

D-Proxy: Reliability in Wireless Networks

David Murray
Murdoch University
D.Murray@murdoch.edu.au

Terry Koziniec
Murdoch University
T.Koziniec@murdoch.edu.au

Michael Dixon
Murdoch University
M.Dixon@murdoch.edu.au

Abstract—Packet corruption negatively affects TCP throughput because losses are interpreted as congestion. To enable TCP to operate effectively over wireless networks, a number of reliability techniques such as Forward Error Correction (FEC) and Automatic Repeat Request (ARQ) are used. These provide reliability at the expense of increased overheads. This study provides experimental results of a new distributed Performance Enhancing Proxy (PEP) called D-Proxy. This proxy can provide reliability to wireless links with minimal overhead. The results show that D-Proxy can provide near-optimal performance in the presence of high loss rates. It is suggested that D-Proxy could be used to replace current ARQ mechanisms.

I. INTRODUCTION

Transmission Control Protocol (TCP) was designed in the 70's to provide reliable end-to-end service for applications. During this period, Internet links were wired and reliable and thus the majority of lost/missing packets were from congestion. These losses occur when packets are transmitted too fast, causing router buffers to fill. This is otherwise known as congestion. When routers become congested, they will drop packets to alleviate congestion. Dropping packets is used as a signal to inform the TCP sender that it is sending too fast. Upon the detection of packet loss, the TCP sender will react by reducing the TCP congestion window, slowing the transmission speed. Since the original design of TCP, there has been considerable deployment of wireless technology across the Internet.

Wireless technologies are fundamentally different from copper and fibre based technologies as they have significantly higher Bit Error Rates (BERs). Higher error rates lead to dropped packets, which unfortunately, get interpreted by TCP as congestion. These “congestion” events cause reductions in TCP's window leading to link underutilisation. To prevent packet losses from being interpreted as congestion, wireless links implement Forward Error Correction (FEC) and Automatic Repeat Request (ARQ) to reduce packet loss.

II. FEC AND ARQ

FEC is the process of adding redundant bits to accommodate small amounts of packet corruption. Varying levels of FEC are added to different technologies based on the degree of unreliability. This leads to a trade-off. Higher levels of FEC will increase reliability but lower the data rate due to overheads.

If FEC is unable to correct errors, ARQ is used to recover the entire frame. Send and wait error detection is the most simple and commonly used form of ARQ. The sender will send a data packet and wait for an individual ack before

sending the next data frame. As the sender has no knowledge of the transmission success, both the data packet and the ARQ ack must be successful.

ARQ does not provide perfect reliability because it will only retransmit packets a finite number of times. In many applications, latency can be more important than reliability. For example, in voice telephony, random losses as high as 2% do not cause audible speech degradation [1] but delayed and jittery delivery do. Another reason why ARQ schemes only transmit a finite number of times is because excessive reattempts of the same transmission may cause fairness issues between TCP flows [2]. When ARQ fails to reliably send a packet, TCP will recover the loss, but; as the loss will be interpreted as congestion, there will be a significant performance penalty.

Newer ARQ mechanisms used in recent WiFi and WiMAX amendments use block acknowledgements to lower the ARQ overhead [3]. BlockAcks reduce the overhead by acknowledging a series of packets with a single acknowledgement. This reduces the number of acknowledgements sent, lowering the overhead. This can create contention and fairness issues [4], [5] because it requires a single transmitting station to have lengthy medium access. During this time, other stations will be unable to transmit or receive packets. BlockAcks also increase link delays. As packets are being sent and received in blocks, the OS cannot process any of the packets until the entire block is successfully received.

The cost of using link layer ARQ to hide link losses from TCP imposes a significant overhead. It has been found that ARQ in 802.11 consumes as much as 22% of the transmission time [6]. The use of BlockAcks can reduce this overhead to 10% but introduces a number of complications described above [4], [5]. Removing acknowledgements yields significant gains in performance. Put simply, the time that was previously spent sending acks can be used for the transmission of actual data. The aim of this paper is to investigate this overhead problem which plagues all wireless technologies. Two lines of investigation are taken; increasing TCP's intelligence to prevent the misinterpretation of loss as congestion, and using Performance Enhancing Proxies (PEPs) to provide wireless reliability with less overhead. We begin with an examination of TCP.

