Adaptive Link Layer Protocols for Shared Wireless Links

George Xylomenos and Micahel Makidis
Mobile Multimedia Laboratory
Department of Informatics
Athens University of Economics and Business
Patision 76, Athens 104 34, Greece
Email: xgeorge@aueb.gr and mikem4600@gmail.com

***Abstract*— The error prone nature of wireless links often necessitates the use of a link layer protocol to ensure acceptable application performance. While traditional link layers assume that they fully control the link, in most emerging wireless networks many sessions may dynamically share the link due to the presence of multiple contending users and/or applications. Such networks require link layers that can automatically adapt to bandwidth variations, offering good performance regardless of contention. To this end, we discuss two adaptive protocols, an Adaptive Selective Repeat (ASR) protocol that dynamically modifies its retransmission timeouts, and the Radio Link Control (RLC) protocol used by UMTS, an advanced protocol without retransmission timers. To assess the applicability of each approach, we measure the throughput achieved by File Transfer and Web Browsing over both protocols, with or without contention from a Media Distribution application, as well as the delay induced by these protocols to the contending application. Our results indicate that the complexity of RLC is not justified by its performance, as ASR nearly always outperforms it.**

I. Introduction

Wireless networks have become an integral part of the Internet, especially as access networks providing wireless connectivity to users. However, the error prone nature of wireless links severely degrades the performance of many Internet applications. The use of reliable link layer protocols is a direct way to locally hide the deficiencies of wireless links and improve the performance of higher layer protocols and applications [11]. This is the option adopted by the *3rd Generation Partnership Project* (3GPP) in its specifications for the *Universal Mobile Telecommunications System* (UMTS) to improve the behavior of the cellular network air interface.

A limitation of most existing link layer protocols is that they set their parameters assuming exclusive access to the underlying wireless link. Many emerging wireless networks however, most notably UMTS and *Wireless Local Area Networks* (WLANs), allow a single physical channel to be dynamically shared by independent users. Furthermore, since different applications have different requirements, multiple link layer protocols should expect to co-exist with each other over the link. In both cases, the bandwidth available to each session fluctuates, necessitating the use of link layers capable of dynamically adapting their operation. Indeed, we have found that a non-adaptive retransmission based link layer protocol cannot offer optimal performance under different contention levels [10].

In this paper we examine two different approaches to link layer adaptivity. The first, exemplified by the *Adaptive Selective Repeat* (ASR) protocol [12], is to start with a standard protocol and make it adapt to prevailing conditions; in the ASR case by adapting its retransmission timers. The second, exemplified by the *Radio Link Control* (RLC) protocol [1], is to adopt the most advanced features in the literature; in the RLC case by avoiding retransmission timers, offering many feedback options and abandoning persistently lost frames. We put both protocols to the test by measuring the throughput of File Transfer and Web Browsing over each one, with or without contention for the link from a Media Distribution application, as well as the delay that they induce to the media packets.

The outline of this paper is as follows. In Section II we provide background on Internet protocol and application performance over wireless links. In Section III we describe the operation of ASR, while in Section IV we describe the operation of RLC. Section V describes our simulation setup. Section VI discusses the performance of File Transfer and Web Browsing over ASR and RLC, first without contention for the link, and then when contention is introduced by Media Distribution; in the latter case, we also discuss media packet delay. We summarize our findings in Section VII.

II. Background and Related Work

The *Internet Protocol* (IP), offers an unreliable packet delivery service: IP packets may be lost, reordered or duplicated. Many real-time applications use the *User Datagram Protocol* (UDP) for direct access to this service, handling error, flow and congestion control themselves, so as to best support their needs. Most other applications delegate these tasks to the *Transport Control Protocol* (TCP) which offers a reliable byte stream service. TCP segments the application data into packets at the sender and reassembles it at the receiver, only passing correctly sequenced data to the application. The receiver generates *acknowledgments* (ACKs) for segments received in sequence, returning duplicate ACKs for segments arriving out of sequence. The sender retransmits the next unacknowledged segment either on receiving three duplicate ACKs or when a retransmission timer expires before an ACK is received.

