Multiple Layer Error Control over Wireless Links

George Xylomenos and Giannis Vasalas Kokkinakis
xgeorge@aueb.gr and jvk21gr@gmail.com
Mobile Multimedia Laboratory
Department of Informatics
Athens University of Economics and Business
Patision 76, Athens 104 34, Greece

***Abstract*— In this paper we compare the wireless performance of TCP with or without selective acknowledgments and in the presence or absence of a reliable link layer protocol, in order to determine whether link layer error control remains beneficial for TCP variants with improved error recovery capabilities, such as TCP SACK. We also examine whether there are adverse interactions between the two protocol layers, that is, whether link layer error control degrades the congestion control performance of the transport layer, and whether transport layer error recovery degrades the error control performance of the link layer. Our results show that, even with TCP SACK, link layer error control remains critical for TCP performance over wireless links. Furthermore, our results show that with a TCP unaware link layer protocol we have a clear separation of concerns, whereby the link layer handles wireless losses and the transport layer handles congestion.**

I. Introduction

The error prone nature of wireless links is known to severely degrade the performance of TCP based applications, such as File Transfer and Web Browsing, due to the inability of TCP to distinguish between congestion and wireless losses. Numerous reliable link layer protocols have been proposed as a way to hide the deficiencies of wireless links, so as to allow TCP to concentrate on congestion control. Our previous work indeed shows that TCP unaware reliable link layers offer dramatic performance improvements for the higher layer protocols and applications commonly used on the Internet [13].

As more advanced variants of TCP are deployed on the Internet however, we need to reconsider the performance of the entire protocol stack over wireless links. Specifically, while TCP with the *Selective Acknowledgment* option (TCP SACK) was originally designed to improve TCP recovery from bursty congestion losses, it may also benefit the performance of TCP over wireless links. Considering that TCP SACK is quite similar to the *Selective Repeat* (SR) protocols used by reliable link layers, it is inevitable to ask whether TCP SACK can handle wireless losses by itself, with no aid from a reliable link layer. Furthermore, it is important to examine whether the co-existence of TCP SACK with a reliable link layer has detrimental side effects; it may be the case that TCP SACK retransmits packets already retransmitted by the link layer, or that the link layer obstructs the congestion control mechanisms of TCP SACK.

This paper attempts to answer these questions via extensive simulation results for the performance of two different TCP applications running over TCP with and without the selective acknowledgment option, coupled either with or without link layer error recovery. In Section II we provide background information about various TCP variants, while in Section III we do the same for reliable link layer protocols. Section IV reviews existing work related to multiple layer error control and states the objectives of this study. Section V describes the simulation setup for the performance evaluation that follows. In Section VI we discuss TCP performance when no congestion exists on the end-to-end path, while in Section VII we do the same when congestion exists. We present our conclusions in Section VIII.

II. Transport Layer Error Control

The heart of the Internet, the *Internet Protocol* (IP), offers an unreliable packet delivery service: IP packets may be lost, reordered or duplicated. While some real-time applications use the *User Datagram Protocol* (UDP) for direct access to this service, handling error, flow and congestion control themselves, most applications prefer delegating these tasks to the *Transport Control Protocol* (TCP) which offers a reliable byte stream service. In all TCP variants, the sender buffers the application layer data stream and segments it into IP packets for transmission. The receiver reassembles the original data stream from the arriving packets and passes it strictly in sequence to the application layer. The receiver returns *acknowledgments* (ACKs) to the sender indicating the latest packet received in sequence and the amount of data beyond that packet that may be transmitted without additional ACKs; this is called the *advertised window*.

The TCP sender also maintains a *congestion window* which represents an estimate of the amount of data that may be transmitted, beyond the latest acknowledged packet, without causing congestion to the network. Since IP networks do not provide explicit congestion indications, the TCP sender dynamically calculates a proper size for the congestion window: the window is increased as new ACKs arrive, indicating that packets are reaching the receiver, and decreased when packets are lost, indicating that they may be dropped due to congestion. The window increases slowly, so as to gently probe the capacity of the network, but decreases sharply, so as to quickly relieve network congestion. The exact details of congestion window handling depend on the TCP variant in use [6]. In all TCP variants however, the sender may only transmit new packets if they lie within both the advertised

window and the congestion window, which provide flow and congestion control, respectively.

Early TCP variants only detected losses when a timer expired before an ACK was received for a specific packet, triggerring a retransmission of the lost packet as well as the congestion control mechanism. As the round trip time of a TCP connection over the Internet may fluctuate, timeouts must be conservative so as to avoid redundant retransmissions, making timeout based loss detection quite slow. Newer TCP variants, and in particular the TCP Reno variant used in this paper, also employ a heuristic loss detection method: whenever three duplicate ACKs arrive for the same packet, meaning that after that packet three more packets were received out of sequence, the next packet in sequence is assumed lost and retransmitted, and the congestion control mechanism is triggered. Provided that enough packets are sent after the lost one to trigger the duplicate ACKs, this loss detection method is much faster than waiting for a timeout.