III. TCP IMPROVEMENTS

Improving wireless performance using TCP is ideal because only end host devices need to be updated to improve capacity.

The caveat is that TCP modifications must improve wireless performance without disadvantaging wired TCP transactions. As a result, TCP improvements tend to be evolutionary.

The aim of TCP congestion control is to send packets from the source to the destination, as quickly as possible, without causing congestion on intermediate routers. TCP does this by building a congestion window. This window dictates the number of packets allowed to be unacknowledged between the source and the destination.

Lost packets are used as a sign of congestion. The TCP receiver will signal packet loss by sending duplicate acknowledgements. Upon reception of three duplicate acks, TCP New Reno will resend the packet and halve the congestion window.

The problem with wireless packet loss is not reliability because TCP ensures reliable delivery end-to-end. The problem is that losses are treated as congestion; causing reductions in the TCP congestion window and lowering the transfer rate. TCP operates in this manner because traditionally, congestion is the primary cause of packet loss. ARQ mechanisms were introduced for the simple purpose of preventing link losses being interpreted as congestion. This section investigates whether TCP optimisations can prevent this mis-interpretation. Numerous solutions have proposed to improve the performance of TCP over wireless links.

A. Westwood

TCP Westwood [7] was created as an alternative congestion control mechanism specifically designed for wireless networks. It mimics TCP Reno operation with exponential growth during slow start and linear growth during congestion avoidance phases, however, Westwood's response in the case of packet loss is different. Based on the packet sizes and Round Trip Time (RTT) estimates, Westwood uses a series of equations to estimate the bandwidth usage of the link. When a packet is lost, the window is reset to the bandwidth estimate rather than halved. Further details of the mathematical derivation of the bandwidth estimates can be found in [7].

B. SACK

Selective ACKnowledgements (SACK) [8] are a TCP extension that enhance recovery when multiple packet losses occur within a RTT window. It is anecdotally recognised that wireless networks experience short bursts of packet loss. Losing multiple packets in the same RTT is problematic for pre-SACK TCP because cumulative acks can only hold the information of the first lost packet. SACK can specify which blocks of data, following the loss have been successfully received. By informing the sender which packets have been received, and which packets must be resent, multiple lost packets can be recovered in one RTT. SACK is proven to be beneficial [9] for error recovery in wireless networks and thus has been implemented in all major OSs. SACK is an acknowledgement mechanism and is therefore capable of being used alongside congestion control mechanisms such as NewReno and Westwood.

C. Explicit Loss Notification

As TCP (mis)interprets packet loss as congestion, many [10], [11] have postulated that if packet loss and congestion could be signalled differently, TCP could make appropriate window adjustments. In the case of congestion, the TCP sender could slow down. Alternatively, if interference or collisions caused the loss, the congestion window could be maintained.

Balakrishnan et al [11] suggested that that some wireless losses could be interpreted at the TCP receiver. If a packet with a corrupt Cyclic Redundancy Check (CRC) is received by a TCP receiver, it could inform the sender that the loss was not caused by congestion. This solution is problematic because if the CRC is incorrect, it is also likely that either; the TCP header is corrupt and unidentifiable or, the packet will be dropped before reaching the TCP receiver due to a corrupt CRC in the IP header.

Base station Explicit Loss Notification (ELN) implementations are another alternative [10]. If wireless base stations could detect packet loss, they could inform senders using ICMP messages or by tagging returning TCP acks. The difficulty is determining whether a transmission was successful. Base station implementations of ELN typically monitor the success of ARQ acknowledgements. As this research is attempting to reduce or remove the ARQ overhead, this option is of little interest. A mechanism that could separate packets lost due to errors and packets lost due to congestion is the panacea that has thus far proved unattainable.

IV. PERFORMANCE ENHANCING PROXIES

PEPs are described in RFC 3135. They can be used in a variety of circumstances, however, their generally described purpose is mitigating link related degradations. This study will investigate whether PEPs are capable of providing reliability to wireless links with less overhead than ARQ mechanisms. Split-TCP and Snoop are commonly described PEPs.

