Performance Study of the GSM Circuit-switched Data Channel

by Almudena Pazo Konrad

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Anthony D. Joseph
Research Advisor

16 December 1999

* * * * * *

Professor Steven McCanne
Second Reader

16 Dercember 1999

1 Abstract

Wireless Communication is the fastest growing area in communications today. Many studies are being performed to understand and improve the performance over wireless connections, a high bit error rate and lossy environment. One of the Iceberg [7] project's research area focuses on the performance of multilayered protocols in a wireless environment. In particular, we are studying the interactions between the Transmission Control Protocol (TCP), a reliable end-to-end transport layer protocol, and the Radio Link Protocol (RLP) [4], [5], a reliable link layer protocol for the wireless connection in the GSM (Global System for Mobile communications) network [2]. Each protocol has its own error recovery mechanisms and we believe that by studying the interactions of these protocols we will improve the performance of the wireless GSM system. We have developed a multi-layer tracing tool to analyze the protocol interactions between the layers. Initially, we hypothesized that delay introduced by RLP would unnecessarily trigger TCP's congestion control algorithm, thus degrading performance. However, our studies show this to be false [2]. Using our multi-layer analysis tool, we have identified some of the causes that degrade performance: (1) inefficient interaction with TCP/IP header compression, and (2) excessive queuing caused by overbuffered links. Furthermore, in a medium where the wireless error rate is low, it is natural to think that a big frame size would increase performance. However, if the wireless medium has high error rate, which would cause numerous retransmissions, performance decreases as the frame size increases. In this masters' report, we present an analysis of how the frame size in the RLP layer affects performance and determine the optimal frame size.

2 Introduction and Motivation

The significant demand for high performance wireless access has lead many people to do performance studies and develop new wireless protocols. In this thesis, we study some of the issues that affect the performance of the circuit-switched data service implemented in GSM. We start by first studying the interaction between RLP, a reliable radio link protocol deployed in the GSM digital cellular telephone network [1], and TCP, the reliable transport protocol currently being used in the Internet. Secondly, we provide a detailed analysis on how to improve the overall performance by choosing an optimal RLP frame size.

Our current research is focused on non-delay sensitive applications. We perform bulk data transfer and collect traces. To analyze the collected traces, we developed a diagnostic tool, *MultiTracer*, to study the protocol interaction between RLP and TCP. *MultiTracer* allows us to visualize the information at both layers correlated in time. We have collected detailed traces representing a variety of wireless scenarios

(e.g., stationary indoors, walking, and driving). By analyzing these traces, we have drawn conclusions about the interaction between the protocols.

We were interested in determing whether there are many inefficient interactions between TCP and RLP during bulk data transfer operations. Our analysis shows that competing error recovery is not a problem, however we have discovered other issues that degrade the overall performance of the data transfer.

Furthermore, in this work we show that the throughput on GSM wireless channel can be improved by using a larger RLP frame size. We show that we can increase performance by 25 percent when the channel quality is good, and by 18 percent when the channel quality is bad. Thus, we determine the optimal fixed frame size in terms of throughtput.

3 Background

In this section, we first provide a background on the Radio Interface in GSM. We describe the Radio Link Protocol (RLP) used by GSM in the radio interface to provide reliability. During a data call, RLP runs between the Terminal Adaptation Function (TAF) of the mobile host and the Interworking Function (IWF) of the Mobile Switching Center [1]. We conclude the section by providing an overview on the functionalities of Transmission Control Protocol (TCP).

3.1 The GSM Radio Interface

GSM is a TDMA-based (Time Division Multiple Access) circuit-switched network, implemented on multiple frequency subbands (TDMA/FDMA). Each user has an assigned carrier frequency number and a time slot. There are two frequency bands allocated for GSM, of 25 MHZ each one.

- Band 890-915 Mhz: allocated for uplink direction (from mobile host to base station).
- Band 935-960 Mhz: allocated for downlink direction (from base station to mobile host).

An additional protocol called the L2R (Layer 2 Relay) protocol is used for flow control, framing and communicating status control signals between the TAF and the IWF.

GSM implements several error control mechanisms, including adaptive power control, frequency hopping, Forward Error Correction (FEC), and Interleaving. FEC (channel coding) in GSM is performed by using a convolutional code, which adds redundancy bits to the data to detect and correct errors. Besides FEC, GSM performs interleaving to avoid error burstiness. Instead of transmitting a RLP frame, the frame is divided into separate small blocks, these blocks are then interleaved into 22 time slots for transmission. A time slot is 5 milliseconds, and can send 114 bits per time slot, producing a data rate of 22.8 kbits/sec. The encoded frame carries 456 bits (sent in 4 time slots), and the encoded frame has 240 bits, resulting in a data rate of 240 bits every 20 ms (12 kbits/sec). Out of the 240 bits, 48 bits are RLP overhead and 192 bits are RLP user data, yielding a user data rate of 9.6 kbits/sec.

3.1.1 The Radio Link Protocol (RLP)

The Radio Link Protocol (RLP) [4] [5], is an HDLC-derived logical link layer protocol. RLP attempts to guarantee correct data delivery by using ARQ (Automatic Repeat Request) mechanisms. In our studies, we show that data loss can occur when the link is reset. The link is reset if an RLP frame has to be retransmitted more than N2 times [1], or in the case of a protocol violation. When a link reset occurs, the sender and receiver empty their buffers and resets their sequence numbers.

The frame size is 240 bits with an overhead of 48 bits. Out of these 48 bits of overhead, 24 bits are used for FEC, 8 bits are used for flow control messages at the L2R layer, and 16 bits are dedicated for control bits. For flow control, RLP limits its window size to 62 frames, which means that 62 is the maximun number of unacknowledged frames that the sender can have outstanding at any given time.

RLP uses three types of frames to carry information. The type of frame is specified in the 16-bit control field. Included below is a description of each type:

- *U-frames* - Unnumbered frames are used to carry initialization, termination and control information.
- *S-frames* - Supervisory frames are used to carry ARQ (Automatic Repeat Request) information.
- *I-S frames* - Information-Supervisory frames are used to carry the data, plus acknowledgment information (i.e., the acknowledgment is piggy-backed onto the frame).

[1] The maximum number of retransmissions is determined by the protocol parameter N2, which is negotiated at call setup. The default value of 6 can be configured through an AT command.

There are two error recovery mechanisms in RLP, (1) Selective Reject (SREJ), and (2) checkpointing. In figure 1 we illustrate the SREJ mechanism. SREJ is initiated by the receiver. The receiver explicitly requests retransmission of missing *I-frames*. Whenever an *I-frame* is received out of order, the receiver will send an *S-frame* with SREJ in the control field, and N(R) equal to the sequence number of the missing frame. The sender is not required to "roll back", but instead retransmits the *I-frame* of the sequence number equal to the N(R) sent by the receiver. The sender then returns to its state before retransmission and continues to transmit frames. It should be noted that every SREJ frame is protected by a single retransmission timer and the sender is limited to retransmitting the same frame no more than N2 times.

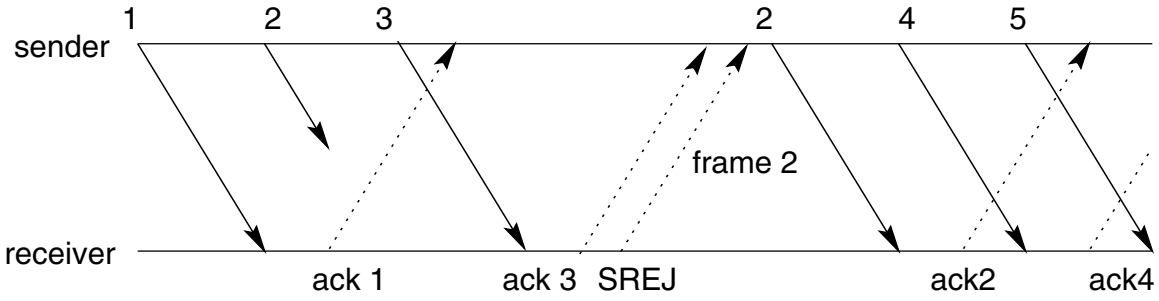

Figure 1: SREJ illustration.