A known limitation of TCP Reno is that when multiple losses occur in neighboring packets, only a single packet can be retransmitted before a new set of three duplicate ACKs is received, thus revealing the identity of the next lost packet. As a result, TCP Reno can either retransmit at most a single lost segment per round trip time or risk retransmitting packets that have already been received [6]. TCP SACK attempts to solve this problem by including in each ACK information for up to three consecutive blocks of data that have been received beyond the last packet acknowledged. This allows the sender to accurately determine the identities of multiple lost packets and retransmit them within a single round trip time, without the risk of wasting bandwidth. This is useful for both congestion and wireless losses that occur in bursts: in an analytical study, TCP SACK has been found to outperform other TCP variants over wireless links without a reliable link layer [2].

III. Link Layer Error Control

The basic limitation of TCP over wireless links is that all losses are interpreted as congestion, as TCP has no way of distinguishing congestion and wireless losses. Due to the fact that the TCP congestion window increases slowly but decreases rapidly, mistaking wireless errors for congestion leads to very degraded TCP performance [14]. While many researchers have proposed TCP modifications that better handle wireless losses, none of these has found widespread acceptance as they generally require modifications to end hosts throughout the Internet and they often interfere with the end-to-end semantics of TCP.

The alternative to modifying TCP is to employ a reliable link layer protocol over the wireless link, so as to hide wireless errors from TCP and thus prevent triggerring congestion control. An early proposal is the *Snoop* protocol, a link layer agent that examines the packets of each TCP stream at the access point bridging the wired and wireless parts of the path. Snoop buffers outgoing packets and retransmits them whenever three duplicate ACKs are received, exactly like TCP; in order to avoid triggerring retransmissions at the TCP sender, Snoop hides these duplicate ACKs from the sender [3]. This TCP awareness has the disadvantage of incompatibility with encrypted IP payloads. In addition, TCP awareness means that a scheme may need modifications as new TCP variants are introduced.

In our previous work we have found that TCP performance can be greatly enhanced with TCP unaware reliable link layer protocols, such as *Selective Repeat* (SR) [13]. In SR, the sender buffers and transmits link layer frames in sequence within a window of N frames. The receiver also accepts frames within a window 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 reception of all frames up to the one delivered. 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 NACKed frames. 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 covering all delivered frames. Since ACKs and/or NACKs may also be lost, the sender starts a retransmission timer after sending each frame; if it expires before an ACK arrives for that frame, the frame is assumed lost and retransmitted. Many SR variants exist, mostly differing on NACK handling [4]. In this paper we use a *multireject* SR variant, that is, each missing frame can be NACKed many times.

While SR and TCP SACK are quite similar in many aspects, their different operating environments have a significant impact on their loss recovery capabilities. TCP SACK operates over an end-to-end path where the round trip delay varies, therefore its timeouts must be conservative, and where packets may be reordered, therefore gaps in the packet sequence do not necessarily imply losses. In contrast, SR operates over a single link where the round trip delay is fixed, therefore its timeouts can be tight, and where frames cannot be reordered, therefore gaps in the frame sequence imply losses, so NACKs can be used.

IV. Related Work and Objectives

While numerous papers evaluate link layer and transport layer approaches for improving TCP performance over wireless links, only a handful deal with combined error recovery at both layers. Early work indicated that the link layer could potentially adversely interact with the transport layer, by locally retransmitting packets that were already retransmitted end-to-end, thus wasting bandwidth and reducing throughput [5]. Later work shows that as the link layer operates at a finer time scale than the transport layer, it may retransmit a packet many times before the transport layer manages to do so [13]. An analytical study found that even a stop and wait link layer scheme may greatly improve the performance of many TCP variants [1].

Another approach is to employ a TCP oriented *Forward Error Correction* (FEC) scheme at the link layer [7], whereby the endpoints of a wireless link combine the error statistics of the link with a model of TCP throughput in order to determine

how much FEC overhead to apply to each transmitted frame: higher overhead means increased error recovery but also decreased available bandwidth, thus the scheme calculates the optimal operating point. This scheme improves the performance of TCP SACK [7] with limited TCP awareness.

