Link Layer Adaptation for Shared Wireless Links

George Xylomenos and Michael Makidis
{xgeorge@aueb.gr and mmakidis05@cs.aueb.gr
Mobile Multimedia Laboratory, Department of Informatics
Athens University of Economics and Business
Patision 76, Athens 104 34, Greece

Published in: Mobile Networks and Applications, Volume 13, Number 3–4, 2008, pp. 259–273

***Abstract*— While traditional link layer protocols assume that they fully control the underlying link, in contemporary wireless networks the link may be dynamically shared by sessions belonging to different users and/or applications. To assess the impact of link sharing, we measure the File Transfer and Web Browsing throughput achieved over a Selective Repeat (SR) protocol, with or without contention from Media Distribution. Our results indicate that the optimal protocol settings strongly depend on the level of contention for the link. We therefore present two link layer protocols that adapt to the available bandwidth, our Adaptive Selective Repeat (ASR) protocol which dynamically modifies its retransmission timeouts, and the Radio Link Control (RLC) protocol specified for use by UMTS networks which does not employ retransmission timers. We first repeat our performance measurements to determine the optimal settings for each protocol, and then compare the fine tuned versions of all protocols with respect to their File Transfer and Web Browsing throughput, as well as to the delay induced to the contending Media Distribution packets. Our results indicate that while both RLC and ASR are more stable than SR, the complex RLC does not match the performance of our simpler ASR. *Index Terms*— ink Layer Protocols, Adaptive Selective Repeat, Radio Link Controlink Layer Protocols, Adaptive Selective Repeat, Radio Link ControlL**

I. INTRODUCTION

Wireless networks have become an integral part of the Internet, especially in the role of access networks providing untethered connectivity to users. The error prone nature of wireless links, however, severely degrades the performance of many Internet applications. Reliable link layer protocols are commonly used to hide the shortcomings of wireless links; the results reported in the literature show that they can indeed improve Internet application performance [30].

Link layer protocols traditionally assume that they have exclusive access to the underlying link. Many wireless networks, however, including the *Universal Mobile Telecommunications System* (UMTS) and *Wireless Local Area Networks*, allow a single physical channel to be dynamically shared by independent users. In addition, applications may have different reliability requirements, hence even the applications of a single user may employ different link layer protocols. Therefore, multiple link layer protocol sessions should expect to coexist over the link, with the available bandwidth fluctuating as sessions start and stop.

Contention for a shared link makes the round trip time visible to each session unpredictable. The problem is that a standard *Selective Repeat* (SR) protocol cannot select an optimal retransmission timeout value without prior knowledge of the level of contention for the link [28]. Link layer protocols for shared wireless links should therefore dynamically adapt to the prevailing level of contention. In this paper we examine two appropriate adaptation approaches. The first, exemplified by our *Adaptive Selective Repeat* (ASR) protocol, is to make a standard protocol adapt to contention; in ASR this is achieved by adapting the retransmission timers using a scheme inspired by TCP [31]. The second, exemplified by the *Radio Link Control* (RLC) protocol of UMTS networks, is to adopt the most advanced features in the literature; in RLC this means avoiding retransmission timers and abandoning persistently lost frames [11].

In order to determine the best adaptation approach, we assess the performance of all protocols via a comprehensive simulation study, using different error models and applications. We first motivate the need for adaptation by showing the throughput of File Transfer and Web Browsing over SR, with or without contention from Media Distribution, showing that the optimal retransmission timeout value depends on the level of contention. We then repeat these measurements for ASR and RLC to find the optimal settings for each protocol. Finally, we compare SR, ASR and RLC in terms of both their File Transfer and Web Browsing throughput and the delay induced to the contending Media Distribution packets, showing that while both RLC and ASR can adapt to different levels of contention, unlike SR which collapses, our simple ASR generally outperforms the complex RLC.

The outline of this paper is as follows. In Section II we discuss related work and outline the contributions of our paper. In Section III we describe our simulation setup. In Section IV we describe the SR protocol and its performance as the level of contention varies. In Section V we describe the operation of ASR and identify its optimal timeout policies, while in Section VI we describe the operation of RLC and identify its optimal retransmission limits. In Section VII we compare the performance of the fine tuned versions of SR, ASR and RLC and their impact on the contending application. We summarize our findings in Section VIII.

II. RELATED WORK AND CONTRIBUTION

The *Internet Protocol* (IP) offers an unreliable packet delivery service: packets may be lost, reordered or duplicated. Many real-time applications use the *User Datagram Protocol*

(UDP) on top of IP, handling error, flow and congestion control themselves. Most other applications delegate these tasks to the *Transport Control Protocol* (TCP) which offers a reliable byte stream service. TCP segments application data into packets at the sender and reassembles it at the receiver. The receiver generates *acknowledgments* (ACKs) for segments received in sequence, returning duplicate ACKs for those arriving out of sequence. The sender retransmits the next unacknowledged segment either on receiving three duplicate ACKs or when a retransmission timer expires.

Due to the high reliability of wired links, TCP assumes that all losses are due to congestion, thus after a loss it reduces its transmission rate and then gradually increases it to probe the network's capacity; details depend on the TCP variant [22]. Unfortunately, losses due to wireless errors are also interpreted as congestion, causing TCP to repeatedly reduce its transmission rate, thus leading to poor performance even at small error rates. Many researchers have proposed TCP modifications to improve its performance over wireless links, falling in two categories. The first is to modify TCP so as to distinguish wireless from congestion losses and react differently in each case [15]; the drawback of this approach is that it can only recover from wireless losses on an end-to-end basis. The second is to split the end-to-end connection to wired and wireless parts and only perform retransmissions over the wireless part [3]; the drawback of this approach is that it breaks the end-to-end semantics of TCP.

The alternative is using a reliable link layer protocol over the wireless link to hide wireless errors from TCP. An early proposal customized to TCP *snoops* inside the packets of each TCP stream at the router bridging the wired and wireless parts of the path and retransmits lost segments when duplicate ACKs arrive, hiding these duplicate ACKs from the sender [4]. Later work shows that TCP application performance can also be enhanced with TCP unaware link layer protocols [30]. Avoiding TCP awareness has many advantages, such as compatibility with encrypted IP payloads that hide TCP headers and prevention of conflicts with TCP with the *Selective Acknowledgement* (SACK) option [25].

Despite the benefits of reliable link layer protocols for TCP applications, delay sensitive UDP applications such as Media Distribution may perform better with faster, albeit limited, error recovery [30]. To serve the needs of both TCP and UDP applications, a network should allow multiple link layer sessions to dynamically share the link; this is the approach taken in UMTS. Link sharing however causes the available bandwidth and the effective *Round Trip Time* (RTT) of the link to fluctuate, meaning that protocols for shared wireless links should be designed to provide some form of adaptation to link contention, otherwise performance will suffer as the effective RTT varies, as explained in Section IV.

Unfortunately, the existing research on link layer adaptation does not address the problems raised by link contention, as its goal is to adapt to the error characteristics of wireless links by, for example, scheduling transmissions depending on channel state or adapting the frame modulation, coding or size. While to the best of our knowledge there is no research directly related to this problem, a few schemes do provide some kind of solution. The TCP aware snoop protocol indirectly adapts to link contention as its timers follow the TCP timers, which themselves adapt to congestion, albeit on an end-to-end basis [4]. Another link layer protocol uses an adaptive timeout to limit the total number of retransmissions for frames belonging to streaming applications, dropping those frames that would miss their deadlines [8]; since individual retransmissions still use fixed timeouts, this scheme does not address the issues raised by link contention.

In contrast, the RLC protocol was explicitly designed to share the channel with other link layer sessions. Due to the large number of RLC parameters discussed in Section VI, many studies of its performance can be found in the literature. Some studies focus on the interplay between the polling and status triggers and their related timers, as the frequency of Status *Protocol Data Units* (PDUs) affects both link overhead and packet delay [1], [21], [27], [32]. Other studies focus on the impact of the RLC window size (in PDUs) or buffer size [in *Service Data Units* (SDUs)] on throughput; buffers must be large enough to avoid stalling during bad channel periods [1], [5], [14], but not too large to avoid building up queues that reduce the TCP transmission rate [26]. Other studies examine the impact of the SDU discard policy on throughput: it has been noted that the discard SDUs after a timeout option strongly depends on the link bandwidth [32]. While most studies employ TCP Reno, the beneficial impact of TCP SACK on performance has also been noted [1]. Most studies also focus on File Transfers; only a few consider Web Browsing [7], [32]. Most of these studies are not relevant to the link contention problem, as only a handful consider competing traffic [26], [27] and even they do not discuss the implications of contention.