Figure 2, exemplifies the checkpointing mechanism. Checkpointing is a phase initiated by the sender whenever there is a timeout of an aknowledgment. The sender sends an *S-frame*, sets the poll bit of the control field to one and waits for the receiver's response. The receiver responds with an *S-frame* indicating the receive sequence number (N(R)) of the frame that it is expecting. Finally, the sender retransmits all the frames in sequence, starting from N(R) (GoBackN).

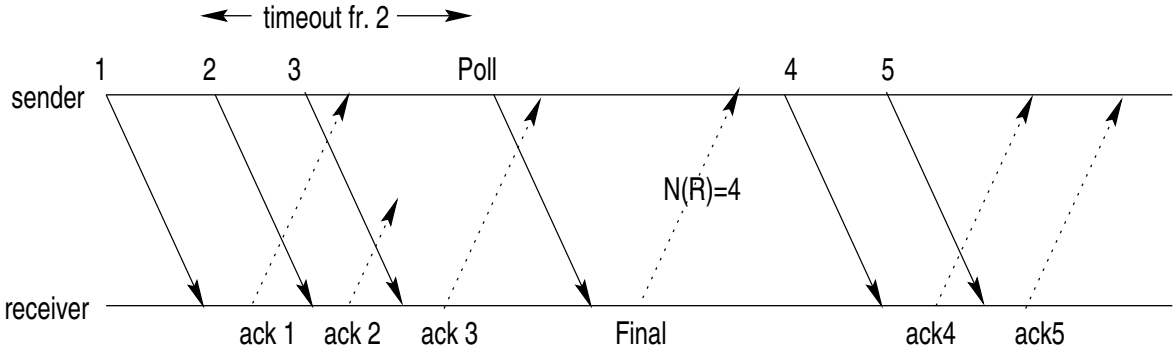

Figure 2: Checkpointing illustration.

3.2 The Transmission Control Protocol (TCP)

The Transmission Control Protocol (TCP) [3], provides a connection-oriented, reliable, byte stream service. TCP assumes that packet loss is a signal for congestion in the network, which is a good assumption since, for most networks, packet loss due to damage is less than 1 percent.

For flow control, TCP maintains the following variables:

- cwnd and ssthresh - congestion window and slow start threshold are the flow control limits imposed by the sender.
- adwnd - advertised window is the flow control limit imposed by the receiver, it depends on the amount of available buffer space at the receiver.
- wnd - current window size (minimun of the congestion window and the advertised window).

Figure 3 illustrates the congestion control mechanism in TCP. There are two indications at the TCP layer of congestion in the network, one is a "timeout" and the other is a "dupack" (duplicate acknowledgment). A "timeout" occurs whenever the timer for an acknowledgment expires, and a "dupack" occurs whenever a TCP segment has been lost at the receiver [2]. There are different states of the sender during congestion control. Upon establishing a connection, the congestion window (cwnd) is set to one. This is a phase known as slow start. During this phase, the congestion window is increased exponentially according to the number of acknowledgments received. Upon reaching the slow start threshold, congestion avoidance is performed and the congestion window increases linearly. Essentially, the congestion window is increased by one per round trip time (RTT). At any time, a duplicate acknowledgment may be sent to indicate out of sequence segments and the sender then enters "fast recovery". At this point, the congestion window of the sender is set to 1/2 of its current size and increases linearly as described above. Whenever a "timeout" occurs, the congestion window is reset to one segment size and the connection re-enters the slow start phase.

4 Methodology

Our study is based upon measurement-based trace analysis. This section describes the testbed and tools we used to collect traces. Then, we explain the different envi-

[2]The receiver will assume that a segment has been lost, whenever it receives a segment out of sequence.

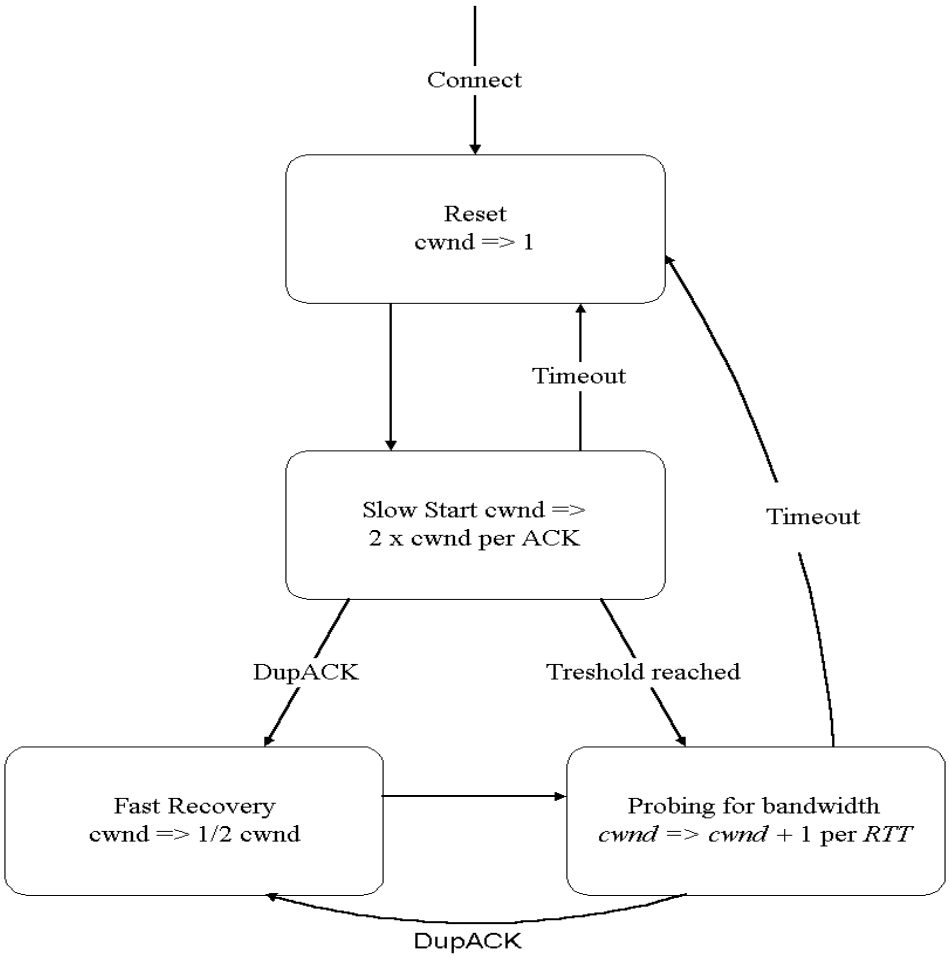

Figure 3: TCP congestion control.

ronments in which we performed measurements and how we collected *block erasure traces*. We conclud with an explanation of the target metrics that we used to perform our analysis.

4.1 Measurement Platform and Software Tools

In Figure 4, we show the testbed used to collect trace information. Our setup consists of a mobile host (MH) which communicates with a fixed host (FH) through a circuit-switched GSM connection (both hosts are running UNIX (BSDI 3.0)). The MH communicates with the GSM Network using a GSM mobile phone and a commercially available PC-Card (Ericsson DC23) running the Terminal Adaptation Function (TAF). RLP runs between the TAF and the Interworking Function (IWF) on the GSM Network [1]. The fixed host terminates the circuit-switched GSM connection.