A. Split-TCP

Split-TCP PEPs [12], [13], segment a TCP connection by capturing the SYN and SYN-ACK packets that are used to setup a TCP session. Split-TCP will rewrite these packets to imitate each side of the transaction. One advantage is a large reduction in TCP perceived Round Trip Times (RTTs). Real end-to-end latencies will be the same, however, by splitting the link, the TCP sender can receive acks more quickly, building the congestion window faster and thereby completing faster. Subsequently, Split-TCP is commonly used in satellite networks which feature large RTTs.

Split-TCP can also be used to hide losses in low RTT wireless by placing the PEP between the wireless link and the TCP sender. If the latency between the PEP and the TCP receiver is small enough, losses on the wireless link do not affect performance because only a small TCP window is required to fill low latency connections.

Unfortunately, Split-TCP has numerous problems. Firstly it breaks TCP end-to-end semantics [14]. This means that when a TCP sender receives a TCP ack, the actual packet

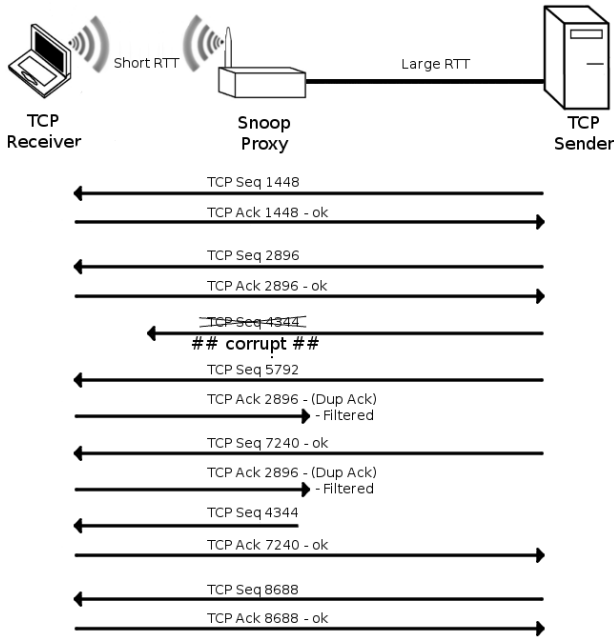


Fig. 1. The operation of Snoop is reactive. Snoop will attempt to resend a packet after a loss has occurred

may not have reached the real TCP receiver. If that packet is currently buffered on the PEP, and the path to the actual TCP receiver breaks, unforeseen application layer problems may arise. Split-TCP is also not capable of accommodating route changes between the TCP sender and the PEP [14]. Finally, numerous security protocols will not work through Split-TCP [14]. For these reasons, Split-TCP is not recommended as a general solution to lossy wireless links [14].

B. Snoop

The Snoop proxy [15], [14] is an alternative to Split-TCP. It avoids the afore mentioned problems because it doesn't split a TCP connection, but instead, detects losses by monitoring TCP acknowledgements. Duplicate acks are a sign of packet loss. Snoop will filter duplicate acknowledgements and retransmit the lost packet. In ideal scenarios, the TCP sender will be oblivious to the loss. Fig 1 demonstrates the basic operation of Snoop.

A problem is that the protocol [15], [14] does not specify what should happen if the retransmitted packet is also lost. It also fails to specify how many duplicate acks should be filtered before resending the lost TCP segment. Furthermore, a number of studies suggest that Snoop is unable to completely hide TCP losses due to interoperability problems between SACK and Snoop [16], [17], [18].

