Reinforcement of TCP Error Recovery for Wireless Communication

Nihal K. G. Samaraweera and Godred Fairhurst
Electronics Research Group, Department of Engineering
University of Aberdeen, Aberdeen, AB24 3UE, UK.
nihal, gorry@erg.abdn.ac.uk

Abstract

When a wireless link forms a part of a network, the rate of packet loss due to link noise may be considerably higher than observed in a modern terrestrial network. This paper studies TCP performance over a range of link environments and highlights the advantage of recent modifications to TCP (e.g. SACK, New-Reno) for wireless communications. It also identifies two key issues which impact the performance of TCP over error prone links: TCP's reliance on timers to recover from a failed retransmission cycle, and TCP's inability to separate congestion packet loss from other types of packet loss. A solution to the first issue is identified and analysed by simulation, and the factors affecting the second issue are outlined.

1 Introduction

TCP [1] has been improved to provide efficient and reliable operation over large networks which suffer potential packet loss. The most important modification was the introduction of congestion control and avoidance techniques using the principle of self clocking. A back-off procedure (Slow Start) was used, which on detection of congestion, 'drains the pipe' (i.e., waits until all transmitted packets have left the network) before transmission of more data [2]. The back-off procedure achieves network stability but is also unduly conservative. The Fast Retransmission and Recovery algorithms [3, 4] have therefore been introduced to drain only a half of the pipe and then recommence transmission, assuming the reception of each duplicate ACK is an indication of a packet leaving the network.

TCP performs poorly when these algorithms operate over a network (or link) with a high rate of packet loss. This is due to their inability to recover multiple packet loss without waiting for a retransmission time out and subsequent Slow Start. TCP performance may be improved when multiple packets are dropped from a window of data by using partial ACKs during the retransmission phase (acknowledging some, but not all, packets outstanding at the start of fast-retransmission) [4] and the SACK option [5, 6] which has been recently defined. This paper provides new data for the performance of the SACK option over wireless links.

When a retransmitted TCP packet is lost (i.e., retransmission fails) most implementations do not have a mechanism to recover the packet without waiting for a retransmission time out and subsequent Slow Start. Such a mechanism may not be important when the dominant form of packet loss is congestion. However, a degradation in the propagation conditions of a wireless link leads to a high Bit Error Rate (BER) resulting in frequent (random) packet loss. This increases the probability of retransmission packet loss (or increases the number of retransmissions required for successful transmission of a packet, which is a function of the bit error rate [7]). This paper therefore proposes a technique which detects the loss of a retransmitted packet and retransmits it efficiently without waiting for the expiry of the retransmission timer.

All current TCP implementations assume that a packet loss is an indication of network congestion and take measures to avoid further congestion in the network by reducing the transmission rate (i.e., reducing window size). This is based on strong evidence that congestion is the main issue in most (terrestrial) networks. Non-congestion packet losses due to link errors are more significant when a wireless link forms a part of a network. The current congestion avoidance algorithm results in very poor performance of TCP error recovery over wireless links [8, 9]. In the case of link errors, the transmitter should, however, persist in utilising a large proportion of the bandwidth to make optimum use of the error-prone link. The importance of distinguishing between these two types of packet losses is discussed and key issues are outlined.

2 TCP Implementations

This section describes congestion control and avoidance techniques, highlighting the current advancements which are particularly important for wireless communications.

2.1 Tahoe TCP

The Slow Start (congestion control) and multiplicative decrease (congestion avoidance) procedures [2] were first implemented in Tahoe TCP. The Fast Retransmission [3, 10] algorithm was also implemented in Tahoe TCP to avoid (inefficient) waiting for the retransmission timer to expire following every packet loss. In this algorithm, a receiver sends an (duplicate) ACK immediately on reception of each out of sequence packet. The transmitter interprets reception of 3 duplicate ACKs (sufficient to avoid spurious retransmissions due to reordering of packets) as a congestion packet loss and performs the Slow Start algorithm.

2.2 Reno TCP

The Reno TCP implementation introduces the Fast Recovery algorithm [3, 10]. When the third duplicate ACK is received, the Reno TCP transmitter sets the slow start threshold size (*ssthresh*) to one half of the current congestion window (*cwnd*) and retransmits the missing packet. The *cwnd* is then set to *ssthresh* plus 3 times the segment size (one per each duplicate ACK). *cwnd* is increased by one segment on reception of each duplicate ACK which continues to arrive after fast-retransmission. This allows the transmitter to send new data when *cwnd* is increased beyond the value of the *cwnd* before the fast-retransmission. When an ACK arrives which acknowledges all outstanding data sent before the duplicate ACKs were received, the *cwnd* is set to *ssthresh* so that the transmitter slows down the transmission rate and enters the linear increase phase.

2.3 New-Reno

If two or more packets have been lost from the transmitted data (window), the Fast Retransmission and Fast Recovery algorithms will not be able to recover the losses without waiting for retransmission time out. Hoe proposed a modification to Reno TCP usually called New-Reno [4] to overcome this problem. New-Reno introduces the concept of a Fast Retransmission Phase, which starts on detection of a packet loss (receiving 3 duplicate ACKs) and ends when the receiver acknowledges reception of all data transmitted at the start of the Fast Retransmission phase.