In the future we will use a testbed which is being developed in the ICEBERG project [7]. The ICEBERG testbed will have a stand-alone GSM base station together with a gateway that "translates" between circuit-switched and IP-based packet-switched voice and data traffic. For this purpose, we are currently implementing the network side of RLP, which terminates RLP in the gateway. For this work, we have used the commercial GSM network for which the network-side of RLP was not accessible.

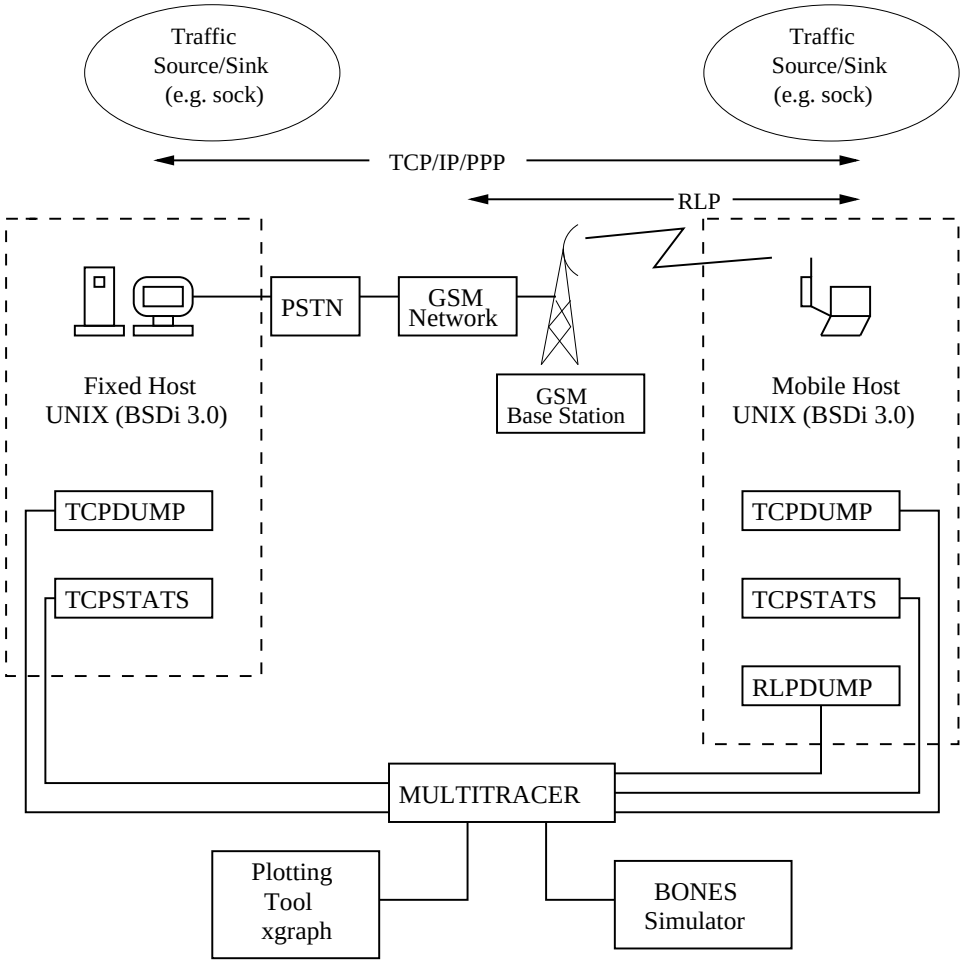

Figure 4: Measurement platform and tools.

In order to study performance in terms of throughput, we performed data bulk transmissions using the *sock* [3] program at the application layer. Going end-to-end, we had TCP running on the transport layer, IP on the network layer, and PPP (point-to-point protocol) [11] at the data link layer. We used several diagnostic tools to collect information at the TCP and RLP layers. To trace information at the TCP sender and receiver, we used *tcpdump* [8] and *tcpstats* [9]. *tcpdump* monitors a host's interface I/O buffers and generates a single log file containing a timestamp specifying when a packet was placed in the sender's buffer and the packet header. From the data generated by *tcpdump*, we can graph time/sequence plots. We used *tcpstats* to

generate information at the sender TCP layer, such as: congestion window, slow start threshold, and the retransmission timeout value.

To collect and correlate TCP and RLP measurements, we ported the RLP protocol of a data PC-Card DC23 to BSDi3.0 UNIX. We, also instrumented the RLP code to log connection related information in the fashion of *tcpdump/tcpstats*. Thus, *rlpdump* logs time/sequence information and also events, like SREJs, retransmissions, flow control signals (XON/XOFF) and RLP link resets.

4.2 Radio Environment

Performance results are highly correlated to the error characteristic of the radio channel. For our studies, we differentiate between "good coverage area" and "bad coverage area", based on the signal strength on the cell phone connected to our mobile host 4. We simply read the mobile phone's visual signal level indicator, which has a range from one to five. The following are the definitions:

- "good coverage area": signal strength on the cell phone was between three and five.
- "bad coverage area"': signal strength on the cell phone was between one and three.

We define three different radio environments:

- Environment A: measurements taken in a "good coverage area" while the mobile host was fixed.
- Environment B: measurements taken while the mobile host was moving (driving a car or walking on the street). In this environment, the signal strength varies from one to four.
- Environment C: measurements taken in a "bad coverage area" while the mobile host was fixed.

4.3 Block Erasure Traces Collection

Using the testbed described in section 4.1, we performed bulk data transfers ranging in size from 230 KBytes to 1.5 MBytes and collected traces at both layers, the TCP

layer and the RLP layer. The traces collected at the RLP layer provide information down to the level of whether an FEC (Forward Error Correction) encoded radio block was decoded successfully or had to be retransmitted. Our block erasure traces consist of binary time series where each element represents the state of an RLP frame. A corrupted block (unsuccessfully decoded) has the value "1", while a non-corrupted block (successfully decoded) has the value "0".

Our analysis is based on 500 minutes of "air-time" traces: 258 minutes of stationary good, 215 minutes of stationary bad, and 44 minutes of mobile. We used three different block erasure traces for our analysis. One which we call *trace-A* is a concatenation of all block erasure traces collected in environment A. Likewise, *trace-C* and *trace-B* are the concatenation of the block erasure traces collected in environments C and B, respectively.

4.4 Target Metrics

In our performance analysis we have used several metrics. We first calculate *throughput* at the TCP layer as an indication of performance on data bulk transfer. However, we were interested in understanding the interaction between TCP and RLP, and by calculating just throughput we could not determine if a low throughput was a consequence of poor TCP/RLP interaction or just congestion on the network. An example of poor TCP/RLP interaction could happen in poor radio channel conditions when RLP retransmissions causes spurious timeouts at TCP (in section 6.1 we show that in our testbed implementation this case is rare).

To analyze TCP/RLP interactions, we used *utilization* as the performance metric. If the TCP sender fully utilizes the bandwidth provided by RLP (which varies over time due to RLP retransmissions) with useful data then this indicates optimal performance (100 percent utilization) and rules out inefficient interaction's between the two protocols. *Utilization* may be *not* optimal if (1) the TCP sender leaves RLP idle, or (2) TCP is doing retransmissions. We provide an analysis tool *MultiTracer* in section 5 which for each measurement checks whether *utilization* was optimal or not. To check to see if *utilization* is optimal, *MultiTracer* uses *rlpdump* to determine idle phases at the RLP sender, and *tcpdump* is used to determine TCP retransmissions. We used *MultiTracer* to isolate the traces where utilization was 95 percent or less and analyzed these traces to identify the causes of the degraded performance. Note that *utilization* can never be 100 percent because of the slow-start phase and the 3-way handshaking used by TCP to connect.

