UNIVERSITY OF CALIFORNIA
SANTA CRUZ

Improving TCP Performance over Wireless Networks at the Link Layer

A thesis submitted in partial satisfaction
of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Christina Parsa

June 1998

The thesis of Christina Parsa is approved:

J.J. Garcia-Luna-Aceves

Anujan Varma

Darrell D. E. Long

Dean of Graduate Studies and Research

Contents

List of Figures

List of Tables

Improving TCP Performance over Wireless Networks at the Link Layer

Christina Parsa

ABSTRACT

We present the transport unaware link improvement protocol (TULIP), which dramatically improves the performance of TCP over lossy wireless links, without competing with or modifying the transport- or network-layer protocols. TULIP is tailored for the half-duplex radio links available with today's commercial radios and provides a MAC acceleration feature to improve throughput. TULIP's timers rely on a maximum propagation delay over the link, rather than performing a round-trip time estimate of the channel delay. The protocol does not require a base station and keeps no TCP state. TULIP is exceptionally robust when bit error rates are high; it maintains high goodput, *i.e.*, only those packets which are in fact dropped on the wireless link are retransmitted and then only when necessary. The performance of TULIP is compared against the performance of the Snoop protocol (a TCP-aware approach) and TCP without link-level support. The results of simulation experiments using the actual code of the Snoop protocol show that TULIP achieves higher throughput, lower packet delay, and smaller delay variance.

Keywords: Link Control, Wireless Networks, ARQ Protocol, TCP

Acknowledgments

I would like to greatly thank and express my gratitude to my advisor, J.J. Garcia-Luna-Aceves for his ideas, support and help with this project. He was never too tired to talk about the link layer. :) I would also like to thank Anujan Varma and Darrell Long for taking time to read this thesis. There are a few additional people to whom I remain grateful and should certainly mention. Many thanks to Rooftop Communications Corporation for donating the CPT toolkit. Chane Fullmer answered my endless stream of questions about CPT and FAMA. Hari Balakrishnan was helpful answering questions about the Snoop protocol. I am most thankful to Anujan Varma and Lampros Kalampoukas for their many discussions with me about TCP. Suzana always took a break with me when I needed one, and Jyoti's fast typing in the chair behind me was an inspiration. My mom, grandma, Bob and Leslie kept up my confidence. My husband, Mike, was a constant source of encouragement.

1. Introduction

Wireless networks are quickly becoming an integral part of the Internet and reliable protocols such as TCP [25] must be supported over these networks. Unfortunately, although TCP works very well for wired networks with minimal losses other than those due to congestion, wired and wireless networks are significantly different in terms of bandwidth, speed, propagation delay, and channel reliability. In particular, wireless channels suffer from bursty error losses that reduce TCP's throughput, because TCP incorrectly interprets packet loss as a sign of congestion that forces TCP to back off from further transmission, reduce its congestion window and enter a congestion avoidance phase (during which the congestion window is incremented by one segment every round-trip time) [11][17].

Maximum throughput occurs in a TCP connection when the TCP congestion window is as large as the bandwidth-delay product of the connection. Current versions of TCP react to losses differently and adjust the TCP congestion window in various ways. With the Reno [26] and Tahoe [17] versions the arrival of three duplicate acknowledgments allows for the fast retransmission of a lost packet. Once the missing packet is retransmitted, however, Tahoe does not maintain the current congestion window, but rather reduces its window size to one segment and begins slow start (during which the congestion window is essentially doubled every round-trip time). Reno follows a different strategy as it interprets the arrival of further duplicate acknowledgments as a sign that packets are in fact progressing through the previously congested network and instead begins fast recovery, during which the congestion window is incremented by one segment for each subsequent duplicate acknowledgment. Once the ACK for the missing packet is received and acknowledged, the congestion window is reduced by half the value at the time the packet was dropped and congestion avoidance begins. Wireless channels present an additional challenge in that losses do not generally occur in isolation, *i.e.*, wireless channels are often characterized by periods of fading in which several losses occur in succession. All versions of TCP are unable to gracefully recover from this situation and must resort to a timeout whenever more than one loss occurs per window of outstanding data. A timeout causes the TCP congestion window to reduce to one segment and the *ssthresh* (slow start can be performed until a threshold congestion window of size *ssthresh* is reached) to be reduced to half the congestion window.

A correctly designed link layer protocol should maintain high throughput by retransmitting only what is absolutely necessary, *i.e.*, only those packets which are dropped on the wireless link. In cases where error losses are high, retransmissions at any location other than the faulty link would result in reduced goodput over all links in the wired path from source to receiver. Arguments against link-layer retransmissions [21] claim that reliable links incur unnecessary transmissions due to the required transmission of acknowledgments in the reverse path (and therefore unnecessary power consumption resulting from transmission of the ACKs). Some previous work has even maintained that reliable link-layer approaches fail for large error rates on the wireless link because of competing retransmissions between the transport layer and the link layer [3]. However, during a TCP session, there are always TCP ACKs flowing in the reverse direction, and an efficient link-layer protocol can surely take advantage of this flow by piggybacking link-layer ACKs with transport layer ACKs. In addition, if the sending side of the link-layer protocol knows the exact state of the connection (at the link level) before retransmissions are performed, then only the missing packets are retransmitted, high goodput is maintained and bandwidth is preserved. Other recent approaches to improve TCP's performance over wireless networks [8] overcome the problem of redundancy in retransmissions by making the link layer protocol aware of the state of the TCP connection.

In this thesis, we show by example that a properly designed link layer (medium access and logical link control) implemented in the "wireless modem" (*e.g.*, WINGS [16], WaveLAN) can allow TCP to operate efficiently over wireless networks, with no changes to the hosts and TCP's semantics, and without requiring proxies between sender and receiver TCP. We present the transport unaware link improvement protocol (TULIP), which provides reliable delivery of packets from the link layer to the network layer. TULIP is *service-aware* in that it provides reliability for only those packets (frames) that require such service, but it is not *protocol-aware*, *i.e.*, it does not know any details of the particular protocol to which it provides its reliable service. TULIP never in any way tries to understand information in the protocol's packet header, but rather either receives instructions from the network layer indicating the type of service a packet should receive, or uses any information from the IP packet header (which would, for example, indicate the presence of a TCP packet). More specifically, TULIP provides reliable service for packets carrying TCP data traffic, and unreliable service for other packet types, such as UDP traffic (which includes routing table updates) and TCP acknowledgments. The receiver buffers packets and passes them up to the next layer in order, thereby preventing TCP from transmitting duplicate acknowledgments in the event that a packet is missing from the expected sequential packet stream. Flow control across the link is maintained by a sliding window, and automatic retransmission of lost packets is accomplished by the sending side's link layer. In addition, the protocol is specially designed for half-duplex radio channels by strobing packets onto the link in a turn-taking manner. We take advantage of TULIP's interaction with the MAC protocol to accelerate the return of link-layer acknowledgments (which are most often piggybacked with returning TCP acknowledgments) by using the MAC protocol to quickly return these packets, without re-negotiating access to the channel. TULIP causes no modification of the network or transport layer software, and the link layer is not required to know any details regarding TCP or the algorithms it uses. The link layer maintains no TCP state whatsoever, and makes no decisions on a TCP-session basis, but rather solely on a per-destination basis. From the transport layer's point of view, the path to the destination through the wireless link simply appears to be a slower link in the presence of packet losses due to channel errors and TCP simply adjusts accordingly.

We present the results of several simulation experiments showing that TULIP can dramatically improve the performance of TCP. Experiments assume a simple configuration with a base station and a single wireless host in order to compare the TULIP's performance against the performance of the Snoop protocol [8], which is a TCP-aware solution. However, in contrast with Snoop and TCP-aware solutions, TULIP is a general solution that can easily be extended for use with ad-hoc mobile networks and any type of reliable end-to-end protocol. The results of our simulations show that TULIP performs better for any bit-error rate than Snoop and TCP with no underlying retransmissions. In addition, at very high error levels, TULIP's throughput is up to three times higher than both Snoop and TCP with no underlying retransmissions. On the other hand, the end-to-end packet delay with TULIP is significantly lower than the other two approaches and, in contrast to Snoop, the standard deviation of delay with TULIP grows only slightly with increasing error rates. Our results have fundamental implications for ad-hoc networks in which no base station can be easily adopted, and for wireless technologies that are currently evolving.

The rest of this thesis is organized as follows. Chapter 2 provides a summary of the mechanisms used by TCP and then discusses prior approaches to improving TCP over wireless networks. Chapter 3 describes TULIP and gives an example of its operation. Chapter 4 shows the correctness of TULIP. Chapter 5 describes the implementation of TULIP used in our experiments. Chapter 6 discusses the simulation experiments used to analyze TCP's performance over wireless links subject to low and high bit-error rates, burst losses, and fading. Chapter 7 summarizes the work in this thesis and also suggests some ideas for future work.

2. Background and Related Work

2.1 Transmission Control Protocol (TCP)

TCP is the primary protocol used in today's Internet to achieve end-to-end reliability. TCP uses a congestion window, which dictates the amount of unacknowledged data that can be outstanding for a given session at any time, to achieve flow control. It provides for the retransmission of packets which have been dropped in the network due to errors or congestion. Different strategies regarding how and when to decide a packet is missing, and thereafter how the algorithm should respond, have been implemented; various versions of TCP are present in the Internet. In 1988 the Tahoe version was implemented in 4.3 BSD Tahoe TCP [17]. It included *slow start*, *congestion avoidance* and *fast retransmit*. In 1990 Reno was implemented in 4.3 BSD Reno TCP [26]. Reno modified Tahoe by implementing *fast retransmit* followed by *fast recovery*.

2.1.1 TCP Tahoe

Round-trip and Timeout Estimations

Original TCP performed a simple round-trip time (RTT) estimate, which was based upon slow increases or decreases to the current estimate, for every packet measured (a single packet was measured per window of outstanding data). The formula used was:

$$R \longleftarrow \alpha R + (1 - \alpha)M \tag{1}$$

where α was typically equal to 0.9. Using this calculation, RFC 793 [25] recommended the retransmission timeout value (RTO) be set to

$$RTO = R\beta \tag{2}$$

where β is a delay variance factor with a recommended value of 2. In addition, this value was rounded up to expire with a 500ms granularity, which is commonly referred to as TCP's coarse-grain timer.

In [17] Jacobson pointed out that the above approach fails to respond to wide fluctuations in RTT variance; in fact, $\beta = 2$ can adapt to loads of at most 30%. Therefore, the RTT and RTO calculation were modified to also include the variance in the RTT estimate. The subsequent equations to calculate the round-trip time and retransmission timeout are:

$$Err \equiv m - a \tag{3}$$

$$a \longleftarrow a + gErr \tag{4}$$

$$v \longleftarrow v + h(\,|Err| - v\,) \tag{5}$$

where a is the smoothed RTT, v is an estimator of the deviation, and m is the measured RTT. Recommended values for g and h are 0.125 and 0.25 respectively. Using Equations 4 and 5 the RTO is calculated as:

$$RTO = a + 4d \tag{6}$$

The TCP Timer

It is important to note that generally there is only one timer in use per session at a given time, i.e. only one packet is timed per window. Therefore, if a segment is transmitted while the timer is in use, then a timeout is not recorded for that segment. Timing is done by incrementing a counter every 500-ms when the TCP timer routine is invoked [26]. When a timeout value is calculated, the corresponding segment sequence number is also recorded. When an ACK for this segment arrives, the timer is turned off. Each time there is a timeout for a segment, an exponential back-off is applied to the RTO. The first time RTO will be multiplied by 2, then by 4 etc.

Karn's Algorithm

When a packet is retransmitted and an ACK is finally received back at the source, there is no way to determine to which actual transmitted packet the ACK corresponds. Therefore, in [19] Karn and Partridge specify that the RTT estimators should not be updated for retransmitted packets. In other words, the RTT estimator only applies to packets which are transmitted successfully the first time. Since ACKs are cumulative, if an ACK arrives with a higher value than the timed segment, the estimate is still performed (providing there were no losses for the included segments).

Slow Start and the Congestion Window

Congestion could be introduced into the network if TCP were to introduce an entire windowful of packets into the network at once. Instead, TCP is required to support *slow start*, an algorithm which causes TCP to inject packets at the same rate in which ACKs are returned by the other end. *Slow start* adds another window to the sender's TCP: the *congestion window* (`cwnd`), which is set equal to one segment when a connection is established or the *slow start* phase is entered [26] and never exceeds the receiver's advertised window. TCP uses a *slow start threshhold* (`ssthresh`) as a value to be used to determine if the *slow start* algorithm should be executed. Whenever an acknowledgment arrives, if `cwnd` < `ssthresh` then *slow start* is performed. Thereafter, each returning acknowledgment increases the window size by one segment, until eventually `cwnd` exceeds `ssthresh` . This leads to an exponential growth in the size of the congestion window. On a per session basis, the minimum value of the congestion window and the receiver's advertised window defines the maximum number of unacknowledged packets that can be outstanding at any time in the network.

Congestion Avoidance

Packets are lost for one of two reasons: they are damaged in transit or they are dropped by the routers as a result of congestion in the network. In wired networks (as opposed to wireless), packet damage is rare ($\ll 1\%$); therefore, at the time TCP was written it was assumed that the most likely cause of packet loss was due to congestion in the network. Jacobson proposed [18] to use the following *congestion avoidance* algorithm if TCP determines congestion exists in the network. *Congestion avoidance* limits the expansion of the congestion window by restricting it to a linear increase of $\frac{1}{cwnd}$ each time an acknowledgment is received (as compared to *slow start's* doubling of the window per round-trip time). *Congestion avoidance* is performed whenever `ssthresh` $>$ `cwnd`.

Fast Retransmit

Fast retransmit attempts to speed up the retransmission process by retransmitting segments which have not yet timed out; however, as a result of duplicate acknowledgments returned by the receiver, it can be inferred by the sender that a packet loss has occurred. After n duplicate ACKs have been received, the segment is retransmitted. The current recommendation for n is 3.

2.1.2 TCP Reno

TCP Reno incorporates all of the algorithms described above in the section on TCP Tahoe, except that *fast retransmit* is followed by *fast recovery.* Once the packet for which three duplicate acknowledgments have been received is retransmitted, *fast recovery* dictates that for each subsequent duplicate acknowledgment, the *cwnd* is incremented by one segment. This allows for the window to grow in an exponential manner (*i.e.*, as in *slow start*). The rationale for this approach is that because acknowledgments are returning, the congestion that caused the packet loss is mitigated. Once the ACK for the missing packet is received, `cwnd` $\longleftarrow$ `ssthresh` (where `ssthresh` is now equal to half the value of `cwnd` when the loss was first detected) and congestion avoidance begins.

2.2 Improving TCP's Performance in Wireless Networks

The quest to solve the ills of TCP over lossy wireless links is an area of active research. Solutions at lower protocol levels attempt to recover losses by using forward error correction (FEC) at the physical layer. FEC is generally considered to be a limited approach (although it remains an area of active research) as it can reverse only a limited number of bit errors, and also is costly in terms of packet delay due to computation time and power consumption in an environment in which power use must be minimized. In addition, while it may alleviate the poor performance of TCP by recovering a few errors, it cannot entirely solve the problem when losses are large. Solutions based on higher-level protocols attempt to fool TCP by hiding the lossiness of the wireless link. Previous solutions fall into three major categories: Link Layer, Split Connection, and Proxy.

The AIRMAIL protocol [4] provides a reliable link layer in conjunction with forward error correction(FEC). In this approach, the base station sends an entire window of data before an acknowledgment is returned by the mobile receiver. The rationale for this approach is to not waste bandwidth on ACKs and to limit the amount of work done by the mobile unit in order to conserve power. Unfortunately, a consequence of this approach is that there is no opportunity to correct errors until the end of an entire window, which can cause TCP to time out if the error rate is large or cause a large variation in delay depending upon the position of the loss in the window.

DeSimone *et al.* [3] conclude that introducing reliability at the link layer introduces unnecessary and redundant retransmissions, because of competing retransmission strategies between the transport and link layers. However, this conclusion was reached based on an analysis that did not take into account the very generous timeout value calculated by TCP nor its granularity of 500ms, but rather an ideal case in which a timeout occurs at the estimated round-trip time value. Balakrishnan *et al.* [7] demonstrate that, as it can be expected, link-layer protocols that fail to provide in-order delivery to the application essentially compete with the upper layers by duplicating retransmissions.

In the split-connection approach, the TCP connection is split between the source and base station and then between the base station and the wireless receiver [5]. The drawback to this approach is that it violates the semantics of TCP, and cannot therefore be easily deployed in the Internet.

In the proxy approach, a proxy is inserted between sender and receiver TCP hosts to help TCP's performance. A well-known example of this approach is the Snoop protocol, which is tailored to the case in which mobile hosts are attached to the Internet through a wireless link to a base station. The proxy at the base station is responsible for retransmitting and intercepting duplicate TCP acknowledgment packets. The proxy residing at the base station, referred to as Snoop [8], intercepts all packets entering an interface before the packets are passed on to IP. The Snoop protocol performs retransmissions of TCP packets when it detects a lost packet. The lost packet is identified by Snoop when TCP transmits a duplicate acknowledgment for a packet which Snoop has seen previously and has in its buffer. Snoop performs a retransmission after the second duplicate acknowledgment for the missing packet has been received. In addition, Snoop maintains two retransmission timers. The first timer is similar to the TCP timer and the same round-trip time estimate is performed. One packet per windowful of data is timed to obtain the round-trip time estimate. The second timer is a persist timer that senses idle periods. Snoop has been shown to improve TCP performance over wireless links with bit-error rates up to 15 bits per millions [8]; however, because Snoop relies only on the same cumulative acknowledgment as TCP and its own timeouts, losses are not recovered in a systematic fashion as Snoop must make reasonable guesses as to the pattern of losses. As we show in Section 6, this leads to large variances in delays at high loss rates. In addition, Snoop relies on the existence of a base station which must maintain state for all TCP sessions going through it.

3. Transport Unaware Link Improvement Protocol (TULIP)

Our design philosophy for TULIP was to provide a link-layer that is transparent to TCP, has no knowledge of TCP's state, takes advantage of TCP's generous timeouts, and makes efficient use of the bandwidth over the wireless link. The generous TCP timeouts can allow a properly designed link-layer to attempt to recover from losses due to noise before TCP notices; therefore, our conjecture was that TCP should perform well over a wireless lossy link if it is allowed to do what it was designed to do: adapt to the speed of a link that delivers packets in sequence, unless congestion occurs. TULIP is designed to improve the reliability of wireless links in either ad-hoc networks or base-station oriented wireless networks. Its design assumes a half-duplex channel between sender and receiver in which transmission errors are as likely to occur in acknowledgments (ACKs) as in data packets.