V. D-PROXY

A. Basic Proxy Design and Implementation

D-Proxy is a new proactive distributed TCP proxy that we designed to overcome the limitations of Snoop and Split-TCP. D-Proxy is distributed because it uses a proxy either side of the lossy link. It is proactive because, instead of using TCP acks

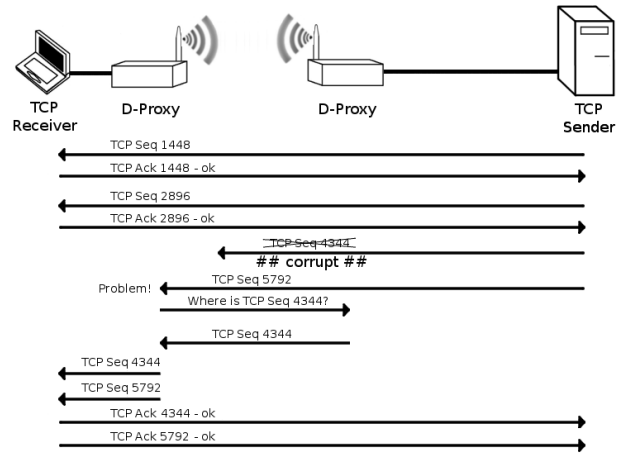


Fig. 2. D-Proxy is proactive; recovering and reordering packets before relaying them to the TCP receiver

as confirmation of packet loss, the TCP sequence numbers in the data packets are used. D-Proxy detects packet losses in the same way that TCP receivers detect packet loss, the: $next_seq_num = curr_seq_num + payload_size$. A packet can be recognised as missing when the received sequence number is higher than expected. When this occurs, a request can be sent to the previous proxy to request retransmission.

Fig 2 shows the basic operation of D-Proxy. The missing packet is discovered because 5792 was sent when 4344 was expected. When a loss is detected, D-Proxy buffers segments from that TCP flow until the lost segment can be replayed and reorganises them such that they are forwarded in sequence.

While the basic concept of D-Proxy is relatively simple, the implementation required significant work. D-Proxy maintains TCP state information and each flow is differentiated based on source IP, destination IP, source port and destination port. The individual packets being cached are identified within their flow based on sequence numbers. D-Proxy was implemented in Linux using the `ip_queue` library which passes packets from kernel space to user space for processing.

B. Inter Proxy Communication

D-Proxy buffers TCP flows until lost packets can be recovered and reordered, but, lengthy buffering can cause TCP timeouts. Therefore, the speed at which D-Proxy can recognise a loss, request retransmission and reorder the recovered packet is of utmost importance. The core problem for D-Proxy is that the burst losses, found in wireless networks, increase the likelihood that D-Proxy packets requesting retransmission are corrupt. Alternatively, packets requesting retransmission may be successful but the data segment being retransmitted may fail. Fig 3 demonstrates this unreliability problem. As a result of this inter proxy unreliability, the mechanism used to request the retransmission of lost packets was critical.

UDP sockets were used to re-request lost packets. We found TCP error recovery too slow in scenarios where errors must be detected and recovered on a millisecond time-scale. D-Proxy uses UDP and its own fast reliability mechanisms. Note these

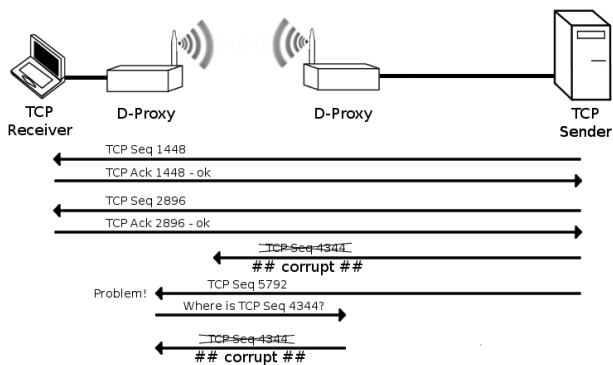


Fig. 3. Requests for packet retransmission and the retransmitted packet are frequently lost

mechanisms had to provide fast two way reliability for both the D-Proxy retransmission request and the actual retransmitted data segment.

C. Timeouts

The scenario shown in Fig 3 is common. Two mechanisms are used to infer that either the UDP retransmission request, or, the retransmitted data segment have been lost. One mechanism was timer based. The amount of time required to recover missing packets is continually averaged. The moving average of retransmission times will adapt to natural link latency and link loads.