5 Multi-Tracer: A Multi-Layer Tracing Tool

Altogether tcpdump, tcpstats and rlpdump (4) generate a total of up to 300 bytes/s of trace data for a connection that is running at about 10 kb/s. It was therefore essential to develop a post-processing tool that enables the rapid correlation and representation of collected trace data in a comprehensive graphical manner for trace analysis. We call this tool *MultiTracer* [2]. *MultiTracer* is a set of script files written in perl that converts the trace data into the input format required by a plotting tool [10]. At a later stage, we plan to use *MultiTracer* to also generate input for trace replay in a simulation environment. Using *MultiTracer* in this manner, we will be able to reproduce various effects that were measured in reality.

5.1 How to Read Time/Sequence Plots

Using *MultiTracer* and the plotting tool *xgraph* [10], we are able to generate complex plots with all the information at both layers (TCP and RLP) correlated in time. Figure 5 shows an example of how to read a time-sequence plot.

In this figure there are two graphs, one corresponds to the sender's sequence numbers and the other corresponds to the acknowledgement received by the sender. The sequence number corresponds to the first byte sent on that specific TCP segment, while the acknowledgement number represents the next byte the receiver is expecting to receive. The Maximun Segment Size (MSS) can be read from the graph by reading the difference between two consecutives sequence numbers. We can also, read the Round Trip Time (RTT), and identify the congestion control phase at each given moment (see section 3.2). For example, in figure 5 the sender is in the slow-start phase, where the congestion window doubles for every acknowledgement recieved.

5.2 Information Generated by MultiTracer

Extracting information from *tcpdump*, *tcpstats*, and *rlpdump*, *MultiTracer* generates the following set of files:

At the sender TCP layer:

- TcpSnd-data: time and sequence number at which segments are sent.
- TcpSnd-ack: time and sequence number at which acknowledgement are received.

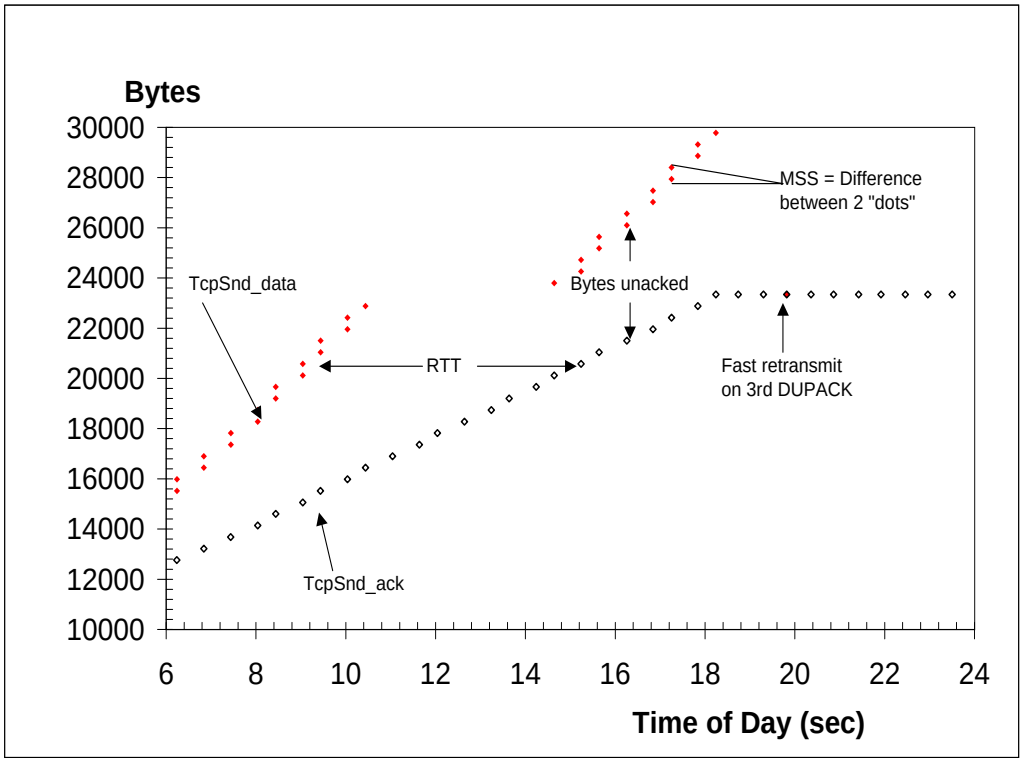

Figure 5: A TCP sender-side time/sequence plot.

- TcpSnd-cwnd: size of the congestion window at each time it changes value.
- TcpSnd-srtt: value of the smooth round trip time (SRTT)[3].
- TcpSnd-vrtt: value of the round trip time varience (VRTT).
- TcpSnd-sstrsh: value of the slow-start threshold.

At the receiver TCP layer:

- TcpRcv-data: time and sequence number at which segments are received.
- TcpRcv-ack: time and sequence number at which acknowledge are sent.

[3]In order to read time in a bytes-versus-time plot, we represent 10,000 bytes as 1 second.

At the sender RLP layer:

- RlpSnd-data: time and sequence number at which frames are sent.
- RlpSnd-ack: time and sequence number at which acknowledgements are received.
- RlpSnd-data-rexmt: time and sequence number at which frames are retransmitted.
- RlpSnd-poll-snd: time at which sender sends a supervisory frame with poll bit set to "1" (this correspons to checkpointing mechanism on section 3.1.1).
- RlpSnd-rpoll-rcv: time at which sender receives a supervisory frame with xxx bit set to "1" responding to the checkpointing mechanism.
- RlpSnd-rst: time at which rlp resets the link.
- RlpSnd-srej-rcv: time at sender receives a select reject (SREJ) from the receiver.
- RlpSnd-xoff: time at which control message xoff arrives to the sender asking the sender to stop sending.
- RlpSnd-xon: time at which control message xon arrives to the sender asking the sender to continue sending.

By using a plotting tool (e.g. xgraph), we can graphically plot the files we want.

6 Measurement Results

In this section, we present the results obtained by using *MultiTracer* [2] on the collected data. We first were interested in finding spurious timeouts and identifying the causes. Spurious timeouts are due to an RLP inefficiency and indicate a poor TCP/RLP interaction. However, we show in section 6.1, that this is not the case. By doing a detailed analysis of the plots generated by *MultiTracer*, we identified other interesting events. In section 6.2, we explain the problem of local buffer overflow. Section 6.3, provides a detailed analysis of the impact of link resets on TCP, which is a cause of poor protocol interaction. The last section, refsec:other effects, explains the xon/xoff effect found in RLP.

6.1 TCP/RLP Interactions are Rare

We have found that TCP and RLP rarely interact in an inefficient way. As depicted in Figure 6, in almost 85 percent of all our measurements, the utilization (see Section 4.4) of the GSM data channel was 98 percent or more. Even in those measurements where we detected protocol interactions, the utilization never dropped below 91 percent. We did not expect to observe such high figures, given that 259 minutes of our measurements were taken in an environment with poor receiver signal strength (see Section 4.3). In such an environment we expected to find cases of competing error recovery between TCP and RLP. In fact, we did not find any incidents of competing error recovery during bulk data transfers, as discussed in Section 6.4. All measurements that yielded a utilization of 95 percent or less suffered from the impact of RLP link resets when TCP/IP header compression [12] was used. This is further explained in Section 6.3.