Due to the high reliability of wired links, TCP assumes that all segment losses are due to congestion, thus after a loss it abruptly reduces its transmission rate to relieve congestion, and then gradually increases it to gently probe the network for available capacity. Unfortunately, losses due to wireless errors are also interpreted as congestion, causing TCP to repeatedly reduce its transmission rate. Many researchers have proposed TCP modifications to improve its performance over wireless links, but they all have two drawbacks: they require modifications to end hosts throughout the Internet and they can only retransmit lost data on an end-to-end basis.

The alternative is to employ a reliable link layer protocol over the wireless link so as to hide wireless errors from TCP. An early proposal customized to TCP *snoops* inside the packets of each TCP stream at the access point bridging the wired and wireless parts of the path and retransmits lost segments when duplicate ACKs arrive, hiding duplicate ACKs from the sender to prevent end-to-end recovery [2]. Later work shows that TCP application performance can be enhanced with standard reliable link layer protocols without TCP awareness [11]. Avoiding TCP awareness has many advantages, such as compatibility with encrypted IP payloads, as well as prevention of adverse interactions with newer versions of TCP, such as TCP with the Selective Acknowledgement option [8].

An important issue with reliable link layer protocols is that not every application requires their services. Delay sensitive UDP applications, such as Media Distribution, may prefer faster, albeit limited, error recovery, therefore reliable link layers should expect to share the wireless link with other protocols. This causes the bandwidth available to the reliable link layer protocol, and therefore its effective *Round Trip Time* (RTT), to fluctuate. The problem is how to set the retransmission timers of a reliable link layer protocol when the effective RTT varies. A TCP aware link layer solves this problem by monitoring TCP retransmission timers [2], but this approach is tied to TCP. The idea of adapting the retransmission timers can be used with other protocols though, as discussed in Section III. Alternatively, retransmission timers can be avoided when regular receiver feedback is available, as discussed in Section IV.

III. Adaptive Selective Repeat

Having found the *Selective Repeat* (SR) protocol to offer good performance for TCP based applications in our past research [11], we used it to experiment with the interaction between link sharing and retransmission timers. In Selective Repeat, the sender sequentially numbers and transmits link layer frames within a transmission window of N frames, buffering them for possible retransmission. The receiver accepts frames that lie within a reception window of N frames. When frames are received in sequence they are passed to the higher layer, the reception window slides upwards and an ACK is returned to the sender, acknowledging all frames received in sequence. When the sender receives an ACK, it drops the buffered frames confirmed by it and also slides its window upwards.

When a frame arrives 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 to the sender, and the sender retransmits each NACKed frame. When missing frames arrive, the receiver delivers to the higher layer all frames that are now in sequence, slides its window upwards and returns an ACK for all delivered frames. 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 the interval expires, the ACK is sent as a separate frame. NACKs on the other hand are always sent immediately as separate frames. Many Selective Repeat variants exist, mostly differing on NACK handling [4]. The variant used here allows each missing frame to be NACKed multiple times, a feature called *multireject*.

If some ACKs and/or NACKs are lost, the sender may exhaust its transmission window and stall. To prevent this, the sender starts a retransmission timer after sending every frame. If the timer expires before an ACK arrives, the frame is assumed to be lost, so it is retransmitted. Based upon experimental results showing how difficult it is to select a proper timeout value in a shared wireless link environment [10], we modified this *Fixed Selective Repeat* (FSR) protocol to adapt its retransmission timers similarly to TCP [5]; this is the *Adaptive Selective Repeat* (ASR) protocol. For every packet (re)transmitted, the sender notes its transmission time. When an ACK arrives, the difference between the current and the transmission time provides an RTT $sample$. We use these samples to update estimates for the RTT, $srtt$, and its variance, $srttvar$:

$$srtt = 0.875 * srtt + 0.125 * sample \quad (1)$$

$$srttvar = 0.75 * srttvar + 0.25 * (sample - srtt) \quad (2)$$

As the effective RTT fluctuates, the estimators (1) and (2) follow its progress in a smoothed manner, i.e., they are not affected by sporadic extreme values. The smoothing factors used are those of TCP, meaning that these calculations can be performed very efficiently in integer arithmetic [5]. After updating the estimators, the new value to be used for the retransmission timers, $rtxto$, becomes:

$$rtxto = \alpha * srtt + \beta * srttvar \quad (3)$$

While this estimator (3) has the same form as the one used by TCP, the actual TCP coefficients ($\alpha = 1$ and $\beta = 4$) offer suboptimal performance in this setting [10]. Our scheme calculates samples from *every* packet acknowledged, with three exceptions: (a) NACKs do not provide samples, since they do not reflect reception of the NACKed frame, (b) when an ACK covers multiple frames, only the last frame acknowledged provides a sample, and (c) when duplicate ACKs arrive, only the first ACK provides a sample.