Interestingly, another study shows that the Snoop protocol can adversely interact with TCP SACK [11]. The first reason for this behavior is that when duplicate ACKs arrive after a loss burst, Snoop can only retransmit a single packet, as it is unaware of the TCP selective acknowledgment option, and it also drops some of these ACKs, thus preventing a TCP SACK sender from exploiting their information. The second reason is that even when a duplicate ACK is not dropped by Snoop, if it triggers multiple retransmissions from a TCP SACK sender, Snoop may drop the retransmitted packets to avoid wasting bandwidth, as it is not yet aware that these packets have been lost. While it is trivial to modify Snoop to properly handle, and indeed benefit from, TCP SACK acknowledgments, these problems indicate a fundamental limitation of TCP aware schemes: a change in TCP may conflict with a TCP aware link layer.

Due to the problems of TCP aware schemes, in this paper we focus on the interplay between a regular SR link layer protocol and TCP SACK. Unlike previous work in this area, we consider both wireless losses and congestion over the wired link. We present simulation results showing the performance of TCP Reno and TCP SACK over either a Raw link layer, that is, without error recovery, or a reliable link layer using SR, in order to answer the following questions:

1) Can TCP SACK handle wireless error recovery without assistance from the link layer?
2) Does SR at the link layer obstruct the congestion recovery performance of TCP SACK?
3) Does TCP SACK obstruct the error recovery performance of SR at the link layer?

In order to come up with conclusive results, we performed experiments over random and bursty wireless error models, local and wide area network topologies and bulk transfer or interactive TCP applications, so as to determine which aspects of TCP performance over wireless links are specifically related to the interaction between link layer and transport layer error control.

V. Simulation Setup

The performance results reported below are based on simulations with ns-2 [10] (version 2.30), extended with additional error models, link layers and applications [12]. Each test was repeated 30 times with different random seeds and the metrics shown reflect average values from these 30 runs, 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. The results given in the following figures and tables are for a LAN topology where the wired link has a bandwidth of 10 Mbps and a propagation delay of 1 ms. We also discuss, but do not present due to space limitations, results from a WAN topology where the wired link has a bandwidth of 2 Mbps and a propagation delay of 50 ms.

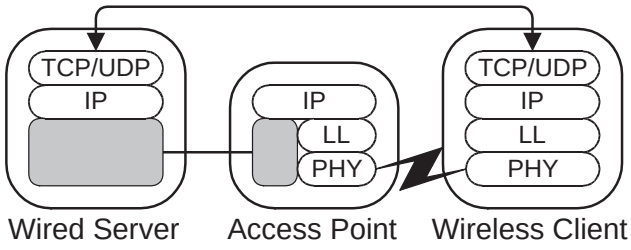

Fig. 1. Simulated network topology

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. To avoid packet fragmentation, the applications also use 250 byte 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 have experimentally 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%, matching the FLRs used for the Uniform model. Note that the error processes in each link direction are identical but independent. To establish a performance baseline, we also show results with no errors.

In order to evaluate TCP performance, we used two very different applications. In File Transfer the Wired Server simply sends a large file as fast as possible to the Wireless Client, with TCP handling flow and congestion control. While longer transfers produce more stable results, in practice users do not initiate huge transfers, therefore we used 10 MByte files as a compromise. We measured File Transfer throughput, defined as the amount of *application* data transferred divided by time taken.

In Web Browsing the Wireless Client accesses *pages* containing text, links to other pages and embedded objects, stored at the Wired Server. The client-server interaction consists of a sequence 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 next transaction begins when the client requests another page. The ns-2 HTTP module provides empirical distributions for request, page and embedded object sizes, as well as for the number of objects per page [8]. Only one transaction is in progress at any time with no pauses between transactions. We measured Web Browsing throughput, defined as the amount of *application* data transferred from the server to the client divided by time taken. Note that client requests only influence throughput indirectly, by introducing delays. The results given reflect the state at the end of the last *completed* transaction during the simulated period, which was 2000 s.

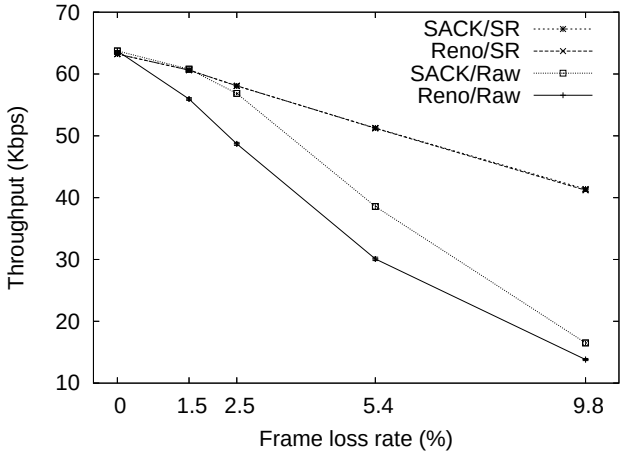

Fig. 2. File Transfer Throughput (Uniform, LAN)

While Web Browsing seems to be nothing more than a set of File Transfers, the two applications behave very differently. File Transfer always has data to send, therefore TCP can eventually reach its peak throughput, while Web Browsing consists of small transfers that rarely last long enough for TCP to achieve high throughputs. Similarly, while in File Transfer the server sends data continuously, in Web Browsing a client request must complete before initiating a server response and vice versa, therefore the server may be idle for extended periods.