The transmitter assumes reception of a partial ACK during the Fast Retransmission phase as an indication that another packet has been lost within the window and retransmits that packet immediately to prevent expiry of the retransmission timer. The implementation proposed by Hoe sets the *cwnd* to one segment on reception of 3 duplicate ACKs (i.e. when entering the Fast Retransmission Phase) and unacknowledged data are retransmitted using the Slow Start algorithm. The transmitter is also allowed to transmit a new data packet on receiving 2 duplicate ACKs. While the transmitter is in the Fast Retransmission Phase, it continues to retransmit packets using Slow Start until all packets have been recovered (without starting a new retransmission phase for partial ACKs). Although this modification may cause unnecessary retransmissions, it avoids unnecessary transmitter time outs and efficiently recovers multiple packet loss using partial ACKs [4].

We have written a full implementation of TCP within a simulated environment [7, 11]. Our implementation of New-Reno is slightly different to the implementation proposed by Hoe. A new state variable has been defined to save the window size for retransmission (*cwnd_rexmit*), allowing new data to be transmitted using the Reno implementation. The sender starts fast retransmission and sets the *cwnd_rexmit* to one segment on reception of 3 duplicate ACKs. *cwnd_rexmit* is increased by one segment on reception of each subsequent new (partial) ACK, and unacknowledged packets are retransmitted, but only if it is allowed by the Slow Start procedure. The sender does not start a new retransmission phase for partial ACKs until all packets have been recovered [4]. The other variables (*cwnd*, *ssthresh*, etc.) are updated using the Reno Fast Retransmission and Recovery algorithms.

2.4 Selective Acknowledgment Option

The SACK option for Reno TCP has been introduced [5] to further enhance TCP performance [6] by allowing (selective) acknowledgment of packets held at the receiver. When the receiver buffer holds in-sequence data packets, the receiver sends duplicate ACKs bearing the SACK option to inform the transmitter which packets have been correctly received [5]. A SACK option field contains a number of SACK blocks (the first block in the SACK option reports the most recently received in-sequence packet). When the third duplicate ACK is received, a SACK TCP transmitter retransmits the packets starting with the sequence number acknowledged by the duplicate ACKs followed by subsequent unacknowledged packets. The New-Reno Fast Retransmission and Recovery algorithms are used for retransmission. These algorithms are modified to avoid retransmitting already SACKed packets. Unnecessary retransmissions caused by the New-Reno modification may be reduced by this selective retransmission of lost packets.

The transmitter is also able to accurately estimate the number of transmitted packets that have left the network [6, 12] by using the explicit information carried by selective ACKs. Mathis proposed an implementation which stores the forward most SACKed sequence number (i.e., the highest sequence number acknowledged by the SACK options) in a new variable called *snd_fack* [12]. The pipe size is accurately calculated using *snd_fack* and another variable called *return_data* which is updated when a packet is retransmitted and when a retransmitted packet is determined to have left the network. This provides efficient retransmission of packets lost from the original data stream.

One potential problem arises when packets are reordered after the initial three duplicate ACKs which initiate the retransmission phase. The subsequent SACKs (indicating holes in the sequence space) may result in unnecessary retransmission [13]. One potential solution is to count the number of SACKs which indicate each missing packet and back-off until receiving a minimum number of SACKs (similar to Reno duplicate ACKs). However, the receiver may not generate sufficient SACKs for this, since the current congestion window restricts the number of new data packets which may be transmitted (*cwnd* is usually small when the packet loss rate is high).

Our implementation addresses this problem by using the new state variable *(cwnd_rexmit*) which is updated using the

slow start algorithm to control the retransmission of lost packets; new data are still transmitted using the variable *cwnd* (updated by the fast retransmission and recovery algorithms, as explained in section 2.3). For example, after retransmission of the first lost packet the 2nd and 3rd lost packets are postponed until an ACK is received. This procedure therefore allows reordered packets to be delayed by approximately one round trip time, and reduces the level of unnecessary retransmissions. Furthermore, since the number of packets in transit is constrained by the *cwnd*, there is only a small possibility of new packets arriving at the receiver after a round trip delay (except packets both delayed and reordered by the network).

This procedure does not compromise the benefit from using the SACK option, since (a) the transmitter is allowed to transmit new data using the fast retransmission procedure (*cwnd* variable and SACKed information are used for transmission of new data), and (b) *cwnd_rexmit* will grow exponentially if the retransmissions are successful. However, this procedure is conservative since the sender is allowed to retransmit only a single packet during the first round trip delay even when the sender determines that more than one packet has left the network. This conservative approach follows that in fast retransmission and recovery. However, SACK may also provide indication of later packets leaving the network (e.g. using the *snd_fack* variable), which may permit retransmission of further segments, if the sender is allowed by the current calculated pipe size and the congestion window. Implementation of this refinement must also consider the possibility of a missing packet still been in transit on a higher delay network path, in which case the retransmission may potentially add to any network congestion.