Based on the above discussion, we can clarify the contributions of this paper. The first contribution is the identification of the problems faced by retransmission based link layer protocols when operating over shared wireless links. In Section IV we show that contention for the link adversely affects the performance of File Transfer and Web Browsing and explain why this problem is inherent to the fixed retransmission timeouts of the SR protocol. Based on this insight, the remainder of the paper discusses possible solutions to the problem of adapting to contention at the link layer.

The second contribution is a study of RLC performance over shared wireless links. Our study in Section VI is differentiated from previous ones in three ways: first, we study generic links rather than UMTS ones, although the performance parameters are similar; second, we are employing the fine granularity TCP timers currently in use, rather than the coarse grained timers of previous studies which make TCP less responsive to losses; third, we consider both the effect of UDP traffic on TCP application performance, and the impact of RLC on the delay of the contending UDP traffic.

The third contribution is our own solution to the link contention problem: a method for dynamically adapting the retransmission timeouts of the SR protocol. In Section V we present and evaluate our ASR protocol, showing how the TCP adaptation scheme can be modified for use in the link layer. We also determine the parameters of the best solution

which, interestingly, depend on the error model. As shown in Section VII, our solution, despite its simplicity, generally outperforms RLC by a wide margin, without introducing additional delay for the competing Media Distribution traffic.

III. Simulation Setup

Our performance results are from simulations with ns-2 [24] extended with additional error models and link layer protocols [2]. Each test was repeated 30 times with different random seeds; the results shown reflect average metric values as well as their 95% confidence intervals. The simulated topology is shown in Fig. 1: a Wired Server communicates with a Wireless Client via an Access Point. In all applications, the server was at the wired end of the network and the client at the wireless end. The wired link had a bandwidth of 10 Mbps and a propagation delay of 1 ms. Simulations using a 2 Mbps wired link with a propagation delay of 50 ms also support the conclusions reached in this paper.

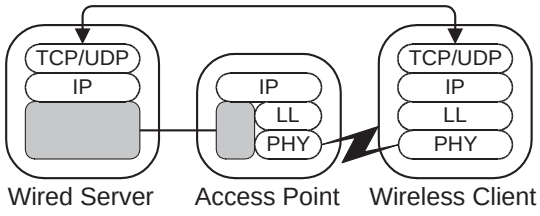

Fig. 1. Simulated network topology.

The wireless link parameters were similar to those of UMTS links, as RLC was designed for that environment. The bandwidth of 64 Kbps is what a regular user can afford, being just sufficient for the applications under study, while the propagation delay of 50 ms represents the interleaving, framing and propagation delays of UMTS [12]. The frame size used was 250 bytes plus a small link layer header; to avoid the need for packet fragmentation and reassembly, each application used 250 byte or smaller packets.

Two wireless error models were used. In the *Uniform* model each frame may be independently lost with a probability of 1.5%, 2.5%, 5.4% or 9.8%. In the *Two State* model the link can be either in a good state, with a bit error rate of 10^{-6}, or in a bad state, with a bit error rate of 10^{-2}. Both states have exponential durations, with the average duration of the good state being 10 s and the average duration of the bad state being 100 ms, 200 ms, 500 ms or 1000 ms. We first found that with these parameters the average *Frame Loss Rate* (FLR) of the Two State model is 1.5%, 2.5%, 5.4% or 9.8%, and then set the FLRs of the Uniform model to the same values, so as to examine the impact of the different error models on application performance. Note that the error processes in each link direction were identical but independent. As a baseline, we also show results with no errors.

The application layer performance offered by each protocol was evaluated via two of the most popular applications on the Internet, File Transfer and Web Browsing [23]. We employed TCP Reno [22] with 10 ms granularity timers, as on most modern systems [20]; the coarse TCP timer granularity of older systems would make TCP react unrealistically slow to losses. In File Transfer, a file was transmitted from the server to the client. While longer transfers produce more stable results, users do not make infinite transfers, so we used 10 Mbyte files which would take 21 minutes to download in the absence of errors. The ns-2 File Transfer module sends data as fast as possible, with TCP handling flow, error and congestion control. The metric of interest was File Transfer throughput, defined as the amount of application data transferred (not retransmissions) divided by total time.

Since Internet traffic studies show the volume of Web Browsing traffic to be an order of magnitude greater than that of File Transfer [23] and File Transfer throughput cannot adequately characterize the performance of Web Browsing [30], we also measured Web Browsing performance. In Web Browsing a client accesses *pages* containing text, links to other pages and embedded objects, stored on a server. The interaction consists of *transactions*: the client requests a page from a server, the server returns the page which contains pointers to embedded objects, the client requests each embedded object, and the server returns them, completing the transaction. The ns-2 Web Browsing module provides empirical distributions for request, page and embedded object sizes, as well as for the number of objects per page [16]. Only one transaction was in progress at any time, with no pauses between transactions. The metric of interest was Web Browsing throughput, defined as the amount of application data transferred from the server to the client divided by time taken. The results shown reflect the state at the end of the last completed transaction within a simulation period of 2000 s.

To examine the suitability of each protocol for shared links, we introduced contention via a UDP based Media Distribution application; it approximates a lecture where the speaker sends audio to a wireless client. The speaker alternates between *talking* and *silent* states with exponential durations, averaging 1 s and 1.35 s, respectively [18]. Packets are transmitted isochronously at a rate of 56 Kbps, consuming 87.5% of the bandwidth in the talking state, but only 37.5% of the bandwidth on average; the bandwidth fluctuates dramatically whenever the speaker changes state. The Media Distribution packets are not retransmitted, bypassing the reliable link layer protocol. However, that protocol introduces contention for the UDP application, which we assessed by measuring the Media Distribution packet delay.

IV. Fixed Selective Repeat

In this section we demonstrate the performance limitations of traditional link layer protocols over shared wireless links by studying the SR protocol, which we have found to provide excellent TCP application performance in our previous work [30]. In SR the sender transmits link layer frames within a transmission window of N frames, buffering them for retransmission. The receiver only accepts frames within a reception window also of N frames; if a frame is received in sequence it is delivered to the higher layer, the window slides upwards and an ACK is returned to the sender, confirming

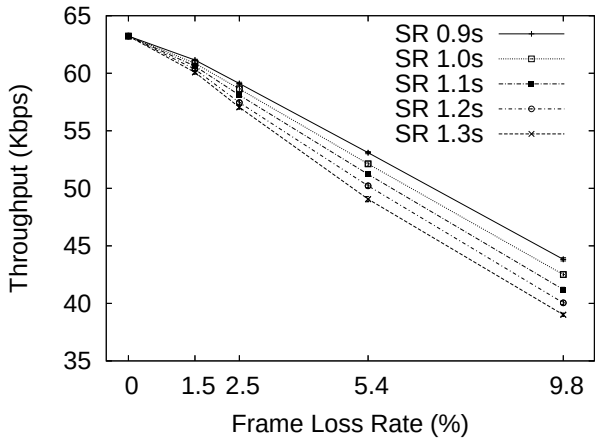

Fig. 2. File Transfer throughput (Uniform errors, no Contention).

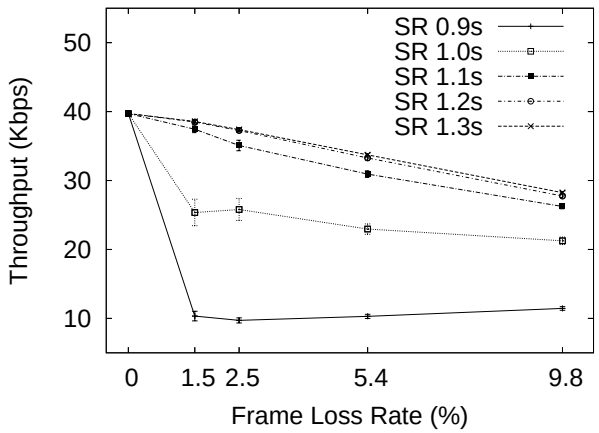

Fig. 3. File Transfer throughput (Uniform errors, Contention).

reception of all frames up to the one delivered. When the sender receives an ACK, it drops the buffered frames covered by it and also slides its window upwards.

When a frame is received out of sequence, it is buffered but not delivered, since the gap in the sequence indicates that some frames were lost; a *negative acknowledgment* (NACK) is returned for each missing frame, and the sender retransmits each NACKed frame. When a missing frame arrives, the receiver delivers to the higher layer all frames that are now in sequence. To reduce protocol overhead, we delay returning an ACK for a short interval, trying to piggyback it into a data frame traveling in the reverse direction. If that interval expires, the ACK is sent as a separate frame. NACKs are always sent immediately as separate frames.