Application performance was measured both with and without congestion so as to separately examine how each transport /link layer protocol combination handles wireless errors and congestion. Congestion was introduced by an on-off source transmitting 250 byte packets over UDP from the Wired Server to the Access Point at a high speed. As a result, the wired link suffered only from congestion, while the wireless link suffered only from wireless errors. The on-off source alternated between two states with exponential durations, with an average on duration of 1 s and an average off duration of 1.35 s [9]. During the on state, packets were transmitted isochronously at a rate of 9.99 Mbps for the LAN topology and 1.99 Mbps for the WAN topology, leaving only 10 Kbps to TCP; during the off state, TCP was limited by the 64 Kbps bandwidth of the wireless link.

VI. Performance without Congestion

In this section we will examine the performance of File Transfer and Web Browsing when there is no congestion over the end-to-end path; in this case, each TCP application operates without contention for the wired link. The goal of this section is to study in isolation the error control behavior of each transport / link layer protocol combination under various scenarios by eliminating the need for congestion control.

Starting with File Transfer, Figure 2 shows the throughput achieved by TCP Reno and TCP SACK over either a Raw or an SR link layer in the LAN scenario with Uniform errors. Table I shows the same data in tabular form: for each FLR, the baseline performance of TCP Reno over a Raw link layer is given as an absolute number, while the performance of the enhanced protocol combinations is given as a factor of improvement over the baseline. In this scenario, without link layer error control TCP SACK provides an improvement of 8.7 to 28.2% over TCP Reno, showing that its improved loss recovery is useful even for wireless links. When SR is added at the link layer however, not only the performance of both TCP Reno and TCP SACK are much better, with improvements of 8.3 to 199.6%, but the two TCP variants perform nearly identically.

TABLE I
File Transfer Throughput (Uniform, LAN)

FLR (%)	Reno/Raw (Kbps)	Improvement over Reno/Raw (%) SACK/Raw	Reno/SR	SACK/SR
1.5	55.95	8.7%	8.3%	8.4%
2.5	48.70	16.7%	19.3%	19.2%
5.4	30.09	28.2%	70.2%	70.4%
9.8	13.81	19.4%	198.3%	199.6%

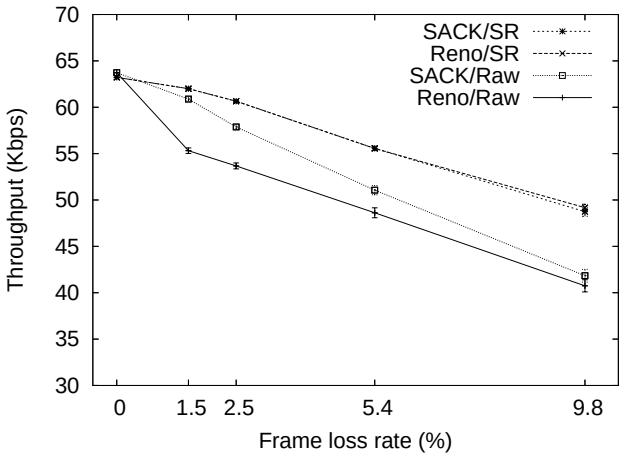

Fig. 3. File Transfer Throughput (Two State, LAN)

Figure 3 and Table II show the corresponding results with the Two State error model, which are quite similar: while TCP SACK provides an improvement of 2.7 to 10.1% over TCP Reno, the addition of SR not only provides an improvement of 12.1 to 20.7%, it also makes TCP Reno and TCP SACK nearly indistinguishable. The results from the WAN scenarios (not shown) differ from the LAN results in one aspect only: the performance improvements due to TCP SACK by itself are relatively smaller. This is reasonable, as all TCP variants can only recover from errors via end-to-end retransmissions, leading to performance degradations when the propagation delay of the path is increased. In contrast, link layer error recovery is not affected by end-to-end path characteristics, therefore it becomes even more beneficial with increasing path delays.