The duration of time to wait before sending repeat retransmission requests is a trade-off. If a replayed packet is not lost but merely waiting in buffers, or awaiting media access, sending further retransmission requests for this packet will only exacerbate delays and create superfluous retransmissions. However, if the re-requested packet has actually been lost, waiting too long before sending further retransmission requests increases buffering and the likelihood of TCP time-outs. In D-Proxy, missing packets are re-requested after two, three and four times the average delay required to retransmit a lost packet. On the fourth retransmission request, D-Proxy will no longer hold the buffer awaiting that packets retransmission. This timeout based retransmission mechanism provided performance benefits, however, transfer rates were still suboptimal. The process of reliably recovering packets, needed to be quickened to reduce the amount of buffering on D-Proxy.

D. Retransmission Order

To reiterate, the problem being solved is the internal reliability of the D-proxy retransmission request packet and the actual data packet being retransmitted. Data packets that have been lost are detected immediately, however, it is difficult to determine the success of a D-Proxy request for retransmission or the success of the retransmitted data segment.

Given that losses occur in bursts, we created a mechanism which uses burst losses as an advantage. For example, lets suggest that 10 packets are sent back-to-back within a TCP flow. Due to the burst loss nature of wireless links, 3 of these 10 packets are lost. Individual UDP D-Proxy retransmission

requests will be sent for each of these three packets. As these retransmission requests are sent in order, it is expected that the retransmitted packets will also be resent in order. In D-Proxy, if the second missing packet is received ahead of the first missing packet, it infers that either the first retransmission request or the first retransmitted packet was lost. A request for the first missing packet will need to be resent. This mechanism reduced the latency of packet recovery. Faster packet recovery resulted in less buffering and significantly better performance.

E. Staggered TCP Catch-up

Under heavy losses, D-Proxy can sometimes queue many packets. Upon the recovery of a missing packet, it is desirable to avoid replaying all the buffered packets immediately as this could be to the detriment of other TCP flows. D-Proxy will send a maximum of five packets from a particular TCP flow before rechecking the input buffer. If the next packet on the ip_queue input buffer is from a different flow, then that flow will be serviced. If it is from the same flow, then another 5 packets can be sent from the buffer to the kernel. The purpose of this mechanism is to stagger the catch-up that occurs after recovering the holes of a TCP sequence. This reduces the impact on other TCP flows and aids fairness.

F. Variable Per Flow Buffer Size

Similar to ordinary routers, buffering increases latency and slows the reaction to congestion, however, some buffering is important to allow missing packets to be re-requested, resent and reordered for delivery. D-Proxy has a buffer size of 150 packets. With one TCP flow, the entire 150 packet buffer will be utilised. The addition of more flows, will divide the buffer equally. Therefore, if there are 5 flows, each flow will have a buffer of 30 packets. Flows may exceed their buffer size. For example, if there is one flow utilising a buffer of 150 packets and then 2 additional flows are added, the buffer size will be reduced to 50 packets per flow. The 100 packets that exceed the buffer size from one flow are not lost or dropped, instead the oversize buffer will continue being processed, but no packets that exceed the allowed buffer size will be reordered in sequence. Retransmission requests may have already been sent to fill the holes in the TCP sequence, however, D-Proxy will no longer reorder packets that are exceeding the buffer size. Receiving the lost packets out of order fortuitously causes the TCP sender to slow down to accommodate the two new flows.

VI. EXPERIMENT

Our experiment shows the performance of D-Proxy under a range of conditions. We believe that D-Proxy is applicable to a range of technologies and scenarios. The experimental aim was therefore, to show the performance benefit of D-Proxy under a range of loss conditions. Our experimental setup is deliberately generic because we believe the results are applicable to many wireless network technologies.

The testing setup is shown in Fig 4 and conceptually emulates a 100Mb/s wireless link connected via 100Mb/s