If some ACKs and/or NACKs are lost over the wireless link, the sender may exhaust its transmission window and stall, waiting for feedback that will never arrive. To prevent this, the sender starts a retransmission timer after sending each frame. If the timer expires before an ACK for that frame arrives, the frame is assumed lost and retransmitted. Ideally, the timeout period should be set slightly higher than the expected RTT, that is, the time between sending a frame and receiving an ACK for that frame. If the timeout period is shorter than the RTT, frames will be needlessly retransmitted; if it is much longer than the RTT, recovery from lost frames will be unnecessarily delayed.

Many variants of SR exist, mostly differing on NACK handling. The variant used in this paper allows each missing frame to be NACKed multiple times if the receiver detects that its retransmissions have also been lost, a feature called *multireject* [6]. We also tested two simpler SR variants, one where only a single NACK can be outstanding at any time, and one where only a single NACK may be sent for each missing frame [6]; apart from providing lower performance, they also support the conclusions reached in this paper.

We now turn to the performance of File Transfer and Web Browsing over SR, with or without contention from Media Distribution. In all tests the SR window was set to 127 frames, the delayed ACK timer was set to 0.5 s and the retransmission timeout ranged from 0.9 s to 1.3 s. Regarding

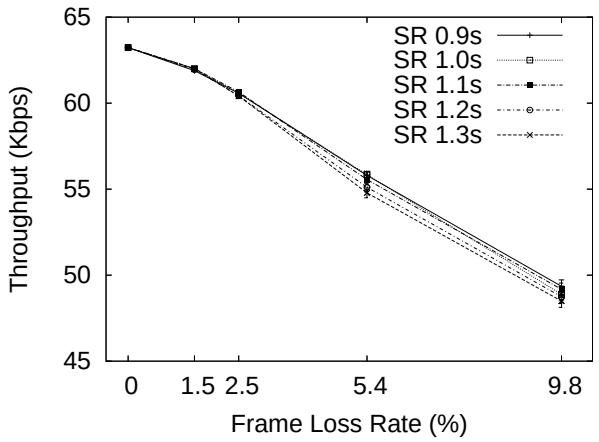

Fig. 4. File Transfer throughput (Two State errors, no Contention).

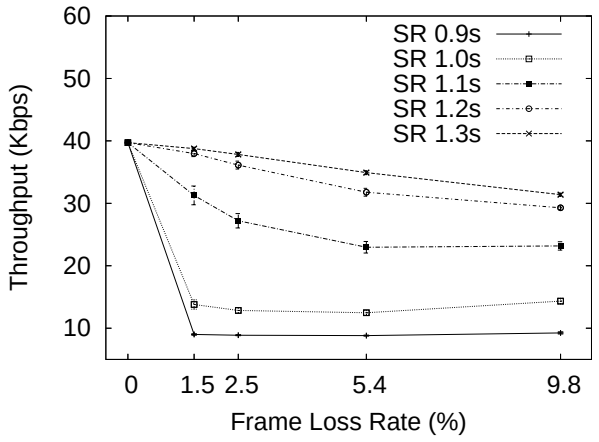

Fig. 5. File Transfer throughput (Two State errors, Contention).

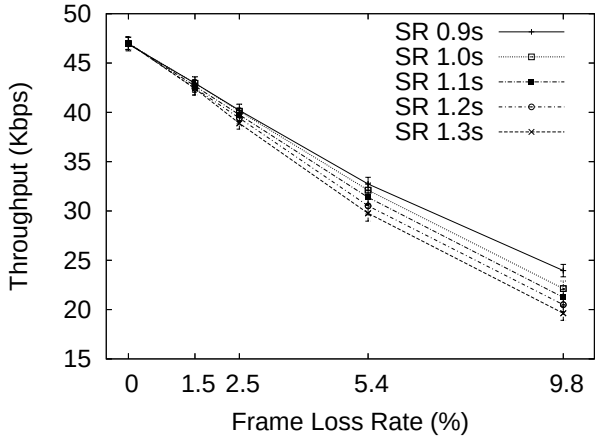

Fig. 6. Web Browsing throughput (Uniform errors, no Contention).

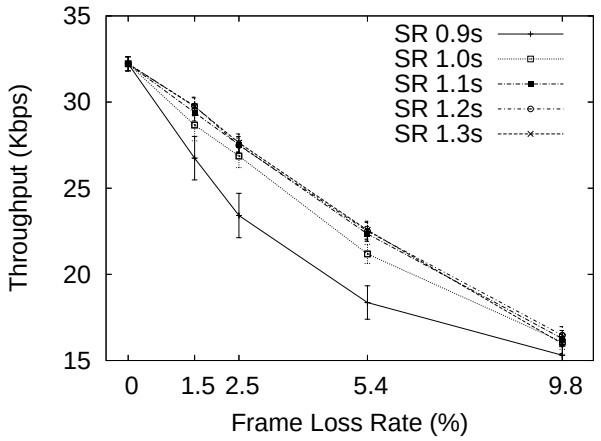

Fig. 7. Web Browsing throughput (Uniform errors, Contention).

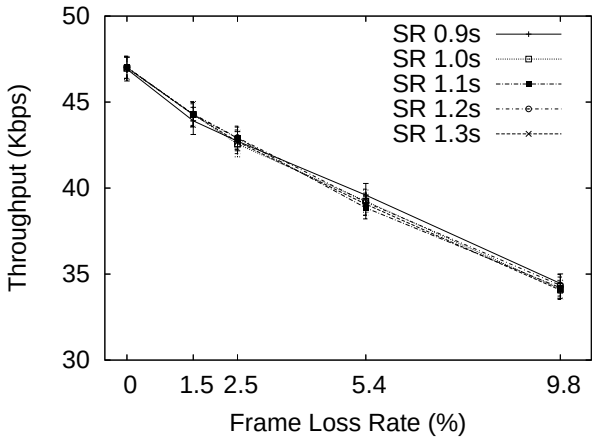

Fig. 8. Web Browsing throughput (Two State errors, no Contention).

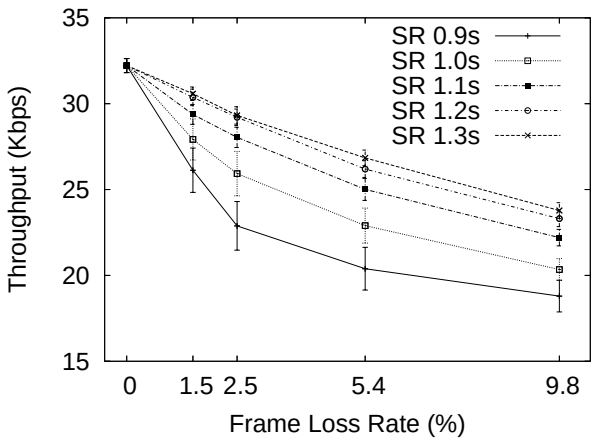

Fig. 9. Web Browsing throughput (Two State errors, Contention).

File Transfer, Fig. 2 shows that with the Uniform error model and without contention, throughput is maximized with the lowest timeout value, progressively decreasing as the timeout period decreases. The *5.4% performance gap*, that is, the difference between the best and worst options at a FER of 5.4%, is 8.23%. When contention is introduced however, the situation is reversed, as Fig. 3 shows: in this case, lower timeout values perform worse, while higher values lead to progressive improvement. In this case the 5.4% performance gap is 228%, but in the opposite direction: the optimal timeout value without contention offers less than one third of the optimal performance with contention.

The performance results with the Two State error model further support these findings. Figure 4 shows that the File Transfer throughput without contention is optimized with the lowest timeout value, although the difference is smaller: the 5.4% performance gap from the highest timeout value is only 1.85%. However, Fig. 5 shows that when contention is introduced, not only the situation is reversed in favor of the highest timeout value, the 5.4% performance gap from the lowest timeout value is even higher at 296%.

Regarding Web Browsing, Fig. 6 shows that with the Uniform error model and without contention, throughput is again maximized with the lowest timeout value; the 5.4% performance gap from the highest timeout value is 9.98%. When contention is added, Fig. 7 shows that the highest timeout value performs best, with a 5.4% performance gap of 22.88%, but in the opposite direction than without contention. With the Two State error model and without contention, Fig. 8 shows that the throughput of Web Browsing is again optimized with the lowest timeout value, albeit with a 5.4% performance gap of just 1.35%. When contention is introduced, Fig. 9 shows that the situation is reversed and the 5.4% performance gap in favor of the highest timeout value is 31.64%. Therefore, the Web Browsing results confirm that the optimal timeout value depends on the level of contention.