Figure 6 also shows the throughput range that sock (see Section 4.1) achieved for measurements that yielded the same utilization. Taking protocol overhead into account, the throughput was consistently close to the bit rate of the channel. These results confirm similar findings from [13]. However, unlike in those studies, our tools provided us with the unique opportunity to measure utilization in addition to throughput. Thus, we could determine that a measurement (using TCP/IP header compression) that resulted in a throughput of only 7.0 kb/s, but yielded an utilization of 99 percent, must have suffered from a non-optimal radio connection. Consequently, the RLP sender must have retransmitted a higher number of frames. The overall throughput results, however, suggest that the GSM data channel is over-protected with FEC.

6.2 Excessive Queuing and local drops

One problem that is evident in the trace plots is the large mismatch between the pipe capacity and the load that the TCP sender puts onto the network. The pipe capacity in this network is reached with 2 segments, assuming a MTU of 512 bytes. However, as can be seen in Figure 7, the TCP sender increases the load up to 8 KBytes or 16 segments. TCP sender has no way to determine the pipe capacity and, thus, will periodically increase the congestion window (the load) until the TCP receiver's advertised window is reached. The latter usually corresponds to the default socket buffer size (a default setting of the operating system; commonly 8 or 16 KBytes). Consequently, the maximum pipe queue is 14 segments (with 8 KBytes socket buffers). In the measurement platform shown in Figure 4, packets (PPP frames) queue up at the mobile host's interface buffer. For downlink transmission, those packets would queue up at the other side of the bottleneck link. Thus, the core of the performance problem is a largely overbuffered link. The default interface buffer size in BSD-

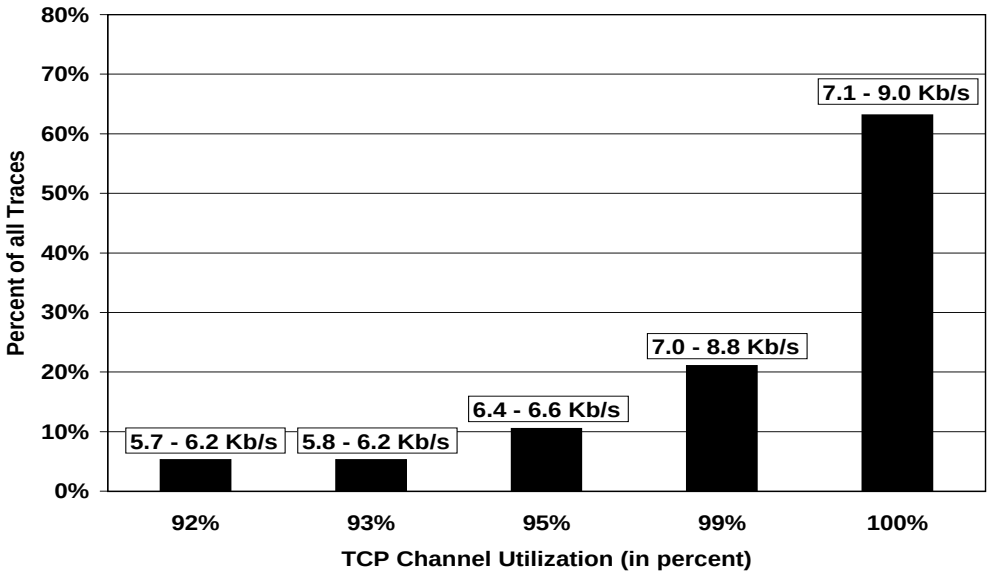

Figure 6: TCP channel utilization.

derived UNIX [3] is 50 packets. Obviously, this is an inappropriate size for a mobile device, which usually does not have a large number of simultaneous connections.

We have purposefully compiled a kernel with an interface buffer that was smaller than 8 KBytes, the default socket buffer size used by BSDi3.0, to provoke a local packet drop as shown in Figure 7. This triggers the "tcp-quench" (source quench [3]) function call to the TCP sender which in response resets the congestion window back to one. After about one half of the current RTT, the sender can again send additional segments until the "dupack" for the dropped packet trigger the fast retransmit algorithm (see Section 3.2). This leads to setting the congestion window to one half of its value before the local drop occurred. At this point, the sender has reached the advertised window and cannot send any additional segments (which it could have otherwise) while further "dupack" return. Thus, when the retransmission is acknowledged, a burst of half the size of the sender-side socket buffer (8 segments) is sent out by the TCP sender at once.

As can be seen from the TCP receiver trace in figure 7, excessive queuing, the ups and downs of the congestion window at the TCP sender, and even retransmissions do not degrade throughput performance. But excessive queueing has a number of other negative effects:

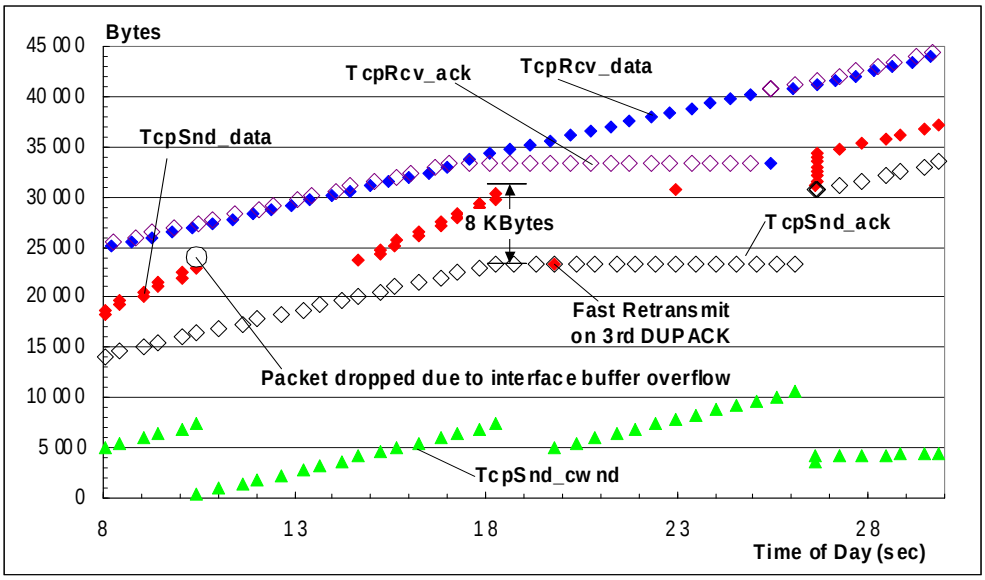

Figure 7: Local buffer overflow.

- It inflates the RTT. In fact, a second TCP connection established over the same link is likely to suffer from a timeout on the initial connect request. This timeout occurs because it takes longer to drain the pipe queue (here up to 14 x MTU or 7 KBytes) on a 960 bytes/s link than the commonly used initial setting for TCP's retransmission timer (6 seconds).

- If the timestamp option is not used, the RTT sampling rate is reduced, leading to an inaccurate retransmission timer value [14].

- An inflated RTT inevitably leads to an inflated retransmission timer value, which can have a significant negative impact on TCP's performance, e.g., in the case of multiple losses of the same packet. The negative impact results from the exponential back-off of the retransmission timer and can be seen in Figure 9.

- For downlink transmissions (e.g., web browsing), where no appropriate limit is imposed on the outbound interface buffer of the bottleneck router, the data in the pipe queue may become obsolete (e.g., when a user aborts the download of a web page in favor of another one). The "stale data" must first drain from the queue, which in case of a low bandwidth link, may take on the order of several seconds.

A simple solution to these problems is to statically adjust the interface buffer size to the order of the interface's bit rate. A more advanced solution is to deploy active queue management [15] at both sides of the bottleneck link. The goal is to adapt the buffer size available for queueing to the bit rate of the interface, a given worst-case RTT, and the number of connections actively sharing the link. Combining active queue management with an explicit congestion notification mechanism [16] would further improve network performance as fewer packets would have to be dropped and retransmitted (in the case of TCP). In fact we regard it as imperative that these mechanisms be implemented at both ends of wide-area wireless links, which we believe will be the bottleneck in a future Internet.