IV. Radio Link Control

In UMTS networks the *Radio Link Control* (RLC) protocol is used for all sessions, therefore it offers three different modes. In the *Transparent Mode* (RLC/TM), frames are passed unmodified from sender to receiver; this mode is suitable for voice calls. In the *Unacknowledged Mode* (RLC/UM), a

sequence number is added to each frame to support (optional) duplicate frame avoidance and reordering; it is suitable for applications requiring low delay, such as UDP based Media Distribution. Finally, in the *Acknowledged Mode* (RLC/AM), a full header is added to each frame to support error detection and retransmission of missing frames; it is suitable for applications requiring a reliable channel, such as TCP based File Transfer and Web Browsing. In the remainder of this paper we focus on RLC/AM, as this is the closest competitor of SR.

RLC/AM is a retransmission based protocol like SR. The sender is passed a sequence of variable size packets from the higher layer, called *Service Data Units* (SDUs), it segments and/or packs them into a sequence of fixed size frames, called *Protocol Data Units* (PDUs), and it transmits these to the receiver. The receiver uses the sequence of incoming frames to reassemble and deliver the original packet sequence to the higher layer. The sender labels each *Acknowledged Mode Data* (AMD) PDU with a sequence number and buffers it for possible retransmission. The receiver advances its window for every frame received in sequence within the window, detecting losses when out of sequence frames arrive, and the sender advances the transmission window based on the ACKs returned.

The first departure of RLC/AM from SR is that the receiver can inform the sender about received and missing frames by returning *Status* PDUs which contain one or more *Super Fields* (SUFIs). Implementations are free to decide which SUFIs to use in each Status PDU. Every implementation must support at least the ACK SUFI, used to acknowledge all frames up to the one indicated; other SUFIs are optional. The BITMAP SUFI indicates the 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 into the padding of the (fixed size) AMD PDUs, or sent as independent frames.

The second departure from SR is that Status PDUs are not automatically returned after every frame received: they are either explicitly requested by the sender or returned by the receiver based on a trigger. Therefore, RLC/AM cannot use retransmission timers to detect losses; it must instead rely on the Status PDUs returned for this purpose. This characteristic of RLC/AM makes it robust to contention for the link, as it does not need to make any assumptions about the effective RTT of the link. On the other hand, it makes the policies used to trigger Status PDU generation critical.

On the sending side, a Status PDU can be requested by setting a polling bit in the header of an AMD PDU. This may be triggered by a number of events: (a) whenever the last PDU in the transmission or (separate) retransmission buffer is sent, (b) whenever x PDUs or SDUs are transmitted, (c) whenever the sender's window occupancy is higher than a configured limit, (d) whenever a periodic timer expires, and (e) whenever a timer set when the latest poll was sent expires before the arrival of new ACKs or NACKs. Any combination of these can be configured for use by the sender.

On the receiving side, in addition to polls from the sender, Status PDUs may be triggered by a number of other events: (a) whenever the receiver detects gaps in the received frame sequence, and (b) whenever a periodic timer expires. Due to the numerous options for triggering poll requests and status replies, two optional functions can limit the number of Status PDUs generated: (a) the sender can start a timer after transmitting a poll, deferring further polls until it expires, and, (b) the receiver can start a timer after generating a Status PDU, deferring further Status PDUs until it expires.