To summarize, we have found that for both File Transfer and Web Browsing and with both the Uniform and the Two State error models, it is not possible to select a single retransmission timeout value for SR that will provide optimal performance regardless of the level of contention. These results confirm that, as we have found in previous work [28], a timeout value that provides excellent performance without contention can only offer suboptimal Web Browsing performance with contention. This paper further shows that contention has a far more pronounced impact on File Transfer performance: the

optimal value without contention provides less than a third of the optimal performance with contention.

V. Adaptive Selective Repeat

In Section IV we showed that contention for the link dramatically affects SR performance: as the available bandwidth fluctuates, so does the effective RTT and therefore the optimal retransmission timeout value. Ideally, a retransmission based link layer protocol for shared wireless links would adapt its timeouts, always keeping them slightly higher than the RTT. To achieve this, we modified the SR protocol to track the RTT in a manner similar to TCP and set its retransmission timeouts accordingly; this is our ASR protocol [31].

The ASR protocol generally operates exactly the same as SR, but in addition, for every packet transmitted or retransmitted the sender stores its transmission time. When an ACK arrives for the packet, the difference between the current time and the transmission time provides an RTT $sample$. These samples are used to update smoothed estimates for the RTT, $srtt$, and its variance, $srttvar$, as follows:

$$srtt = 0.875 * srtt + 0.125 * sample$$

$$srttvar = 0.75 * srttvar + 0.25 * |srtt - sample|$$

These estimators react to the RTT fluctuations with a time lag, without being dramatically affected by sporadic extreme values. The equations and smoothing factors used are those of TCP [19], allowing the calculations to be performed efficiently using only integer arithmetic [13]; these calculations are performed with a 10 ms granularity as in TCP. We have also tested a wide range of other values for the smoothing factors but found that the TCP values provide good performance under most conditions. After updating the estimators, the new timeout value, $rtxto$, is calculated as follows:

$$rtxto = \alpha * srtt + \beta * srttvar$$

While this equation is also borrowed from TCP, the actual values recommended for use with TCP, that is, $\alpha = 1$ and $\beta = 4$ [19], were found to lead to suboptimal performance. We therefore tested an extended set of values for α and β, referring to the resulting adaptive timeout policies as $\alpha + \beta$; for example, the TCP policy is referred to as ASR $1 + 4$.

Our ASR protocol calculates samples from every acknowledged packet, with three exceptions. First, NACKs are not used to calculate samples, since they do not reflect reception of the frame indicated. Second, when an ACK covers multiple frames, only the last frame acknowledged is used to calculate a sample; the previous ones may have been received much earlier. Third, when duplicate ACKs arrive, only the original ACK is used to provide a sample.

The ASR retransmission timeout policy also diverges from TCP in that the timeout value is *not* modified after a timeout occurs, unlike TCP which performs binary exponential backoff. The reason is that consecutive timeouts over a wireless link are more likely to indicate wireless losses rather than severe congestion, therefore they do not reflect a dramatic change to the RTT. Furthermore, when setting the retransmission timer for a frame, ASR ensures that it will not expire before any previously set retransmission timer by increasing the timer (not the $rtxto$) if needed; this ensures that the SR correctness proofs remain valid for ASR.

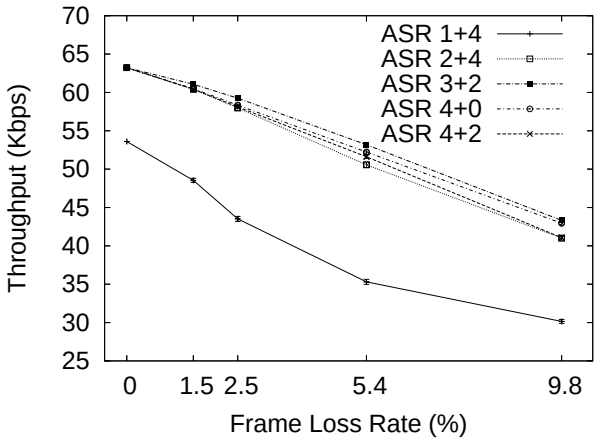

Fig. 10. File Transfer throughput (Uniform errors, no Contention).

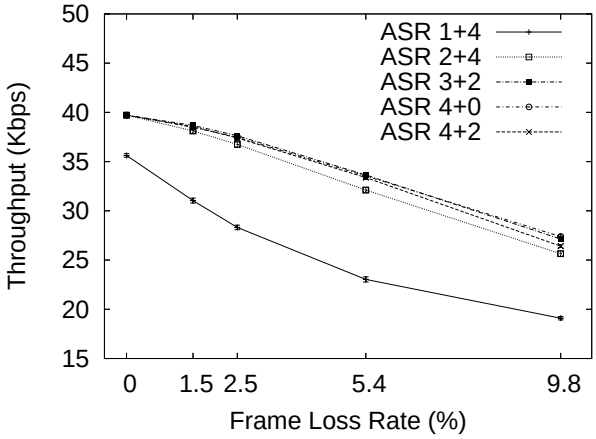

Fig. 11. File Transfer throughput (Uniform errors, Contention).

We now turn to the performance of File Transfer over ASR, with or without contention from Media Distribution. All ASR settings are the same as for SR, with the initial $srtt$ set to 1.1 s and the initial $srttvar$ set to 0. In addition to the TCP policy ASR $1 + 4$ we show results for the ASR $2 + 4$, $3 + 2$, $4+0$ and $4+2$ policies. Figure 10 shows that with the Uniform error model and without contention throughput is maximized with ASR $3 + 2$; the TCP policy ASR $1 + 4$ performs worse than the other policies even with no errors. The simulation logs indicate that ASR $1+4$ causes many timeouts to occur, despite the absence of errors and contention, indicating that this policy tends to calculate very short timeouts. The 5.4% performance gap between the best and worst policies is 50.64%. When contention is added, Fig. 11 shows that ASR $3+2$ remains the best and ASR $1 + 4$ remains far worse than all other policies, with a 5.4% performance gap between the two of 46.06%. It is important to note that the same policy, ASR $3+2$, provides the best performance regardless of the level of contention.

The performance results with the Two State error model lead to similar findings. Figure 12 shows that the throughput of File

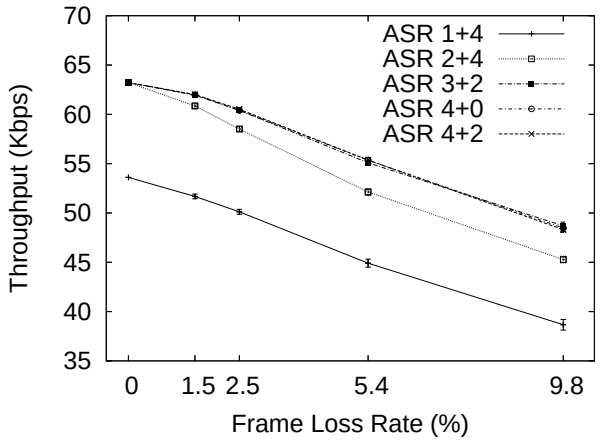

Fig. 12. File Transfer throughput (Two State errors, no Contention).

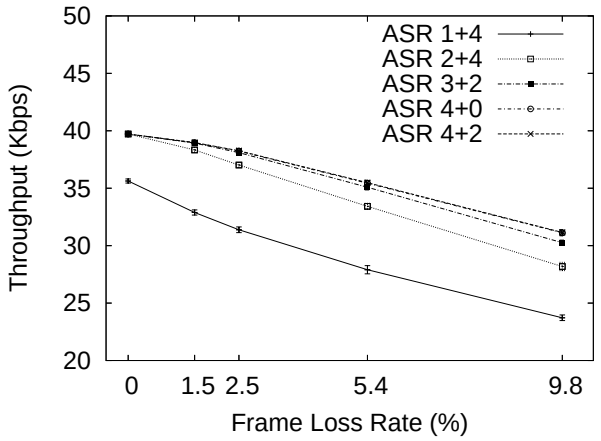

Fig. 13. File Transfer throughput (Two State errors, Contention).

Transfer without contention is optimized with ASR $4+0$. The TCP policy ASR $1+4$ again performs worse than the other policies, even with no errors, with a 5.4% performance gap between it and the $4+0$ policy of 23.25%. Figure 13 shows that when contention is introduced ASR $4+0$ remains the best policy, ASR $1+4$ remains by far the worst, and the 5.4% performance gap between them is 26.99%. Again, the same policy, ASR $4+0$, provides the best performance regardless of the level of contention.