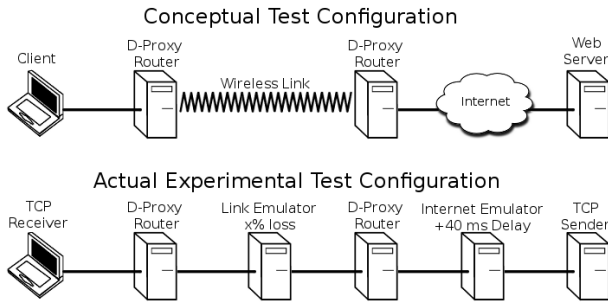


Fig. 4. The conceptual and actual experimental test configurations

Ethernet to the Internet. The wireless link was emulated with a dual NIC Linux machine which used Netem to create varying levels of loss. The Internet was emulated by inserting 40ms of delay. These link emulators were Intel Celeron 700Mhz machines. The D-Proxy machines were P4 2.4GHz machines with dual NICs. All machines used a variant of Ubuntu and the TCP sender was running the Apache 2.0 web server. Two types of lossy networks were emulated, very high loss and high loss. The very high loss tests feature packet loss rates linearly descending from 10% to 1% packet loss. The other test results show packet loss rates on a logarithmic scale from 1×10^{-1} to 1×10^{-6} .

Multi flow tests were performed whereby one host started five simultaneous downloads of a file from an Apache web server. The results are the amount downloaded (five copies of the file), divided by the time when the last file download completes. We believe this five flow test was a reasonable test of TCP fairness. The limited Bandwidth Delay Product (BDP) means that one flow typically does not fill the link, and thus; to complete quickly, it is advantageous if all flows finish simultaneously. Mechanisms which unfairly distribute bandwidth between flows achieved lower scores. Single flow tests were also performed, however, they have been omitted due to space limitations. D-Proxy was compared with Standard TCP Reno, TCP Westwood and Snoop.

VII. RESULTS AND DISCUSSION

The results shown in Fig. 5 and 6 show the performance of the different solutions. There was a maximum capacity of 93.2 Mb/s imposed by the limits of the 100 Mb/s link.

Reno and Westwood show the worst performance. They are end-to-end congestion control mechanisms and are clearly not designed to hide very high loss rates between 10% and 1%. In these scenarios, Every lost packet is interpreted as congestion, causing constant window reductions. At loss rates between 10% and 1%, Westwood outperformed Reno by a small margin, however, as loss rates decreased from 1×10^{-1} to 1×10^{-6} TCP Reno began to outperform Westwood.

Our results suggest that Snoop is unable to completely hide losses from the TCP sender, resulting in suboptimal performance. There are numerous reasons why we suggest that Snoop is unable to hide losses end-to-end. Interoperation with SACK is problematic because duplicate acks containing SACK

information are filtered and Snoop only recovers the packet referred to in the cumulative ack [9]. Despite our attempts to modify Snoop to utilise SACK, no significant performance benefits were made. Another problem is that, after Snoop's initial retransmission of the lost packet specified in duplicate acknowledgements, there is no mechanism to confirm or deny the success of the retransmitted packet.

D-Proxy significantly outperforms Snoop in all scenarios due to its proactive approach to packet recovery. By analysing TCP sequence numbers, holes can be detected and filled before they reach the final destination. This means that packets are delivered to the TCP sender in order. The D-Proxy code and the code used to test Snoop can be found at <http://bridgingthelayers.org/>.

Despite D-Proxy's obvious advantages over Snoop, it is debatable as to whether D-Proxy performs better than link layer ARQ technologies. This was impossible to test in this experiment as we were artificially introducing loss by dropping packets over an Ethernet link. Based on the adequate performance of modern wireless networks, ARQ is able to hide link layer losses using acknowledgements. This is evident because the results in Fig 6 demonstrate that dropping 0.1% of packets over a 50ms link would reduce the throughput by half. The next section does not argue that D-Proxy can hide higher loss rates from TCP, but that D-Proxy hides losses more efficiently.

A. D-Proxy/ARQ Analysis

D-Proxy is able to maintain and recover losses in very high loss situations. It can perform this task using negative acknowledgements. Unlike ARQ D-Proxy, will send a message when a packet has been lost rather than for every successful packet. Thus, the overhead of D-Proxy is minuscule ($< 1\%$) compared with traditional ARQ that consumes 22% of transmission time [6].

More recent ARQ developments use BlockAcks to reduce the ARQ overhead. With BlockAcks, groups of packets can be acknowledged by a single ack. Despite the obvious efficiency advantages over standard ARQ mechanisms, BlockAcks are still a positive acknowledgement mechanism, whereas D-Proxy is a negative ack mechanism. Prior work has suggested that BlockAcks can reduce the ARQ overhead to 10% [4], [5] in 802.11a/g technology.