Finally, RLC/AM departs from SR in that it may abandon persistently lost PDUs, thus trading off reliability for delay and preventing conflicting retransmissions between the link and transport layers. This *SDU Discard* function can be triggered by a number of events: (a) whenever a frame is not acknowledged for a period of time after its first transmission, and (b) whenever a frame is not acknowledged after a number of transmissions. When SDU Discard is activated, the sender transmits a *Move Reception Window* (MRW) SUFI in a Status PDU to the receiver, the receiver advances its window and acknowledges with a Status PDU containing an MRW_ACK SUFI, and then the sender advances its window.

V. Simulation Setup

The performance results reported in Section VI are based on simulations with ns-2 [7], extended with additional wireless error models and link layers [9]. Each test was repeated 30 times with different random seeds and the results shown reflect average metrics from these 30 runs, as well as their 95% confidence intervals. The simulated topology is shown in Figure 1: a Wired Server communicates with a Wireless Client via an Access Point. In all applications, the server was located at the wired end of the network and the client at the wireless end, hence the naming convention used. The wired link has 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.

The wireless link has a bandwidth of 64 Kbps, a propagation delay of 50 ms and uses a frame size of 250 bytes plus a header; these are typical characteristics for cellular links, where bit interleaving inflates delay. To avoid packet fragmentation, each application uses 250 byte or smaller packets. Two error models were used for the wireless link. In the *Uniform* error model each frame may be independently lost with a probability of 1.5%, 2.5%, 5.4% or 9.8%. In the *Two State* error 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 found experimentally 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%, matching the FLRs used for the Uniform model. Note that the error processes in each link direction were identical but independent. To establish a baseline, we also show results with no errors.

To evaluate the link layer protocols under study, we used File Transfer and Web Browsing, two of the most popular applications on the Internet, over TCP Reno with 10 ms

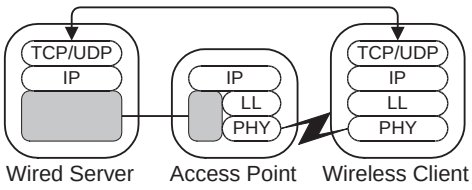

Fig. 1. Simulated network topology.

granularity timers. In File Transfer, a file was transmitted from the wired server to the wireless client. While longer transfers produce more stable results, users do not make infinite transfers, so we used a file size of 10 Mbytes. The ns-2 File Transfer module sends data as fast as possible, with TCP handling flow, error and congestion control. We measured File Transfer throughput, defined as the amount of application data transferred (not retransmissions) divided by total time.

As in our past work we found that File Transfer throughput cannot characterize the performance of interactive applications [11], we also employed Web Browsing. 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. Only one transaction was in progress at any time, with no pauses between them. We measured 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 the simulation period of 2000 s.

In order to examine the suitability of each protocol for a dynamic link sharing environment, we introduced contention via a UDP based Media Distribution application, approximating a lecture where a 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. Packets are transmitted isochronously at a rate of 56 Kbps, consuming 87.5% of the available bandwidth in the talking state, but only 37.5% on average. No retransmissions are performed for Media Distribution: its packets bypass the link layer protocol used by the TCP packets. However, that protocol introduces contention for the UDP application, which we assessed by measuring Media Distribution packet delay.

For the TCP based applications we examined four error control options: *Raw Link*, or no error control at all, *Fixed Selective Repeat* (FSR), *Adaptive Selective Repeat* (ASR), and *Radio Link Control* (RLC) in the Acknowledged Mode. Both FSR and ASR used a window size of 128 frames. FSR used a retransmission timeout of 1.1 s, a value found to provide good performance with or without contention for the link [10]. ASR set its retransmission timeouts by using equation (3) with the values $\alpha = 3, \beta = 2$ for the Uniform error model and $\alpha = 4, \beta = 0$ for the Two State error model, found to offer

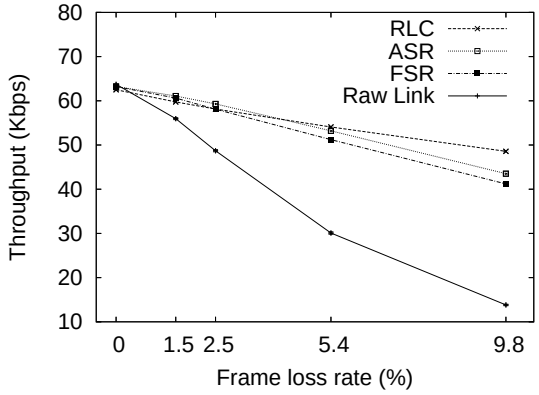

Fig. 2. File Transfer throughput (uniform, no contention).