3.1 Service Provided

Although all link-level packets could be delivered reliably, this approach could unnecessarily increase delays across the wireless link; therefore, the service provided by TULIP consists of the unreliable or reliable transmission of packets according to the type of service requested by the network layer. The reliable service provided by TULIP consists of the delivery of reliable link-level packets (RLP) to the receiver in order and without duplication within a finite time. The unreliable service consists of a single transmission by the source of any unreliable link-level packet (ULP). In TULIP, TCP data packets are encapsulated in RLPs, while TCP ACKs with no data, UDP packets, and link-level ACKs are encapsulated in ULPs. TCP acknowledgment packets are transmitted with unreliable service, because the TCP ACKs are cumulative and there is generally more than one packet in flight at a time; therefore, any subsequent acknowledgment packet can cover for the loss of a previous one.

3.2 Basic TULIP Operation

The half-duplex nature of the wireless links over which TULIP is meant to operate dictates that the sender stop to listen for acknowledgments (ACKs) from the receiver. In contrast to prior reliable link protocols, in TULIP, the sender does not stop and wait to receive a correct ACK after either every packet (as in the traditional Stop-and-wait ARQ strategy [9]) or after an entire window of packets (*e.g.*, AIRMAIL [4]). Instead, the sender simply allows enough time between the transmission of two data packets for the receiver to send either an ACK or an ACK together with a data packet, which leads to the interleaving of packets from the two ends of the logical link on a turn-taking basis, as depicted in Figure 3.1. In the figure, a connection between nodes A and B is assumed to exist; packets from A to B or from B to A are labeled accordingly, and the signals between TULIP and the underlying MAC protocol are indicated.

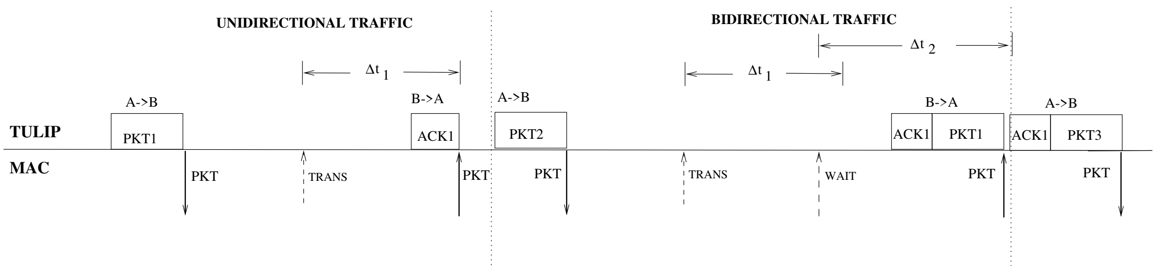


Figure 3.1: Packet interleaving in over half-duplex link in TULIP

The interleaving of data packets in TULIP requires each end of the link to pace its transmissions assuming a maximum propagation delay over the link (τ), and characteristics of the physical layer. For TULIP's implementation, we assume the specifications of the physical layer proposed in the IEEE 802.11 standard [1].

Our design allows at most one packet in transit at the MAC layer, which means that TULIP passes one packet at a time to the medium-access control (MAC) sub-layer. To accomplish packet interleaving, the service required by TULIP from the MAC layer is very simple, as Figure 3.1 indicates. First, TULIP requires that the MAC layer notify TULIP when to start a timer to wait for a minimum of Δt_1 seconds before sending the next packet.

This is indicated in Figure 3.1 by the MAC layer sending a signal denoted by TRANS, which notifies TULIP that the transmission attempt for the packet passed by TULIP has started. The timeout Δt_1 is given by the following equation:

$$\Delta t_1 = t_{PKT} + 3\tau + t_{ACK} + 2t_{TR} + 2t_c + t_p \tag{1}$$

where t_{PKT} is the time to transmit a data packet, t_{ACK} is the time to transmit a TULIP ACK (which could contain data), τ is the propagation delay over the wireless link, t_c is the time to get capture (including framing and preamble bits), t_{TR} is the radio ramp-up/ramp-down time, and t_p is the overhead and processing time at sender and receiver. Second, because either side may have variable-length data packets to send, and because such packets are much longer than simple ACKs, the MAC layer must inform TULIP of the need to wait longer than Δt_1. TULIP requires the MAC layer to specify the additional amount of time (Δt_2 in Figure 3.1) that a node sending packets needs to hold down beyond Δt_1, before sending the next packet. In the case of TDMA this time could be passed with the TRANS signal. However, because we are interested in contention-oriented MAC protocols (e.g., CSMA, DFWMAC [1], FAMA [14]), the MAC is assumed to pass a signal WAIT that allows the TULIP sender to wait long enough for a data packet to arrive before it sends the next packet. If CSMA and ALOHA were to be used, they would have to be modified to use priority ACKS [28], for example, in order to provide the WAIT signal needed by TULIP. This procedure of interleaving allows the two sources to be self-clocking during the bidirectional transfer of data over the link. The complete sender pseudo-code which describes the bidirectional sharing of the link is given in Figure 3.3.

Flow control and error recovery are managed by the sender through a sliding window of size W, depicted in Figure 3.2 that is essentially the same as the window of a typical selective repeat ARQ strategy [9]. Data packets are assigned sequence numbers modulo $2W$ and sender and receiver maintain a buffer of W packets to ensure correct operation [9]. The sender maintains a transmission buffer with all unacknowledged packets in its transmission window, and a retransmission list specifying the sequence numbers of those packets that must be retransmitted. The sender advances the low end of the window based on the highest sequence number that the receiver reports having received in sequence in an ACK; it advances the high end of the window with each new data packet it transmits, until the difference between the smallest and largest sequence number in the window equals W, at which time the sender retransmits from the beginning of the window in the next time slot.

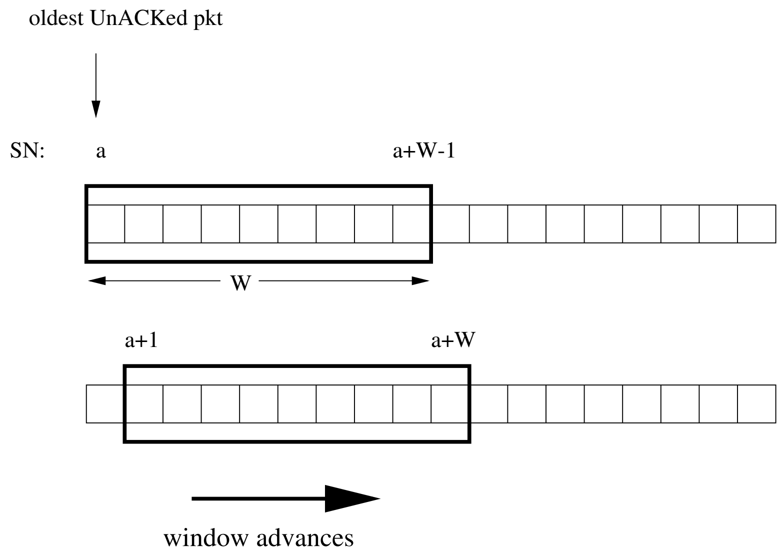


Figure 3.2: Sliding Window maintained by Sender

The Sender transmits data packets according to the algorithm shown in Figure 3.3. A packet transmission is triggered by a timer that expires every Δt_1 (see Figure 3.7) or by the reception of an ACK. At this time, Procedure send_packet() is used to choose the next packet to transmit. The highest priority is given to packets that are apparently lost, as indicated by the ACKs obtained from the receiver, and therefore the current retransmission list is always checked first. If, however, the retransmission list has been completely transmitted, or there is in fact no current retransmission list, then the next packet chosen for transmission is a new incoming packet (if there is one), provided that the transmission window is not exhausted. This is an important feature in TULIP, because the sender does not simply retransmit the last data packet whenever a correct ACK covering the packet fails to arrive. Instead, if the transmission window is not full, the sender transmits the next incoming

```
Definition of Terms
    ACK = received pkt is an ACK
    RTS = RTS received by MAC layer
    TRANS = MAC has acquired channel and
    pkt is to be transmitted
    macState = 1 if MAC layer has a packet,
    0 otherwise

[this procedure is called when the sender receives
a signal or ACK packet from the MAC layer]
procedure receive_from_MAC ( incoming_pkt or signal )
    begin
        switch packet or signal type
            ACK:
                cancel Timer T1
                process_received_ack()
                if(data to send)
                    send_packet()
            RTS:
                cancel Timer T1
                set timer T2 = Δt2(RTS.data_length)
            TRANS:
                set timer T1 = Δt1(mypacket.length)
                macState=0
    end process_receive_from_MAC

[this procedure is executed upon timer expiration]
procedure process_timer_expire
    begin
        if (data to send) AND (macState=0)
            send_packet()
            macState=1
        else
            return
    end process_timer_expire

[This procedure is called when the sender receives
an ACK from the sender.]
procedure process_received_ack ( incoming_pkt )
    begin
        if pkt.CumAck > oldestUnAckedPkt
            oldestUnAckedPkt = incoming_pkt.CumAck + 1
            advance transmission window
        end if
        if pkt.BitVector ≠ ∅
            free_received_packets ( incoming_pkt.BitVector)
            create new retransmission list
        end if
    end process_received_ack

procedure send_packet ( )
    begin
        if (untransmitted packets remain in Retransmission list)
            send next pkt in list
            else if (Window is not exhausted) AND (new pkt available)
                    if (new pkt is RDP)
                        update Window and structures
                        send new packet
                    else if (new pkt us URDP)
                        send new packet
                else if (Retransmission List exists)
                    retransmit first pkt in list, i.e. start over
                    else (no retransmission list)
                        retransmit oldest unacknowledged packet
    end send_packet
```

Figure 3.3: Complete Sender Algorithm

packet. The worst case is when there is no retransmission list available and there is no room in the transmission window to send new packets. At this point, in order to preserve liveness of the protocol, the sender retransmits the oldest unacknowledged packet in its window, which eventually forces an ACK back from the receiver.

The algorithm followed at the receiver is shown in Figure 3.5. The receiver buffers all out-of-order data packets it receives. Upon reception of a correct data packet, the receiver passes all data packets received in order to the network layer. The receiver sends an ACK every time it receives a data packet. To improve throughput, ACKs in TULIP consist of both a sequence number and a bit vector of length $W-1$. The sequence number, which we call cumulative acknowledgment (CACK), specifies the highest sequence number of a data packet received in sequence. The bit vector is a succinct way to specify negative acknowledgments (NACKs) and is taken from prior well-known selective-repeat protocols (e.g., XTP [27]). Starting with the sequence number specified in the CACK, the bit vector notifies the sender about the successful or unsuccessful reception of up to $W-1$ additional data packets. The bit vector is represented as:

$$\overline{B} = \{b_0, b_1, \ldots, b_n\}$$

where

$$b_i \in \begin{cases} 0 & \text{if packet has not been received correctly} \\ 1 & \text{if packet is correctly received} \end{cases}$$

and the complete acknowledgment is depicted in Figure 3.4.

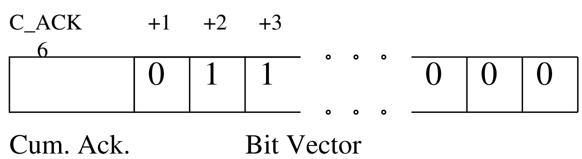


Figure 3.4: Acknowledgment from Receiver

TULIP uses a connection establishment strategy that is much the same as those reported in the literature for selective repeat protocols [9].

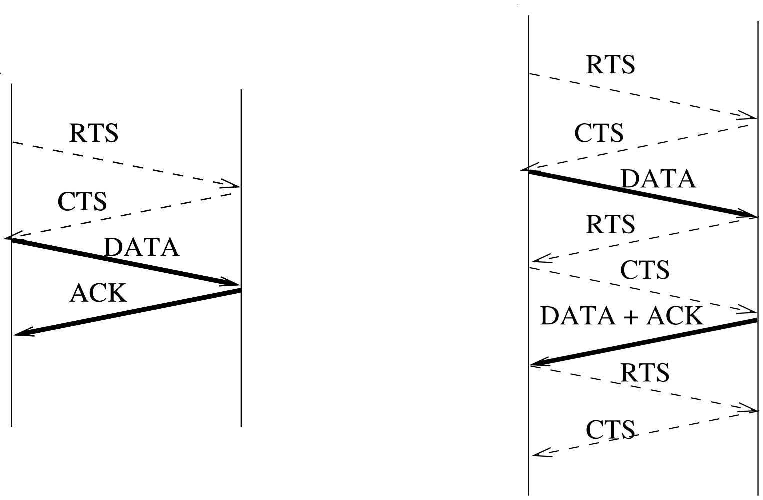
3.3 MAC-Level Acceleration

For the case of reservation-oriented MAC protocols, transmission delays can be reduced by a closer interaction between the MAC protocol and TULIP. In particular, protocols based on four-way handshakes of the form: request-to-send (CTS), clear-to-send (CTS), data, and MAC-level ACK can be improved. The four-way handshake common to protocols such as DFWMAC [1] is effective only in one-way traffic scenarios, in which the MAC-level ACK can be used to also transfer a TULIP ACK (or the ACK for any other link-control protocol). However, in two-way traffic scenarios, the MAC-level ACK is unnecessary, because a round of RTS-CTS handshakes has to take place for both sides of the logical link to transmit data.

Our implementation of TULIP runs on top of FAMA-NCS [14] and includes a MAC-acceleration feature, depicted in Figure 3.6(a), aimed at reducing link delays. In essence, after a data packet has been received by FAMA and passed up to TULIP, TULIP informs FAMA of the size of the packet it now needs to send. If the packet payload is 40 bytes or less (which is large enough to carry a TCP ACK packet), then the packet is piggy-backed with the TULIP ACK, which in turn would be encapsulated in the MAC-level ACK sent as part of a four-way handshake in DFWMAC. (FAMA-NCS does not provide a MAC layer ACK). On the other hand, if the packet is larger than 40 bytes, FAMA is instructed to immediately send an RTS to reserve the channel for a long packet, as shown in Figure 3.6(b). This permits FAMA to reassign the channel quickly, without having to assume worst-case long packets, and also reduces the number of MAC-level control packets.

```
[this procedure is called when a pkt is received]
procedure process_incoming_pkt ( incoming_pkt )
  begin
    set bit in Bit Vector corresponding to received sequence num.
    calculate new CumACK
    if (noDataPkt in Queue)
      prepare_ACK_pkt( CumACK, BitVector )
      send_ACK_pkt( sender_address )
    else
      prepare_PiggyBack_ACK_pkt( CumACK, BitVector, DataPkt )
      send_PiggyBack_ACK_pkt( sender_address, Piggyback_dgParms )
    end if
    if any pkts are in-order and can be passed up to higher layer
      pass up packets
    end if
  end process_received_ack
```

Figure 3.5: Complete Receiver Algorithm



(a) (b)

Figure 3.6: (a) MAC Acceleration: FAMA transmits TULIP data packet and returns ACK without another RTS/CTS exchange. Returning TULIP ACK may contain a TCP ACK. (b) FAMA exchange with large ACK packet, *i.e.*, encapsulated data packet, requires another RTS/CST exchange.

MAC acceleration requires a level of communication between the link-layer and the MAC layer. Specifically, the MAC layer must provide the link-layer with two signals:

- **TRANS** indicates that the MAC layer has captured the channel and the data packet received from the link layer is about to be transmitted.
- **WAIT** indicates that the MAC layer has received an RTS, and passes the sender address and the size of the packet which will occupy the channel.

With MAC acceleration, the interleaving of data packets and ACKs for unidirectional traffic, depicted in Figure 3.7 with the label **Unidirectional Traffic**, is straightforward: The link-layer passes a packet to the MAC layer which causes the MAC layer to initiate an RTS-CTS handshake. When the MAC layer has successfully negotiated the channel, it sends the signal **TRANS** to the link-layer which sets its timer T_1 equal to the value Δt_1 calculated by Equation 1. Upon receiving the signal **TRANS** the sender must wait for the data packet to be transmitted and propagate to the receiver and also for the receiver to get capture.

With MAC Acceleration, an additional RTS/CTS exchange is not needed to transmit the corresponding ACK, so Δt_1 includes only the ACK packet transmission time, propagation, and capture for the sender. Both sender and receiver incur an additional turnaround time as they switch from send to receive mode and receive to send mode, respectively. When the ACK for the data packet arrives, timer T_1 is cancelled and the link-layer passes the next packet to the MAC layer. However, if the ACK is lost, or if the data packet was lost, timer T_1 expires and the sender algorithm, described in Figure 3.3, selects the next packet for transmission.

It is shown in Figure 6.4(a) of our simulation results (Section 6), that with no errors on the link TULIP provides higher throughput than Snoop precisely because of the interaction between TULIP and the MAC protocol, which leads to a much better scheduling of the channel.

3.3.1 Bidirectional traffic

In this section we discuss support for bidirectional traffic over half-duplex radio links by interleaving traffic streams in both directions. This approach is important because if we do not provide any type of scheduling, then the opposing data streams will constantly compete for the channel, causing RTS packets to collide, and as a result unnecessarily waste bandwidth and increase delays.

Assuming a bidirectional data flow between two sources A and B, the portion of Figure 3.7 labeled **Bidirectional Traffic** shows the timing that occurs at the link layer in order to interleave packets, as well the resulting data flow at the MAC layer, from the perspective of a Sender A.

To interleave bidirectional traffic streams the MAC layer sends an additional signal WAIT to the link-layer when the MAC layer receives an RTS signal. An additional timer, T_2, triggered by the signal WAIT, is set for Δt_2, where Δt_2 must last long enough to receive a CTS, an incoming packet of length specified by the RTS, plus the necessary capture and turnaround times discussed previously.

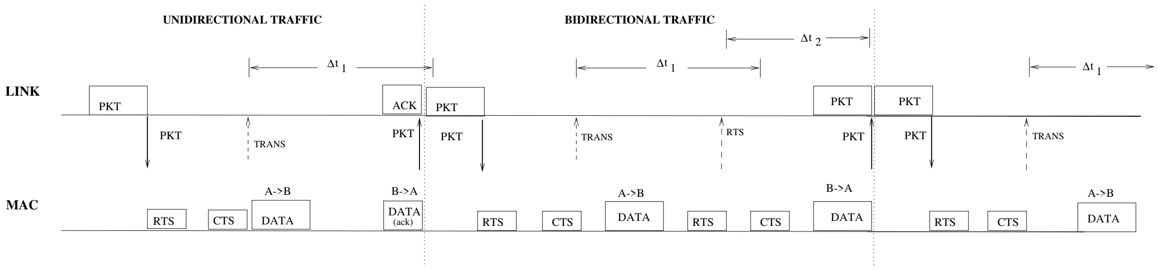


Figure 3.7: Unidirectional and bidirectional Traffic from the perspective of Node A in a logical link with Node B.