Due to space limitations, we omit the corresponding figures for Web Browsing; the results however confirm our previous observations, that is, that the optimal policies are ASR $3+2$ for Uniform errors and ASR $4+0$ for Two State errors, regardless of the level of contention, while the TCP policy ASR $1+4$ is always the worst. The 5.4% performance gaps for the Uniform error model are 33.71% without and 37.89% with contention, while for the Two State error model they are 20.54% without and 25.45% with contention.

To summarize, we have found that the ASR protocol reaches its optimal performance with the same policy, regardless of the application and the level of contention. These results confirm that, as we have found in previous work [31], the optimal adaptation policy for Web Browsing depends on the error model: ASR $3+2$ performs best with Uniform errors and ASR $4+0$ performs best with Two State errors. This paper further shows that these findings also apply to File Transfer, which exhibits even higher 5.4% performance gaps between the best and worst policies than Web Browsing. It also shows that the use of integer arithmetic and 10 ms granularity timers to calculate the smoothed estimates, rather than the floating point previously used [29], [31], only slightly improves the performance of the TCP policy ASR $1+4$.

VI. Radio Link Control

The RLC protocol is mandatory for all UMTS link layer sessions, therefore three modes are supported. In *Transparent Mode* the protocol simply passes through all frames unmodified; this is used for voice calls. In *Unacknowledged Mode* the protocol only performs duplicate frame avoidance and reordering; this is useful for applications requiring low delay, such as media streaming. In *Acknowledged Mode* the protocol performs flow control, error detection and retransmission of missing frames; this is suitable for applications requiring a reliable link, such as TCP based ones. We focus below exclusively on the Acknowledged Mode of RLC.

The basic operation of RLC is similar to that of SR. The sender is passed a sequence of variable size packets from the higher layer, called SDUs, it segments and/or packs the SDUs into a sequence of fixed size frames, called PDUs, and transmits these frames to the receiver. The receiver uses the sequence of incoming PDUs to reassemble the original sequence of SDUs and deliver it to the higher layer. The sender (receiver) may only transmit (accept) PDUs within a sender (receiver) window. The receiver advances its window as *Acknowledged Mode Data* (AMD) PDUs are received in sequence; it notifies the sender about received AMD PDUs by returning *Status* PDUs and the sender advances its window accordingly.

The first departure of RLC from SR is that Status PDUs may contain additional information in the form of one or more *Super Fields* (SUFIs). All RLC implementations must support the ACK SUFI, used to acknowledge all PDUs up to the one indicated; other SUFIs are optional. The BITMAP SUFI indicates the complete status of the receiver window after the latest frame received in sequence, with each bit indicating whether the corresponding frame has been received or not. Status PDUs may be piggybacked in the padding of the fixed size AMD PDUs, or sent as independent frames.

The second departure of RLC from SR is that Status PDUs are *not* returned after every frame received: they are either requested by the sender or returned by the receiver based on various triggers. Therefore, RLC cannot use retransmission timers to detect lost PDUs; it must instead rely on the information returned in Status PDUs. The absence of retransmission timers makes RLC robust to contention for the link, as the protocol makes no assumptions about the RTT of the link. At the same time, it makes the policies for Status PDU generation critical for performance.

The sender can ask for a Status PDU by setting a polling bit in the header of an AMD PDU. The polling function may

TABLE I
RLC CONFIGURATION PARAMETERS.

Parameter	Uniform	Two State
Window size	128 frames	128 frames
Piggybacked Status	No	No
Poll triggers	Poll Timer, Window Based	Poll Timer, Window Based, Every Last PDU in Retr. Buffer
Poll Timer timeout	200 ms	200 ms
Window threshold	70%	80%
Poll Prohibit	Yes (100 ms)	Yes (100 ms)
Status Report triggers	Missing PDU, Timer based	Missing PDU, Timer based
Status Report timeout	400 ms	500 ms
Status Prohibit	No	Yes (90 ms)
SDU Discard mode	Discard after x transmissions	
Maximum retransmissions	1 to 5	1 to 5
MRW timeout	500 ms	110 ms
Piggybacked Status	No	No

be triggered by a number of events: (a) when the last PDU in the transmission (or retransmission) buffer is transmitted, (b) after every x PDUs (or SDUs) are transmitted, (c) when the sender's window usage is higher than a configured limit, (d) when a periodic timer expires and, (e) when a timer set on sending a poll expires before either the frame which contained the poll and all previous ones have been acknowledged, or that frame has not been negatively acknowledged.

The receiver may return a Status PDU if any of the following events occur: (a) when gaps are detected in the received AMD PDU sequence, (b) when a periodic timer expires and, (c) when a request arrives from the lower layers. Due to the numerous options for triggering polls and status replies, RLC also defines two options to limit the number of Status PDUs. If *Poll Prohibit* is used, the sender starts a timer after transmitting a poll and defers further polls until the timer expires. If *Status Prohibit* is used, the receiver starts a timer after returning a Status PDU and defers further Status PDUs until the timer expires. Any combination of these mechanisms can be configured; normally, at least one side must periodically poll for or return Status PDUs [21].

The third departure of RLC from SR is that RLC can abandon persistently lost PDUs. Since TCP will eventually timeout and retransmit the corresponding packets, repeated RLC retransmissions only lead to a waste of bandwidth [9]. The sender may drop unacknowledged data via the *SDU Discard* function, whereby the sender notifies the receiver to advance its window by transmitting a *Move Reception Window* (MRW) SUFI in a Status PDU, the receiver advances its window and acknowledges by returning a Status PDU with an MRW_ACK SUFI, and the sender finally advances its window. Note that the persistent loss of a PDU causes the entire SDU of which that PDU is a part of to be discarded. The sender can be configured to use one of three SDU Discard modes: (a) discard PDUs when a period of time elapses after their first transmission, (b) discard PDUs after a number of unsuccessful transmissions, or (c) reset the peer RLC entities after a number of unsuccessful PDU transmissions.

We fine tuned the RLC parameters for our wireless links based on an extensive simulation study [17]. Table I lists the RLC parameters used in our simulations. We only study below the SDU Discard after a number of transmissions policy as it automatically adapts to the prevailing RTT, unlike the SDU Discard after a period of time policy which requires choosing a fixed timeout value; in a shared wireless link this is as hard as setting the SR retransmission timeout.

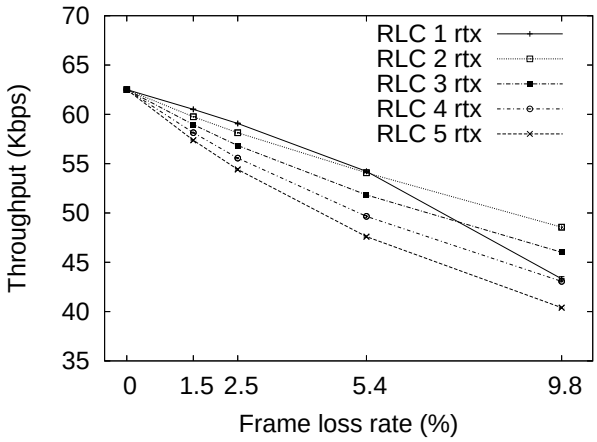

Fig. 14. File Transfer throughput (Uniform errors, no Contention).

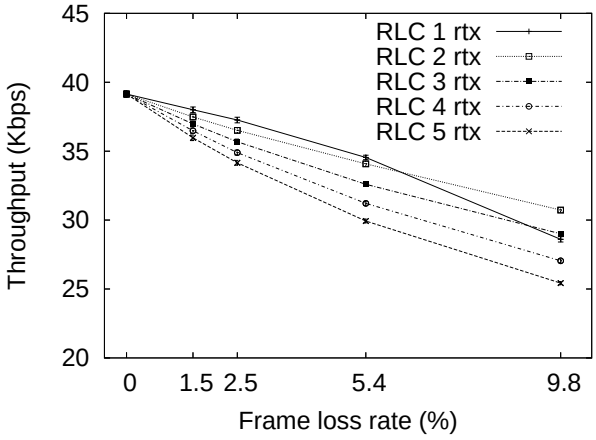

Fig. 15. File Transfer throughput (Uniform errors, Contention).

We now turn to the performance of File Transfer over RLC, with or without contention from Media Distribution, while varying the retransmission limit from 1 to 5. Figure 14 shows that with the Uniform error model and without contention, throughput is maximized with 1 retransmission at the lower FLRs, while for higher FLRs 2 retransmissions are required for optimal performance. As the performance gap between these policies at lower FLRs is very small, the latter policy should be preferred due to its stability. The 5.4% performance gap between the best and worst policies is 13.59%. When contention is added, Fig. 15 shows that RLC again performs best with 1 retransmission at lower FLRs and 2 retransmissions at higher FLRs; the latter is again the preferable policy. The 5.4% performance gap between the best and worst policies is 13.85%. Note that the relative performance of each RLC policy is not affected by the level of contention.

