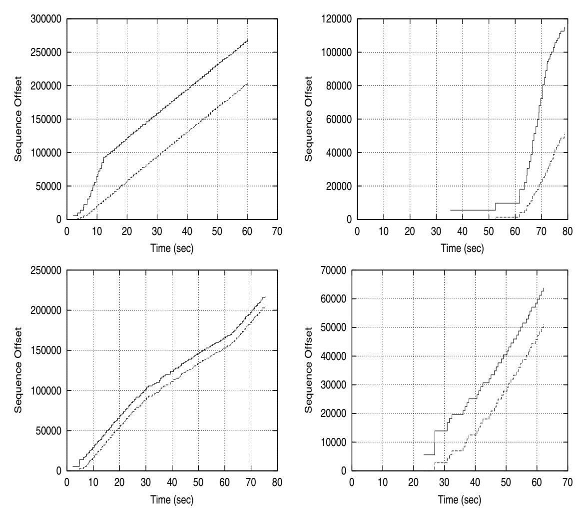

Fig. 10. The plot on the top left show the TCP sender side sequence and ack-sequence number time plot for a transfer, with a second connection (top right plot) initiated after 20 seconds, without using the proxy. The bottom left and right plot is a repeat but using our mobile proxy.

B. *Achieving Fairness 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 20 seconds later.

Without the proxy, the top half of figure 10 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 time 60 seconds. With the proxy, the second flow connects quickly, then receives a fair share of the bandwidth. The bottom left graph shows the bandwidth to the first flow being halved during the duration of the second flow. 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 600KB 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 minimise chances of packet loss.

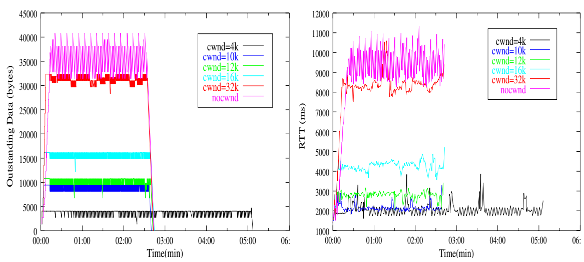

Fig. 11. Figures showing (a) Outstanding (inflight) TCP data and (b) sender perceived RTTs during a 600KB file transfer. Queueing delay can be reduced by clamping the congestion window without effecting throughput.

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

D. Faster Recovery

We developed our TCP aggregate error recovery scheme using NS2 simulation of test cases. Further work to provide a thorough real-world evaluation of the scheme is on-going. Here, we show the implementation's response to a simple loss scenario (see figure 12). Around 90 seconds into a file transfer the link stalls, presumably due to a cell hand-off. A single retransmission occurs, and then the link recovers swiftly. This should be compared with the similar scenario without the proxy previously shown in figure 3 – recovery is significantly quicker.

VII. Related Work

The academic literature contains a plethora of solutions for elevating performance over wireless links. Berkeley's SNOOP [14], delayed dupAcks scheme [26], M-TCP [23], I-TCP [20],

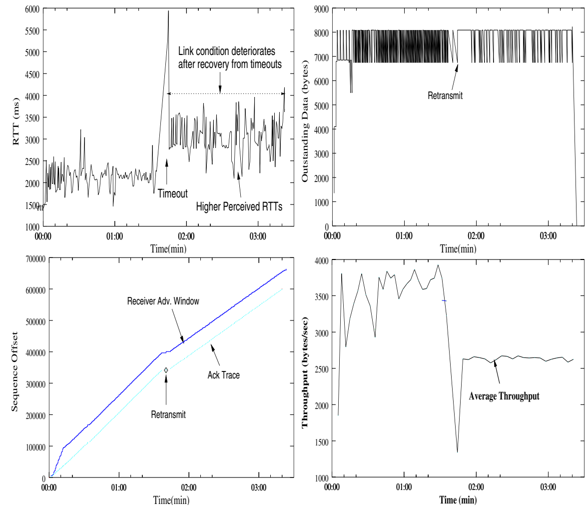

Fig. 12. Swift recovery from TCP timeout during a 600KB file transfer while using the proxy. Plots showing (top-left and clockwise) (a) RTT plot of TCP timeout (b) Outstanding (inflight) data (c) receiver perceived throughput and (d) sender trace.

Freeze-TCP [25], FDA [27], W-TCP [16], WTCP [24] and TCP Westwood [21] 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], [26]), split connection based approaches (e.g. I-TCP [20], W-TCP [16]) and early warning based approaches (e.g. FreezeTCP [25]) and finally those necessitating end system changes (e.g. WTCP [24], Freeze-TCP [25], TCP Westwood [21]).