The timing for the bidirectional case begins in the same way as the unidirectional case; however, once the link-layer at B receives a packet from A (labeled as A→B), instead of building and returning an ACK packet, it instead passes a data packet which is encapsulated in a TULIP ACK to the MAC layer. Upon receipt of this packet, the MAC layer at B must send an RTS to negotiate the channel. Once this RTS propagates to A, the MAC layer sends the WAIT signal to the link-layer. The receipt of the WAIT indicates to Sender A that there is a bidirectional data stream and instead of passing the next packet to the MAC layer, sender A instead cancels timer T_1 and sets timer T_2 to a timeout value determined from the packet length specified in the WAIT signal. Once the data packet from B has been received or the timer expires (in the event A was not the recipient of the packet), sender A can pass its next packet to the MAC layer and the process is repeated. As the diagram indicates, no special control signal is needed to indicate a bidirectional stream; each link simply adapts as it receives either WAIT or TRANS signals.

3.4 Packet Headers

In accordance with the types of data traffic passed through TULIP, Figure 3.8 shows the packet headers used by the protocol.

Packet headers (a) through (c) are used for unreliable data packets(URDP), reliable data packets(RDP), and acknowledgment packets(ACK), respectively. The packet header shown

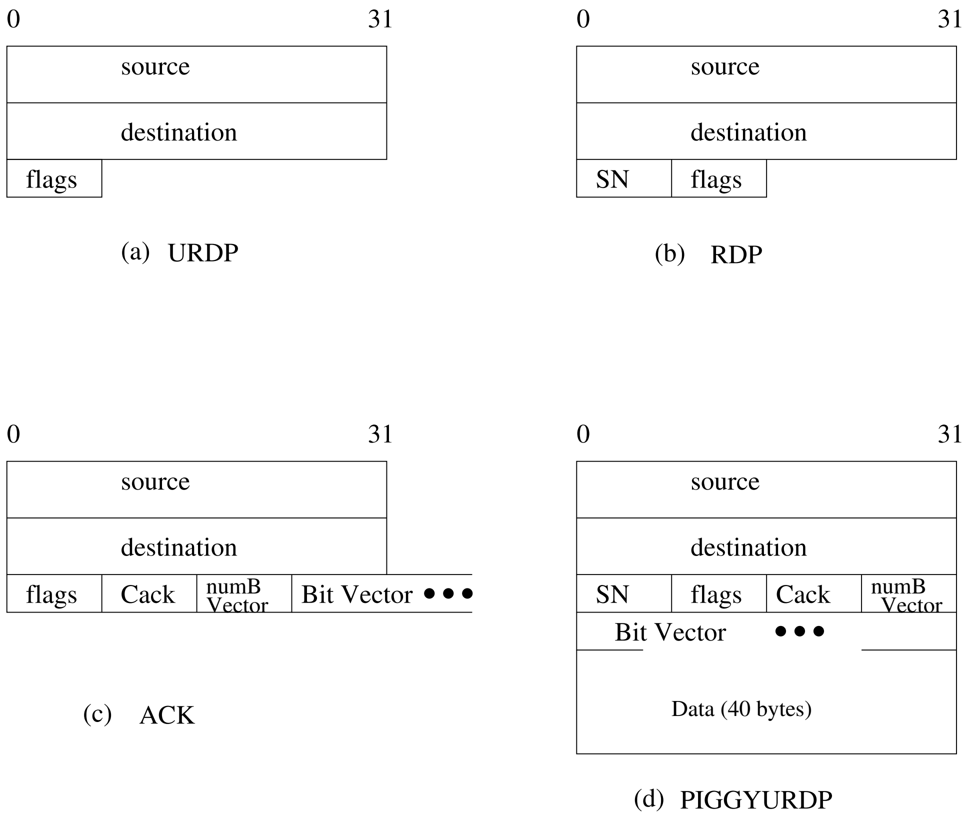


Figure 3.8: Packet Headers used by TULIP

in (d) is a piggy-back packet(PIGGYURDP), which may encapsulate a TCP acknowledgment packet of 40 bytes. The meaning of the individual packet fields is the following:

- **source**: IP address of source of packet
- **destination**: IP address of destination of packet
- **SN**: sequence number
- **Cack**: cumulative acknowledgment
- **numB Vector**: number of bytes in bit vector
- **BitVector**: bit vector
- **flags:** packet type (RDP,ACK,URDP,PIGGYURDP)

In general, the number of bits in the bit vector is equal to the window size; however, if less than a windowful of data is outstanding, it is not necessary to transmit the maximum size bit vector. Therefore, the number of bits in the vector is not necessarily equal to the window size. It should be noted, however, that because of the low bandwidth-delay product of the link and the desire to quickly retransmit packets within the window without causing a TCP timeout, the typical size of the sender's window is 8 packets (producing a one byte bit vector).

3.5 Transmission Example for Unidirectional Traffic

A sample transmission session is given in Figure 3.9 to illustrate TULIP's use of the cumulative acknowledgment, bit vector and the retransmission list, using unidirectional traffic for simplicity. On the left of the time line the data packets are transmitted by the base station, and on the right side acknowledgments are returned by the mobile receiver. On the sender side the retransmission list created by the Base Station is shown as $R[sn_i \cdots sn_n]$, indicating the sequence numbers that must be retransmitted. A number followed by a star indicates a packet that has already been retransmitted. On the receiver side, the cumulative acknowledgment is to the left of the bit vector that is contained in brackets. A 1 indicates a correctly received packet and 0 indicates a missing packet. To the right of the bit vector, an up arrow is followed by the sequence number of packets that are in order and can be passed up to the higher-layer protocol.

At time t, packet #7 is transmitted successfully. The receiver returns a cumulative acknowledgment of 7 and a one byte bit vector indicating no holes in the sequential stream. Packet #9 is transmitted at time $t+2$ and it is lost on the channel. At time $t+3$, the transmission timer expires when the acknowledgment for packet #9 fails to arrive. At this point, a new packet is available in the input queue, and packet #10 is transmitted accordingly. When the sender receives an ACK at time $t+4$, the bit vector indicates that packet #9 is missing. Therefore, the sender creates a retransmission list and retransmits packet #9; however, this packet is again dropped on the channel. When the transmission timer expires

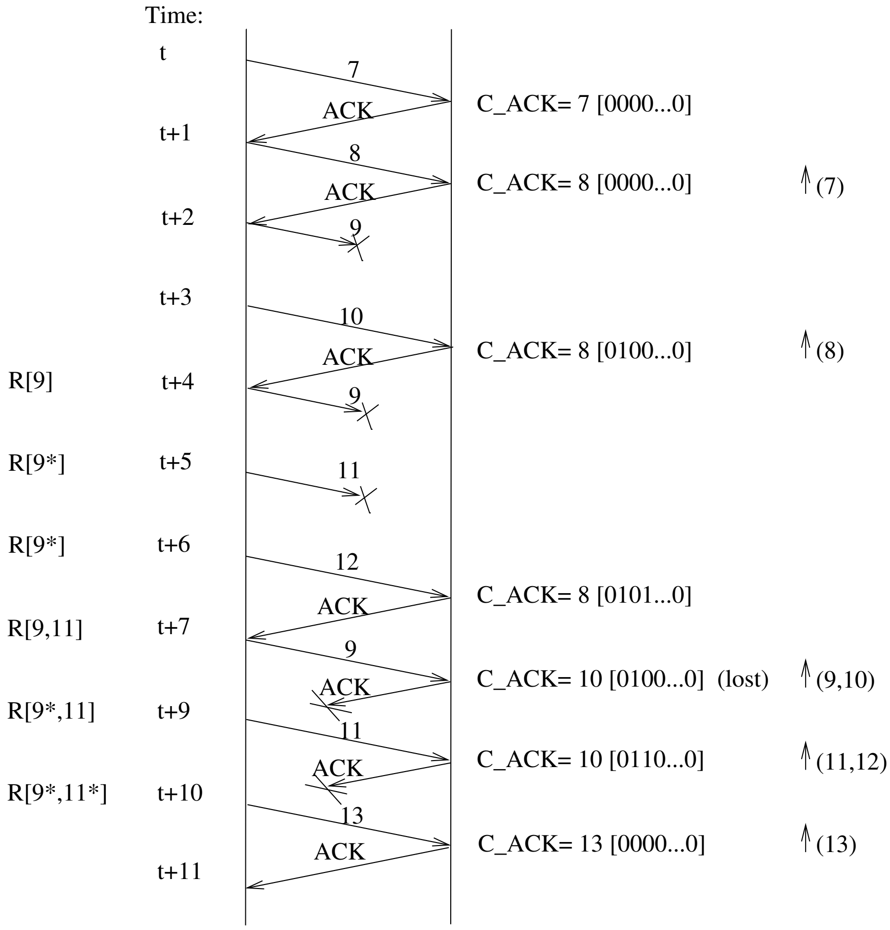


Figure 3.9: Example of Transmission. Window Size = 8.

at time $t+5$, the retransmission list indicates that the entire list has been retransmitted and a new packet, #11, is selected, transmitted, and subsequently lost. Packet #12 is transmitted at time $t+6$; however, this packet is successful and an acknowledgment is received at time $t+7$ which indicates packet #9 is still missing and also #11. A new retransmission list is created and the packets are retransmitted at times $t+7$ and $t+8$ respectively. This time the data packets are received correctly, however the corresponding acknowledgment packets are lost. Once packet #9 is received, the receiver can now pass packet #9 and #10 up to the next layer and when packet #11 arrives, packets #11 and #12 can be released. At time $t+10$ the current retransmission list has been completed so a new packet (#13) is selected for transmission. When its acknowledgment is received at time $t+11$ the bit vector indicates that all packets have been received correctly.

4. Correctness of TULIP

In this chapter we show that TULIP is both Safe and Live. A protocol is safe when it provides the "correct" results. In the case of ARQ protocols, safety implies that the receiver never releases packets to the network layer that are out of order. In addition, the protocol does not release multiple copies of the same packet, nor does it allow for the omission of packets. An ARQ protocol is live as long as it supplies the network with the correct results as long as the network layer requires the service, which implies that the protocol is free of deadlocks. The proofs follow the method presented by Bertsekas and Gallager [9].

The pseudocode covering TULIP's packet selection algorithm is given in Chapter 3. The sender's window is of size W and is managed through two sequence numbers, SN_{min} and SN_{max} which are maintained mod n, where $n = 2 * W$. In other words, the range of SN_{min} and SN_{max} is $\{0, 1, \cdots, 2W{-}1\}$. Let S be an ordered and increasing set of modulus numbers such that, $S = \{SN_{min}, \cdots, SN_{max}\}$. SN_{min} and SN_{max} are updated as follows:

Sender Window:

1. Initialize SN_{min} and SN_{max} to 0.
2. A new packet can be transmitted iff $|S| < n$. Assign $sn = SN_{max}$ to the new packet. Set $SN_{max} = (SN_{max} + 1) mod\ n$.
3. If an acknowledgment is received with C_{ACK} and $C_{ACK} \in S$, then SN_{min} is updated to $(C_{ACK} + 1) mod\ n$.

The receiver maintains a buffer of size at least W to accomodate the sender's entire window. In addition, it manages a cumulative acknowledgment field, C_{ACK}, which represents the highest sequence number that has been received in order, and a bit vector which indicates holes in the sequential traffic stream. The receiver buffer is updated as follows:

Receiver buffer:

1. Initialize $C_{ACK} = -1$ and $\overline{B}$ to a vector of all zeros.
2. Any packet from the sender with sn in the range of increasing modulus numbers $C_{ACK}{+}1, \cdots, C_{ACK}{+}n$ is accepted into the buffer. If sn is equal to $(C_{ACK}{+}1) mod\ n$,

release the packet and any subsequent in order packet to the network layer. Update C_{ACK} to the sequence number of the last released packet. Shift left bit vector for each packet released to network layer.

The proof of correctness for TULIP assumes the following:

- The system is correctly initialized. This implies that initially there are no frames in transit on the link. In addition, the sender is initialized such that $SN = 0$ and the receiver is initialized with $C_{ACK} = -1$ and $\overline{B}$ is a vector of all zeros.
- All frames with transmission errors are detected by the CRC at the MAC layer and are not passed to the link-layer.
- At some time t, a frame is transmitted and received error-free with probability q, where $q > 0$.
- Frames do not get out-of-order on the link, i.e. the channel provides FIFO delivery of frames.
- The receiver can buffer at least W packets.

The proof of correctness for TULIP proceeds in two parts. First, Theorem 1 shows that TULIP provides in order delivery of packets from source to destination with no repetitions or omissions. Next we show that TULIP operates free of deadlocks as long as the network layer requires its service.

Lemma 1: *Sequence numbers transmitted mod $2W$ provide a unique and distinguishable numbering mechanism.*

Proof: Assume that sender and receiver label packets with consecutive sequence numbers $0, 1, 2, \cdots$ and that the sender keeps track of its window with SN_{min} and SN_{max} and that the receiver returns a C_{ACK} as described above.

At time t_1 in Figure 4.1 the sender has transmitted packet i. According to the sender's rules for advancing its window, in order to transmit packet i, an acknowledgment covering packets up to and including packet $i - W$ must have been received previously. Therefore, at time t_1 the sender can have no packets in its buffer with sequence numbers less than $i - W - 1$. This implies that retransmitted packets can arrive at the receiver with

$$sn \geq i - [W - 1] \tag{1}$$

At time t_3 the acknowledgment C_{ACK} arrives at the sender covering packet i. Because C_{ACK} only covers through packet i, subsequent packets transmitted by the sender are limited to

$$i < sn \leq i + W \tag{2}$$

Combining equations 1 and 2 gives the range of possible sequence numbers that the receiver could expect to receive:

$$i - [W - 1] \leq sn \leq i + W \tag{3}$$

We see that the range of numbers that could potentially arrive at the receiver is 2W, and therefore ambiguity is not possible if the sequence numbers are stored and labeled modulo 2W. □

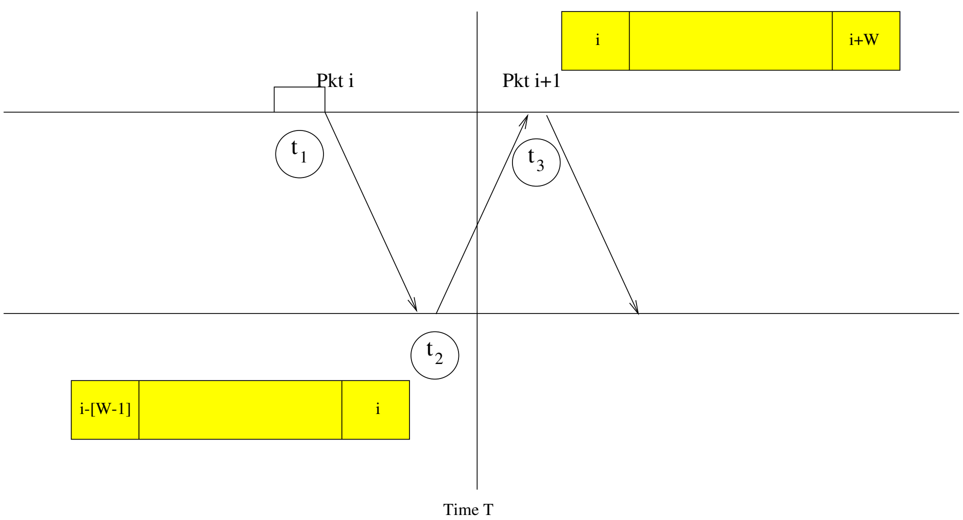


Figure 4.1: Sequence numbers from the perspective of receiver and sender.

Theorem 1: *TULIP is safe.*

Proof: The proof is by induction on the packet sequence numbers. The first packet that the sender transmits is numbered 0; likewise, the first packet expected by the receiver is also packet 0. The receiver cannot release to the network layer any packet with a sequence number higher than 0, until it has received a group of packets that are in order and begin with sequence number 0. Following the sender's algorithm presented in Figure 3.5, the sender retransmits packet 0 either upon receipt of an acknowledgment packet for a subsequent packet (which will indicate the missing packet in the bit vector $\overline{B}$), or when it reaches the end of its window and must retransmit the first packet in the window, namely packet 0. By Lemma 1, there is no ambiguity in sequence numbers of the retransmitted packes; therefore, with probability q, packet 0 must arrive error-free at the receiver and be released to the network layer. It follows that TULIP is safe for the first packet.

Assume that TULIP is safe up to the nth packet, i.e., the receiver has correctly delivered all packets from 1 to n in order and without repetitions or deletions. According to the receiver's algorithm, the next packet that the receiver can release to the network layer is the packet with sequence number $nmod2W$. When the first error-free packet witht this sequence number arrives at the receiver, it is released to the network layer. An acknowledgment is then generated to cover all in order packets which are subsequently released to the network layer and C_{ACK} is increased to the last packet released and transmitted to the sender. No replica of the same packet n can then be released again. □

Theorem 2: *TULIP is live.*

Proof:

To verify the liveness property we show that TULIP is deadlock-free for as long as the network layer desires its service. Note that the protocol is only required to operate when there are packets available from the network layer. Figure 4.2 shows the timing that will be used to demonstrate liveness, and shows the sequence numbers of packets as they are transmitted by the sender and the corresponding values of C_{ACK} returned by the receiver via acknowledgment packets. Let t_1 be the time at which packet i is first transmitted by the sender; this implies that $SN_{min} \leq SN_{max} = i$. Packet i and returning acknowledgments can undergo several losses; however, at time t_2 packet i is received error-free and released to the network layer. Let $t_2 = \infty$ if this event never occurs. Let t_3 be the time when the sender receives an acknowledgment covering packet i and advances the low end of the window to $SN_{min} > i$. Let $t_3 = \infty$ if this event never occurs. We proceed to show that t_3 is finite, and that $t_1 < t_2 < t_3$.

Let $SN_{min}(t)$, $SN_{max}(t)$, and $C_{ACK}(t)$ denote the values of SN_{min},SN_{max}, and C_{ACK} at time t, respectively. Because the channel has a finite propagation delay and since time t_1 is the time when packet i is first transmitted, it is impossible for the receiver to release this packet to the network layer at any time before t_2. It follows that $t_1 < t_2$.

According to the sender's algorithm, SN_{min} is updated to $C_{ACK} + 1$ after an acknowledgment in which $C_{ACK} \in S$ is received; therefore, by modulus counting and Lemma 1, $SN_{min}(t) \leq C_{ACK}(t) + 1$. The receiver can send a $C_{ACK} = i$ only at time t_2. The receipt of this acknowledgment allows the sender to advance the low end of its window to $SN_{min} = i + 1$; it follows that $t_2 < t_3$.