Turning now to Web Browsing, Figure 4 and Table III show the throughput achieved by TCP Reno and TCP SACK over either a Raw or an SR link layer in the LAN scenario with Uniform errors. Without link layer error recovery TCP SACK only has a slight advantage, if any, over TCP Reno:

TABLE II
FILE TRANSFER THROUGHPUT (TWO STATE, LAN)

FLR (%)	Reno/Raw (Kbps)	Improvement over Reno/Raw (%) SACK/Raw	Reno/SR	SACK/SR
1.5	55.32	10.1%	12.1%	12.1%
2.5	53.67	7.8%	13.0%	13.0%
5.4	48.62	5.0%	14.3%	14.3%
9.8	40.74	2.7%	20.7%	19.6%

TABLE III
WEB BROWSING THROUGHPUT (UNIFORM, LAN)

FLR (%)	Reno/Raw (Kbps)	Improvement over Reno/Raw (%) SACK/Raw	Reno/SR	SACK/SR
1.5	36.09	2.1%	18.2%	18.4%
2.5	30.75	3.5%	29.3%	29.7%
5.4	18.28	8.9%	71.6%	69.7%
9.8	8.53	6.5%	149.1%	146.2%

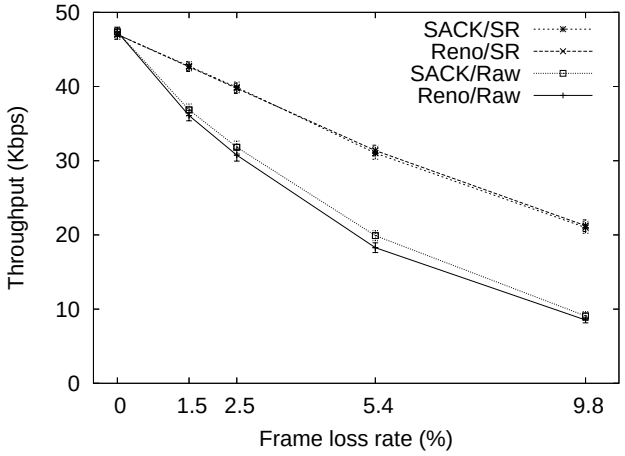

Fig. 4. Web Browsing Throughput (Uniform, LAN)

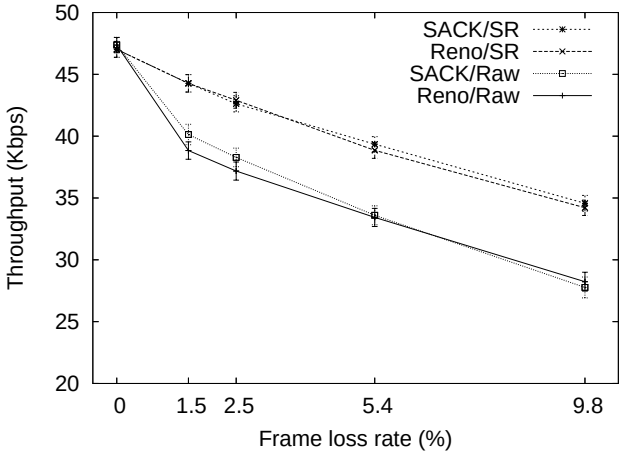

Fig. 5. Web Browsing Throughput (Two State, LAN)

the improvement is 2.1 to 8.9%. In contrast, when SR is introduced performance improves by 18.2 to 149.1%, factors comparable to those with File Transfer; also as in File Transfer, with SR both TCP variants perform nearly the same. The reason that TCP SACK by itself is not as beneficial with Web Browsing is that this application mostly consists of short transfers, therefore after a loss it is often the case that not enough packets follow to trigger the three duplicate ACKs required by the TCP sender to detect a loss, thus making TCP resort to timeout initiated recovery; this is a problem affecting all TCP variants, but not SR which uses short timers optimized for the underlying link.

The same situation appears in Figure 5 and Table IV for the Two State error model: without link layer error recovery TCP SACK is slightly better (0.5 to 3.3%) or worse (1.6%) than TCP Reno; adding SR leads to performance gains of 14.0% to 22.5%, again comparable to those with File Transfer, as well as nearly indistinguishable performance for the two TCP variants. The results from the WAN scenarios (not shown) differ from the LAN results in one aspect only: the performance of all protocol combinations is relatively reduced in the WAN case, in contrast to File Transfer where LAN and WAN performance results are similar. The reason is that with File Transfer data are sent in one direction only, therefore the increased propagation delay of the WAN path is mostly hidden by the pipelined transmissions. In contrast, with Web Browsing the direction of transfer continuously switches from client to server and vice versa, and as each transfer must complete before the next one begins, the pipeline is emptied, therefore the application runs relatively slower.

VII. PERFORMANCE WITH CONGESTION

In this section we will examine the performance of File Transfer and Web Browsing when there is congestion over the wired part of the path; in this case, each TCP application contents for the wired link with the UDP on-off source. The goal of this section is to study the differences induced in the behavior of each transport / link layer protocol combination by the alternation between congested and uncongested periods.