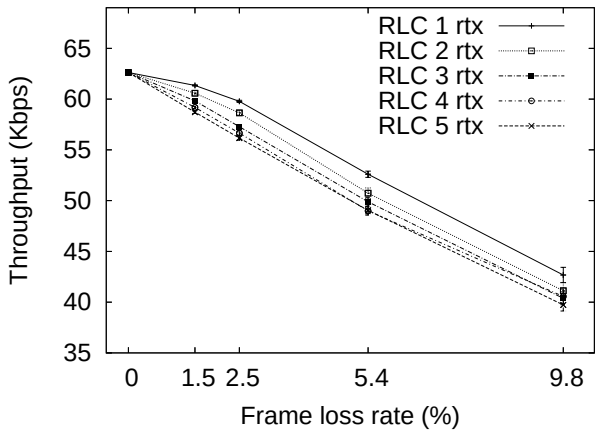

Fig. 16. File Transfer throughput (Two State errors, no Contention).

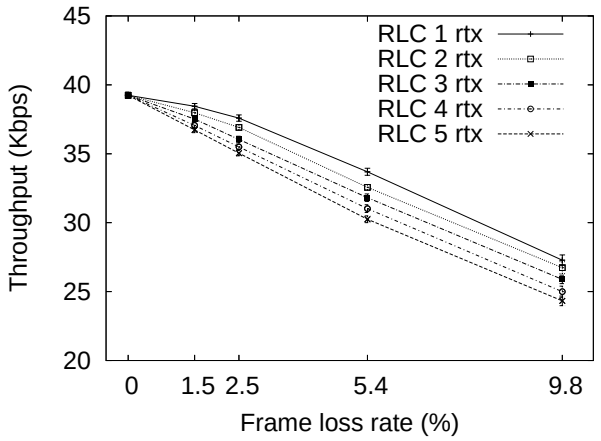

Fig. 17. File Transfer throughput (Two State errors, Contention).

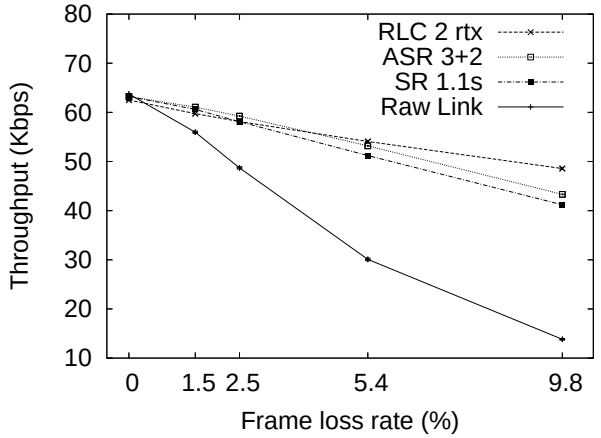

Fig. 18. File Transfer throughput (Uniform errors, no Contention).

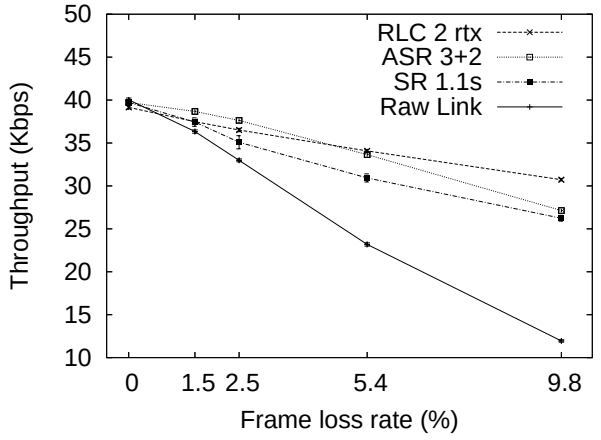

Fig. 19. File Transfer throughput (Uniform errors, Contention).

The performance of File Transfer with the Two State error model is only slightly different. Figure 16 shows that the throughput of File Transfer without contention is optimized with 1 retransmission, regardless of the FLR. As the number of retransmissions increases the throughput drops, with the 5.4% performance gap between the best and worst policies being 7.25%. Figure 17 shows that when contention is introduced, 1 retransmission remains the best option, with the 5.4% performance gap with the worst performing policy being 11.37%. Again, the relative performance of each RLC policy is unaffected by the level of contention.

Due to space limitations, we omit the corresponding figures for Web Browsing; the results are similar to those of File Transfer, indicating that the optimal RLC policy is 2 retransmissions for Uniform errors and 1 retransmission for Two State errors, regardless of the level of contention. The 5.4% performance gaps between the best and worst policies for the Uniform error model are 10.71% without and 11.75% with contention, while for the Two State error model they are 4.56% without and 5.25% with contention.

To summarize, we have found that the RLC protocol achieves its optimal performance with the same policy, regardless of the application and the level of contention. For the Uniform error model, 1 retransmission provides the best performance for lower FLRs, but 2 retransmissions provide nearly the same performance for lower FLRs and better performance at higher FLRs. For the Two State error model the best performance is always offered with 1 retransmission. As the number of retransmissions increases, application performance decreases for two reasons: first, the same packet may be retransmitted by both layers leading to wasted bandwidth [9]; second, the increased bandwidth-delay product of the path reduces the TCP transmission rate [10]. Therefore, the SDU discard option is beneficial for TCP performance.

VII. Overall Performance Comparison

In this section we compare the fine tuned variants of each protocol against each other, as well as with the *Raw Link*: this is what TCP can achieve without any link layer error recovery. For the SR protocol there is no single optimal retransmission timeout value, therefore we use the 1.1 s value as a compromise. For the ASR protocol we use the $3 + 2$ policy for the Uniform error model and the $4 + 0$ policy for the Two State error model, while for the RLC protocol we use a limit of 2 retransmissions for the Uniform error model and 1 retransmission for the Two state error model.

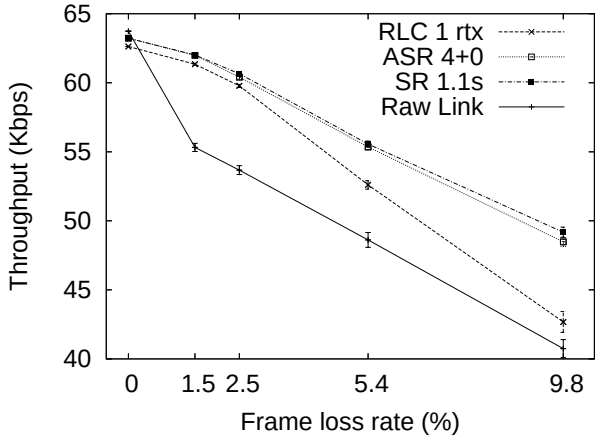

Fig. 20. File Transfer throughput (Two State errors, no Contention).

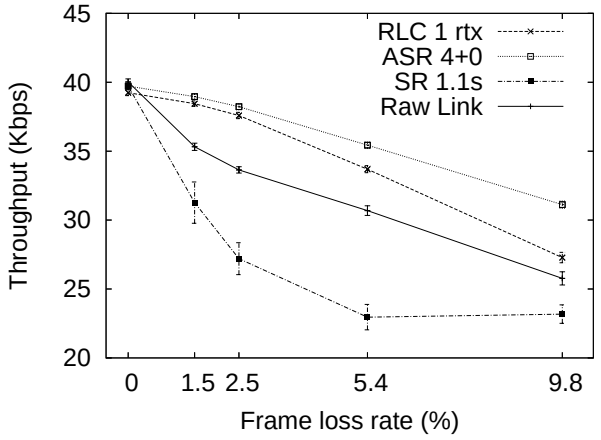

Fig. 21. File Transfer throughput (Two State errors, Contention).

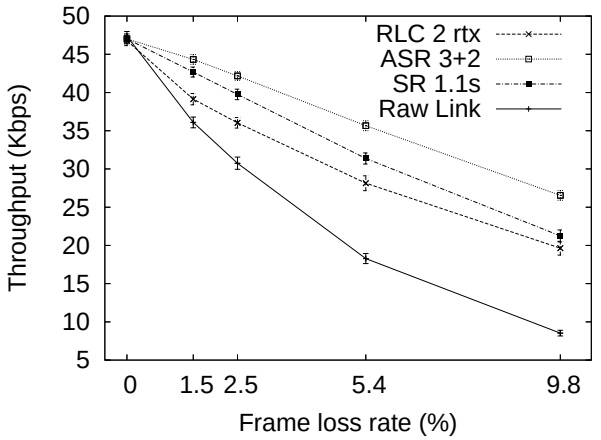

Fig. 22. Web Browsing throughput (Uniform errors, no Contention).