6.3 The Impact of RLP Link Resets

One of the key findings of our measurements and analysis is an understanding of the impact of RLP link resets (see Section 3.1.1) when TCP/IP header compression [12] is used to reduce the per segment overhead. As with other differential encoding schemes, header compression relies on the fact that the encoded "deltas" are not lost or reordered on the link between compressor and decompressor. Lost "deltas" will lead to false headers being generated at the decompressor, yielding TCP segments that have to be discarded at the TCP receiver because of checksum errors. This effect is described in [12], which proposes the use of uncompressed TCP retransmissions as a means for re-synchronizing compressor and decompressor. Thus, once a "delta" is lost, an entire window worth of data is lost and has to be retransmitted. Even worse, since the TCP receiver does not provide feedback for erroneous TCP segments, the sender is forced into a timeout. This effect is further exacerbated by excessive queuing as described in Section 6.2, since queuing leads to unreasonably large windows and a large retransmission timer.

Figure 8 depicts this problem as perceived by the TCP receiver. We have plotted the RLP link reset found at the RLP layer. As can be seen, the link reset leads to a gap of 18 seconds. During the gap, no data is received correctly. The reset apparently caused the loss of 5 segments. Recall from Section 3.1.1 that RLP transports user data (PPP frames) transparently. Thus, if only the first or last few bytes of a PPP frame are lost when the RLP sender and receiver flush their buffers after the link reset, the whole PPP frame is discarded by the PPP receiver because of a checksum error. This causes the header decompressor to be off by 5 segments, so that segment i+5 is decoded as segment i and so forth. Thirteen of the segments shown in the plot are not acknowledgment by the TCP receiver because they are discarded due to checksum errors. These segments should actually have been plotted with an offset of 5 x MSS parallel to the y-axis.

Another variant of the same problem is shown in Figure 9. This time the ac-

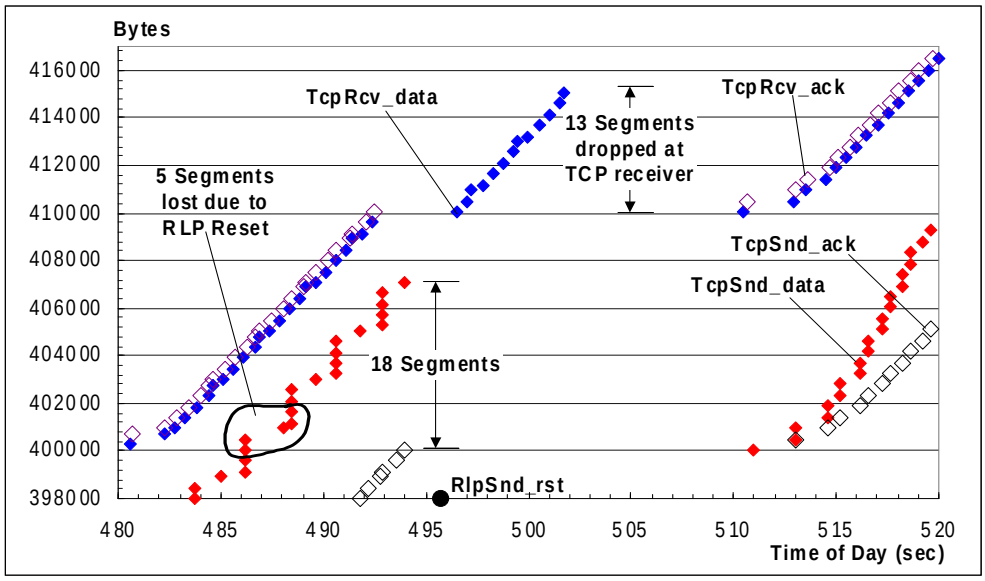

Figure 8: Impact of link reset on TCP layer (example 1).

knowledgment get lost, including the one for the first retransmission; again due to a RLP link reset. This loss leads to an exponential timer back-off of the retransmission timer. Since the retransmission timer value is significantly inflated (see Section 6.2), this has a particularly bad effect.

We want to point out, though, that RLP link resets are very rare events. We have captured 14 resets, all of which occurred when the receiver signal strength was extremely low. In all cases, the link reset was triggered because a particular RLP frame had to be retransmitted more than 6 times (the default value of the RLP parameter N2, "maximum number of retransmissions"). Our results suggest that this default value is too low and needs to be increased. TCP connections before and after the link reset usually progress without problems and there is no apparent reason why the link should be reset. Increasing N2 is also supported by the fact that we did not find any sign of competing error recovery between TCP and RLP during bulk data transfers (see Section 6.4). Initial results indicate that TCP can tolerate a fairly high N2 without causing competing error recovery. This initial result and the negative interactions with header compression suggest that link layer retransmissions should be more persistent when transmitting fully reliable flows, e.g., TCP-based flows. This not only pertains to RLP [4] [5], but also to comparable protocols which are intentionally designed to operate in a semi-reliable mode [17]. Recent studies of TCP over WLAN

(Wireless Local Area Network) links report similar results [18]. On the other hand, persistent link layer retransmissions are not tolerable for delay-sensitive flows.

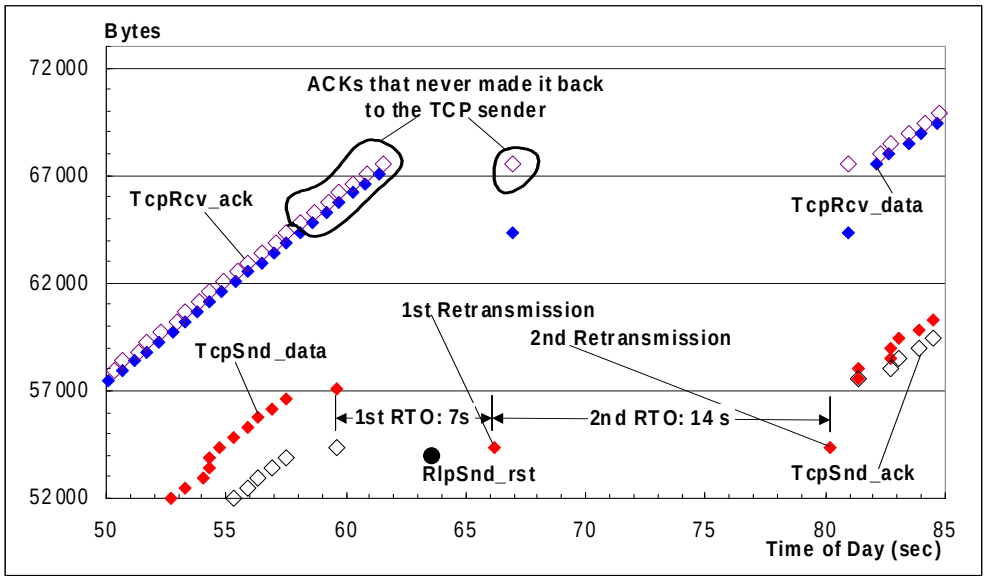

Figure 9: Impact of link reset on TCP layer (example 2).