The sender's algorithm is such that upon the expiration of the transmission timer, a packet is always chosen for transmission. In addition, this timer is active whenever there is new data to be sent or when there are previously transmitted packets outstanding in the sender's window. Because the transmission or subsequent retransmission(s) of packet i has a finite probability of success $q > 0$, the time interval (t_1, t_2) is finite. Likewise, the transmission of acknowledgments is also prompted by the arrival of data packets; therefore, acknowledgments will be transmitted in finite intervals corresponding to the arrival of data packets. Acknowledgments also have a finite probability of success q, and therefore the time interval (t_2, t_3) is finite. Because $t_1 < t_3$ and $t_2 < t_3$, t_3 is finite. $\Box$

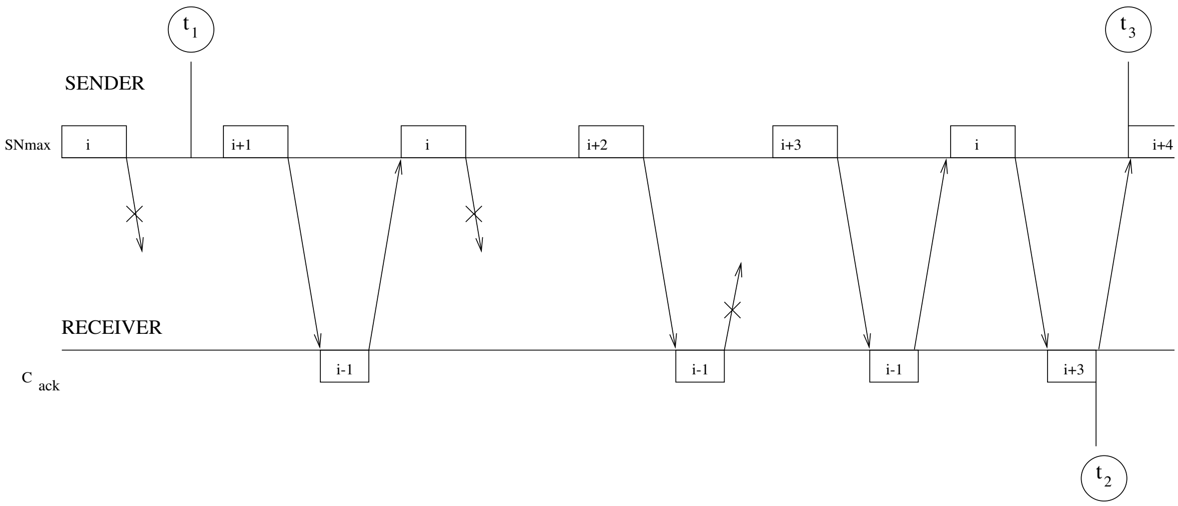


Figure 4.2: Liveness scenario.

5. Implementation

We have implemented TULIP and Snoop [8] in the C++ Protocol Toolkit (CPT) [10]. Wireless nodes in our simulation run the protocol stack shown in Figure 5.1. A key feature of our simulation is that it is based on the exact same source code that runs in the WING prototypes (which are wireless IP routers) [16], and in hosts attached to the WINGs. The IEEE 802.11 specifications are used at the physical layer to simulate the broadcast medium. CPT simulates the wireless and wired transmission media with specific parameters and channel characteristics specified through script files read at runtime. Various parameters such as propagation delay, bit error rate, and offered load can be altered and are also specified through scripts. Either TULIP or a dummy link-layer that does not perform retransmissions is used at Layer 2. Our implementation of TULIP runs on top of FAMA-NCS [14] with the MAC acceleration feature described in Chapter 3. TULIP in turn interacts with IP [24] and the wireless Internet routing protocol (WIRP) [23] for packet forwarding. WIRP control packets are treated by TULIP as any other UDP packet (*i.e.*, as unreliable link packets). The only higher-level protocol requesting reliable link service from TULIP is TCP, which in our implementation, has been ported from the TCP Reno code contained in the REAL simulator [20].

The code of the Snoop protocol [8] was modified from the on-line FreeBSD implementation[1] to run in CPT. Logically, Snoop is at the network layer as it monitors packets at the incoming interface and then passes them to IP. The code was maintained in its original on-line form and only when absolutely necessary were data structures specific to the FreeBSD implementation modified for use with CPT; however, hard coded values for the minimum timeout values for the retransmission timer and the persist timer were changed to 40ms and 200ms, respectively, and the retransmission timer's calculated value was set to $2 \cdot rtt$ to match that of published work [8]. For optimization purposes retransmissions are performed after the receipt of 2 duplicate acknowledgments [6]. In addition, as specified [8], only

[1] On-line source code available at ftp://daedalus.cs.berkeley.edu/pub/snoop/

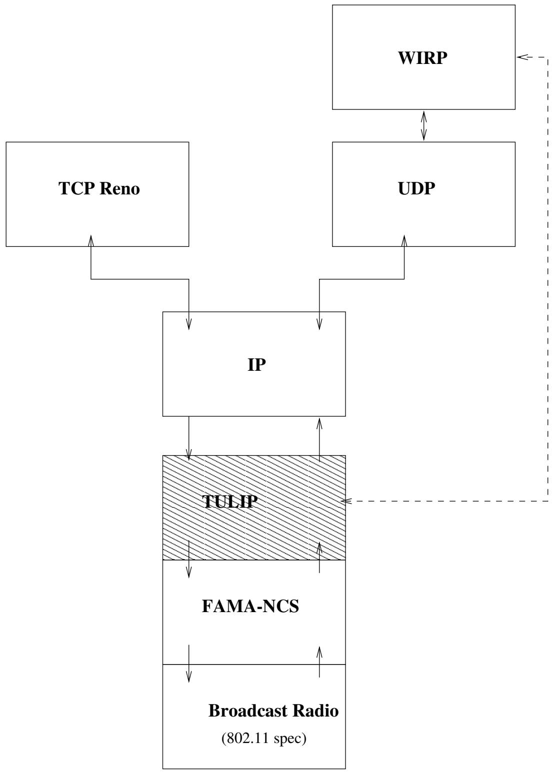


Figure 5.1: Protocol stack of wireless node.

one *rtt* estimate is made per transmission window. For Experiment 2, the on-line version of Snoop was changed to fix a minor coding error. The algorithm followed by the Snoop protocol is described in Chapter 2.2.

Although TULIP can operate in ad-hoc networks, the following section concentrates on experiments that assume the last hop through a base station, because our intent is to compare TULIP's performance against Snoop's, which is one of the best performing TCP improvements for wireless networks reported to date.

6. Performance

In this section we present the results of a series of simulation experiments used to compare the performance of TULIP, the Snoop protocol [8] and TCP implemented with no underlying link-layer protocol. We chose to compare TULIP against Snoop because the latter has been shown [8] to give very positive results in terms of throughput at low to moderate error rates in the radio link. For both steady-state and in the early stages of a TCP data transfer, the performance metrics of primary interest are throughput, delay and goodput. Goodput is the ratio of successfully transmitted packets (*i.e.*, those accepted by the link layer to pass up to the transport layer) to the total number transmitted and is an important measure because it reveals if a protocol makes unnecessary retransmissions.

Because the Snoop protocol relies on the existence of a base station where the Snoop module resides, the test case for the simulation experiments is the last-hop case, where an Internet source sends data to a wireless host via a base station, as shown in Figure 6.1.

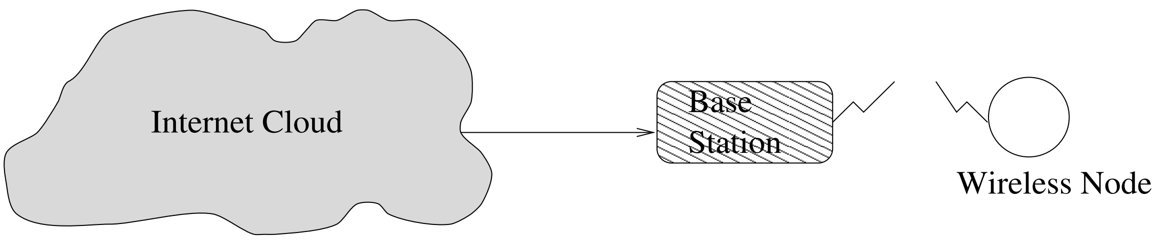


Figure 6.1: Logical view of simulation configuration.

Of primary interest is traffic flowing toward the wireless receiver, as it is the case typical of Web traffic. Accordingly, the simulation topology used in all experiments is shown in Figure 6.2. The topology is such that a TCP source transfers 10Mbyte of data from a host located on an Ethernet segment via a base station to a wireless receiver two hops away (one wired hop and one wireless hop). All experiments are run using either TULIP, a dummy link layer that performs no retransmissions (referred to as No LL) in the link layer, or the Snoop protocol. Because we are interested in TCP's reaction to losses in the wireless link, we have insured that there are no losses due to congestion by making the queue at the link-layer larger than the TCP receiver's advertised window. In Experiments 1 and 2, the TCP receiver window is varied and losses are applied to the wireless channel with an exponential distribution to examine the performance of the retransmission policies of TULIP and the Snoop protocol, and to determine their affect on TCP. Experiment 1 examines low to moderate error levels and Experiment 2 examines performance as the error rate is increased to very high levels. Experiment 3 examines performance when Raleigh fading [29] and burst losses are present on the channel.

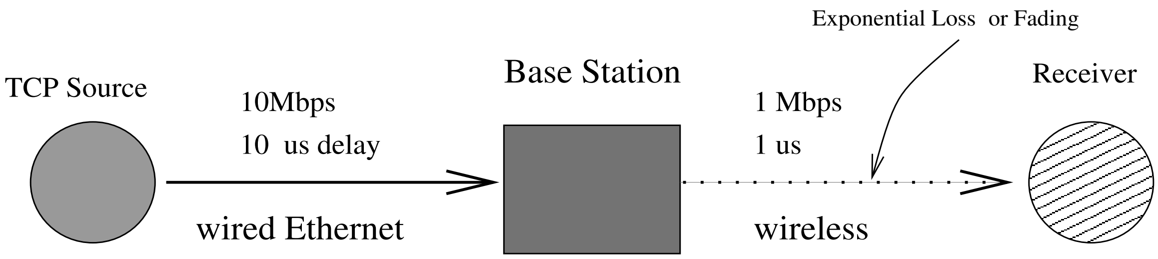


Figure 6.2: Topology for Experiments 1 – 3

Table 6.1 gives the simulation parameters that are fixed for all experiments. These specifications are the same as those used in the Snoop experiments [8], except for the radio characteristics and a lower transmission rate for our experiments.

6.1 Experiment One: Low Error Rates

In this experiment the bit error rate is varied from 0 to 15 bits/million and the TCP receiver window is varied between 42kbytes and 16kbytes. TCP's behavior is examined in detail to gain insight into its operation during moderate loss periods. The throughput, packet delay, and goodput of all the protocols are also examined.

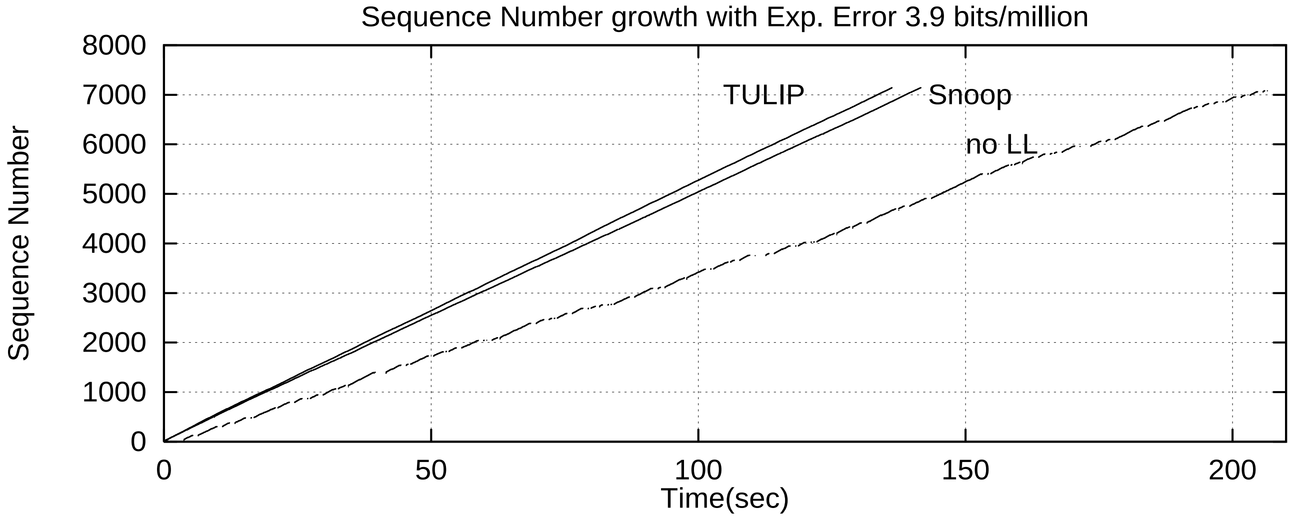
Parameter	Value
Physical Layer	802.11 specifications
Wired channel capacity	10Mbps
Wireless channel capacity	1Mbps
Transfer size	1Mbyte
TULIP window	8 packets
Offered Traffic	Poisson, 1Mbps
Packet Size	1400 bytes

Table 6.1: Fixed Simulation Parameters for Experiments 1 – 3

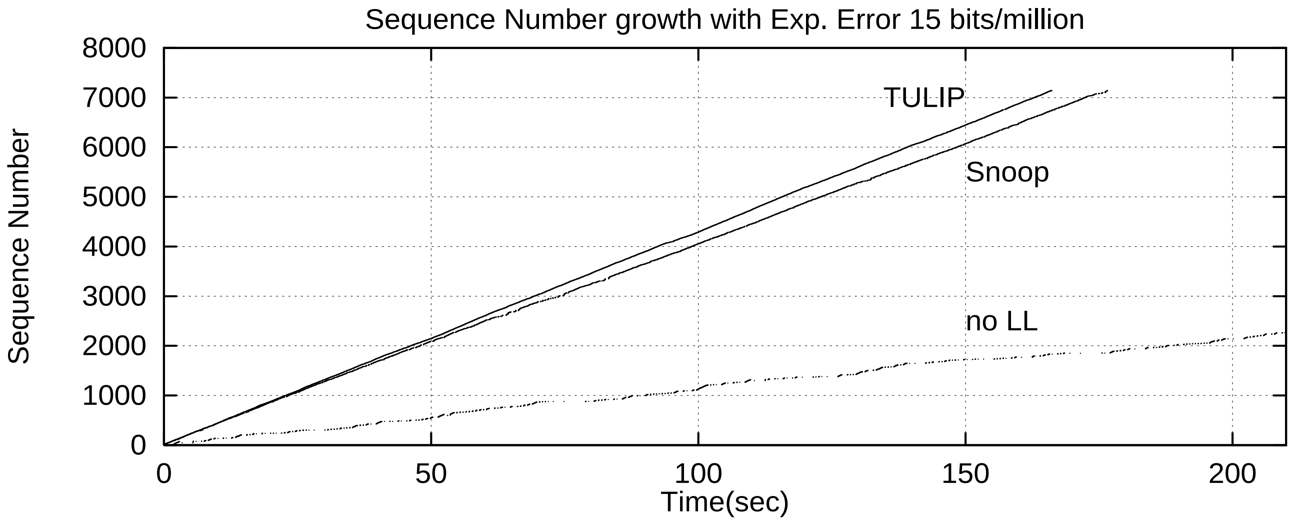
6.1.1 Throughput

Figure 6.3(a) shows the progression of sequence numbers for a 10Mbyte TCP file transfer with a BERR of 3.9 bits per million. Though this is in fact a modest error rate, corresponding to a packet error rate of approximately 4%, it is evident that the TCP session with no assistance from the link layer (the no LL line) suffers from timeouts and as a result degraded throughput. The plot also shows that TULIP performs slightly better than the Snoop protocol, most likely because Snoop is waiting to retransmit a lost packet until it has received 2 duplicate acknowledgments, whereas TULIP retransmits on the first indication of a loss. In addition, TULIP shows improved throughput because it returns its ACKs from the receiver to the sender with MAC Acceleration, described in Chapter 3, whereby the underlying channel access protocol (FAMA) returns link-layer acknowledgment packets (often piggy-backed with TCP ACK packets) without requiring an additional RTS/CTS exchange. When the Snoop protocol is running at the network layer, the TCP acknowledgment packets from receiver to sender must of course compete with the data packets flowing in the forward direction and compete for access to the channel.

Figure 6.3(b) shows the progression of sequence numbers with a BERR of 15 bits/million, or a packet loss rate of approximately 16%. It is again apparent in this transfer that TCP is suffering considerably due to losses on the wireless link. TULIP and the Snoop protocol, on



(a)

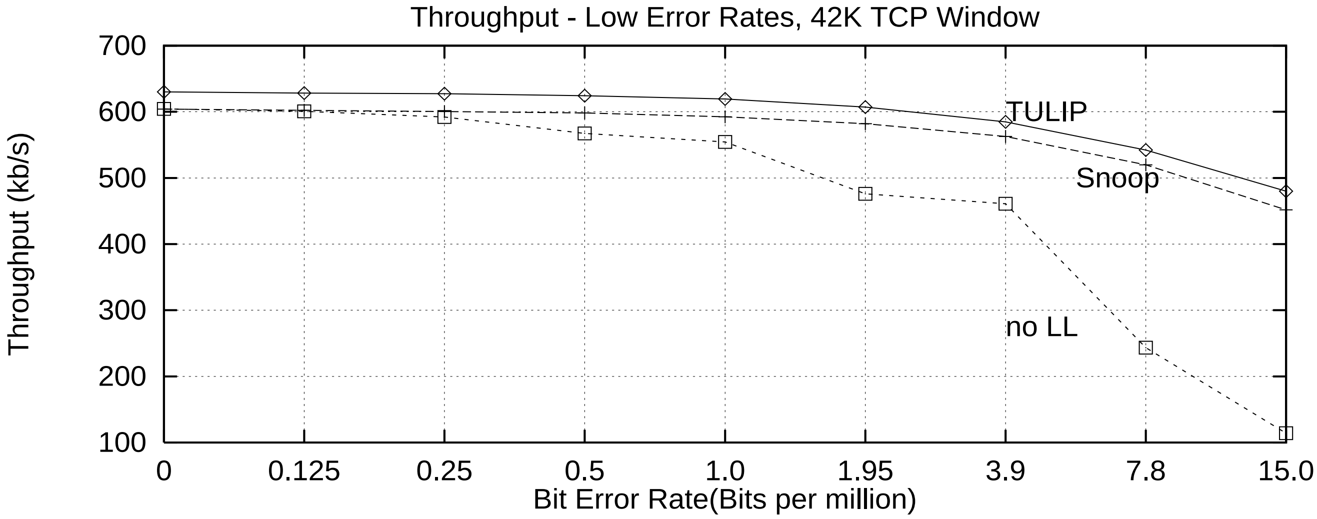


(b)

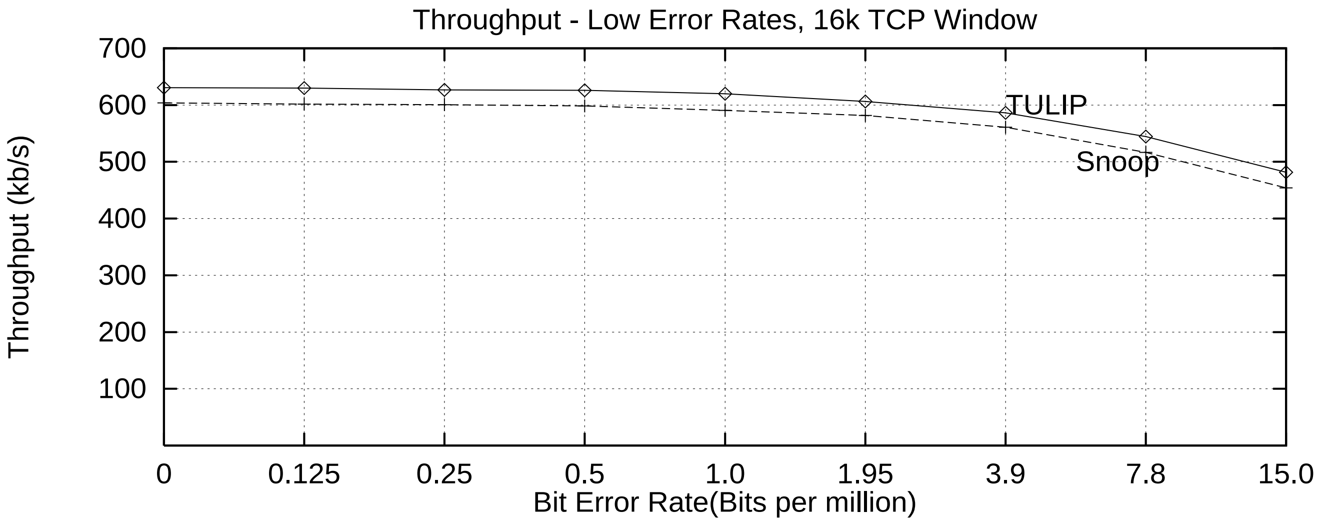
Figure 6.3: Experiment 1: TCP Sequence number growth. (a) BERR = 3.9 bits/million = 1/256Kbytes (b) BERR = 15 bits/million = 1/64Kbytes.

the other hand, achieve nearly identical throughput and are able to complete the transfer with no TCP timeouts. At the time these two protocols are finished, unassisted TCP has progressed only 30% into the transfer.