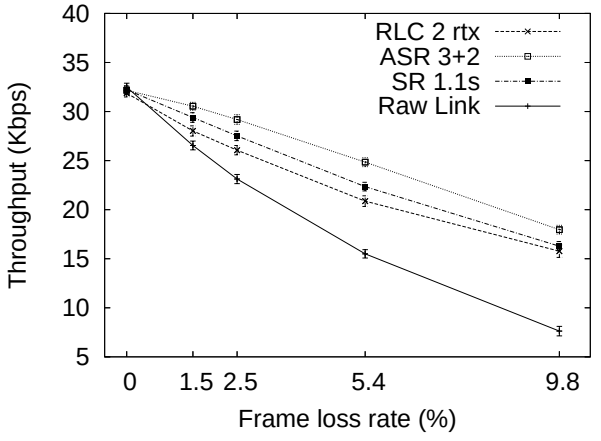

Fig. 23. Web Browsing throughput (Uniform errors, Contention).

We begin with the performance of File Transfer, with or without contention from Media Distribution. Figure 18 shows that with the Uniform error model and without contention, all protocols provide a considerable performance improvement over the Raw Link. ASR outperforms SR, albeit by a small margin, also outperforming RLC at lower FLRs, while at higher FLRs RLC has a performance advantage over ASR. The 5.4% performance gap between ASR and Raw Link is 76.73%. When contention is introduced, Fig. 19 shows that the situation is quite similar: ASR outperforms RLC at lower FLRs, RLC outperforms ASR at higher FLRs, and ASR outperforms SR. The 5.4% performance gap between ASR and Raw Link is 45.11%. Note that RLC performance is slightly reduced due to the non piggybacked Status PDUs, as is evident in the no loss case.

With the Two State error model, the situation is quite different. Figure 20 shows that without contention ASR always outperforms RLC, with the performance gap widening at higher FLRs, where RLC performance is closer to that of Raw Link; in this scenario, SR performs slightly better than ASR. The 5.4% performance gap between ASR and Raw Link is 13.84%. When contention is introduced, Fig. 21 shows that while ASR and RLC perform nearly the same as without contention, that is, ASR outperforms RLC by a widening margin as the FLR is increased, SR performs worse even than Raw Link. This clearly shows the limitations of SR: with the same timeout, it is the best performer without contention but the worst with contention. The 5.4% performance gap between ASR and Raw Link is 15.46%.

We then turn to the performance of Web Browsing. Figure 22 shows that with the Uniform error model and without contention, not only ASR outperforms SR and RLC by a considerable margin, even SR outperforms RLC at all FLRs. The 5.4% performance gap between ASR and Raw Link is 95.09%. Figure 23 shows that the situation remains the same when contention is added: ASR clearly outperforms both SR and RLC, while SR outperforms RLC, albeit by a smaller margin. The 5.4% performance gap between ASR and Raw Link is 60.29%. It should be clear that with Uniform errors RLC cannot provide good Web Browsing performance, despite providing excellent File Transfer performance.

The situation is even worse for RLC with the Two State error model. As shown in Fig. 24, in the absence of contention RLC provides performance closer to Raw Link than to ASR, while ASR outperforms SR. The 5.4% performance gap between ASR and Raw Link is 20.42%. When contention

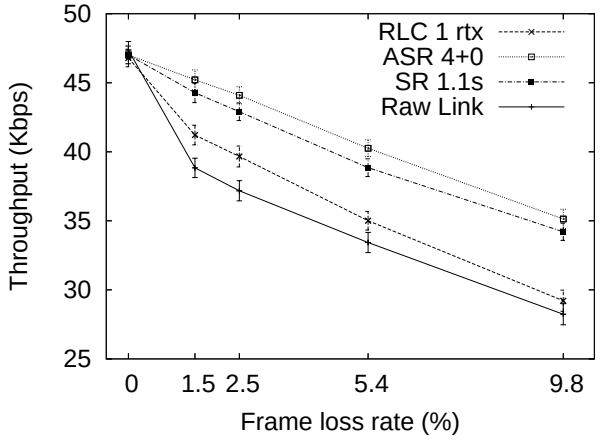

Fig. 24. Web Browsing throughput (Two State errors, no Contention).

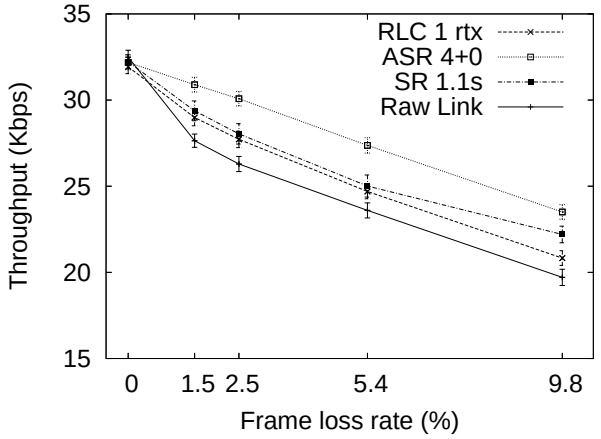

Fig. 25. Web Browsing throughput (Two State errors, Contention).

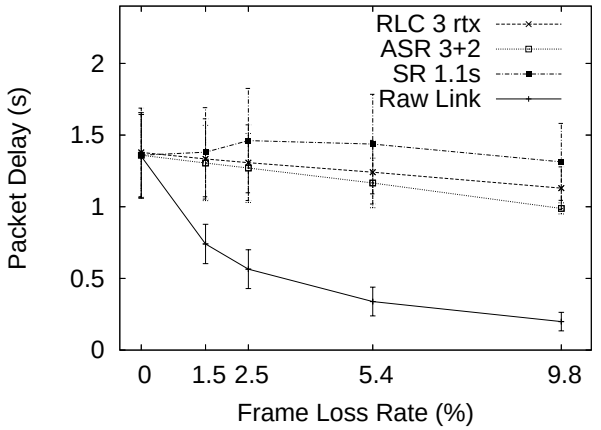

Fig. 26. Media Distribution delay (Uniform errors, Contention from File Transfer).

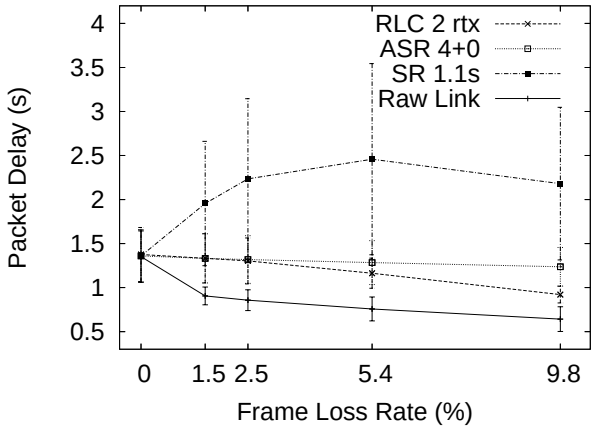

Fig. 27. Media Distribution delay (Two State errors, Contention from File Transfer).

is introduced, Fig. 25 shows that RLC is again closer to Raw Link than to ASR; in this case SR performs nearly the same as RLC. The 5.4% performance gap between ASR and Raw Link is 15.98%. Therefore, with the Two State error model RLC cannot provide good performance for either File Transfer or Web Browsing.

Finally, we examine the impact of each protocol to Media Distribution. Figure2 26 and 27 show Media Distribution packet delay when contending against File Transfer with Uniform and Two State errors, respectively. For most protocols the delay is proportional to the File Transfer throughput achieved: with the Uniform error model RLC introduces more delay than ASR, while with the Two State error model ASR introduces more delay than RLC; in both cases, Raw Link introduces the lowest delays. The exception is SR which introduces the highest delays, even though it provides worse performance than ASR and RLC.

Due to space limitations, we omit the corresponding figures for Media Distribution packet delay when contending against Web Browsing. The results are similar to those with File Transfer: ASR introduces higher delays, as it provides the highest Web Browsing performance, followed by RLC and Raw Link. Again, SR introduces the highest delays even though it provides worse performance than ASR and RLC. It should be noted that with both File Transfer and Web Browsing, ASR and RLC *never* increase the delay of Media Distribution packets compared to the no loss case, therefore, the adaptive protocols not only improve TCP performance, they also lead to lower UDP delays than the fixed SR.