3 Retransmission Packet Loss Detection

Conventional implementations of TCP do not provide efficient retransmission of lost *retransmitted* packets. Congestion losses normally occur in bursts [14], but retransmission loss is not a major concern because the retransmitted packets are usually successfully received (mainly due to the transmission rate being reduced following a congestion loss). The probability of loss over a wireless link depends only on the packet size and the link BER. Therefore retransmission loss is just as probable as original loss and needs to be considered. We introduce the Retransmission Packet Loss Detection algorithm which identifies loss of retransmitted packets and recovers them efficiently *without the need for transmitter time outs*. The new algorithm works by correlating the time order of the transmitted packets and received acknowledgments. When a packet has been successfully retransmitted (i.e., the receiver acknowledges the packet), this algorithm will not be invoked. However, if the receiver does not acknowledge the packet and instead acknowledges (SACK) a subsequently transmitted packet, the transmitter considers that this is a good indication of loss of the retransmitted packet. The algorithm therefore relies on the presence of the SACK option.

This mechanism is implemented by modifying the Reno algorithms. During the Fast Retransmission phase, the transmitter records the first sequence number of a retransmitted packet and the transmission time. The transmitter also records the first sequence number of each subsequent packet, the corresponding transmission time, and marks it as either a retransmission or new data.

On reception of an ACK which covers a recorded sequence number, the corresponding records are cleared. The transmitter detects loss by receiving 3 duplicate ACKs. This prevents any mis-ordering of packets triggering the algorithm (a similar procedure is used in the Fast Retransmission algorithm).

On reception of 3 duplicate selective ACKs which cover one of the records (tagged as new data), the transmitter compares the transmission time of this record with the transmission time of the other records (which were tagged as retransmitted packets) to check whether the selectively acknowledged packet was transmitted after any retransmitted packets (i.e., the algorithm considers not just the packet at the low window edge, but any SACKed packet). If this happens, the transmitter deduces that the retransmitted packets were lost and retransmits them using the New-Reno Fast Retransmission and Recovery algorithms. The time stamp entries are updated on retransmission, allowing further retransmission losses to be detected if this retransmission should also fail.

RPLD may therefore efficiently recover from even high rates of packet loss without reliance on time out retransmission (performance data is presented in section 5). If the retransmission timer does expire, the transmitter also clears all recorded entries to avoid the ambiguity which may arise due to time-out retransmission. The RPLD algorithm may also be implemented using a list of packets sorted in the transmission order instead of time stamps.

The separation of the retransmission processes using the *cwnd_rexmit* variable allows the RPLD algorithm to achieve conservative behaviour. On the start of the retransmission phase, the transmitter is allowed to retransmit only a small number of segments (one in the protocol described). *cwnd_rexmit* will not grow during the retransmission phase if this retransmitted packet has not been ACKed. Since the *cwnd* is also reduced by the Fast Retransmission and Fast Recovery procedures, on detection of each packet loss, the rate of retransmission of retransmitted packets is constrained to one packet per round trip time. For severe congestion, this limitation causes the retransmission timer to expire and slow start. This extension does not alter the congestion avoidance procedure

of the fast retransmission algorithm and will be shown to improve TCP error recovery performance for wireless links.

4 Behaviour of the TCP Extensions

The extensions have been implemented in a TCP/IP simulator (which contains a full implementation of TCP and a model for the wireless link) [7, 11, 15]. The implementation was based on RFC 793 [1], RFC1122 [16] and Tahoe TCP. The TCP/IP algorithms were initially taken from [17] and implemented using Simula [18] and have since been validated [19]. The Reno [20], New-Reno [4] and SACK [5] modifications were also implemented. The New-Reno modification with Reno was adopted as the reference implementation of Reno TCP for the remaining discussion.

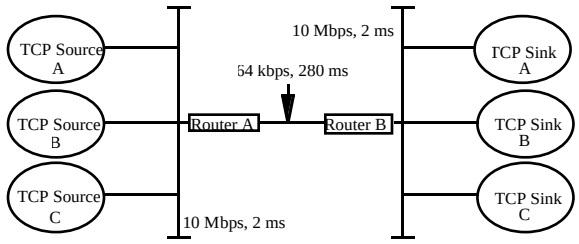


Figure 1: LAN interconnection using a satellite link.

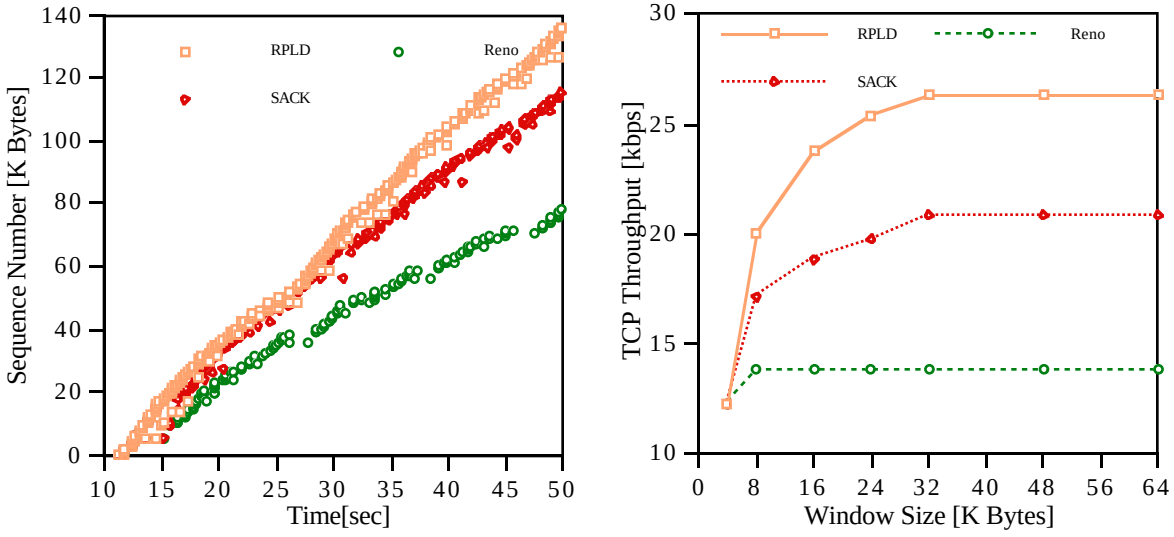
The first set of results compares TCP performance using each extension to highlight the importance of the SACK option and the proposed extensions when TCP operates over a network with an error prone link (Graph A of figure 2). The network shown in figure 1 is used for the simulation. Two local area networks (10 Mbps) were connected using a 64kbps (satellite) link with a propagation delay of 280ms. The data link queues are configured to hold 100 packets (i.e. no congestion losses). Packet losses were caused by simulating a random white noise source on the 64 kbps link, and the error rate of the link was $3x10^{-5}$ (a reasonably high BER typical of a wireless link).