Snoop [14] is a TCP aware link-layer scheme that 'sniffs' packets in the base station and buffers them. If duplicate acknowledgements are detected, incoming packets from the mobile host are retransmitted if they are present in a local cache. 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 behaviour with TCP's retransmission mechanisms.

The Snoop protocol scheme was originally designed for wireless LANs rather than 'long-thin' wide-area wireless links. As such, it does not address the problems of excess queueing at base stations or proxies. Jian-Hao *et. al.* developed FDA [27], 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 TCP's ability to detect error free conditions.

Delayed DupAcks [26] 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, interval d is difficult to determine as it depends on frequency of losses over the wire-

Proposals	SNOOP[14]	M-TCP[23] & I-TCP[20]	Freeze-TCP[25]	FDA[27]	WTCP[24]	W-TCP[16]	TCP Westwood [21]	**Our approach**
TCP change in end-systems required	×	×	√	×	√	×	√	×
Avoids Excess Queuing at Proxy/Base Station	×	×	×	√	√	×	√	√
Faster Start-up for short flows	×	×	×	×	√	×	×	√
Maintains TCP Fairness for all types TCP flows	×	×	×	×	×	×	×	√
Handle variable bit error environment	√	√	√	×	√	√	√	√
Handle small link "stalls"	×	√	×	×	√	√	√	√
Quick Recovery from Long Disconnections	×	√	√	×	×	√	×	√
Maintains End-to-End Protocol Semantics	√	×	√	√	√	√	√	×

TABLE I

FEATURE COMPARISION OF WIRELESS TCP SOLUTIONS [YES (√), NO (×)]

less medium. The base station is not TCP aware, thus a receiver can not 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.

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 [20] 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 [23] 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 under-utilization for want of packets to send. Holding back ACKs also affects sender's RTT estimates, affecting TCP's ability to recover from non-wireless related packet losses.

Ratnam and Matta propose W-TCP [16], which also splits the connection at the base station. However, it acknowledges a packet to the sender only after receiving an acknowledgement 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 underutilised.

WTCP [24] is an end-to-end scheme which primarily uses inter-packet separation as the metric for rate control at the receiver. Congestion related loss detection is also provided as a backup mechanism. The biggest drawback of 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 [21] 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 (ZWP) 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.

VIII. CONCLUSIONS

In this paper we have proposed a flow aggregation scheme to transparently enhance performance of TCP over wide area wireless links such as GPRS. The proposed scheme was implemented in a split TCP proxy and evaluated on a GPRS network testbed. The key features of the scheme are summarised below:

- During connection startup, slow start is avoided, thus quickly bringing the link up to full utilization, to the particular benefit of WWW sessions.
- Error recovery is improved due to the extension of the SACK mechanism to cover the entire aggregate. Losses are detected faster and recovery is done without unnecessary backoff.
- The flow control algorithm manages proxy buffer space to ensure sufficient data is buffered to keep the wireless link fully utilised, but trys to adjust flows smoothly, and limits the buffer space committed to each mobile host.
- Scheduling of packets in the aggregate allows fair bandwidth allocation to flows, regardless of duration.

IX. CURRENT LIMITATIONS AND FUTURE WORK

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 during cell re-selections (and/or routing-area updates), we feel that a more dynamic scheme may be required for next generation 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.

We have used the proxy with only a limited set of concurrent clients. It would be interesting to perform tests (part of our ongoing work) to identify how well the transparent proxy scales to a wider client base. We are recording tcpdump traces of all GPRS traffic generated by our user community, which will assist in evaluating system scalability and resulting user experience from using the proxy.

Work is also ongoing to perform more thorough evaluation of the flow control and error recovery schemes, using a combination of simulation and empirical analysis of the long-term packet traces we are collecting. Fine tuning of the scheme is bound to result.

Our current work has concentrated on the downlink direction as we perceive this as being the most critical direction for performance of most applications. We are 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 issues posed by the downlink.

Acknowledgments

We wish to thank Vodafone Group R&D, Sun Microsystems Inc. and BenchMark Capital for supporting this work. Thanks also go to Tim Harris for comments on an earlier version of this paper and the INFOCOM'03 reviewers for their constructive input.

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] B. Walke, Mobile Radio Networks, Networking and Protocols (2. Ed.), John Wiley & Sons, Chichester 2001

[3] C. Bettssetter, H. Vogel and 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.

[4] Michael Meyer, "TCP Performance over GPRS", In Proceedings of IEEE WCNC, pages 1248-1252, 1999

[5] R. Ludwig, B. Rathonyi, A. Konrad, K. Oden and A. Joseph, "Multi-Layer Tracing of TCP over a Reliable Wireless Link", In Proceedings of ACM SIGMETRICS 1999.