6.4 Competing Retransmissions are Rare

Various related studies [13] [19], [20], mention the potential problem of competing error recovery between TCP and a reliable link layer protocol resulting from spurious timeouts at the TCP sender. However, we did not find this problem in our measurements during bulk data transfers. A spurious timeout can be easily seen in a TCP sender-side time/ sequence plot: the acknowledgment for a correctly received segment reaches the TCP sender after the retransmission timer covering that segment has expired. We only found 2 such instances in all our traces. However, in both cases the spurious timeout occurred at the beginning of the connection when the TCP sender had not yet converged to an appropriate retransmission timer value. Also, in both cases the receiver signal strength was very low and the RLP sender had already performed several retransmissions at that time. All other timeouts that we found were related to RLP link resets. In contrast, we found several instances that show that the TCP retransmission timer is conservative enough to even allow for extra delays due to link layer error recovery beyond 1200 ms. This is depicted in Figure 11 which

shows a burst of retransmissions on the RLP layer of 1325 ms leading to a "hole" of 2260 ms at the TCP receiver. One reason for the difference in these values is that the end of a segment could have been affected by the retransmissions, which would require a full round-trip time on RLP layer (about 400 ms, see [6]). It cannot be the case that the returning acknowledgments were delayed in addition to the segment, as the plot shows no sign of acknowledgment compression [21].

We were curious to understand why [13] did find spurious timeouts in their study which used almost the same network setup as ours. The authors of that study believed that these spurious timeouts were caused by excessive RLP retransmissions (i.e., because of competing error recovery between TCP and RLP). While it appears as if our results contradict the results of [13], our in-progress work indicates that this is not the case. The reason apparently lies in differences between the implementations of TCP that were used in both studies. Some implementations of TCP seem to maintain a more aggressive retransmission timer than others. Moreover, the TCP implementation we used (BSDi 3.0) uses the timestamp option [14], yielding a more accurate estimation of the RTT and consequently also a more accurate retransmission timer. Timing every segment instead of only one segment per RTT (which is done when the timestamp option is not used) enables a TCP sender to more quickly adapt the retransmission timer to sudden delay increases. Thus, we believe that timing every segment is an attractive enhancement for TCP in a wireless environment. However, we are not convinced that this requires the overhead of 12 bytes for the timestamp option field in the TCP header.

6.5 xon/xoff Effects

The data rate provided by RLP is 1200 bytes/s. We were therefore surprised when we saw gaps in the RlpSnd-data plots in some of our traces. However, after we traced the flow control messages at the L2R layer (see Section 3.1), it became clear what was occurring (see figure 11). Due to limitations in some commercial GSM networks, the data rate appears to be limited to only 960 bytes/s (9.6 kb/s asynchronous).

In these networks, the RLP sender is flow controlled from the remote side so that the average data rate becomes 960 bytes/s. Figure 12 shows that the RLP sender sends at the maximum rate of almost 1200 bytes/s at times when it is not flow controlled, but the linear regression line shows that the real throughput is throttled by 20 percent down to about 960 bytes/s. However, the periodic gaps of 950 - 1300 ms did not trigger spurious timeouts in TCP.

It is worth mentioning that the GSM standard [4] [5] also allows implementations where, instead of a link reset, the data call is completely dropped. We have measured this effect several times in some commercial GSM networks. Simply dropping the call

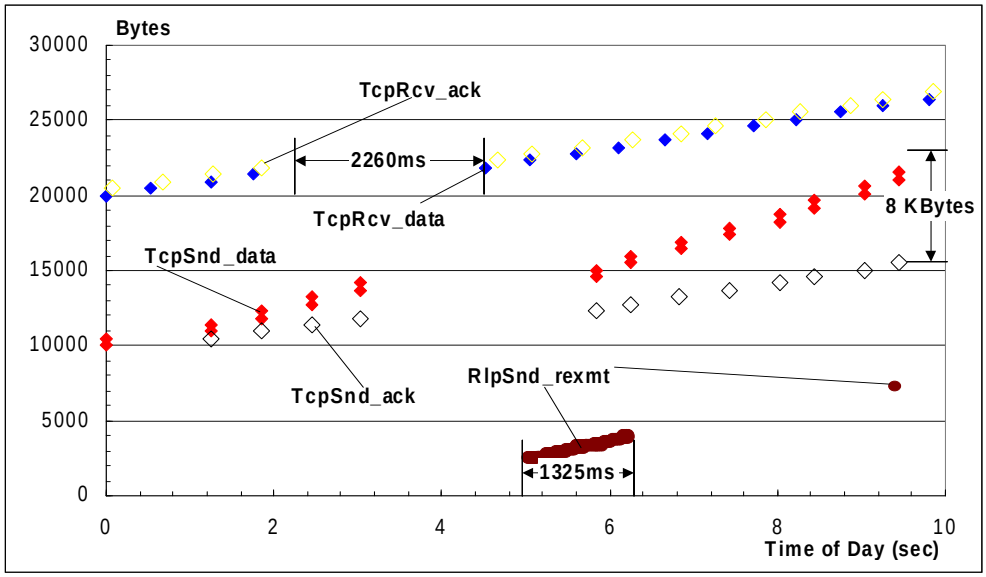

Figure 10: TCP spurious timeout.

is, however, an unacceptable alternative. Not only will the user in many cases have to re-initiate the data transfer (e.g., a file transfer), but will also be charged for air time that yielded an unsuccessful transmission.

7 Optimizing RLP Frame Size in terms of Throughput

We focused have explored several ways in which we could improve RLP to achieve higher performance. In particular, we have analyzed how to choose the optimal RLP frame size. There is a trade off between using larger frames to reduce overhead and using smaller frames to reduce retransmissions in noisy channels. In this section we find an optimal frame size in terms of throughput.

The current RLP implementation [4] uses as the frame size the size of the uncoded data block (30 bytes). However, any multiple of 30 bytes could be chosen as the frame size. For the next generation of GSM, the RLP frame size has been chosen to be 60 bytes [5]. We have created a technique, *retrace analysis*, which calculates the

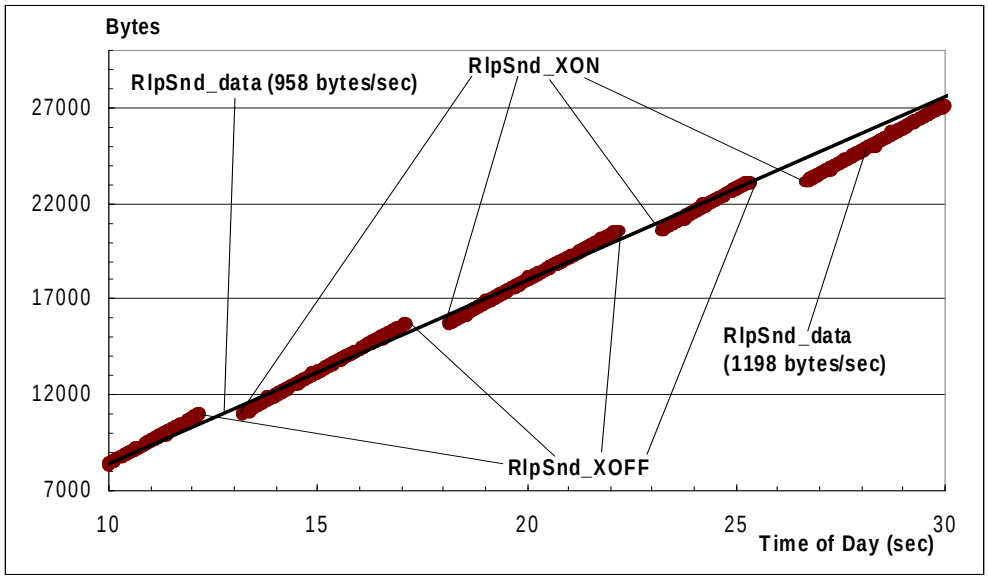

Figure 11: RLP/L2R flow control.

throughput for different frame sizes (from 30 to 1500 bytes) for a given block erasure trace. The optimal frame size corresponds to the frame size which yield the maximum throughput value.