The first sequence number of each transmitted packet has been recorded with the transmission time. The results demonstrate a clear improvement of TCP performance when using the SACK option over the simulated link and are further improved by adding the RPLD extension. The TCP throughput is plotted in graph B for a range of window sizes (simulation for 120 secs, corresponding to transfer of a 512Kbyte file).

Reno TCP was not able to utilise the available window size (buffer space) due to triggering of the congestion avoidance algorithms. SACK TCP gained a higher throughout, making use of the explicit information carried by the SACK option and uses the available window, correctly estimating the pipe size. RPLD TCP further improved TCP performance for wireless links, by avoiding unnecessary retransmission and subsequent Slow Starts. The following sections continue this analysis by zooming into a part of the graph A shown in figure 2.

4.1 Reno TCP Vs SACK TCP

Figure 3 compares the performance using Reno against Reno with the SACK option by observing the first 22 seconds of the connection. Similar analysis which demonstrates the advantage of using SACK over a network with only congestion packet loss can be found elsewhere [6, 12].



Graph A Graph B

Figure 2: Performance improvement using SACK and RPLD extensions compared to Reno TCP .

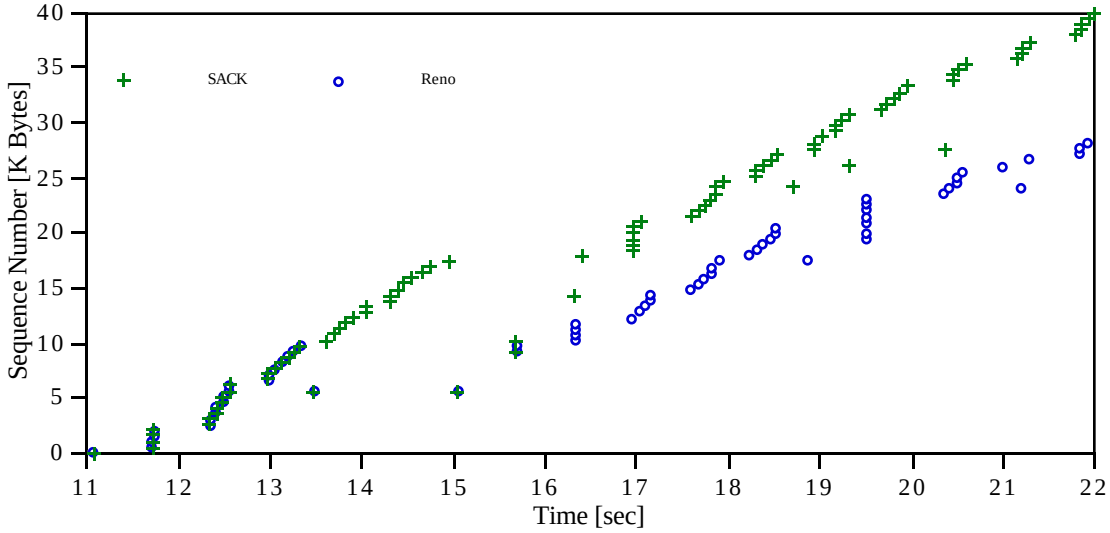


Figure 3: Operation of SACK TCP and Reno TCP (with New-Reno extensions). A simulated 64kbps satellite link (280ms propagation delay) with BER of $3x10^{-5}$.

These results demonstrate two main advantages of the SACK option over an error prone link. Both Reno TCP and SACK TCP transmit packets in a similar pattern and achieve the same throughput until the point where the transmitter receives 3 duplicate ACKs for packet 5644. In both implementations, this triggers a fast-retransmission of the packet. However, Reno TCP only acknowledges (cumulatively) in-sequence packets. By the time the Reno transmitter receives 3 duplicate acknowledgments, it has already sent a full window of data (constrained by the congestion window). In contrast, SACK TCP sends selective ACKs for each individual correctly received packet and therefore the SACK transmitter was able to calculate the pipe size accurately and resume the transmission of new data after the fast retransmission event.