Figures 6.4(a) and (b) are summary plots of the average throughput for all three protocols with bit error rates varying from 0 to 15 bits/million and receiver window sizes of 42Kbytes and 16Kbytes, respectively.



(a)



(b)

Figure 6.4: Experiment 1: Average throughput for all three protocols with varying BERR. (a) 42K receiver window (b) 16K receiver window

The graph shows first and foremost that, for error rates under 0.5 bits/million, TCP is able to keep up with the losses and does not show appreciable performance degradation. However, as losses begin to rise, it is apparent that traditional TCP can no longer keep up with the losses and the throughput degrades considerably. Both the Snoop protocol and

TULIP show decreased throughput as the BERR rises; however, their overall throughput is consistently better than unassisted TCP. Because a TCP congestion window size of both 42Kbytes and 16Kbytes is more than enough to accommodate the bandwidth-delay product of this connection, the graphs show no appreciable difference in throughput for different window sizes.

6.1.2 Goodput

Figure 6.5 shows the goodput for the three protocols. Goodput gives a sense of the redundant information transmitted over the channel as a result of losses and subsequent retransmissions. Ideally every loss would be retransmitted at most once; however, as error rates increase it is possible that, for example, TCP could timeout and retransmit a packet which has already undergone retransmission at the link layer or by the Snoop protocol. In this plot, the ideal goodput is indicated by the top line and the goodput achieved over the wireless link is then plotted for each protocol. It is clear from the figure that both the Snoop and TULIP protocols are able to maintain perfect goodput, which means they retransmit only what is absolutely necessary to insure all the packets reach the destination without error. Regular TCP, however, diverges from ideal goodput as the errors on the link increase. Table 6.2 indicates for each protocol the link losses, theoretical and achieved goodput, number of TCP timeouts and number of redundant transmissions over the wireless link for a BERR of approximately 15 bits per million. The numbers verify that Snoop and TULIP have no redundant transmissions and therefore achieve ideal goodput. The reduced goodput of regular TCP is due to the transmission of 158 redundant packets.

6.1.3 A Closer Look at TCP

Table 6.2 shows that with an average error rate of 15 bits per million, in which TCP suffers from multiple losses per window of outstanding data, TCP is unable to achieve ideal goodput. It is well-known that TCP cannot recover from two losses per window without resorting to a timeout. Figure 6.6(a) shows the sequence number growth and returning

Figure 6.5: Experiment 1: Goodput for all three protocols with varying packet error rates and a receiver window of 42Kbytes.

Comparison of Goodput for TULIP Protocol, Snoop and no LL						
Protocol	BERR (bits / million)	Packet Loss (percent)	Ideal Goodput	Achieved Goodput	#TCP timeouts	#redundant packets
TULIP	15.1	0.159	0.841	0.840	0	0
Snoop	15.5	0.169	0.831	0.829	0	0
No LL	15.2	0.166	0.834	0.814	732	158

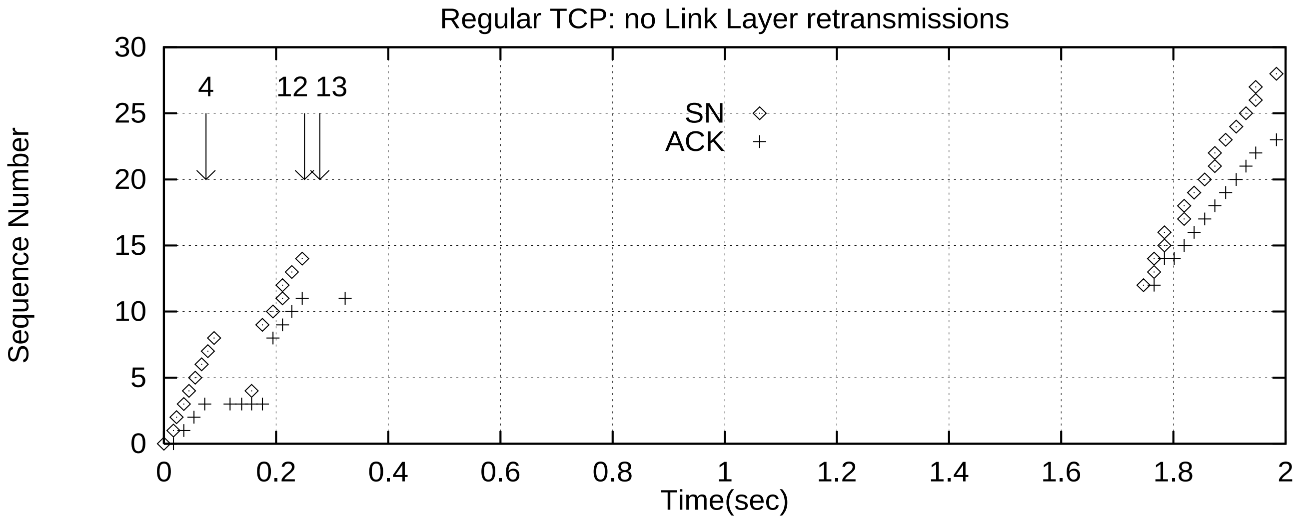
Table 6.2: Experiment 1: Ideal and achieved goodput, number of TCP timeouts and redundant packets transmitted over the wireless link for a BERR of 15 bits per million and a receiver window of 42Kbytes.

acknowledgments for the first two seconds of the transfer when TCP has no link-level retransmissions. The packets dropped by the wireless link are indicated by arrows and are labeled with their sequence numbers. At time $t = 0.15$, the third duplicate acknowledgment for packet 4 arrives, a fast retransmission occurs and the TCP congestion window is reduced by half. At time $t = 0.24$ and $t = 0.25$, packets 12 and 13 are both dropped. Packet 12 is not recovered by a fast retransmission because the reduced window size does not produce enough outstanding packets to return duplicate ACKs and as a result there is a TCP timeout at $t = 1.75$ prompting the retransmission of packet 12. When the ACK for this packet arrives it is determined that packet 13 is also missing. Because the session is now in the slow start phase (following a timeout), the current congestion window of 2 allows for the transmission of two packets from the beginning of the window, namely 13 and 14. The retransmission of packet 14, however, was redundant because it was received correctly the first time, and as a result both the wired and wireless goodput decrease accordingly.

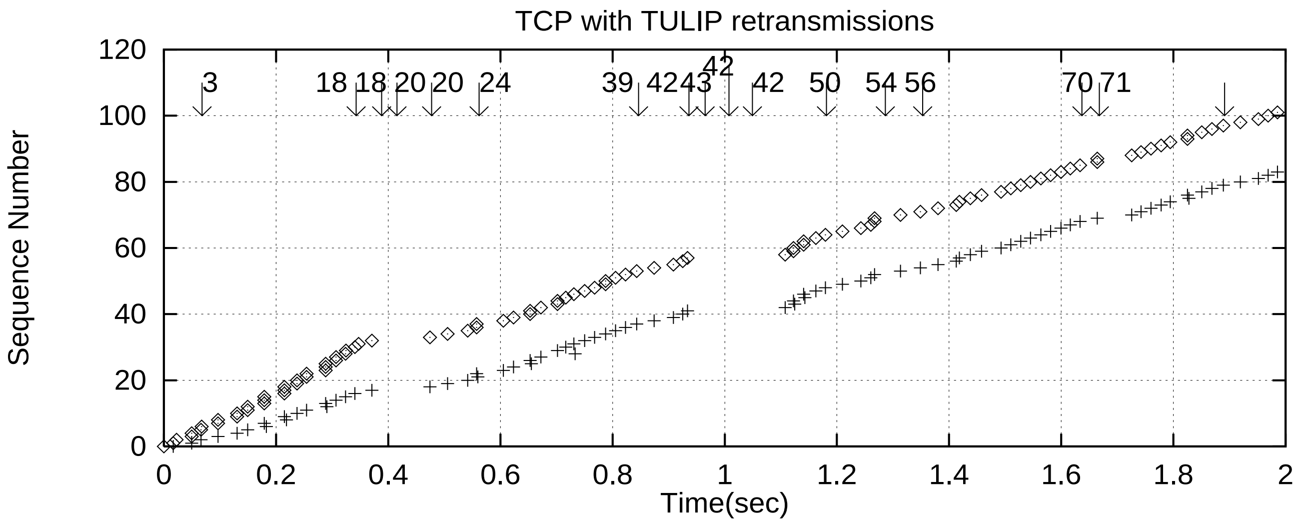
In contrast, as depicted in Figure 6.6(b), TULIP is able to handle many sequential losses within a window, perform the necessary retransmissions, and prevent TCP from having a timeout. When this plot is compared to Figure 6.6(a), we see that TULIP's retransmissions improve the situation considerably: TULIP suffers from 17 packet losses (compared to only 3 losses in Figure 6.6(a)) and though there are noticeable idle periods lasting not more than 200ms (during which time TULIP is retransmitting lost packets), there are no TCP timeouts and the transfer makes considerable progress. It is interesting to note that, beginning at time $t = 0.9$, packet 42 is dropped from the channel three times. This series of losses would have required a minimum of three timeouts for TCP without underlying link layer retransmissions; instead, the dropped packets are quickly retransmitted and the losses are not noticed by TCP.

6.1.4 TCP Round-trip Time

Figure 6.7 shows the estimated RTT and deviation calculated by TCP throughout the session, the actual measured RTT and retransmission timeout value for TCP with a channel BERR of 15bits/million and TULIP retransmissions. The TCP retransmission timer is calculated as the estimated round-trip time plus four times its deviation, has a granularity of 500ms, and generally produces a TCP timeout value which gives plenty of slack for TULIP to retransmit packets without TCP timeouts. It is interesting to note in the plot that the variation in RTT is generally within the 500ms granularity of the TCP timer. Arguments against link-layer approaches [3] point out an increased variance when retransmissions are performed at the link layer; however, increased variance decreases TCP's susceptibility to timeouts (because the timeout value increases) and in effect reduces the number of timeouts!



(a)



(b)

Figure 6.6: Experiment 1: Sequence number and ACKs at the source for the first 2 seconds of the TCP transfer. Packets dropped on the channel are shown with arrows. BERR = 15 bits per million. (a) no link retransmissions (b) TULIP protocol

This is a well-known phenomenon in public wireless networks such as Metricom's Ricochet [2] which have a large delay variance through the RF portion of the network, which in effect causes very large TCP timeout values.[1]

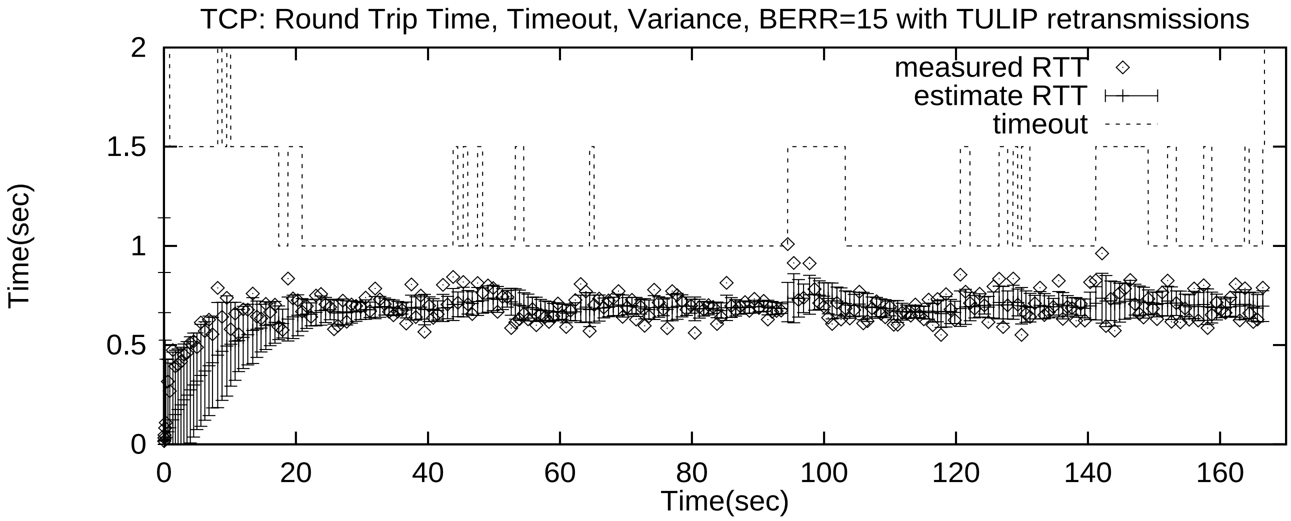


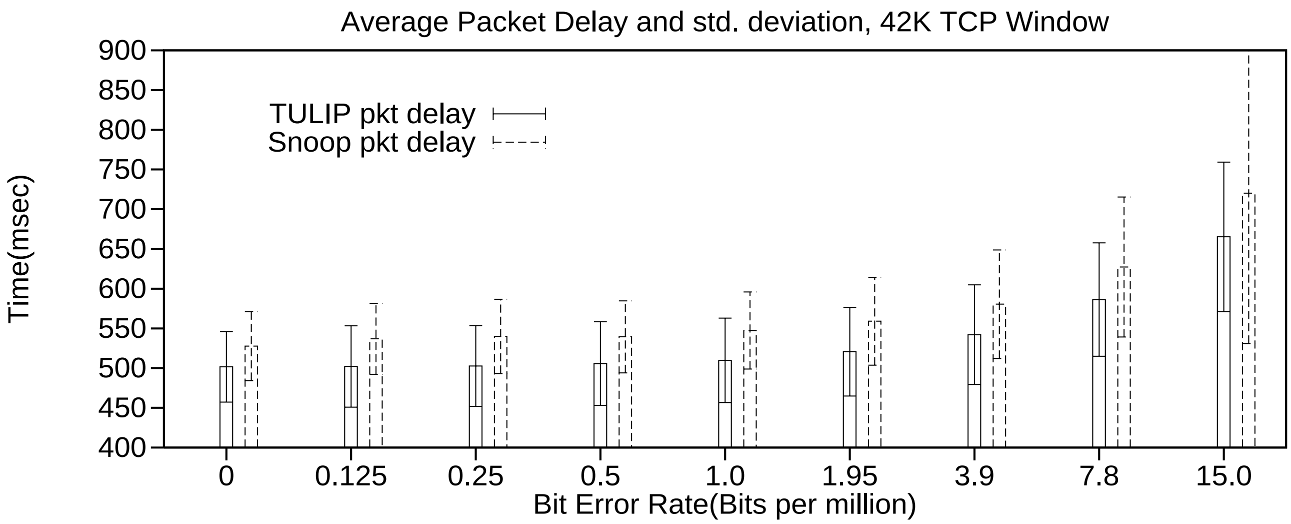
Figure 6.7: Experiment 1: RTT measurement and estimate for TULIP with a BERR of 15 bits/million with receiver window of 42Kbytes.

6.1.5 Delay

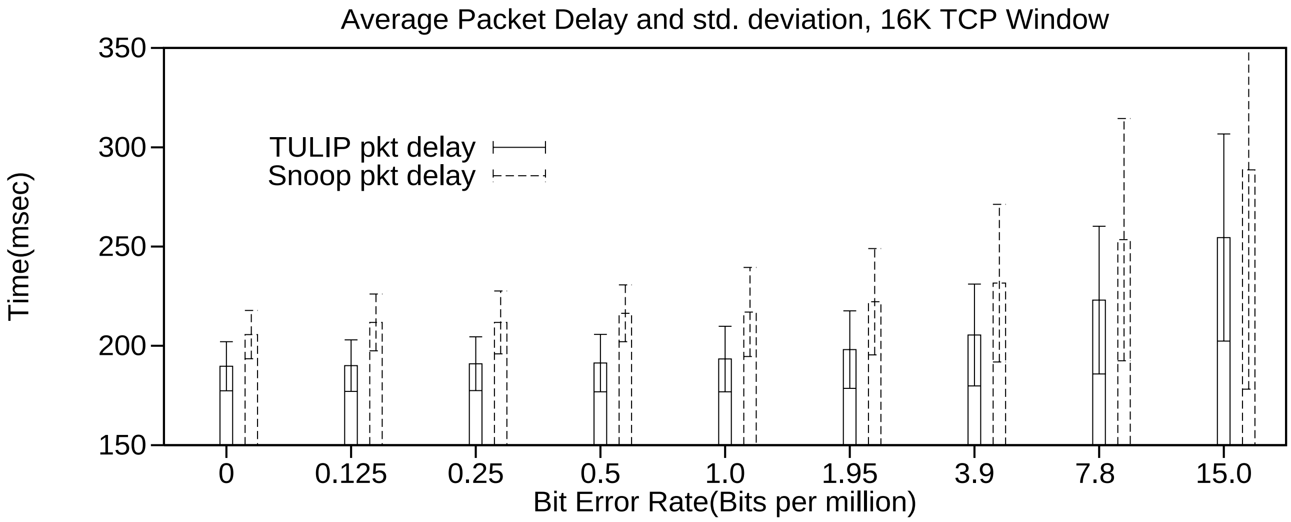
Figure 6.8(a) shows the average packet delay with TULIP is lower than the Snoop protocol in every instance, and as the error rates increase, TULIP's standard deviation is also lower than the Snoop protocol. TULIP's delay is smaller because, as discussed previously, TULIP has a faster retransmission mechanism. This becomes apparent as the error rate increases and Snoop's variance jumps up at a BERR of 15 bits per million when it has trouble with scattered losses within a window and with the occasional loss of retransmissions.