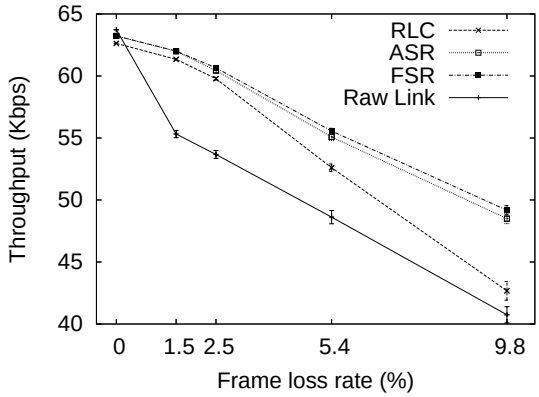

Fig. 3. File Transfer throughput (two state, no contention).

the best performance for each type of link [12].

Previous studies have found that the SDU discard policy [3] is critical for RLC performance, something confirmed by our extensive simulation study of RLC [6]. Based on that study we selected the following RLC settings; all settings apply to both Uniform and Two State links unless otherwise indicated. The RLC window size was 128 frames, with Status PDUs transmitted as separate frames (not piggybacked). We used SDU discard with a finite number of transmissions, 3 for Uniform links and 2 for Two State links. Polls were triggered at the sender whenever the transmission window occupancy reached 70% for Uniform links and 80% for Two State links, or when a 200 ms poll timer expired; on Two State links polls were also triggered whenever the retransmission buffer was exhausted; after triggering a poll, further polls were deferred for 100 ms. Status PDUs were also triggered at the receiver whenever a missing PDU was detected, as well as every 400 ms for Uniform links and every 500 ms for Two State links; after triggering a Status PDU, further Status PDUs were deferred for 90 ms on Two State links. Finally, the retransmission timeout for MRW SUFIs was 500 ms for Uniform links and 110 ms for Two State links.

VI. Performance Evaluation

We begin with the performance of File Transfer and Web Browsing when operating without contention for the link.

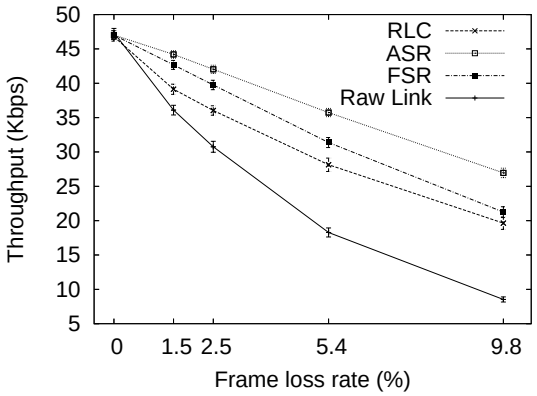

Fig. 4. Web Browsing throughput (uniform, no contention).

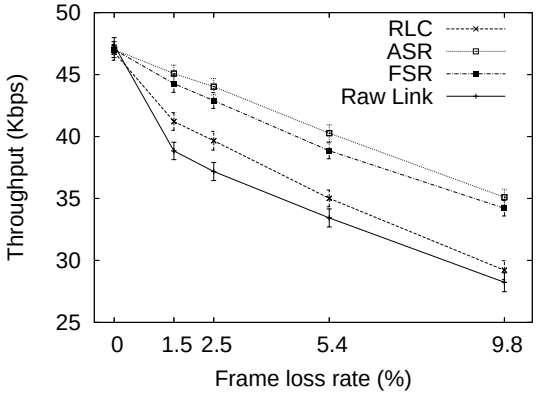

Fig. 5. Web Browsing throughput (two state, no contention).