[6] O. Spatscheck, J. Hansen, J. Hartman and L. Peterson, "Optimizing TCP Forwarder Performance", IEEE/ACM Transactions on Networking, Vol. 8, No. 2., April 2000

[7] R. Chakravorty, J. Cartwright and I. Pratt, "Practical Experience With TCP over GPRS", In Proceedings of IEEE GLOBECOM 2002, November 17-21, Taipei, Taiwan
Source: http://www.cl.cam.ac.uk/users/rc277/gprs.html

[8] R. Chakravorty and I. Pratt, "WWW Performance over GPRS", in Proceedings of the IEEE International conference in Mobile and Wireless Communications Networks (IEEE MWCN 2002), September 9-11, Stockholm, Sweden
Source: http://www.cl.cam.ac.uk/users/rc277/gprs.html

[9] R. Chakravorty and I. Pratt, "Performance Issues with General Packet Radio Service", to appear in IEEE/KICS Journal of Communication and Networks (JCN), Special Issues on "Evolving from 3G deployment to 4G definition", 2003.
Source: http://www.cl.cam.ac.uk/users/rc277/gprs.html

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

[11] C. Waldspurger and W. Weihl, "Stride Scheduling: Deterministic Proportional-Share Resource Management", Technical Report 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", IEEE/ACM Trans. on Networking, Vol. 5, No.6, Dec. 1997.

[13] H. Balakrishnan, H. Rahul, and S. Seshan, "An Integrated Congestion Management Architecture for Internet Hosts", In Proceedings of ACM SIGCOMM 1999.

[14] H. Balakrishnan, R. Katz and S. Seshan, "Improving TCP/IP performance over Wireless Networks", In Proceedings of ACM MOBICOM, November 1995

[15] L. Zhang, S. Shenker and D. Clark, "Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic", In Proceedings of ACM SIGCOMM 1991.

[16] K. Ratnam and I. Matta, "W-TCP: An Efficient Transmission Control Protocol for Networks with Wireless Links", In Proceedings of Third IEEE Symposium on Computer and Communications (IEEE ISCC), 1998.

[17] L. Kalampoukas, A. Varma and K. K. Ramakrishnan, "Explicit Window Adaptation: A Method to Enhance TCP Performance", In Proceedings of INFOCOM 1998

[18] L. L.H. Andrew, S. Hanly and R. Mukthar, "Analysis of a Flow Control Scheme for Rate Adjustment by Managing flows", In Proceedings of the 4th Asian Control Conference, September 25-27, 2002, Singapore
Source: http://www.ee.mu.oz.au/staff/lha/LAicp.html

[19] Z. Wang and P. Cao, "Persistent Connection Behaviour of Popular Browsers", Source: http://www.cs.wisc.edu/cao/papers/persistent-connection.html

[20] A. Bakre and B. R. Badrinath, "I-TCP: Indirect TCP for mobile hosts", In Proc. of the 15th IEEE International Conference on Distributed Computing Systems, pages 136-143, Vancouver, BC, May 1995.

[21] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links", In Proceedings of ACM Mobicom 2001

[22] M. Kim and B. D. Noble, "Mobile network estimation", In Proceedings of the ACM MOBICOM 2001

[23] K. Brown and S. Singh, "M-TCP: TCP for Mobile Cellular Networks ", ACM Computer Communication Review, 1997

[24] P. Sinha, N. Venkitaraman, R. Sivakumar and V. Bhargavan, "WTCP: A Reliable Transport Protocol for Wireless Wide-Area Networks", In Proceedings of ACM MOBICOM 1999.

[25] T. Go , J. Moronski, D. S. Phatak and V. Gupta, "Freeze-TCP: A true end-to-end enhancement mechanism for mobile environments," In Proceedings of IEEE INFOCOM 2000, Israel.

[26] V. Bakshi, P. Krishna, N. H. Vaidya and D. K. Pradhan, "Improving Performance of TCP over Wireless Networks", Texas A&M University Tech. Report TR-96-014, May 1996

[27] Jian-Hao Hu and K. L. Yeung, "FDA: A Novel Base Station Flow Control Scheme for TCP over Heterogeneous Networks", In Proceedings of IEEE INFOCOM 2001

[28] A. Gurtov, M. Passoja, O. Aalto and M. Raitola, "Multi-Layer Protocol Tracing in a GPRS Network", In Proceedings of the IEEE Vehicular Technology Conference (Fall VTC2002), Vancouver, Canada, Sepember 2002.

[29] P. Stuckmann, N. Ehlers and B. Wouters, "GPRS Traffic Performance Measurements", In Proceedings of the IEEE Vehicular Technology Conference (Fall VTC 2002), Vancouver, Canada, Sepember 2002.

[30] J. Cartwright, "GPRS Link Characterization",
Source: http://www.cl.cam.ac.uk/users/rc277/linkchar.html

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

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

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