The retransmitted packet (packet 5644) has been lost during the transmission (when using both Reno TCP and SACK TCP). Until this packet has been received, the receiver continues to acknowledge data up to 5644 bytes. This causes the retransmission timer subsequently to expire and the transmitter to retransmit the same packet at 15.05 seconds. The Reno transmitter then demonstrates a classic Slow Start by transmitting one packet followed by two packets, and then by four packets. However, a SACK TCP transmitter selectively retransmits only the packets lost during the transmission (i.e., only those which are not selectively acknowledged) and then resumes transmission of new data. The other fast-retransmissions of the SACK TCP transmitter at 18.69 sec (packet 24076), 19.32 sec (packet 26124), and 20.36 sec (packet 27660) were successful.

The Reno TCP transmitter also performs a fast-retransmission at 18.86 (packet 17420) after transmission of a packet with first sequence number 20492. It also retransmits two packets (packets 19468 and 19980) at 19.5 seconds using Slow Start. Although the first-retransmission is caused by a partial ACK, the second retransmitted packet is unnecessary caused by the Slow Start procedure used for New-Reno retransmissions. If the SACK option had been used, this unnecessary retransmission could have been avoided.

4.2 SACK TCP Vs RPLD TCP

The Retransmitted Packet Loss Detection (RPLD) algorithm avoids unnecessary expiry of the retransmission timer and subsequent Slow Start due to a loss of a retransmitted packet. After a packet is retransmitted, the RPLD transmitter checks whether the subsequently transmitted packets have been selectively acknowledged on reception of each duplicate ACK. Figure 4 compares performance using SACK TCP and SACK TCP with RPLD by observing the first 18 seconds of the connection.

The SACK TCP and RPLD TCP implementations transmit packets in a similar way until the RPLD TCP transmitter receives three duplicate ACKs for packet 10252 (which is the first new data packet after the fast retransmission of packet 5644). The transmitter then deduces the previously retransmitted packet has been lost during transmission (because a later packet was SACKed by the receiver). The RPLD transmitter therefore fast-retransmits packet 5644 again at 14.32 seconds. In comparison, the SACK transmitter only recovers this packet loss after expiration of the retransmission timer at 15.05 seconds which results in a rapid slow down of the transmission rate at appreciable BER.

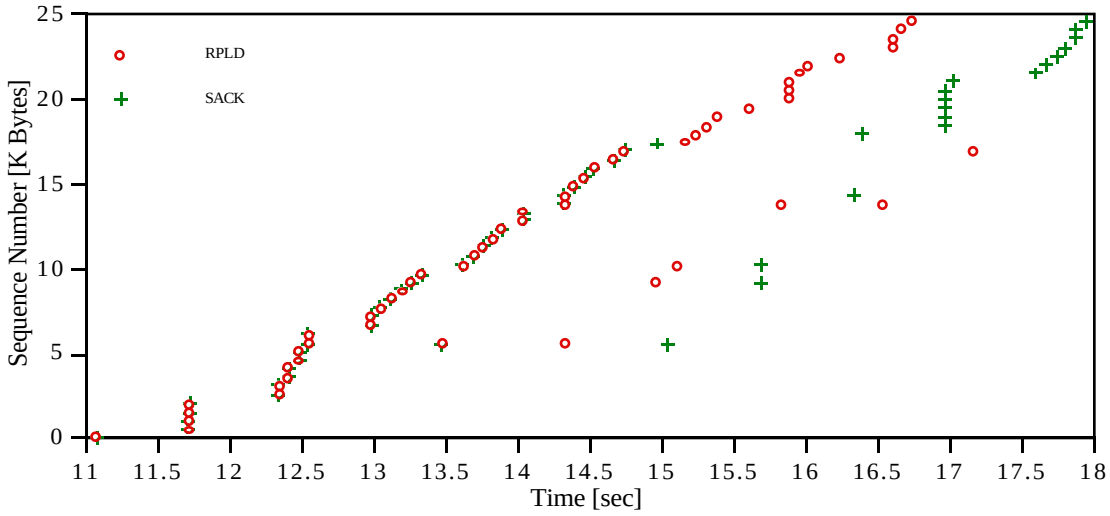


Figure 4: Operation of SACK TCP and SACK TCP with RPLD extension. A simulated 64kbps satellite link (280ms propagation delay) with BER of $3x10^{-5}$.

At 16.53 seconds, the loss of the retransmitted packet 13836 was again detected by the RPLD TCP transmitter. Similar transmitter time outs for the SACK transmitter can be noted at 31 and 41 seconds (as shown in the figure 2). This expiration of retransmission timer followed by Slow Start is inefficient when TCP is used over wireless links which may exhibit a high BER (e.g., satellite links) and is even more significant when the link has a high propagation delay. It may also be very inefficient if the retransmission time out estimation is too high.

5 End-to-End Performance

The benefit from implementing the new extensions is dependent on the network environment (such as the network congestion condition and propagation conditions of a wireless link when such a link forms part of the network). The end to end throughput measurements using different combinations of these modifications are, therefore, presented for two scenarios one with little congestion in the network and one for a congested network.