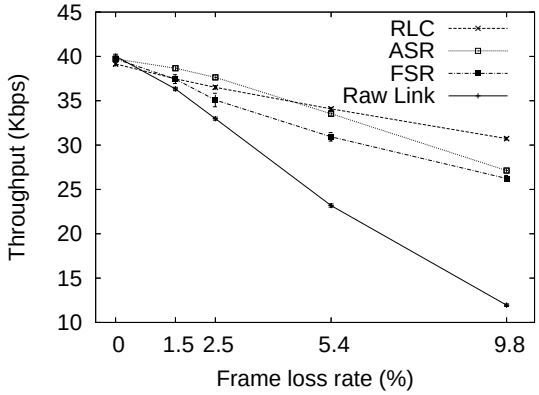

Fig. 6. File Transfer throughput (uniform, contention).

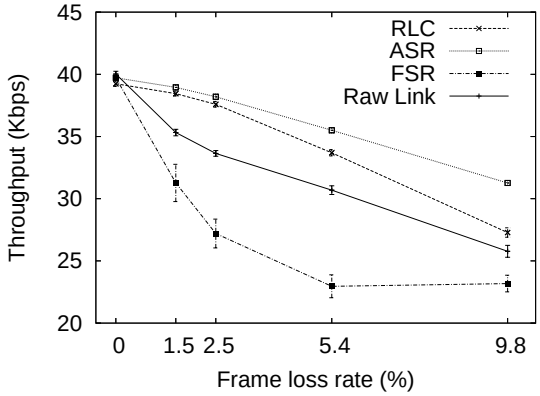

Fig. 7. File Transfer throughput (two state, contention).

Regarding File Transfer, Figure 2 shows that with the Uniform error model all link layer protocols provide a considerable performance improvement over the Raw Link. ASR consistently outperforms FSR, albeit by a small margin, also outperforming RLC at the lower loss rates; at the highest loss rate, RLC has a clear performance advantage over ASR, despite its additional overhead due to the non piggybacked Status PDUs. With the Two State error model though, Figure 3 shows that the situation is different, since both ASR and FSR outperform RLC, with the performance gap widening at higher loss rates; in this scenario, FSR performs slightly better than ASR.

Regarding Web Browsing, Figure 4 shows that with this application and the Uniform error model, ASR not only consistently outperforms FSR by a considerable margin, FSR also clearly outperforms RLC throughout the loss rate scale; only at the highest loss rate does RLC approach FSR. The situation is similar with the Two State error model, as shown in Figure 5; indeed, in this case the performance advantage of ASR and FSR over RLC is so large that RLC is closer to the Raw Link than to the SR protocols.

From these results we can make four observations. First, while RLC is robust to higher FLRs with the Uniform error model, the situation is reversed with the Two State error model, unlike ASR and FSR which exhibit similar behavior with both error models. Second, RLC works better with File Transfer than with Web Browsing. Third, ASR consistently outperforms RLC, except with File Transfers over Uniform links and high loss error rates. Fourth, in the absence of contention, FSR also outperforms RLC in most cases.

We now turn to the performance of File Transfer and Web Browsing when contention is introduced by the Media Distribution application, thus reducing the bandwidth available to the TCP based applications and increasing their effective RTT, making the fixed retransmission timers of FSR more likely to expire prematurely. Regarding File Transfer, Figure 6 shows that with the Uniform error model the situation is quite similar to the corresponding no contention case: ASR outperforms RLC at the lower loss rates, RLC outperforms ASR at the highest loss rate, and ASR outperforms FSR, but by a higher margin. With the Two State error model though, Figure 7 shows that while ASR and RLC perform more or less as in the no contention case, that is, with ASR outperforming RLC by a widening margin as the error rate is increased, in this scenario FSR performs worse even than the Raw Link.

Regarding Web Browsing, Figure 8 shows that with the Uniform error model the situation is again quite similar to the corresponding no contention case, with ASR outperforming FSR, which in turn outperforms RLC, albeit by a smaller margin. The situation is similar with the Two State error model, as shown in Figure 9, with ASR providing the best performance, followed by FSR and RLC, with both performing nearly the same; in this case though, FSR does not perform

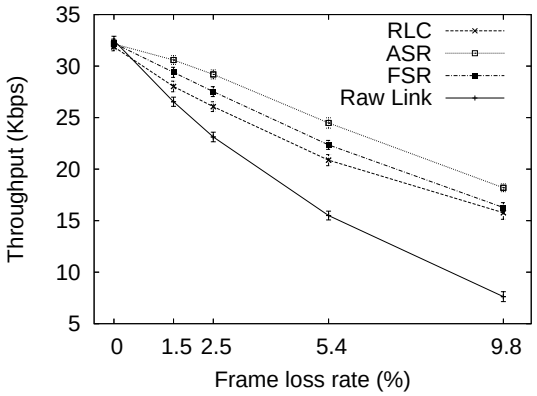

Fig. 8. Web Browsing throughput (uniform, contention).