Figure 6.8(b) shows the delay when the receiver's advertised window is reduced to 16Kbytes in an attempt to lower the average end-to-end delay by reducing the number

[1] This avoids unnecessary retransmissions when packets are simply delayed, but is highly undesirable if packets are in fact lost because so much time lapses before the timeout occurs.



(a)



(b)

Figure 6.8: Experiment 1: Average packet delay and std. deviation for TULIP and Snoop protocols. (a) Receiver Window is 42Kbytes. (b) Receiver Window is 16Kbytes.

of packets in the queue at the bottleneck link. The plot shows that the end-to-end delay is, in fact, reduced by more than half for both protocols by reducing link layer queuing. This leads to our recommendation that mobile nodes should, in general, advertise a smaller window size to reduce delay and lessen the possibility of congestion at the base station.

6.2 Experiment 2: High Error Rates

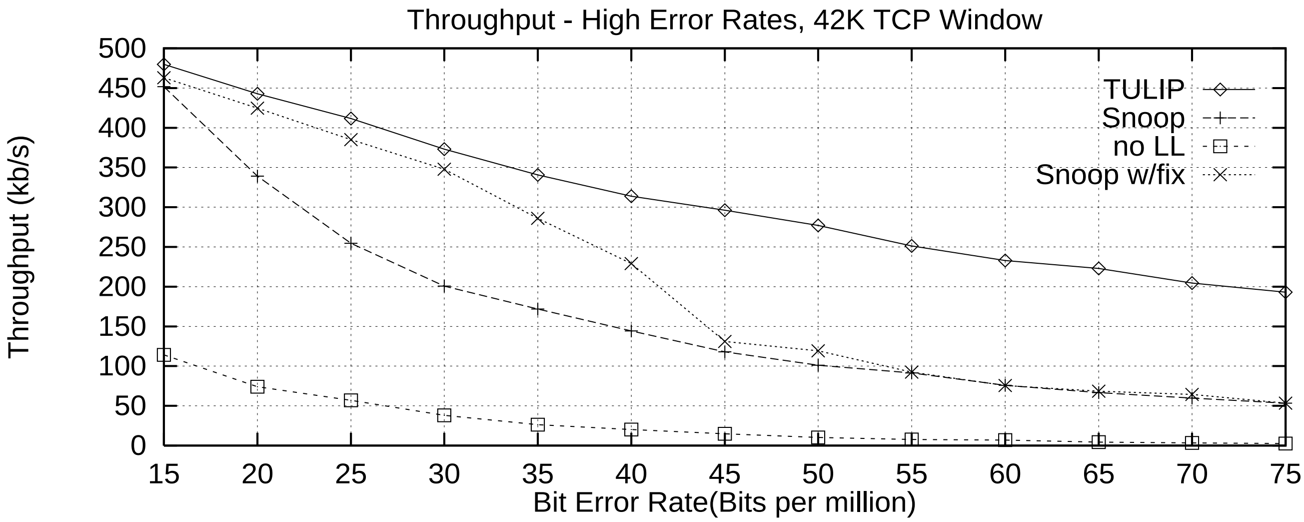
In this experiment we investigate the effect of high error rates on throughput and end-to-end packet delay. Some public wireless communication providers [15][22] report typical wireless one-hop packet loss rates between 10 – 40%. For this reason, we increase the bit error rates on the channel to very high values, *i.e.*, from 15 to 75 bits per million (corresponding to packet loss rates from 16 to 85%). We also examine the end-to-end delay experienced by packets when the TCP congestion window size is reduced.

This section shows two different plots for the Snoop protocol. After running simulations with high error rates, Snoop appeared to perform poorly. After examining the source code, we discovered that when many losses occurred in succession, causing Snoop to retransmit multiple lost packets per window, it would inadvertently update its link *rtt* estimate to an incorrect value, *i.e.*, to a value during which time one or more retransmissions occurred. This caused the *rtt* estimate to grow to very large values, which when uncorrected caused any subsequent retransmissions to be driven only by the persist timer and not the round-trip timer. The plots labeled as Snoop w/fix are run with the error corrected.

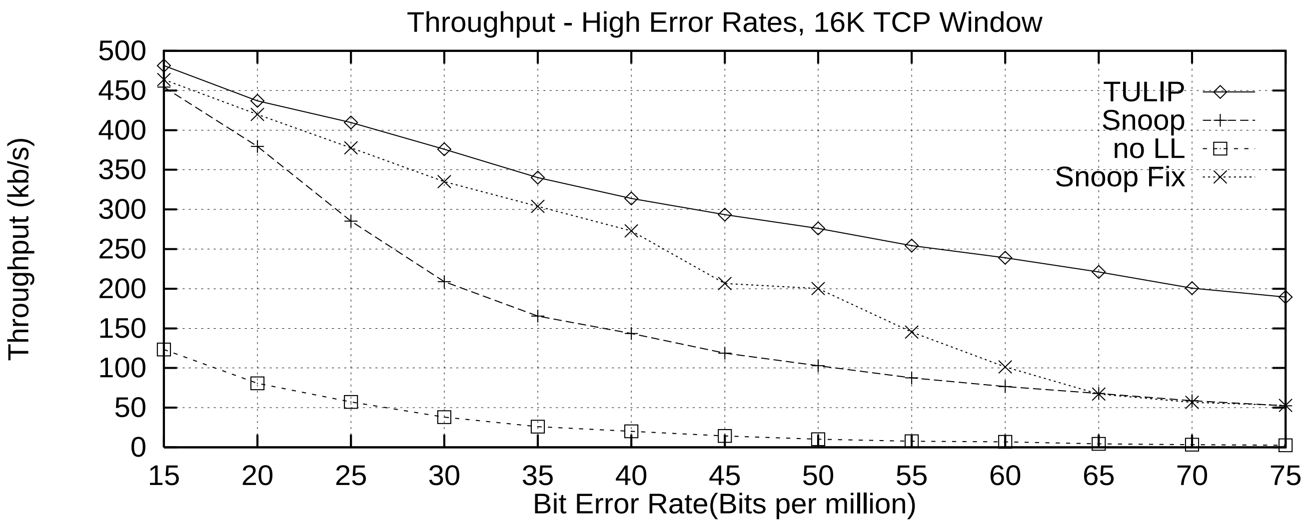
6.2.1 Throughput

The graphs of throughput under very high error rates are shown in Figures 6.9(a) – (c) for receiver window sizes of 42K, 16K and 8Kbytes, respectively. These graphs clearly show that, as the bit error rate increases, all three protocols show a reduced turn in throughput. With a BERR of 15 bits/million the TCP connection with no link-layer retransmissions has degraded significantly and comes to a near standstill for error rates above 30 bits/million.

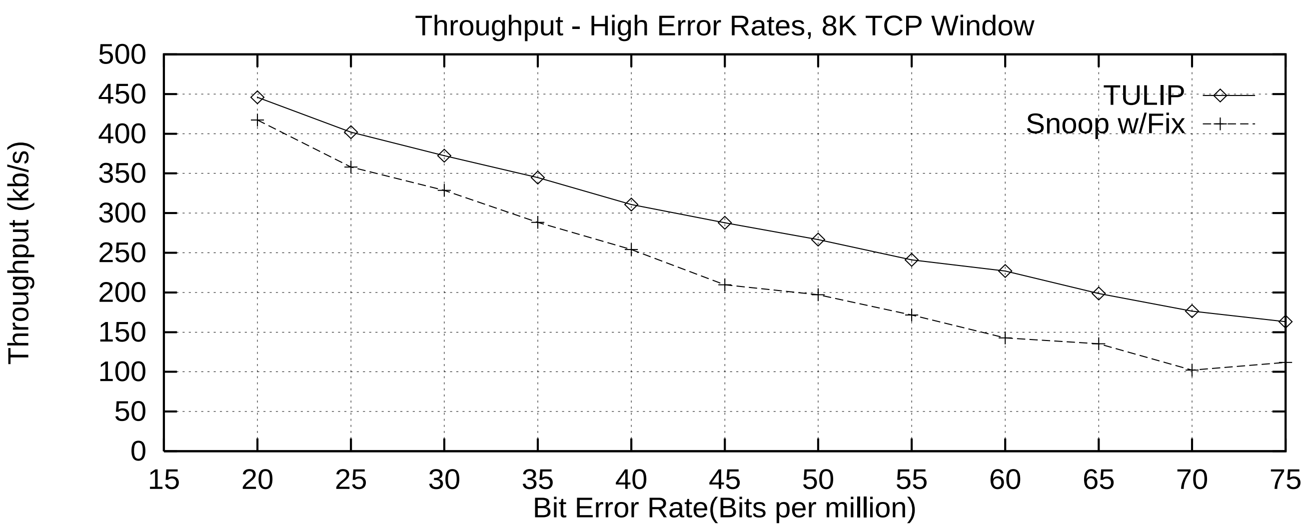
After a BERR of 30 bits/million Snoop's throughput drops off quickly and diverges from TULIP's throughput, which decreases slowly as error rates increase. With TULIP the connection simply appears increasingly slower to the transport layer. Because a TCP congestion window size of both 42Kbytes and 16Kbytes is more than enough to accommodate the bandwidth-delay product of this connection, the graphs (a) and (b) show no appreciable difference in throughput for different window sizes. However, once the error rate is above 50



(a)



(b)



(c)

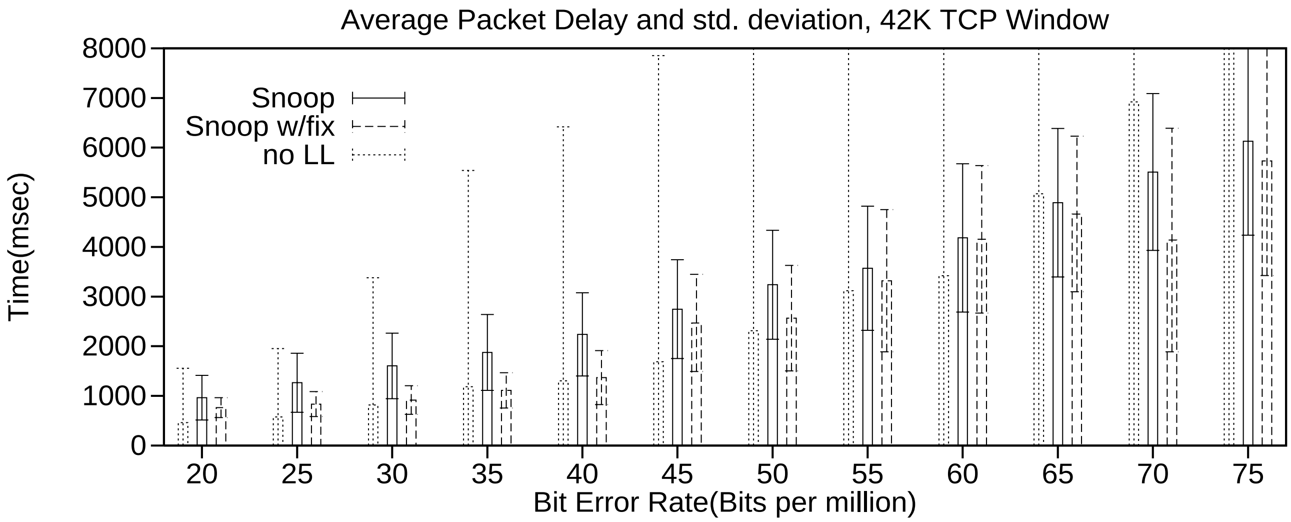
Figure 6.9: Average throughput for all three protocols with high error rates. (a) 42Kbyte window (b) 16Kbyte window. (c) 8Kbyte window

bits/million, the throughput for Snoop actually improves due to the smaller window size. With very large error rates the larger window sizes can cause large blocks of data to be dropped or for many subsequent packets to be transmitted before an error earlier in the window is recovered. Since Snoop recovers from these errors sequentially, it takes a longer time to recover from errors, especially if the errors are distributed throughout a window of data. This causes the source to remain idle and no new packets are transmitted (see Appendix Figure A.1 and A.2). Furthermore, the larger window size leads to bursty traffic because once the errors are finally recovered and the acknowledgments are returned to the sender, large bursts of new data from the sender are transmitted (because Snoop suppresses duplicate acknowledgments the ones that arrive after losses often cover many packets).

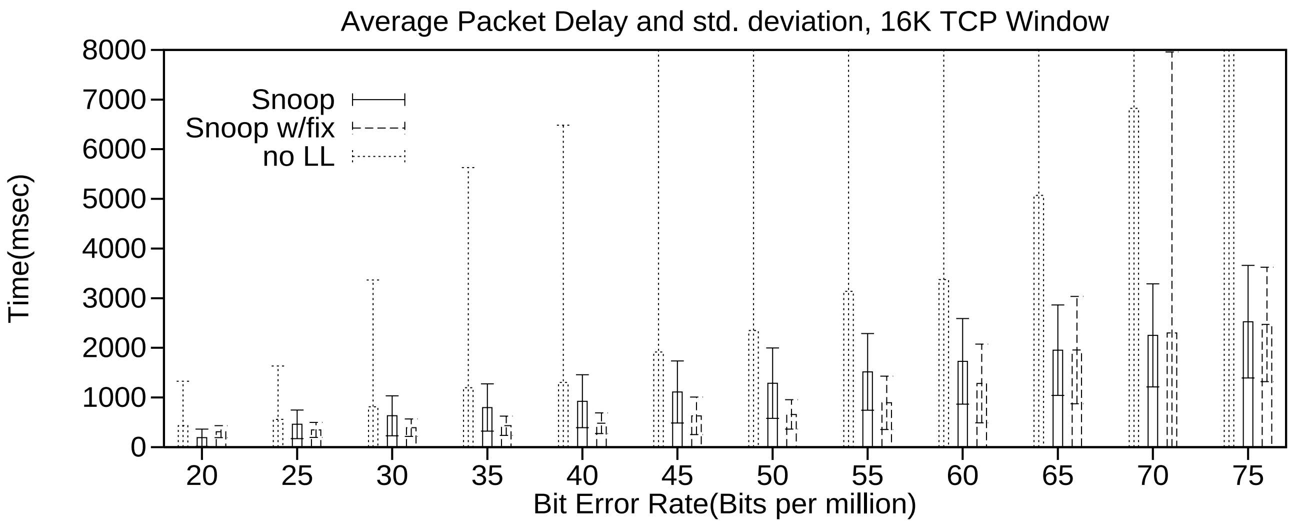
6.2.2 Delay

Figures 6.10(a) and (b) show the average end-to-end packet delay and standard deviation for Snoop, Snoop w/fix and TCP with no underlying retransmissions, with a receiver window size of 42K and 16Kbytes, respectively. In Figure 6.10(a) the TCP transfers with no link layer retransmissions display a lower average delay for bit error rates up to 60 bits/million; however, when the average delay is considered in conjunction with the large standard deviation, we see that the overall delay is, in fact, much larger than the delay seen by the Snoop protocol. The large standard deviation occurs because the packets encounter either no queuing (during periods immediately after a timeout the queue has a chance to empty and the congestion window is small) or significant queuing delays when the queues at the base station are large and the congestion window reaches its maximum size.

The plots show that the delay and deviation for Snoop with the code modification are significantly lower than without. Because the performance of Snoop with the fix shows improved performance, it is used to compare with TULIP in Figures 6.11(a)-(c) for receiver windows of 42K, 16K and 8Kbytes, respectively. Several observations can be made from these plots. First, it is clear that regardless of the protocol used, lowering the TCP receiver's advertised window can greatly reduce the end-to-end packet delay. The delays experienced



(a)



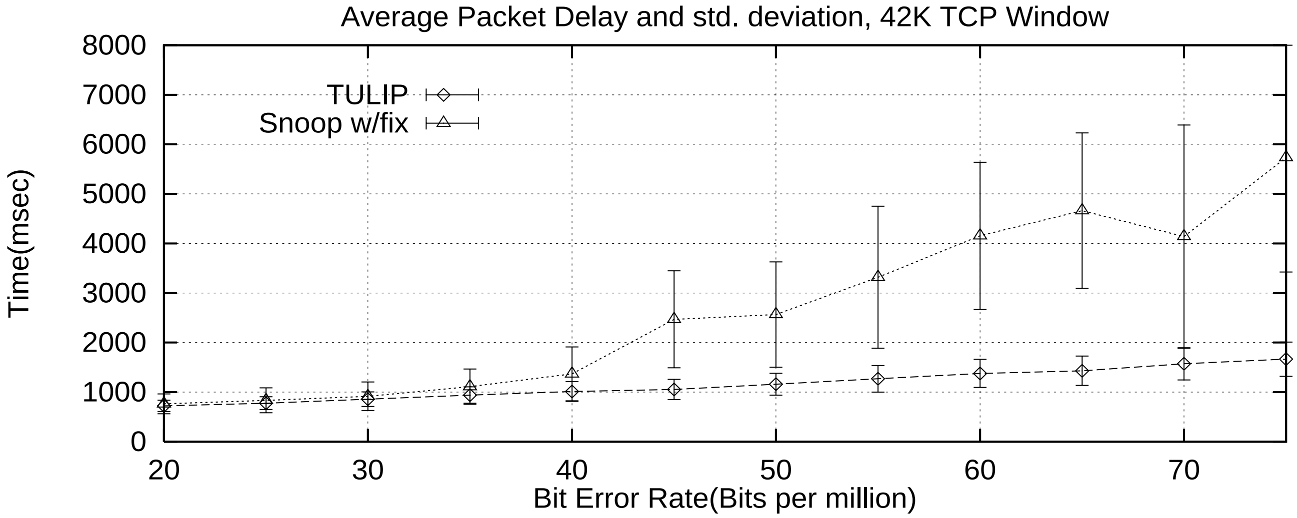
(b)

Figure 6.10: Experiment 2:Average packet end-to-end delay and std. deviation for Snoop, Snoop w/fix, and No LL (a) 42Kbytes receiver window (b) 16Kbytes receiver window

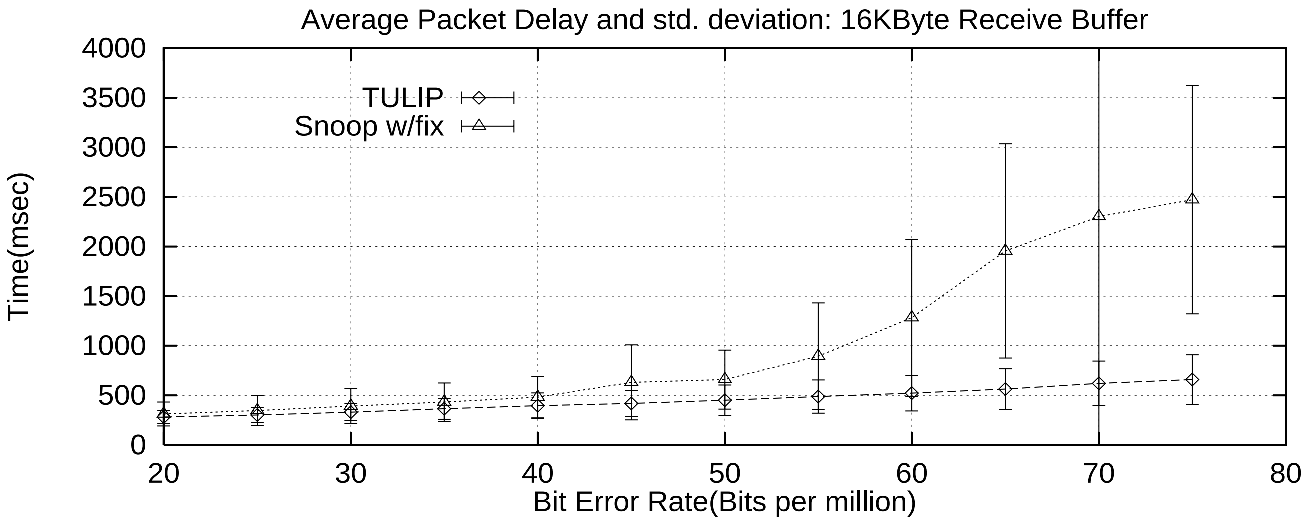
by packets with an 8Kbyte window and the highest error rates are close to the delays experienced by sessions with moderately high loss rates and a larger window size. This result is expected since a window size larger than the bandwidth delay product of the connection will certainly introduce delay.