Starting again with File Transfer, Figure 6 and Table V show the throughput achieved by TCP Reno and TCP SACK over either a Raw or an SR link layer in the LAN scenario with Uniform errors. The main differentiation with the corresponding uncongested scenario is that with congestion TCP SACK always provides an improvement over TCP Reno, regardless of the underlying link layer protocol. When SR is used at the link layer, the difference between TCP Reno and TCP SACK is nearly constant, even with no wireless losses, therefore it can be attributed to the improved congestion control mechanisms of TCP SACK. When the Raw link layer is used, TCP SACK outperforms TCP Reno by 10.1 to 25.2% due to its improved handling of both congestion and wireless losses. By adding SR, TCP SACK improves performance by 12.8 to 176.8%.

Figure 7 and Table VI show the corresponding results with the Two State error model, which are quite similar: with SR there is a nearly constant gap between TCP Reno and TCP SACK, reflecting the improved congestion control of TCP SACK, while without SR the larger gap between the two TCP variants reflects the improved congestion control and wireless loss handling of TCP SACK. TCP SACK without SR improves upon TCP Reno by 6.3 to 13.0%, but when SR

TABLE IV
WEB BROWSING THROUGHPUT (TWO STATE, LAN)

FLR (%)	Reno/Raw (Kbps)	Improvement over Reno/Raw (%) SACK/Raw	Reno/SR	SACK/SR
1.5	38.84	3.3%	14.0%	14.0%
2.5	37.18	3.0%	15.4%	14.6%
5.4	33.43	0.5%	16.2%	17.7%
9.8	28.23	-1.6%	21.2%	22.5%

TABLE V
FILE TRANSFER THROUGHPUT (UNIFORM, LAN)

FLR (%)	Reno/Raw (Kbps)	Improvement over Reno/Raw (%) SACK/Raw	Reno/SR	SACK/SR
1.5	50.54	10.1%	7.1%	12.8%
2.5	44.84	15.2%	15.0%	20.9%
5.4	29.05	25.2%	52.9%	60.5%
9.8	13.72	18.8%	161.8%	176.8%

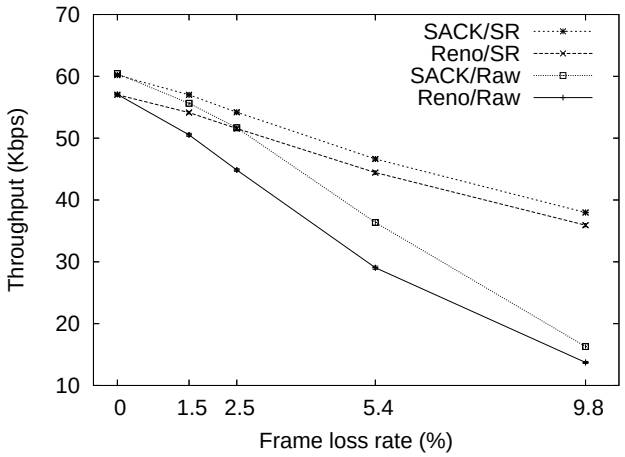

Fig. 6. File Transfer Throughput (Uniform, LAN)

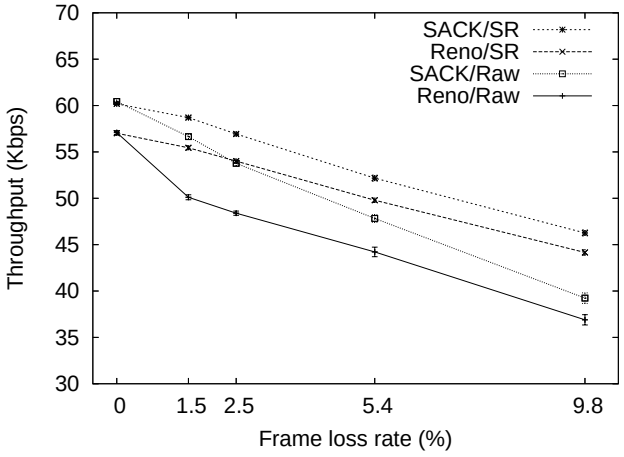

Fig. 7. File Transfer Throughput (Two State, LAN)

is also introduced, the performance with TCP SACK improves by 17.2 to 25.4%. The results from the WAN scenarios (not shown) differ from the LAN results in two ways: first, the gap between TCP SACK and TCP Reno when SR is used is larger, since with higher propagation delays the importance of improved congestion recovery is more pronounced; second, the performance improvements in both TCP variants due to the introduction of SR are even more dramatic, since link layer error control becomes more important with higher end-to-end propagation delays.