5.1 End-to-End Performance With Low Congestion

The first scenario (figure 5) shows the performance with little (or no) network congestion, that is where random loss is more significant than congestion loss (error bars show the 95% confidence intervals of a series of measurements). The network configuration described in section 4 (figure 1) was used with a single TCP session. The data link queues are configured to hold 30 packets. When packet losses are present in the network, the SACK extension provides higher throughput than Reno due to its efficient retransmission strategy and the ability to accurately estimate the pipe size. The advantage of the SACK option diminishes when the packet loss rate increases due to the link errors (e.g., when BER exceeds $3x10^{-6}$), since the congestion control algorithms dominate the operation of error recovery.

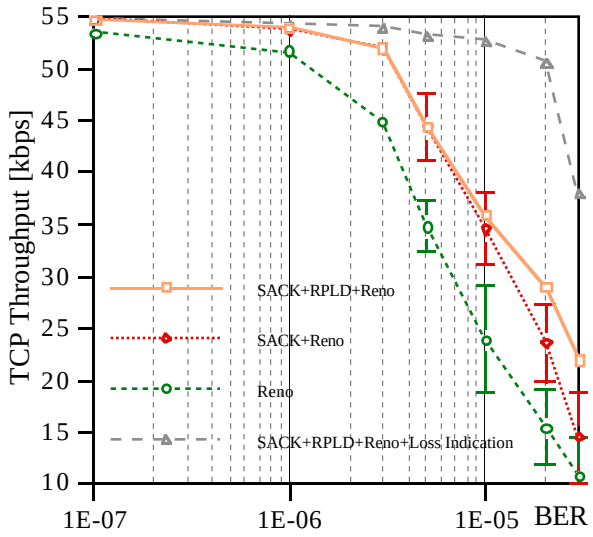


Figure 5: TCP performance for different extensions over a network of interconnecting LANs using a Satellite link. The potential benefit from differentiating link/congestion loss when using RPLD is also shown (SACK+RPLD+Reno+Loss Indication).

The addition of the RPLD extension has little impact at moderate to low BER. When the propagation conditions further degrade (e.g., BER higher than 10^{-5}), retransmission packet loss becomes significant and the RPLD extension therefore significantly improves TCP performance ("SACK+RPLD+Reno" of figure 5). TCP performance has

been degraded when the packet loss rate increases due to link errors, since the congestion control algorithms dominate the operation of error recovery.

5.2 End-to-End Performance With Congestion

A wide area network (figure 6) was also simulated to compare different extensions when the network is subjected to both congestion loss and link errors.

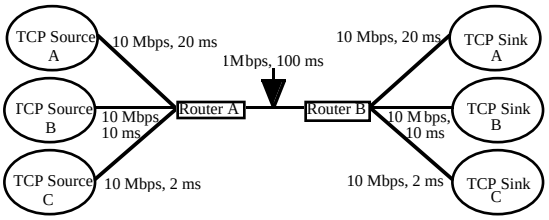


Figure 6: Network interconnecting LANs using a wireless WAN.

Three TCP sessions (sources at host A, B and C) share the 1Mbps link, with the router queue configured to hold 30 packets. The TCP window size was again 64kbytes and MSS was 512 bytes. A noise source was introduced to both directions of the 1Mbps link. The aggregated throughput of the three sessions is shown in figure 7 (error bars show the 95% confidence interval).

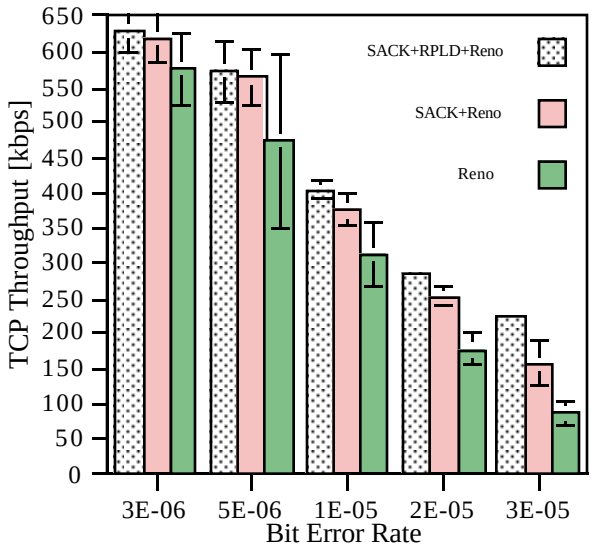


Figure 7: TCP performance of different extensions over a network interconnecting LANs using a wireless WAN.

The results demonstrate that SACK (and RPLD) improves TCP performance for wireless links even when congestion is present ("SACK+RPLD+Reno" of figure 7). Futher improvement in performance requires accurate differentiation between link packet loss and network congestion.

6 Discussion

When a wireless link forms a part of a network, link packet losses (due to noise) are often more significant than congestion losses. However, existing implementations of TCP assume that a packet loss is an indication of network congestion and take measures to avoid further congestion by reducing the transmission rate (i.e. reducing window size). This results in a very low utilisation of the link when there is an appreciable rate of losses due to link errors (figure 5 and 6). This issue is very significant for wireless links, particularly those with a long propagation delay (e.g. satellite links). Some authors have even suggested removing congestion control for such links [21], however such action is potentially very dangerous.

TCP performance over networks subjected to both link and congestion loss can be improved by enhancing the wireless link or extending TCP to accommodate the needs of the wireless link.