Second, once error rates exceed 35 bits/million, the delays and deviation with the Snoop protocol are significantly larger than with TULIP. When the receiver's window is reduced from 42Kbytes to 16Kbytes, we notice that as error rates increase Snoop suffers from a very high standard deviation and an average packet delay of more than twice that of TULIP. Likewise, as the window is further reduced to 8Kbytes, the delay with TULIP is below 400ms for all error rates, whereas the Snoop protocol approaches this delay with error rates of 55 bits/million. The larger delay is because the Snoop protocol has trouble recovering multiple losses per window and also recognizing when retransmissions are also lost. In the case of high losses, Snoop must rely heavily on its timer for retransmissions and gets stuck trying to retransmit the first packet in a series of losses. The deviation is high here because losses are tackled one by one and in order, *i.e.*, once the first loss is recovered, then the next is tackled and so on. TULIP, on the other hand, creates a retransmission list upon the first receipt of an ACK and knows exactly which packets are missing because each ACK specifies the complete state at the receiver's buffer. All packets in the retransmission list are then retransmitted. In addition, if retransmissions need to be retransmitted, TULIP is able to do this as soon as it has received any ACK. The deviation is smaller in TULIP because, with multiple losses per window, it is often the case that errors further down in the window are recovered before the first error. Therefore, by the time the first error is recovered, all the packets can be released to the higher layer in sequence. Note that this will not result in the same degree of burstiness as Snoop (see previous section) because each acknowledgment packet, from the first loss to the last packet in the window, is returned to the TCP source.

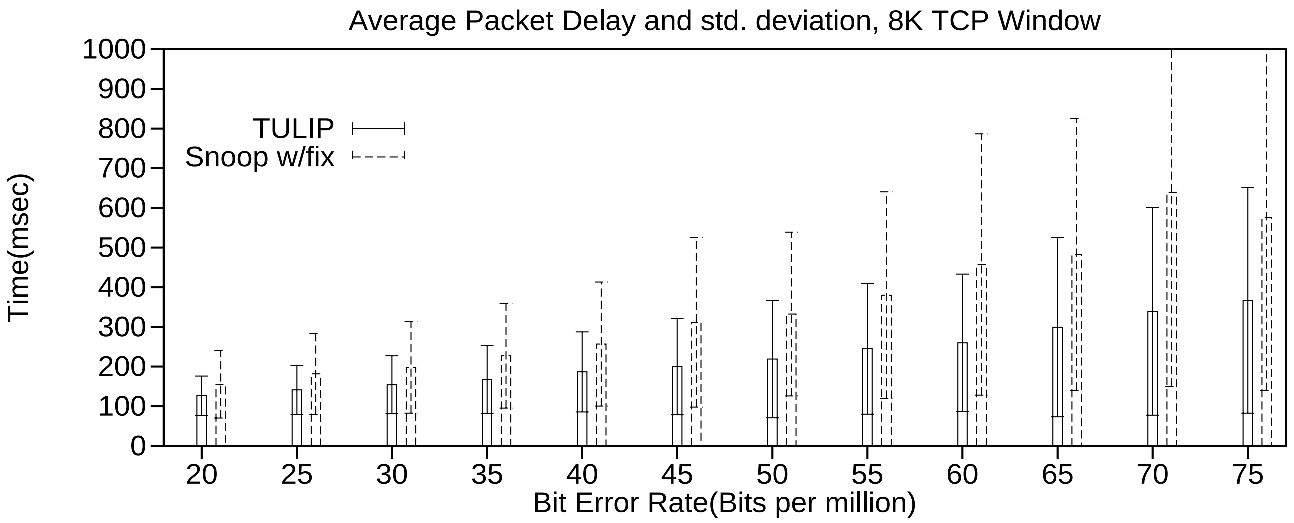
Finally, Figure 6.11(c) shows an increased standard deviation (proportional to the delay) for TULIP as compared to the larger window sizes. This is because with the smaller window size there is less buffering at the base station, and therefore, on the average, the delay due to queuing is lessened, but delays due to errors on the channel and the subsequent retransmission of lost packets becomes a more dominant effect in the deviation term.



(a)



(b)



(c)

Figure 6.11: Experiment 2:Average end-to-end delay and std. deviation for Snoop, Snoop w/fix, and No LL (a) 42Kbytes receiver window (b) 16Kbytes receiver window (c) 8Kbytes receiver window

6.2.3 Goodput

Figure 6.12 displays the ideal goodput versus achieved goodput for all three protocols for high packet error rates. The plot shows that while TULIP achieves nearly ideal goodput, Snoop and the no LL approach often fall below the ideal line.

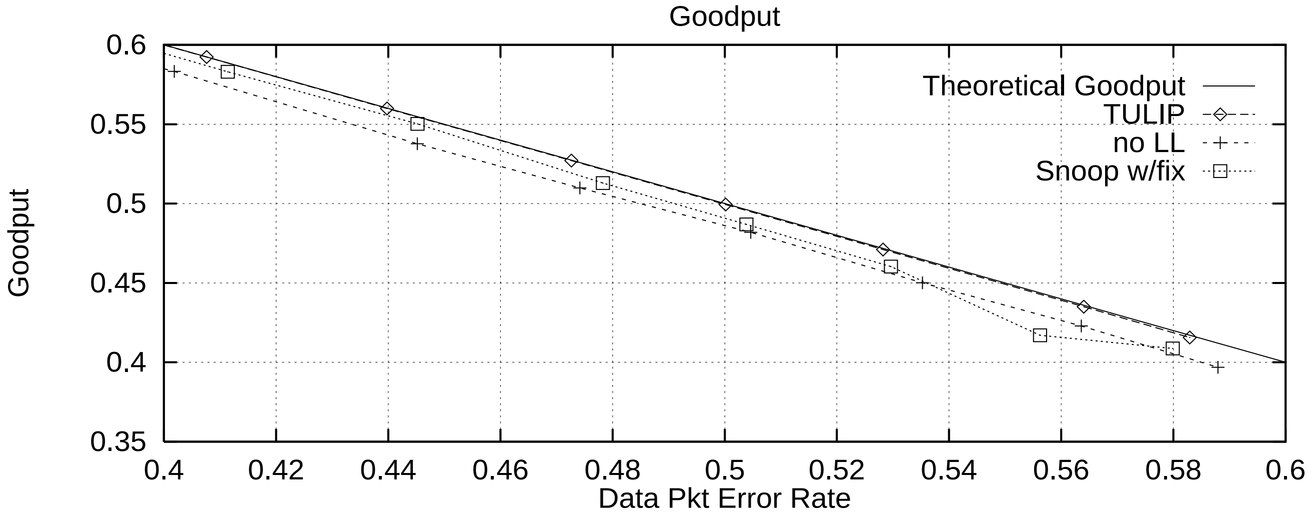


Figure 6.12: Experiment 2: Goodput for all three protocols with high packet error rates and a receiver window of 16Kbytes.

Table 6.3 shows a more detailed example of what happens with a BERR of 50 bits/million. Note that because the desired BERR is specified for the simulation at run time, and because the errors are exponentially distributed for each simulation, there is some slight variation in the actual resulting packet error rates. The table shows that TULIP retransmits 3 packets unnecessarily. Snoop also does well and achieves near ideal goodput; however, there is one TCP timeout in which nearly the entire window is retransmitted by TCP in its attempt to recover errors, and also one loss detected via a fast retransmission, which also results in nearly the remainder of the window being retransmitted. Of course these retransmissions were unnecessary as the packets were buffered in the Snoop module. In addition, Snoop performs an additional 40 retransmissions over the wireless which were actually not necessary because the packets had already been delivered to the receiver. As shown, TCP with no underlying retransmissions produces many more unnecessary retransmissions, which thereby causes waste of both the wired and wireless bandwidth.

Comparison of Goodput for TULIP Protocol, Snoop and no LL							
Protocol	BERR (bits / million)	Pkt Loss (percent)	Ideal Goodput	Achieved Goodput	#TCP timeouts	#redundant TCP packets	#redundant link packets
TULIP	49.2	0.44	0.56	0.559	0	0	3
Snoop	50.0	0.445	0.56	0.550	1	19	40
No LL	50.2	0.445	0.56	0.53	3699	187	-

Table 6.3: Experiment 2: Ideal and achieved goodput, number of TCP timeouts and redundant packets transmitted over the wireless link for a BERR of 50 bits per million and a receiver window of 16Kbytes.

6.3 Experiment 3: Fading and Burst Losses

In this section we examine TCP's performance in the presence of packet burst losses and fading on the wireless link. Fading causes periods of silence on the channel, during which time neither sender nor receiver can hear each other and in effect packets are dropped in both directions on the channel. Fading is often caused by the movement of mobile nodes, but can also be caused by objects which move in front of and around a mobile node.

As described by Garg [29], the average fade duration T for which a signal is below a specified signal level R is approximately calculated as:

$$T(R) = \frac{e^{\rho^2} - 1}{\sqrt{2\pi} f_m \rho} \tag{2}$$

where $f_m = \frac{v}{\lambda}$ is the maximum Doppler frequency, and ρ is the ratio between the specified level and the RMS amplitude of the fading envelope.

The level-crossing rate, $N(R)$ is the average number of times per second that the signal envelope crosses the fading level in a positive-going direction is:

$$N = \sqrt{2\pi} \cdot f_m \cdot \rho \cdot e^{\rho^2} \tag{3}$$

6.3.1 Experiment 3a: Uniform Distribution of Burst Losses

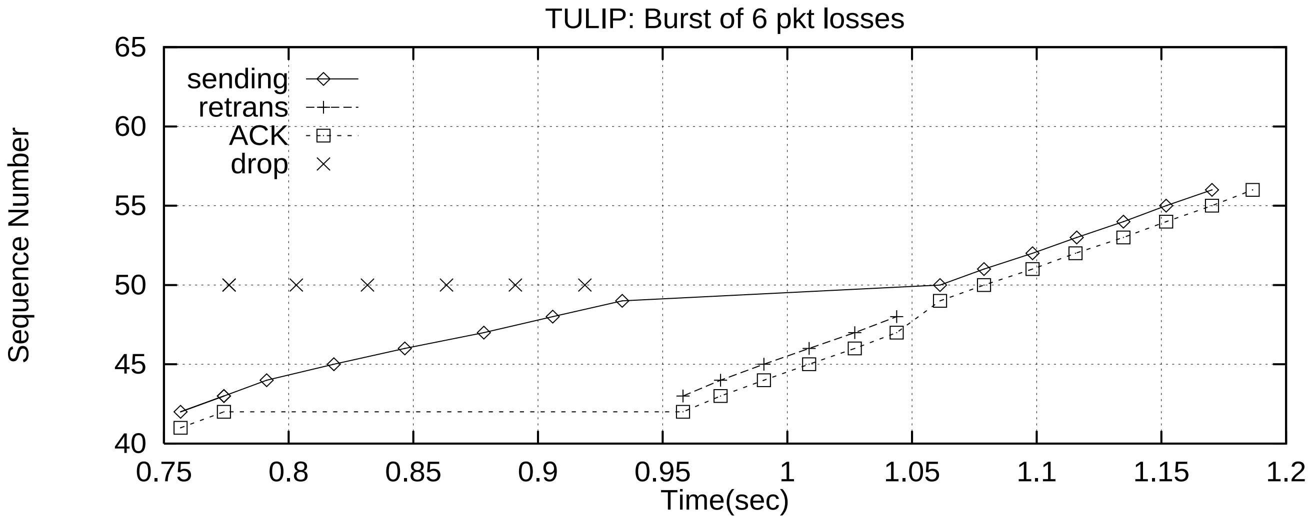
In the method used by Balakrishnan et. al. [7], burst losses of a specific size are distributed uniformly over the run of the experiment. The results when bursts of sizes 2,4 and 6 data packets are spread every 64Kbytes of data are shown in Table 6.4.

Because of FAMA's handshake, the sender would not send more than one packet into the channel during a fading period, because it would not receive the necessary CTS to send any more data packets. The case would be the same for DWFMAC[1]. However, this experiment is still interesting to show, because it indicates that TULIP provides smaller delays and slightly better throughput than Snoop, even in such rare cases in which the MAC layer manages to send multiple packets into the channel that cannot reach the receiver without error, even though the corresponding RTS and CTS packets could.

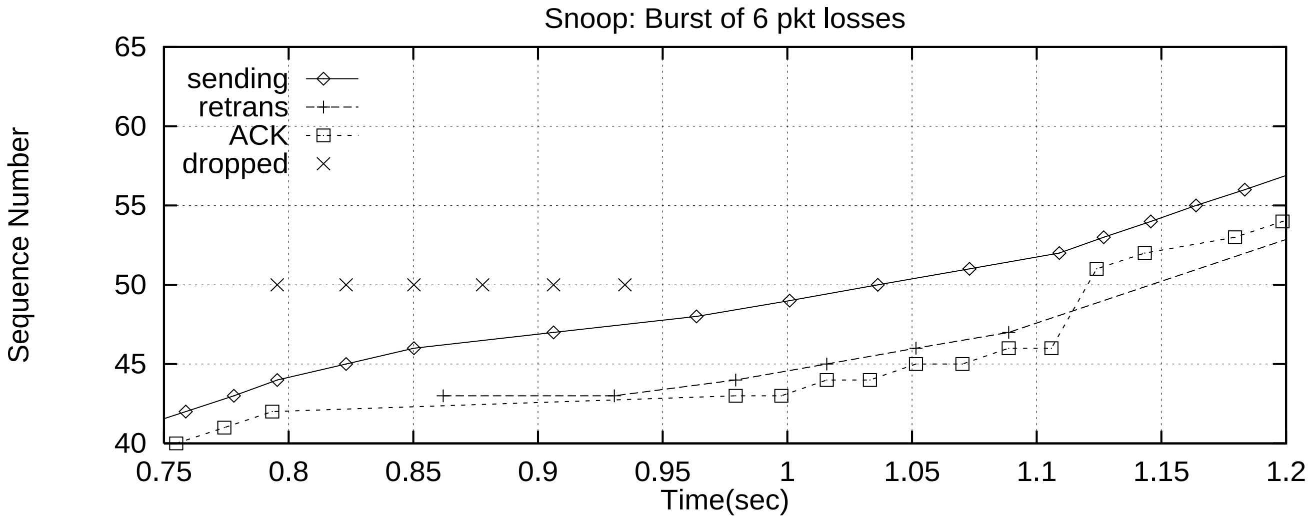
Bursts Distributed every 64Kbytes					
Burst Size #packets	TULIP Throughput(Kbps)	Snoop Throughput(Kbps)	Δ (Kbps)	TULIP Delay $\pm$ dev.(ms)	Snoop Delay $\pm$ dev.(ms)
2	587.3	562.6	24.7	540±56	582±60
4	550.0	527.6	22.4	579±74	621±84
6	516.1	496.4	19.7	618±98	660±114

Table 6.4: Experiment 3a: Throughput of TULIP and Snoop in the presence of bursts of length 2,4 and 6 packets. Burst periods are distributed every 64Kbytes of data. Receiver Window is 42Kbytes.

Figure 6.13 (a) and (b) show how TULIP and Snoop retransmit lost packets during a period of 6 consecutive data packet losses. It takes Snoop slightly longer to recover from the errors than TULIP, because TULIP has a more systematic approach to recovery, *i.e.*, it is absolutely clear upon receipt of the first ACK at time $t = 0.95$, that packets with SN #43 through 48 are missing, because the bit vector returned with the cumulative acknowledgment indicates the precise losses. The protocol can then immediately retransmit all necessary packets. Snoop, on the other hand, must try to figure out which packets are missing based upon timeouts, which occur at $t = 0.87$ and $t = 0.93$ for the first loss, and it thereafter must interpret the values of returned cumulative acknowledgments to guess if additional errors have occurred. It should also be noted in this transfer, Snoop only suffers from 4 original packet losses (and 2 retransmissions), whereas 6 original packets are dropped for TULIP.



(a)



(b)

Figure 6.13: Experiment 3a: Burst loss of 6 packets. (a) TULIP (b) Snoop

6.3.2 Experiment 3b: Markov Model of Burst Losses

This method to simulate channel fading consists of a two-state Markov model taken directly from the work by Lettieri *et al.* [21] which is also discussed by Wang et al. [30] and Swarts *et al.* [13]. The Markov model shown in Figure 6.14 is used to model the channel during periods of fading; the model consists of a two-state Markov chain representing *Good* and *Bad* states on the channel, transition probabilities into and out of the states, and channel loss probabilities associated with each state. As bits arrive on the channel, the error probability for the given state is first applied to the bit, and then a decision is made as to whether or not a change of state should occur. Thus, it is possible to change states many times within one packet.