Using BlockAcks requires frames to be sent and received in blocks. OSs cannot start routing received frames onto the next network segment until the entire block has been received. In D-Proxy, packets are not grouped and are processed as received.

With BlockAcks, the loss of one packet will cause the entire group of packets to be buffered awaiting the recovery of lost frames. This occurs because BlockAck is not transport layer aware. It must maintain order to prevent delivering mis-ordered packets. Comparatively, D-Proxy packets get separated into different TCP flows. The loss of one packet from a TCP flow will only hold up subsequent packets from the same TCP flow. We believe that D-Proxy will have lower overheads and will be more TCP friendly than current ARQ technology, however, it must be acknowledged that current comparisons

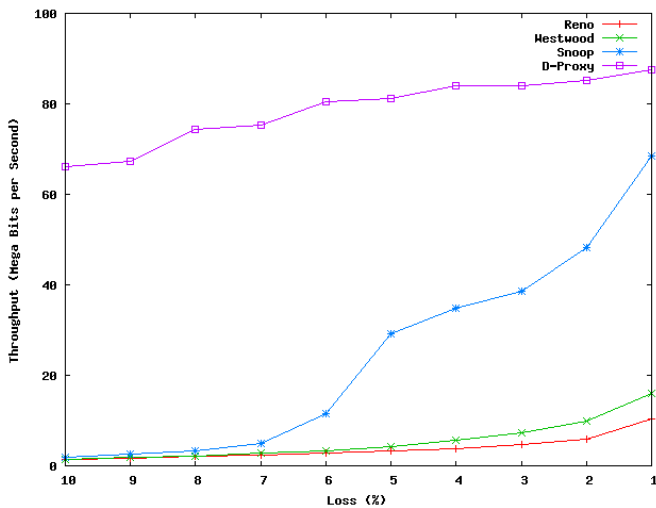


Fig. 5. Throughput of five TCP flows given losses of 10% to 1%

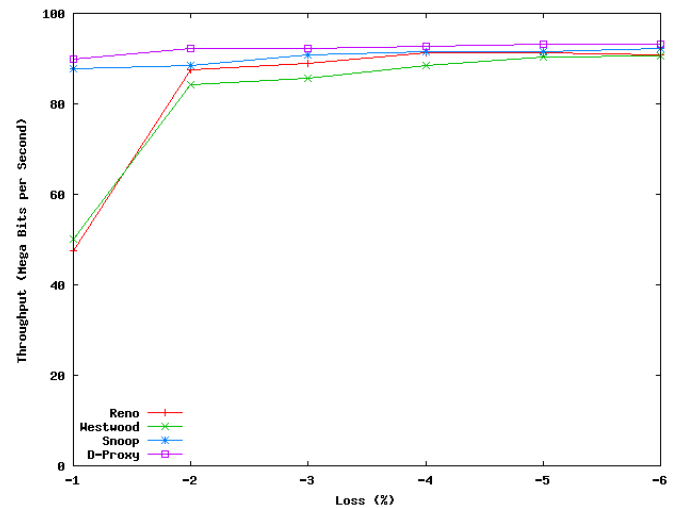


Fig. 6. Throughput of five TCP flows given losses of 1×10^{-1} to 1×10^{-6}

with ARQ are purely analytical. Future work must compare basic ARQ and BlockAck with D-Proxy to provide substantive evidence.

D-Proxy’s drawbacks are consistent with most PEPs: complexity and layer violation. D-Proxy is more complex than basic ARQ mechanisms and Snoop. However, D-Proxy is no more complex than modern ARQ optimisations like BlockAck. Transport layer security must also bypass the proxy mechanism, however this function will occur automatically.

VIII. CONCLUSION

This research introduced a new TCP proxy capable of hiding wireless losses from TCP. The experiments show that D-Proxy can recover packets, over highly lossy links, and can significantly outperform other competing solutions. The scenario used to test D-Proxy was deliberately generic because we believe it to be broadly applicable to a multitude of wireless technologies. D-Proxy is proactive. It analyses the sequence of data frames, recovering and reordering the packets for in order delivery to the TCP receiver. The key benefit is that D-Proxy is a negatively acknowledging proxy; sending messages only when a packet is missing. Overall, we believe that the performance of D-Proxy necessitates serious consideration as a potential replacement for ARQ mechanisms.