Implementation of Forward Error Correction (FEC) and Automatic Repeat Request (ARQ) techniques at the link level (or below), which may transparently enhance the link quality by hiding the effect of loss in the wireless link [7, 9]. Such techniques allow existing TCP protocols (optimised to recover congestion packet loss) to efficiently operate over the entire network. A transport layer gateway at the interface to the wireless link may provide a similar service [9, 22, 23].

Alternatively, TCP (in the Internet hosts) may be extended allowing it to determine the type of packet loss and respond accordingly. Most suggested extensions adopt one of two approaches [24]:

(i) Explicit indication of the type of packet loss sent by the network routers
(ii) Extension of TCP to infer the presence of link loss (implicit indication)

6.1 Explicit Indication

Explicit schemes are designed to exploit the additional information available locally at the equipment directly connected to the wireless link [8, 9, 25]. Such equipment is aware of any local congestion and may be able to identify packet lost due to link errors. This information however needs to be passed to the sender to modify TCP behaviour.

One example is the protocol suggested by Durst [8] in which the router connected to the wireless link maintains a weighted moving average of the number of corrupted packets received over the link. When this number exceeds a threshold, the router generates "corruption-experienced" ICMP messages to each destination. Each TCP receiver then informs the corresponding sender that corruption has been experienced by sending a new TCP option. The sender may then choose to inhibit congestion measures until it

receives an ACK without the corruption-experienced TCP option. A similar technique has been described in [9].

It is possible that most wireless links (from time to time) suffer congestion loss as well as link errors. To utilise explicit corruption indications, the sender must also verify the lack of network congestion along the intended retransmission path. Explicit congestion notification (e.g. RED [25], DECbit [26]) may ease this detection of congestion.

6.2 Implicit Indication

Instead of relying upon modification to the network, the sender may attempt to infer the type of packet loss from the arrival of received ACK and/or data packets. Implicit indications have the advantage of not requiring additional network and processing overhead at intermediate routers, and therefore off-the-shelf equipment may be used. The drawback is that the detailed status of the wireless link is not visible to the sender or the receiver, which must be inferred by other means.

One approach is to measure the variation in the round trip delay from a measured (minimum) reference delay. A change (increase) in delay may indicate congestion in the network [27, 28]. Paxson [14] observed that a congestion condition usually persists for an appreciable time (higher than the RTT), and it is therefore possible for a sender to detect queuing fluctuations and imply the presence of congestion (the variation in the one way packet transit delay measurement may more accurately estimate the queuing delay [14]).

However, it is very difficult to accurately infer the type of loss by only using round trip delay variation because:

(i) A sender may never experience a low delay due to persistent congestion in the network. This may result in an artificially high reference delay measurement wrongly indicating a lack of congestion.

(ii) The packets belonging to a session are not necessary routed along the same path which introduces delay variation [14].

(iii) Some routers employ active queue management (rather than drop-tail routing) which complicates congestion detection. For example, a Random Early Detection (RED) router [29] may discard packets due to persistent congestion, even though the instantaneous queue size is small (low delay), and the delay for a router with Weighted Fair Queuing (WFQ) [30] may depend on the session bandwidth allocation.

6.3 Challenges to Implementation

The development of a robust algorithm to differentiate the type of packet scheme is likely to result in significant performance gain for TCP over wireless links. Any implementation using implicit or explicit indication needs to adopt a conservative approach which applies congestion avoidance if any congestion *may be present* in the network, but may relax this when the network is know not to be congested.

Since TCP congestion avoidance algorithms restrict the transmission of new data after retransmission of a packet (i.e., window closes), a non-congestion packet loss indication technique would also improve performance of the suggested RPLD algorithm enabling it to achieve high throughput when the BER is higher than $3x10^{-6}$ (see suggested benefit with loss indication in figure 5).

The challenge is to design a technique which reliably identifies link loss (where rapid retransmission benefits the session), while also providing minimal chance of confusing this with congestion (where conservative retransmission is required for network stability). A failure to respond to congestion will inevitably exacerbate any network congestion [24].

7 Conclusions and Future Work

The SACK option and New-Reno modifications have been studied in detail and were found to significantly improve TCP performance especially over wireless links (e.g., satellite links). The explicit information provided by the SACK option allows the sender to more efficiently transmit packets which are lost by either congestion or random (link) errors.

The SACK option also provides additional information which allows the transmitter to detect a loss of a retransmitted packet when using the suggested RPLD extension. RPLD requires only modification to the sender, and allows efficient recovery of lost retransmitted packets without reliance on the operation of TCP timers. This extension provides improved performance for a network which experiences a high rate of packet loss (e.g., a BER higher than 10^{-6}).

The benefit from using SACK (and RPLD) for an appreciable random (link) error rate ($>10^{-6}$) is limited by an interaction between the error recovery procedures and the congestion avoidance algorithms. An implicit or explicit random loss indication method may significantly improve performance, however it is fundamentally difficult for a TCP sender to reliably discriminate random (link) loss from occasional congestion loss. The development of an accurate algorithm to infer type of packet loss is a topic of current research.

8 References

1. J. Postel, 'Transmission Control Protocol', *Information Sciences Institute, University of Southern California*, RFC 793, September 1981.