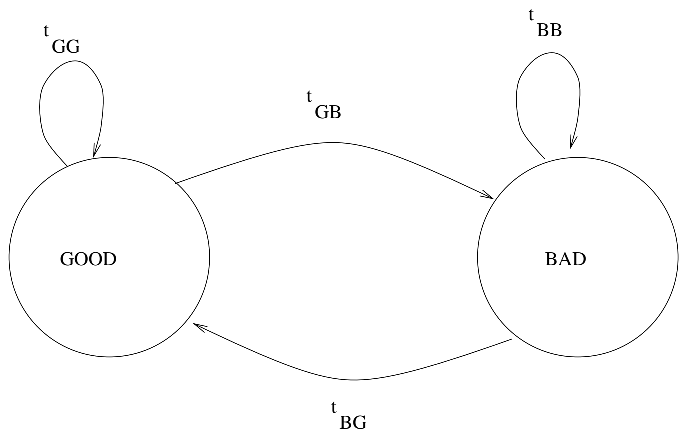


Figure 6.14: Experiment 3b: Markov chain to represent channel fading.

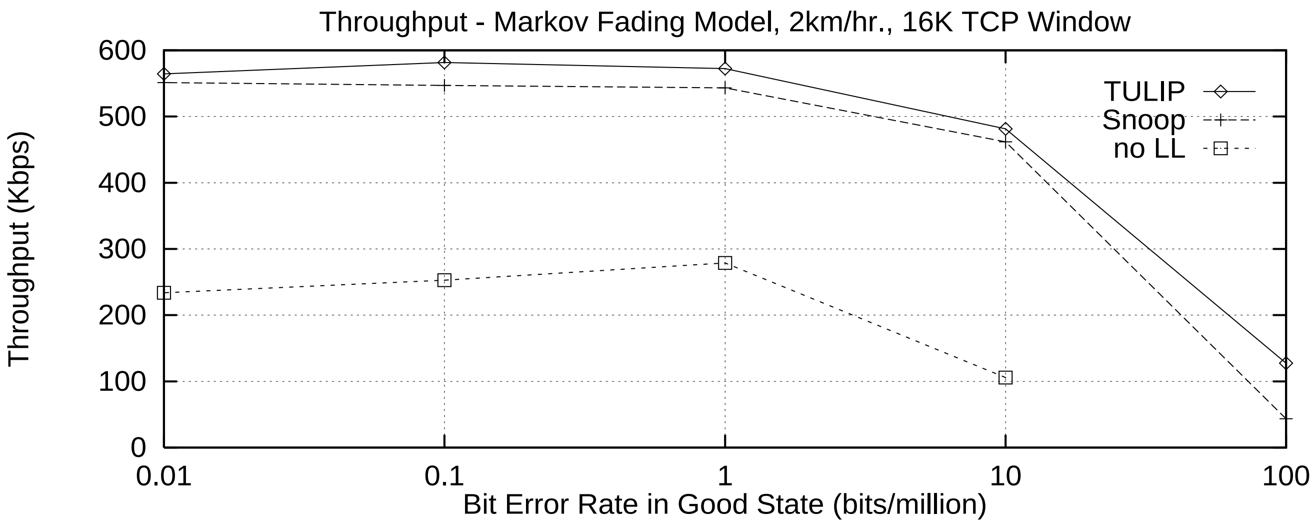
The transition probabilities are calculated directly from the Equations 2 and 3 as:

$$\mu = \begin{bmatrix} \mu_0 \\ \mu_1 \end{bmatrix} = \begin{bmatrix} P(Good) \\ P(Bad) \end{bmatrix} = \begin{bmatrix} \frac{(\frac{1}{N} - T)}{\frac{1}{N}} \\ \frac{T}{\frac{1}{N}} \end{bmatrix} \tag{4}$$

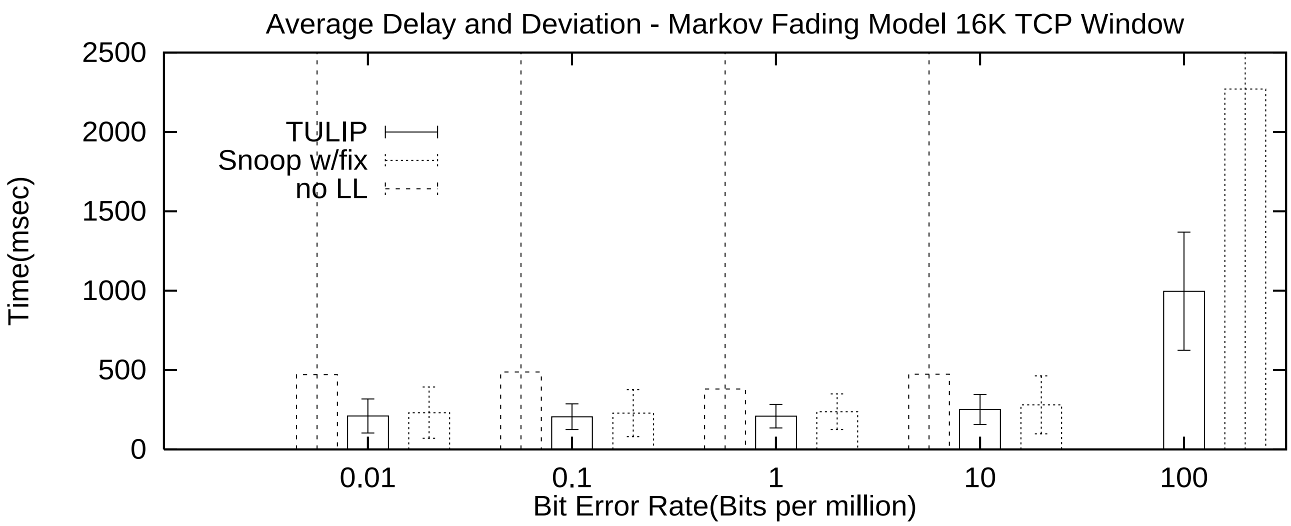
for a given channel frequency f, a Raleigh fading envelope ρ, and a particular speed v.

We have performed this experiment for the case of pedestrian speed with parameters f = 915MHz, v = 2km/hr, ρ = -20dB. The transition probabilities to be in either a good or bad state were calculated by Lettieri et al. [21] to be $t_{GB} = 6.67e^{-7}$ and $t_{BG} = 6.65e^{-5}$, respectively.

The results of this experiment are presented in Figures 6.15(a) and (b) as the BERR in the *Good* state is varied from 0.01 to 100 bits/million and the loss probability in the **Bad** state is held constant at a 50% packet loss rate. The plot shows that TULIP and the Snoop protocol display similar performance as long as the error rates are low; however, as shown in Figure 6.15(a), Snoop's throughput begins to diverge and fall below TULIP once error rates exceed 10 bits/million. The performance of TCP with no underlying retransmissions degrades once error rates exceed 0.1 bits/million, or approximately 1/8Mbytes. The end-to-end packet delay, depicted in Figure 6.15(b), shows that the average delay for TULIP and the Snoop protocol are similar; however, the standard deviation for Snoop is again higher. For the highest error rate shown, 100 bits/million, TULIP provides a much lower delay and a significantly tighter bound on the deviation.



(a)



(b)

Figure 6.15: Experiment 3b: Throughput for TULIP and Snoop protocols during Markov Fading Model. Loss probability in bad state is 50% and BERR in good state is varied. Pedestrian speed 2km/hr. (a) Throughput (b) Average end-to-end delay and standard deviation.

7. Conclusion

We have presented a link-level solution to TCP's performance degradation over lossy wireless links, which stems from TCP's interpretation of packet loss due to link errors as a sign of congestion. Our transport unaware link improvement protocol (TULIP) hides the wireless losses from TCP by taking advantage of the speed with which feedback is provided at the link level and TCP's generous timeouts. TULIP is designed to work over half-duplex wireless links and interleaves traffic from both ends of a link very efficiently, even when the underlying MAC protocol is contention based and provides no channel-access delay guarantees.

We have shown through simulation that when errors are exponentially distributed over the channel our approach not only minimizes TCP timeouts for all bit error rates, but also provides improved throughput over the Snoop protocol, which is one of the best performing prior published approaches to the problem. End-to-end delay becomes a problem as the losses on the link increase; however, reducing the size of the receiver's advertised window can help to alleviate the queuing delay. In all simulations TULIP provided lower end-to-end delay and smaller delay variance than both the Snoop protocol and unassisted TCP. We have looked at the effects of burst losses and channel fading and our results show that again the TULIP approach quickly retransmits the dropped packets once the channel is active again, yielding reduced though consistent throughput. The simulations show that during fading TULIP provides higher throughput and lower end-to-end delays compared to both Snoop and TCP with no underlying retransmissions.

We have demonstrated through simulations that current TCP does not correctly respond to wireless losses and that when appreciable losses occur (particularly multiple losses per window), TCP unnecessarily reduces its window and its throughput comes to a near standstill. New proposals for TCP, such as TCP-SACK [12], will help to identify the precise lost segments; however, this algorithm will still interpret these losses as congestion and reduce the sender's window. In addition, for connections with long end-to-end delay, a lengthy round-trip time must be incurred before the transport layer can respond to the loss of packets. We would expect that lower end-to-end packet delay and smaller delay variance could be achieved by providing underlying link-level retransmissions via TULIP. It is also important to point out that by recovering from these errors on the wireless link locally, we conserve bandwidth in the wired segments of the transmission path.

The advantage of our approach over other published approaches is that we keep no TCP state and therefore do not need to look into the TCP packet headers. This means that TULIP works correctly even if TCP headers are encrypted. TULIP works with both IPv4 and IPv6; in the latter case, TCP data packets can be identified as requiring reliable service from the NextHeader field in the IPv6 header. In addition, because our approach does not restrict the network to the presence of a base station, it can easily be applied to multi-hop wireless networks. Furthermore, TULIP conserves wireless bandwidth by piggybacking TCP ACKs with link-layer ACKs and returning them immediately across the channel through MAC Acceleration.

The simulation experiments presented in this thesis focus on TCP performance over lossy wireless links with no network congestion. It would be interesting to investigate end-to-end performance when TULIP is used and congestion is present in the wired and wireless segments of the transmission path. Another interesting comparison would be to examine the effect of underlying link-level retransmissions on paths with long end-to-end delay versus those with small delay.

Appendix A. Appendix

A.1 Receiver Window Size

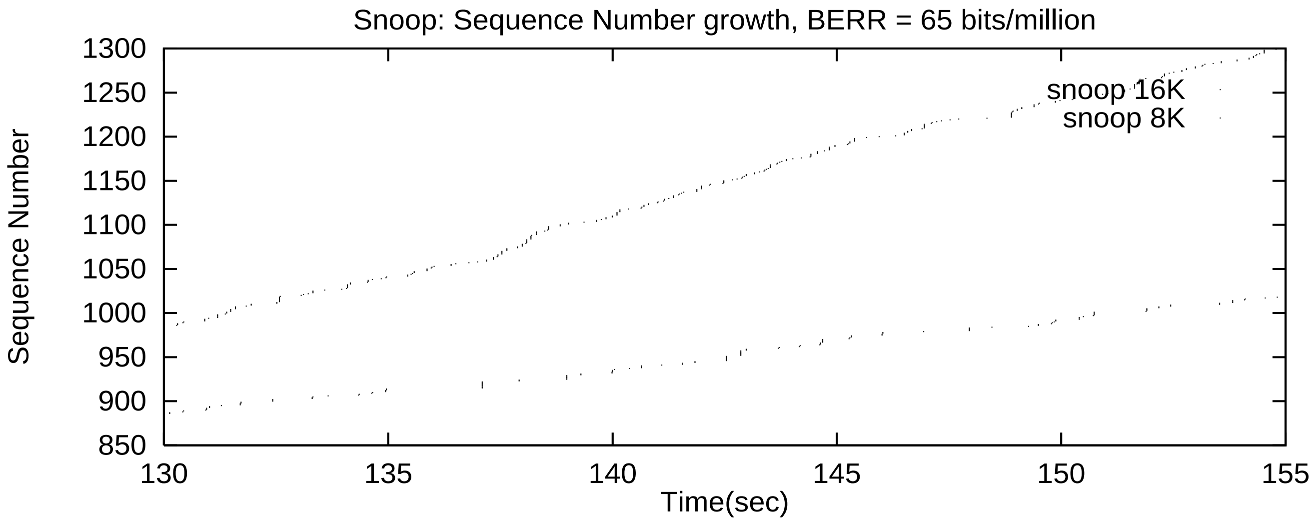
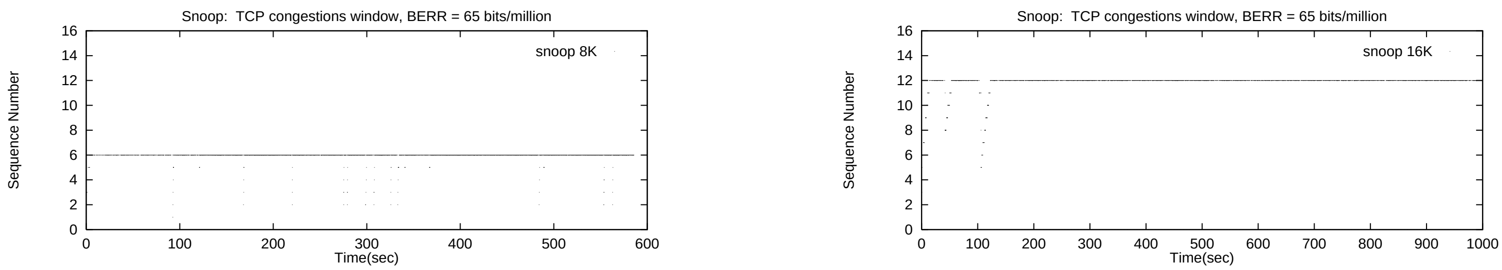


Figure A.1: Experiment 2: Sequence number growth for 16K versus 8Kbyte receiver window for Snoop.



(a) (b)

Figure A.2: (a) 8Kbyte window (b) 16Kbyte window

References

[1] P802.11–Unapproved Draft: Wireless LAN Medium Access Control (MAC) and Physical Specifications. Technical report, IEEE, January 1996.

[2] Metricom Ricochet Network, April 1998. http://www.metricom.com, Metricom Inc., Los Gatos, CA 95032.

[3] A. DeSimone AND M.C. Chuah AND O.C. Yue. Throughput performance of transport-layer protocols over wireless lans. In *Proc. IEEE Globecom '93*, pages 36–46, Houston,TX, Nov. 1993.

[4] E. Ayanoglu, S. Paul, T. LaPorta, K. Sabnani, and R. Gitlin. Airmail: A link-layer protocol for wireless networks. *Wireless Networks*, 1(1):47–60, 1995.

[5] A. Bakre and B. R. Badrinath. I-TCP: indirect TCP for mobile hosts. In *Proc. 15th IEEE Int'l Conf. on Distributed Computing Systems*, pages 136–43, Vancouver, BC, Canada, May 1995.

[6] H. Balakrishnan. Personal communication, Nov. 1997. UC Berkeley, Berkeley, CA.

[7] H. Balakrishnan, V. Padmanabhan, S. Seshan, and R. Katz. A comparison of mechanisms for improving TCP performance over wireless links. *IEEE/ACM Transactions on Networking*, 6(5):756–69, Dec. 1997.

[8] H. Balakrishnan, S. Seshan, and R. Katz. Improving reliable transport and handoff performance in cellular wireless networks. *ACM Wireless Networks*, Dec. 1995.

[9] D. Bertsekas and R. Gallager. *Data Networks.* Prentice Hall, 2nd edition, 1992.

[10] D. Beyer and B. Nguyen. *The C++ Protocol Toolkit: Overview.* Rooftop Communications Technical Manual, 1995.

[11] R. Caceres and L. Iftode. Improving the performance of reliable transport protocols in mobile computing environments. *IEEE Journal on Selected Areas in Communications*, 13(5), 1995.

[12] Kevin Fall and Sally Floyd. Simulation-based comparisons of tahoe,reno, and sack tcp. In *Computer Communication Review*, volume 26 No. 3, pages 5 – 21, July, 1996.

[13] F.Swarts and H.C. Ferreira. Markov characterization of digital fading in mobile VHF channels. *IEEE Transactions on Vehicular Technology*, 43(4):977–985, Nov. 1994.

[14] C. L. Fullmer and J.J. Garcia-Luna-Aceves. Solutions to hidden terminal problems in wireless networks. In *Proc. SIGCOMM'97*, pages 39–49, Cannes, France, Sept. 1997.

[15] C.L. Fullmer. Personal communication, April 1998. Rooftop Communications Corp., Mountain View, CA 94041.

[16] J.J. Garcia-Luna-Aceves, C.L. Fullmer, E. Madruga, D. Beyer, and T. Frivold. Wireless Internet Gateways (WINGS). In *Proc. MILCOM'97*, Monterey,California, Nov. 1997.

[17] V. Jacobson and M. Karels. Congestion avoidance and control. In *Proc. SIGCOMM'88*, pages 16–19, Stanford, CA, Aug. 1988.

[18] Van Jacobson and Michael J. Karels. Congestion avoidance and control. In *SIGCOMM Symposium on Communications Architectures and Protocols*, pages 314 – 329, 1988.

[19] P. Karn and C. Partridge. Improving round-trip time estimates in reliable transport protocols. In *Computer Communication Review*, volume 17 No. 5, pages 2 – 7, August 1987.

[20] S. Keshav. REAL: A network simulator. Technical report, University of California, Berkeley, Berkeley, California, 1988. Technical Report 88/472.

[21] P. Lettieri, C. Fragouli, and M. Srivastava. Low power error control for wireless links. In *Proc. Third ACM/IEEE MobiCom Conference*, pages 139–150, Budapest,Hungary, Sept. 1997.

[22] M.Ritter and R.Friday. Personal communication, April 1998. Metricom Inc., Los Gatos, CA 95032.

[23] S. Murthy and J.J. Garcia-Luna-Aceves. An efficient routing protocol for wireless networks. *ACM Mobile Networks and Applications Journal*, 1(2), 1996.

[24] J.B. Postel. Internet Protocol. Technical report, SRI Network Information Center, September 1981. RFC 791.

[25] J.B. Postel. Transmission Control Protocol. Technical report, SRI Network Information Center, September 1981. RFC 793.

[26] W. Richard Stevens. *TCP/IP Illustrated, Volume 1.* Addison-Wesley, 1996.

[27] W.T. Strayer, B.J. Dempsey, and A.C. Weaver. *XTP The Xpress Transfer Protocol.* Addison-Wesley, 1st edition, 1992.

[28] F.A. Tobagi and L. Kleinrock. The effect of acknowledgment traffic on the capacity of packet-switched radio channels. *IEEE Transactions on Networking*, COM-26(6):815–826, 1978.

[29] V.Garg and J. Wilkes. *Wireless and Personal Communications Systems.* American Telephone and Telegraph Co., 1st edition, 1996.

[30] H.S. Wang and N. Moayeri. Finite-state Markov channel-A useful model for radio communication channels. *IEEE Transactions on Vehicular Technology*, 44(1):163–171, Feb. 1995.