Turning now to Web Browsing, Figure 8 and Table VII show the throughput achieved by TCP Reno and TCP SACK over either a Raw or an SR link layer in the LAN scenario with Uniform errors. In this application TCP SACK only provides a slight improvement over TCP Reno without SR, which is even smaller with SR; in contrast, the introduction of SR greatly improves the performance of both TCP variants. With the Raw link layer, TCP SACK outperforms TCP Reno by 3.0 to 12.4%, while with SR the improvements due to TCP SACK are 18.2 to 144.1%. As explained in the uncongested case, the relatively small gains of TCP SACK over TCP Reno are due to the short transfers of Web Browsing that often prevent the triggerring of TCP retransmissions by duplicate ACKs; this obstructs both congestion and error control. Another factor diminishing the congestion control gains of TCP SACK is that Web Browsing is bidirectional, therefore it faces congestion only in the server to client direction, unlike File Transfer which is unidirectional, thus continuously facing congestion.

A similar situation appears in Figure 9 and Table VIII for the Two State error model: with SR at the link layer, TCP SACK is slightly better than TCP Reno, while without SR at the link layer TCP SACK has a slightly bigger advantage. Without SR the gains of TCP SACK over TCP Reno are 2.6 to 4.4%, while when SR is introduced the TCP SACK gains are 13.4 to 24.6%. The relatively smaller performance improvements compared to the Uniform error model are consistent with those experienced by the File Transfer application, while the overall behavior is consistent with that of Web Browsing with Uniform errors. Finally, the results from the WAN scenarios (not shown) are very similar, with the only visible difference being an overall reduction in Web Browsing throughput, due to the higher end-to-end propagation delay compared to the LAN scenario.

VIII. CONCLUSIONS

We have presented results from a comprehensive set of simulations of TCP Reno and TCP SACK with or without SR at the link layer, using different applications, error models and network topologies. We summarize our findings as follows:

- Without congestion and without SR at the link layer, TCP SACK provides a performance improvement over TCP Reno, but only for File Transfers. When SR is used at the link layer, TCP SACK performs the same as TCP Reno.
- With congestion TCP SACK provides a performance improvement over TCP Reno, higher for File Transfers and lower for Web Browsing. When SR is used at the link layer, this advantage is relatively independent of the FLR.

TABLE VI
FILE TRANSFER THROUGHPUT (TWO STATE, LAN)

FLR (%)	Reno/Raw (Kbps)	Improvement over Reno/Raw (%) SACK/Raw	Reno/SR	SACK/SR
1.5	50.11	13.0%	10.7%	17.2%
2.5	48.39	11.1%	11.6%	17.6%
5.4	44.22	8.2%	12.6%	18.0%
9.8	36.90	6.3%	19.7%	25.4%

TABLE VII
WEB BROWSING THROUGHPUT (UNIFORM, LAN)

FLR (%)	Reno/Raw (Kbps)	Improvement over Reno/Raw (%) SACK/Raw	Reno/SR	SACK/SR
1.5	34.31	3.0%	16.1%	18.2%
2.5	29.60	3.2%	25.6%	27.0%
5.4	18.34	7.4%	60.1%	62.6%
9.8	8.44	12.4%	141.1%	144.1%

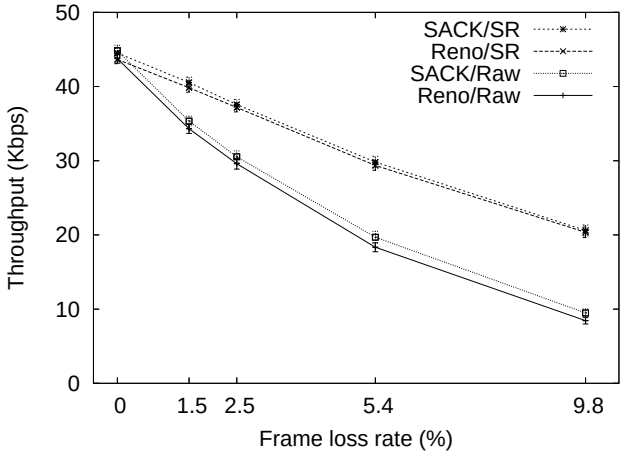

Fig. 8. Web Browsing Throughput (Uniform, LAN)

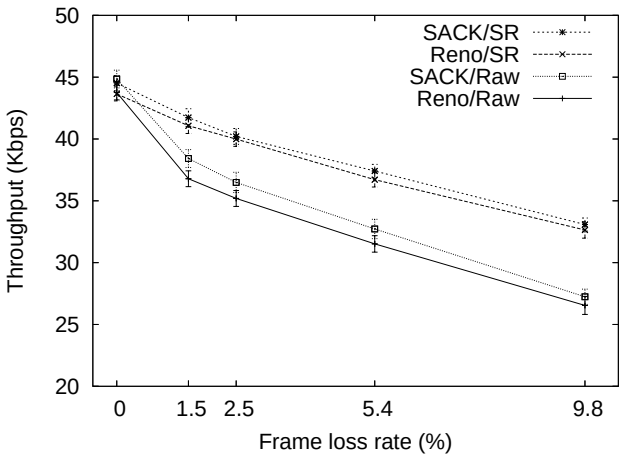

Fig. 9. Web Browsing Throughput (Two State, LAN)