We can summarize these results as follows. First, while RLC provides good performance for File Transfer with Uniform errors, as it is more robust to higher FLRs than ASR, the situation is reversed with Two State errors, with RLC being closer to Raw Link than ASR at higher FLRs. An explanation of this phenomenon is that while RLC can recover from multiple losses within the same RTT due to its BITMAP SUFIs, the SDU Discard function makes it less robust to the long error bursts of the Two State model. Second, RLC works better with File Transfer than with Web Browsing, since with the latter it is outperformed not only by ASR, but even by FSR. This phenomenon can be explained by the short transfers typical of Web Browsing: when the last packet of a transfer is lost, RLC will not detect the loss until a Status PDU is asked for or returned due to a timer expiration. As these timers are conservative to avoid overloading the link,

RLC cannot react to these losses as fast as ASR and SR with their tight retransmission timers. Third, ASR and RLC do not increase the delay experienced by contending Media Distribution packets, despite the performance gains offered to File Transfer and Web Browsing, unlike SR which increases delays despite its lower performance.

VIII. Conclusions

In this paper, we first presented the problems faced by reliable link layer protocols when sharing a wireless link with competing link layer sessions, using the SR protocol as an example. Our measurements indicate that the optimal retransmission timeout values for SR strongly depend on the level of contention: for both wireless error models tested, a single value cannot provide optimal File Transfer and Web Browsing performance both with and without contention. We then presented two protocols designed for shared wireless links, our own ASR protocol, an adaptive timeout extension of SR, and the RLC protocol, a complex protocol with numerous options and parameters. Regarding ASR, our measurements indicate that while the TCP adaptation policy does not work well, the ASR $3+2$ policy for Uniform errors and the ASR $4+0$ policy for Two State errors provide optimal application performance, regardless of the level of contention. Regarding RLC, our measurements indicate that 2 retransmissions for Uniform errors and 1 retransmission for Two State errors provide optimal application performance, again regardless of the level of contention.

We then compared the fine tuned versions of SR, ASR and RLC. While we found both ASR and RLC to be insensitive to the level of link contention, in contrast to SR that collapses under some circumstances, their behavior is quite different. RLC performs well with File Transfer and Uniform errors at high loss rates, while with Two State errors it barely improves upon plain TCP at high loss rates. With Web Browsing, RLC performs worse even than SR with both error models. In contrast, ASR provides excellent performance regardless of the application and error model, nearly always outperforming RLC. Finally, both RLC and ASR did not induce additional delay to the contending Media Distribution packets, unlike SR that inflated delays despite its lower performance. We therefore conclude that adaptivity at the link layer is critical for performance over shared wireless links and that our simple ASR is superior to the far more complex RLC for both bulk transfer and interactive applications, regardless of the level of contention and error model.

References

[1] Alcaraz JJ, Cerdan F, Garcia-Haro J (2006) Optimizing TCP and RLC interaction in the UMTS radio access network. IEEE Network 20(2): 56–64

[2] AUEB/MMLAB: Multi service link layers for ns-2. Available at http://www.mm.aueb.gr/~xgeorge/codes/codephen.htm

[3] Badrinath B, Bakre A, Imielinski T, Marantz R (1993) Handling mobile clients: A case for indirect interaction. In: Proc. of the 4th workshop on workstation operating systems. IEEE Computer Society Press, Silver Spring, pp 91–97

[4] Balakrishnan H, Padmanabhan VN, Seshan S, Katz RH (1996) A comparison of mechanisms for improving TCP performance over wireless links. In: Proc. of the ACM SIGCOMM. Association for Computer Machinery, New York, pp 256–267

[5] Bestak R, Godlewski R, Martins P (2002) RLC buffer occupancy when using a TCP connection over UMTS. In: Proc. of the IEEE PIMRC, vol. 3. IEEE, Piscataway, pp 1161–1165

[6] Brady PT (1987) Evaluation of multireject, selective reject, and other protocol enhancements. IEEE Trans Commun 35(6): 659–666

[7] Cano-Garcia J, Gonzalez-Parada E, Casilari-Perez E (2006) On the impact of RLC layer configuration parameters in UMTS internet access. In: Proc. of the IEEE vehicular telecommunications conference fall. IEEE, Piscataway, pp 1–5

[8] Chen L, Kapoor R, Lee K, Sanadidi M, Gerla M (2004) Audio streaming over Bluetooth: an adaptive ARQ timeout approach. In: Proc. of the international conference on distributed computing systems workshops. IEEE, Piscataway, pp 196–201

[9] DeSimone A, Chuah MC, Yue OC (1993) Throughput performance of transport-layer protocols over wireless LANs. In: Proceedings of the IEEE GLOBECOM '93. IEEE, Piscataway, pp 542–549

[10] Fall K, Floyd S (1996) Simulation based comparisons of Tahoe, Reno and SACK TCP. Comput Commun Rev 26(3): 5–21

[11] 3rd Generation Partnership Project (3GPP) (2006) Radio link control (RLC) protocol specification (Release 7). Technical Specification 25.322, V7.0.0

[12] Hernandez-Valencia EJ, Chuah MC (2000) Transport delays for UMTS VoIP. In: Proc. of the IEEE wireless communications and networking conference. IEEE, Piscataway, pp 1552–1556

[13] Jacobson V (1988) Congestion avoidance and control. In: Proc. of the ACM SIGCOMM. Association for Computer Machinery, New York, pp 314–329

[14] Lefevre F, Vivier G (2001) Optimizing UMTS link layer parameters for a TCP connection. In: Proc. of the IEEE vehicular telecommunications conference spring, vol. 4. IEEE, Piscataway, pp 2318–2322

[15] Ludwig R, Katz R (2000): The eifel algorithm: making TCP robust against spurious retransmissions. Comp Commun Rev 30(1): 30–36

[16] Mah BA (1997) An empirical model of HTTP network traffic. In: Proc. of the IEEE INFOCOM. IEEE, Piscataway, pp 592–600

[17] Makidis M (2007) Implementing and evaluating the RLC/AM protocol of the 3GPP specification. Master's thesis, Dept. of Informatics, Athens University of Economics and Business, Available at http://mm.aueb.gr/archive.html

[18] Nanda S, Goodman DJ, Timor U (1991) Performance of PRMA: a packet voice protocol for cellular systems. IEEE Trans Veh Technol 40(3): 584–598

[19] Paxson V, Allman M (2000) Computing TCP's retransmission timer. Request For Comments 2988

[20] Rewaskar S, Kaur J, Smith FD (2007) Performance study of loss detection/recovery in real-world TCP implementations. In: Proc. of the IEEE international conference on network protocols. IEEE, Piscataway

[21] Rossi M, Scaranari L, Zorzi M (2003) On the UMTS RLC parameters setting and their impact on higher layers performance. In: Proc. of the IEEE vehicular telecommunications conference fall, vol. 3. IEEE, Piscataway, pp 1827–1832

[22] Stevens W (1997) TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. Request For Comments 2001

[23] Thompson K, Miller G, Wilder R (1997) Wide-area Internet traffic patterns and characteristics. IEEE Netw 11(6): 10–23

[24] UCB/LBNL/VINT: Network Simulator - ns (version 2). Available at http://www.isi.edu/nsnam

[25] Vangala S, Labrador M (2007) Shielding TCP from last hop wireless losses. Wirel Commun Mob Comput 7(6): 679–688

[26] Xu H, Chen YC, Xu X, Gonen E, Liu P (2002) Performance analysis on the radio link control protocol of UMTS system. In: Proc. of the IEEE vehicular telecommunications conference fall, vol. 4. IEEE, Piscataway, pp 2026–2030

[27] Xu X, Chen YC, Xu H, Gonen E, Liu P (2002) Simulation analysis of RLC timers in UMTS systems. In: Proc. of the winter simulation conference, vol. 1. IEEE, Piscataway, pp 506–512

[28] Xylomenos G (2006) Limitations of fixed timers for wireless links. In: Proc. of the int. symposium on parallel and distributed processing and applications. Springer, Heidelberg, pp 159–170

[29] Xylomenos G, Makidis M (2007) Adaptive link layer protocols for shared wireless links. In: Proc. of the ACM international mobile multimedia communications conference, Nafpaktos, 27–29 August 2007

[30] Xylomenos G, Polyzos GC (2004) A multi-service link layer architecture for the wireless Internet. Int J Commun Syst 17(6): 553–574

[31] Xylomenos G, Tsilopoulos C (2006) Adaptive timeout policies for wireless links. In: Proc. of the int. conference on advanced information networking and applications, vol. 1. IEEE, Piscataway, pp 497–502

[32] Zhang Q, Su HJ (2002) Performance of UMTS radio link control. In: Proc. of the IEEE international conference on communications, vol. 5. IEEE, Piscataway, pp 3346–3350