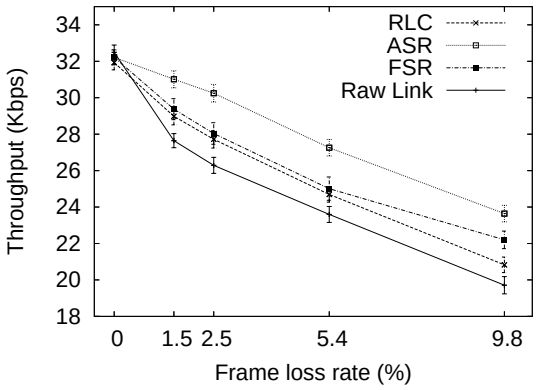

Fig. 9. Web Browsing throughput (two state, contention).

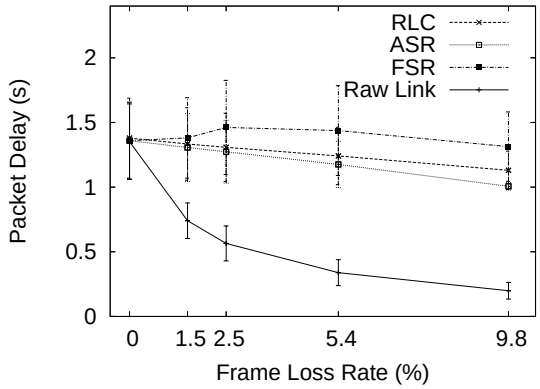

Fig. 10. Media Distribution delay (uniform, contention).

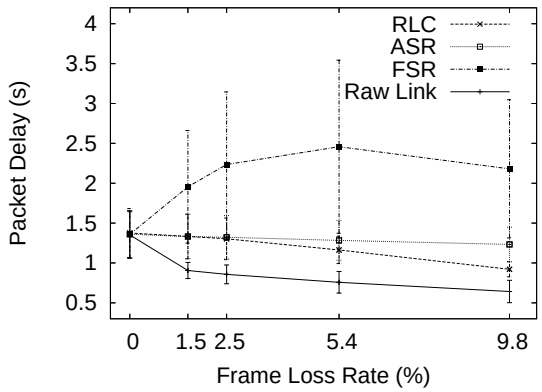

Fig. 11. Media Distribution delay (two state, contention).

worse than the Raw Link, as in the File Transfer case.

From these results we can make three observations. First, RLC exhibits the same overall behavior as without contention, that is, better performance with File Transfer and the Uniform error model, especially at the highest loss rates. Second, ASR also exhibits the same overall behavior as without contention, that is, similar performance with both applications and error models. Third, while the adaptive protocols (ASR and RLC) are relatively insensitive to contention for the link, the fixed timeout protocol (FSR) is not: at best it performs worse than in the corresponding no contention case, and at worst it is outperformed even by the Raw Link.

Finally, we examine the impact of each link layer protocol to the Media Distribution application. Figure 10 and Figure 11 show Media Distribution packet delay when contending against File Transfer with the Uniform and Two State error model, respectively; Figure 12 and Figure 13 show the same metric when Media Distribution contends against Web Browsing. Both ASR and RLC do not increase the delay experienced by Media Distribution packets, but the protocol offering the best TCP based application performance induces higher delays to the UDP based application. In contrast, FSR not only induces the highest delays, it even inflates them with the Two State error model. Therefore, the adaptive link layer protocols not only improve performance for the TCP based applications, they also lead to lower delays for the UDP based application.