REFERENCES

- [1] Phil Karn, “The qualcomm cdma digital cellular system”, 1993.
- [2] G Fairhurst and L Wood, “Advice to link designers on link automatic repeat request (arq)”, IETF RFC 3366, August 2002.
- [3] IEEE, “802.11e - ieee standard for information technology-telecommunications and information exchange between systems-local and metropolitan area networks-specific requirements-part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications-amendment 8: Medium access control (mac) quality of service enhancements”, IEEE Standards, 2005.
- [4] Orlando Cabral, Alberto Segarra, and Fernando J Velez, “Implementation of multi-service ieee 802.11e block acknowledgement policies”, *IAENG International Journal of Computer Science*, vol. 36:1, pp. 1, June 2009.

- [5] Tianji Li, Qiang Ni, Thierry Turletti, and Yang Xiao, “Performance analysis of the ieee 802.11e block ack scheme in a noisy channel”, in *IEEE BroadNets*, 2005, pp. 551–557.
- [6] David Murray, Terry Koziniec, and Michael Dixon, “Solving ack inefficiencies in 802.11 networks”, in *2009 IEEE International Conference on Internet Multimedia Services Architecture and Applications (IMSAA)*, 2009, pp. 1–6.
- [7] Saverio Mascolo, Claudio Casetti, Mario Gerla, M. Y. Sanadidi, and Ren Wang, “Tcp westwood: Bandwidth estimation for enhanced transport over wireless links”, in *ACM Mobicom*, July 2001, pp. 287–297.
- [8] Matthew Mathis, J Mahdavi, Sally Floyd, and A Romanow, “Tcp selective acknowledgment options”, IETF RFC 2018, October 1996.
- [9] Farooq Anjum and Leandros Tassiulas, “Comparative study of various tcp versions over a wireless link with correlated losses”, *IEEE/ACM Trans. Netw.*, vol. 11, no. 3, pp. 370–383, 2003.
- [10] Wenqing Ding and Abbas Jamalipour, “A new explicit loss notification with acknowledgment for wireless tcp”, in *2001 12th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, Sep 2001, vol. 1, pp. B-65–B-69 vol.1.
- [11] Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan, and Randy H. Katz, “A comparison of mechanisms for improving tcp performance over wireless links”, *IEEE/ACM Trans. Netw.*, vol. 5, no. 6, pp. 756–769, 1997.
- [12] Ajay Bakre and B R Badrinath, “I-tcp: Indirect tcp for mobile hosts”, in *15th International Conference on Distributed Computing Systems*, 1995, pp. 136–143.
- [13] M Luglio, M Y Sanadidi, M Gerla, and J Stepanek, “On-board satellite “split tcp” proxy”, *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 2, pp. 362–370, February 2004.
- [14] J Border, M Kojo, J Griner, G Montenegro, and Z Shelby, “Performance enhancing proxies intended to mitigate link-related degradations”, IETF RFC 3135, June 2001.
- [15] Hari Balakrishnan, Srinivasan Seshan, Elan Amir, and Randy H. Katz, “Improving tcp/ip performance over wireless networks”, in *1st Annual International Conference on Mobile Computing and Networking*, New York, NY, USA, 1995, pp. 2–11, ACM.
- [16] Jaehoon Kim and Kwangsue Chung, “C-snoop: Cross layer approach to improving tcp performance over wired and wireless networks”, *International Journal of Computer Science and Network Security*, vol. 7(3), pp. 131–137, 2007.
- [17] Sarma Vangala and Miguel A. Labrador, “The tcp sack-aware snoop protocol for tcp over wireless networks”, in *IEEE Vehicular Technology Conference*, 2002.
- [18] Fanglei Sun, Victor O.K. Li, and Soung C. Liew., “Design of sack mechanism for wireless tcp with new snoop”, in *Wireless Communications and Networking Conference*, March 2004, vol. 2, pp. 1051–1056.