- The use of SR at the link layer provides a considerable performance improvement for both TCP variants, regardless of the application in use. These performance gains are much higher than those due to TCP SACK by itself.
- The only difference between LAN and WAN results is that in the WAN scenarios Web Browsing performance is lower for all transport / link layer protocol combinations, due to the increased path propagation delay. In File Transfer this delay is mostly hidden by pipelining.

Based on these findings, we can now provide answers to the questions posed in Section IV that are independent of the network topology, wireless error model and application used:

1) TCP SACK does *not* fully handle wireless error recovery. While it manages to improve upon TCP Reno without link layer error recovery, this improvement only appears with File Transfer and not Web Browsing. In contrast, when SR is used at the link layer, the performance of both TCP variants is improved much more than with TCP SACK by itself, therefore link layer error recovery remains essential for performance even with TCP SACK.
2) SR at the link layer does *not* obstruct the congestion recovery performance of TCP SACK. Whether with or without SR at the link layer, TCP SACK outperforms TCP Reno when congestion exists; the gains may be major, as is the case with File Transfer, or minor, as is the case with Web Browsing. Therefore, TCP SACK provides improves congestion control performance even when link layer error recovery is used.
3) TCP SACK does *not* obstruct the error recovery performance of SR at the link layer. The introduction of SR at the link layer leads to considerable performance improvements for both TCP variants, with or without congestion; without congestion, SR makes TCP SACK and TCP Reno indistinguishable. Therefore, SR improves wireless error recovery even when TCP SACK is used.

We therefore conclude that the use of SR at the link layer is as beneficial for TCP SACK as it has been found to be for TCP Reno [13]. Furthermore, it appears that SR at the link layer and TCP SACK at the transport layer offer largely orthogonal improvements, with SR handling wireless losses and TCP SACK handling congestion.

REFERENCES

[1] F. Anjum and R. Jain. Performance of TCP over lossy upstream and downstream links with link-level retransmissions. In *Proc. of the International Conference on Networks (ICON)*, pp. 3–7, 2000.
[2] F. Anjum, L. Tassiulas. Comparative study of various TCP versions over a wireless link with correlated losses. *IEEE/ACM Trans. on Networking*, 11(3):370–383, June 2003.
[3] H. Balakrishnan, V.N. Padmanabhan, S. Seshan, and R.H. Katz. A comparison of mechanisms for improving TCP performance over wireless links. *IEEE/ACM Trans. on Networking*, 5(6):756–769, June 1997.
[4] P.T. Brady. Evaluation of multireject, selective reject, and other protocol enhancements. *IEEE Trans. on Communications*, 35(6):659–666, June 1987.
[5] A. DeSimone, M.C. Chuah and O.C Yue. Throughput performance of transport layer protocols over wireless LANs. In *Proc. of the Global Telecommunications Conference (GLOBECOM)*, vol. 1, pp. 542–549, 1993.
[6] S. Floyd, K. Fall. Simulation based comparisons of Tahoe, Reno and SACK TCP. *Computer Communications Review*, 26(3):5–21, July 1996.

TABLE VIII

WEB BROWSING THROUGHPUT (TWO STATE, LAN)

FLR (%)	Reno/Raw (Kbps)	Improvement over Reno/Raw (%)		
		SACK/Raw	Reno/SR	SACK/SR
1.5	36.79	4.4%	11.7%	13.4%
2.5	35.19	3.7%	13.6%	14.2%
5.4	31.52	3.8%	16.5%	18.7%
9.8	26.54	2.6%	23.0%	24.6%

[7] B. Liu, D.L. Goeckel and D. Towsley. TCP cognizant adaptive forward error correction in wireless networks. In *Proc. of the Global Telecommunications Conference (GLOBECOM)*, vol. 3, pp. 2128–2132, 2002.

[8] B.A. Mah. An empirical model of HTTP network traffic. In *Proc. of the IEEE INFOCOM '97*, pp. 592–600, 1997.

[9] S. Nanda, D.J. Goodman, and U. Timor. Performance of PRMA: a packet voice protocol for cellular systems. *IEEE Trans. on Vehicular Technology*, 40(3):584–598, 1991.

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

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

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

[13] 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.

[14] G. Xylomenos, G.C. Polyzos, P. Mahonen and M. Saaranen. TCP Performance Issues over Wireless Links. *IEEE Communications Magazine*, 39(4):52–58, April 2001.