VII. Conclusions

We have presented two link layer protocols designed to operate in a shared wireless link environment, the ASR protocol, a straightforward adaptive timeout extension of traditional Selective Repeat, and RLC, a completely new protocol design with numerous options and parameters, and evaluated their performance under a wide range of scenarios. While we found both protocols to be insensitive to the level of contention for the link, in contrast to the non adaptive FSR protocol that collapses under some circumstances, their behavior is quite different. RLC performs best with random rather than with bursty errors and with bulk transfer rather than with interactive applications, unlike ASR which offers good performance with both types of application and error model. Furthermore, we found that in most cases ASR outperforms RLC, in some cases by a wide margin, despite the higher complexity and numerous tuning parameters of RLC. Finally, both RLC and ASR did not induce additional delay to the contending media packets, again unlike FSR. We therefore conclude that while adaptivity at the link layer is critical for performance in a shared wireless link environment, our simple ASR protocol is superior to the far more complex RLC protocol.

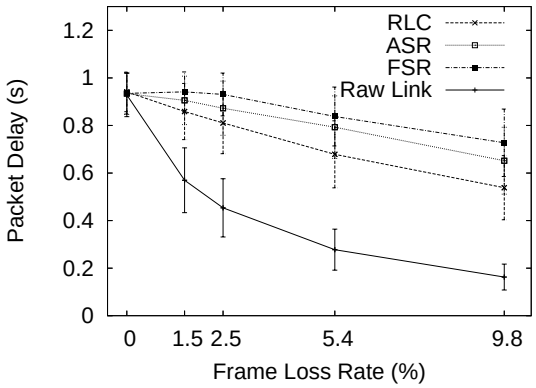

Fig. 12. Media Distribution delay (uniform, contention).

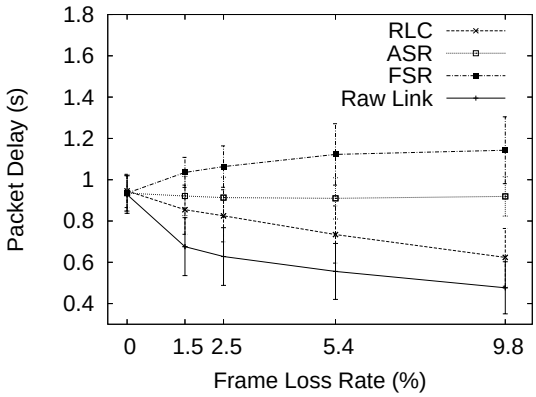

Fig. 13. Media Distribution delay (two state, contention).

REFERENCES

[1] 3rd Generation Partnership Project (3GPP). Radio Link Control (RLC) protocol specification (Release 7). Technical Specification 25.322, V7.0.0, March 2006.

[2] H. Balakrishnan, V.N. Padmanabhan, S. Seshan, and R. H. Katz. A comparison of mechanisms for improving TCP performance over wireless links. In *Proc. of the ACM SIGCOMM*, pages 256–267, August 1996.

[3] R. Bestak, P. Godlewski, and P. Martins. RLC buffer occupancy when using a TCP connection over UMTS. In *Proc. of the IEEE PIMRC*, volume 3, pages 1161–1165, September 2002.

[4] P. T. Brady. Evaluation of multireject, selective reject, and other protocol enhancements. *IEEE Transactions on Communications*, 35(6):659–666, June 1987.

[5] V. Jacobson. Congestion avoidance and control. In *Proc. of the ACM SIGCOMM*, pages 314–329, August 1998.

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

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

[8] S. Vangala and M. Labrador. Shielding TCP from last hop wireless losses. *Wireless Communications and Mobile Computing*, 2007. to appear.

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

[10] G. Xylomenos. Limitations of fixed timers for wireless links. In *Proc. of the Int. Symposium on Parallel and Distributed Processing and Applications*, pages 159–170, 2006.

[11] G. Xylomenos and G. C. Polyzos. A multi-service link layer architecture for the wireless Internet. *International Journal of Communication Systems*, 17(6):553–574, 2004.

[12] G. Xylomenos and C. Tsilopoulos. Adaptive timeout policies for wireless links. In *Proc. of the Int. Conference on Advanced Information Networking and Applications*, volume 1, pages 497–502, 2006.