2. V. Jacobson, 'Congestion Avoidance and Control', *SIGCOMM '88*, ACM, USA, 314-329 (1988).
3. W. R. Stevens, 'TCP Slow Start, Congestion Avoidance, Fast Retransmission, and Fast Recovery Algorithms', *IETF*, RFC 2001, January 1997.
4. J. C. Hoe, 'Improving the Start-up Behavior of a Congestion Control Scheme for TCP', *SIGCOMM '96*, ACM, California, USA, 270-280 (1996).
5. M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, 'TCP Selective Acknowledgment Options', *IETF*, RFC 2018, October 1996.
6. K. Fall and S. Floyd, 'Simulation-based Comparisons of Tahoe, Reno, and SACK TCP', *ACM Computer Communication Review*, **26**(3), 5-21 (1996).
7. N. Samaraweera and G. Fairhurst, 'Robust Data Link Protocols for Connection-less Service over Satellite Links', *Int J. Satellite Communications*, **14**(5), 427-437 (1996).
8. R. C. Durst, G. J. Miller, and E. J. Travis, 'TCP Extensions for Space Communications', *MOBICOMM 96*, ACM, USA, 15-26 (1996).
9. H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, 'A Comparison of Mechanisms for Improving TCP Performance over Wireless Links', *SIGCOMM '96*, ACM, California, USA, 14 (1996).
10. V. Jacobson, 'Modified TCP Congestion Avoidance Algorithm', End-to-End mail, April 1990.
11. N. Samaraweera and G. Fairhurst, 'Explicit Loss Indication and Accurate RTO Estimation for TCP Error Recovery using Satellite Links', *IEE Proceedings - Communications*, **144**(1), 47-53 (1997).
12. M. Mathis and J. Mahdavi, 'Forward Acknowledgment: Refinng TCP Congestion Control', *SIGCOMM '96*, ACM, California, USA, 281-291 (1996).
13. S. Floyd, 'Private communications', 1997.
14. V. Paxson, 'End-to-End Internet Packet Dynamics', *SIGCOMM 97*, ACM, France, 139-152 (1997).
15. G. Fairhurst, 'A Simulation of a Satellite Link Employing a Protocol Based on X.25', *Fourth UK Teletraffic Symposium*, IEE, Bristol, UK, (1987).
16. R. Braden, 'Requirements for Internet Hosts - Communication Layers', *IETF*, RFC 1122, October 1989.
17. D. E. Comer and D. L. Stevens, *Internetworking with TCP/IP: Design, Implementation, and Internals*, Vol:**2**, Prentice Hall, Englewood Cliffs, USA, 1991.
18. R. J. Pooley, *An Introduction to Programming with Simula*, Blackwell Scientific Publications, 1987.
19. N. Samaraweera, 'Robust Connection-less Service over a Packet Satellite Link', *University of Aberdeen, Aberdeen, UK*, PhD Thesis, 1995
20. W. R. Stevens, *TCP/IP Illustrated: The protocols*, Vol:**1**, Addison Wesley, New York, 1994.
21. A. Pirovano and G. Maral, 'Congestion Avoidance in TCP/IP Applied to LAN Interconnection over Satellite Links', *Final Workshop of COST 226 Integrated Space/Terrestrial Networks*, EC, Budapest, Hungary, 79-87 (1995).
22. A. V. Bakre and B. R. Badrinath, 'Implementation and Performance Evaluation of Indirect TCP', *IEEE Transactions of Computers*, **64**(3), 260-278 (1997).
23. J. A. Cobb and P. Agrawal, 'Congestion or Corruption? A Strategy for Efficient Wireless TCP sessions', *IEEE Symposium on Computers and Communications*, IEEE, USA, 262-268 (1995).
24. C. Partridge and T. Shepard, 'TCP/IP Performance over Satellite Links', *IEEE Network*, **11**(5), 44-49 (1997).
25. S. Floyd, 'TCP and Explicit Congestion Notification', *ACM Computer Communication Review*, **24**(5), 10-23 (1994).
26. K. K. Ramakrishnan and R. Jain, 'A Binary Feedback Scheme For Congestion Avoidance In Computer Networks With A Connectionless Network Layer', *SIGCOMM 88*, ACM, USA, 303-313 (1988).
27. R. Jain, 'A Delay-Based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks', *ACM Computer Communication Review*, **19**(5), 56-71 (1989).
28. L. Brakmo S. and L. Perterson L., 'TCP Vegas: End to End Congestion Avoidance on a Global Internet', *IEEE Journal on Selected Areas in Communication*, **13**(8), 1465-1480 (1995).
29. S. Floyd and V. Jacobson, 'Random Early Detection Gateways for Congestion Avoidance', *IEEE/ACM Transactions on Networking*, **1**(4), 397-413 (1993).
30. A. K. Parekh and R. G. Gallager, 'A Generalized Processor Sharing Approach to Flow Control in Intergrated Services Networks: The Multiple Node Case', *IEEE/ACM Transactions on Networking*, **2**(2), 137-150 (1994).

Acknowledgments

The authors wish to thank the European Space Agency (ESA) and Defence Research and Evaluation Agency (DERA Defford) in the UK for their use of the satellite network which provided insight into the behaviour of applications using TCP/IP over a satellite Internet. This work greatly benefited from constructive comments from Sally Floyd and the CCR reviewer.