We used *trace-C* and *trace-A* (see section 4.3) to perform *retrace analysis*. We also wanted to understand the impact of error burstiness. For this purpose, we generated artificial traces with evenly distributed errors. *Trace-C-even* is an artificial generated trace with same error rate as *trace-C*, but with the errors being evenly distributed across the trace. These artificial traces show that error burstiness allows for larger frame sizes.

Figure 12 shows the results of our *retrace analysis* on *trace-A*, *trace-C*, and *trace-C-even*. For *trace-A* the optimal frame size is 1410 bytes, yielding 25 percent increased in throughput. For *Trace-C* the optimal frame size is 210 bytes with an 18 percent increased in throughput. From this, we concluded that the frame size chosen for RLP was too conservative. Even in the worst case scenario (*trace-C*), performance can be improved by 18 percent.

It is interesting to compare the plot for *trace-C* to the plot for *trace-c-even*. *Trace-C-even* yields an optimal frame size of 60 bytes. This results show that error burstiness

in the channel allows for larger frame sizes. Assuming errors in the channel are evenly distributed leads to the *wrong* frame size choice. Given that the choice for the frame size in the next generation of GSM is 60 bytes, this decision raises the interesting question of what error model was used to make the decision.

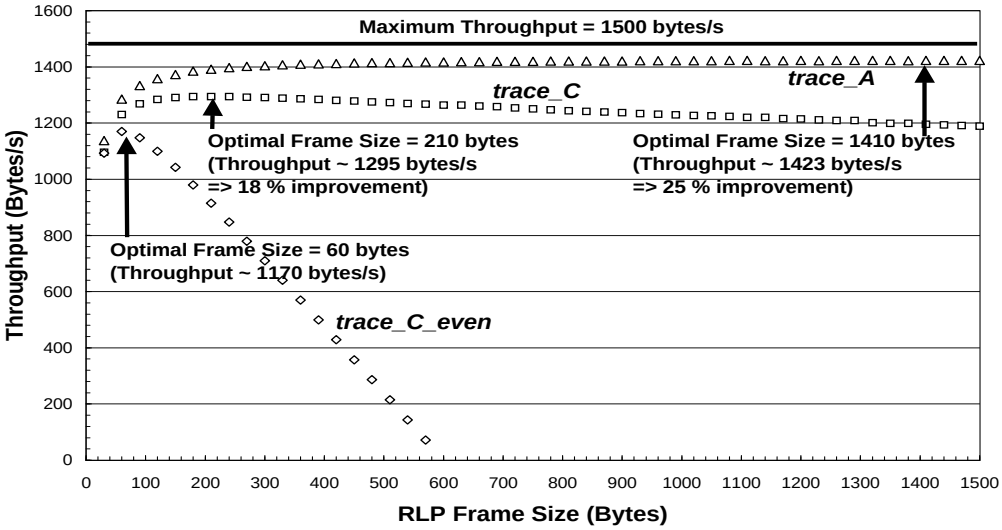

Figure 12: Throughput versus frame size

8 Conclusion

In this project we have presented an analysis tool *MultiTracer*, which allows protocol developers the study of complex interaction between TCP and RLP. We have used *MultiTracer* to analyze 500 minutes of collected traces, and draw conclusions on the interaction efficiency. The main result of the study of the two protocols is that competing error recovery is a rare event. We instead found poor protocol interaction whenever RLP link resets occur and TCP uses VJ header compression. We also, demonstrate the negative impact of overbuffered links.

The second part of this project focused on optimizing the RLP frame size to increase throughput. We show that the throughput can be improved by up to 25 percent by increasing the frame size on RLP. We also argue that the GSM channel has an error burstiness nature which allows larger frame sizes.

9 Acknowledgments

I would like to thank Prof. Anthony Joseph and Reiner Ludwig for their support and valuable contribution to the work presented in this thesis. Special thanks to Keith Sklower for all his help and time. Thanks to Kimberly Oden for her help in developing MultiTracer. I also would like to thank Bela Rathonyi for implemeting and instrumenting the RLP code for BSDI. And many thanks to all the members of the ICEBERG group.

References

[1] M. Mouly, M. B. Pautet, *The GSM System for Mobile Communications*, Cell and Sys, France 1992.

[2] R. Ludwig, B. Rathonyi, A. Konrad, K. Oden, A. Joseph, *Multi-Layer Tracing of TCP over a Reliable Wireless Link*, In Proceedings of ACM SIGMETRICS 99.

[3] W. R. Stevens, *TCP/IP Illustrated, Volume 1 (The Protocols)*, Addison Wesley, November 1994.

[4] ETSI, *Radio Link Protocol for data and telematic services on the Mobile Station - Base Station System (MS-BSS) interface and the Base Station System - Mobile Switching Center (BSS-MSC) interface*, GSM Specification 04.22, Version 5.0.0, December 1995.

[5] ETSI, *Digital cellular communications system (Phase 2+); Radio Link Protocol for data and telematic services on the Mobile Station - Base Station System (MS-BSS) interface and the Base Station System - Mobile Switching Center (BSS-MSC) interface*, GSM Specification 04.22, Version 6.1.0, November 1998.

[6] R. Ludwig, B. Rathonyi, *Link Layer Enhancements for TCP/ IP over GSM*, In Proceedings of IEEE INFOCOM 99.

[7] *The ICEBERG project*, CS Division, EECS Department, University of California at Berkeley, http://iceberg.cs.berkeley.edu/.

[8] Jacobson V., Leres C., McCanne S., *tcpdump*. Available at http://ee.lbl.gov/.

[9] Padmanabhan V., *tcpstats*, Appendix A of Ph.D. dissertation, University of California, Berkeley, September 1998.

[10] *Xgraph*, available at http://jean-luc.ncsa.uiuc.edu/Codes/xgraph/index.html.

[11] W. Simpson, *The Point-to-Point Protocol*, RFC 1661, July 1994.

[12] Jacobson V., *Compressing TCP/IP Headers for Low-Speed Serial Links*, RFC 1144, February 1990.

[13] M. Kojo, K. Raatikainen, M. Liljeberg, J. Kiiskinen, T. Alanko, *An Efficient Transport Service for Slow Wireless Telephone Links*, IEEE JSAC, Vol. 15, No. 7, pp. 1337-1348, September1997.

[14] R. C. Durst, G. J. Miller, E. J. Travis,*TCP Extensions for Space Communications*, In Proceedings of ACM MOBICOM 96.

[15] B. Braden, et al.,*Recommendations on Queue Management and Congestion Avoidance in the Internet*, RFC 2309, April 1998.

[16] K. K. Ramakrishnan, S. Floyd,*A Proposal to add Explicit Congestion Notification (ECN) to IP*, RFC 2481, January 1999.

[17] P. Karn, *The Qualcomm CDMA Digital Cellular System*, In Proceedings of the USENIX Mobile and Location-Independ- ent Computing Symposium, USENIX Association, August 1993.

[18] D. A. Eckhardt, P. Steenkiste,*Improving Wireless LAN Per- formance via Adaptive Local Error Control*, In Proceedings of IEEE ICNP 98.

[19] Balakrishnan H., Padmanabhan V., Seshan S., Katz R. H.,*A Comparison of Mechanism for Improving TCP Performance over Wireless Links*, In Proceedings of ACM sigcomm 96.

[20] DeSimone A., Chuah M. C., Yue O.-C, *Throughput Performance of Transport-Layer Protocols over Wireless LANs*, In Proceedings of IEEE globecom 93.

[21] Paxson, V., *End-To-End Routing Behavior in the Internet*, IEEE/ACM Transactions on Networking, Vol.5, No.5, pp. 601-615, October 1997.

[22] R. Ludwig, A. Konrad, A. Joseph, *Optimizing the End-To-End Performance of Reliable Flows over Wireless Links*, In Proceedings of ACM/IEEE MobiCom 99.