Improving TCP Performance over Wireless Networks at the Link Layer

Christina Parsa J.J. Garcia-Luna-Aceves

Computer Engineering Department
Baskin School of Engineering
University of California
Santa Cruz, California 95064
E-mail: chris,jj@cse.ucsc.edu

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 applicable to collision-avoidance MAC protocols (*e.g.,* IEEE 802.11) 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 retransmission 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

1. Introduction

With the need to support end-to-end communication services to mobile hosts, wireless networks are quickly becoming an integral part of the Internet and reliable protocols such as TCP [24] must be supported over these networks. Mobile users requiring remote access to corporate LANs, file access and Web

transfers over wireless links must rely upon TCP to support their transactions. 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 as a result the overall throughput of the connection is drastically reduced. Methods to hide these losses from TCP is an active area of research [3][4][7][12][10][11].

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 [25] and Tahoe [18] 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 exponentially increased. 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 an acknowledgment (ACK) covering the missing packet is received, the congestion window is reduced by half the value at the time the packet was dropped and `congestion avoidance` begins, during which the congestion window increases linearly. 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. This becomes the predominant shortcoming of TCP over wireless links: the connection suffers long idle periods in which the sender is idle waiting for a timeout, and when the packet is finally retransmitted and recovered, the congestion window is reduced to one segment, thereby reducing throughput until the congestion window again grows to its optimal size.

In this paper, we present the transport unaware link improvement proto-

col (TULIP), and show by extensive simulation studies that TULIP allows 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. 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. More specifically, TULIP provides reliable service for packets carrying TCP data traffic, and unreliable service for other packet types, such as UDP traffic (*e.g.*, routing table updates and DNS packets) and TCP acknowledgments. TULIP doesn't provide reliable service to TCP ACKs because subsequent cumulative acknowledgments supersede the information in the lost ACK. The receiver buffers packets and passes them up to the next layer in order, thereby preventing TCP from generating duplicate acknowledgments in the event that a packet is missing from the expected sequential packet stream. This approach eliminates the need for a transport-level proxy [7], which must actively monitor the TCP packets and suppress any duplicate ACKs it encounters. An important feature of TULIP is its ability to maintain local recovery of all lost packets at the wireless link in order to prevent the unnecessary and delayed retransmission of packets over the entire path and a subsequent reduction in TCP's congestion window. 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. This aspect of TULIP is very much the same as prior selective repeat ARQ protocols [8]. However, TULIP is designed for efficient operation over the half-duplex radio channels available in commercial radios today by strobing packets onto the link in a turn-taking manner. We introduce a new feature, MAC Acceleration, in which TULIP interacts with the MAC protocol to accelerate the return of link-layer acknowledgments (which are most often piggybacked with returning TCP acknowledgments) 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. TULIP maintains no TCP state whatsoever, and makes no decisions on a TCP-session basis, but rather solely on a per-destination basis. This approach greatly reduces the overhead of maintaining state information when multiple TCP sessions are active for a given destination (as is common with Web traffic). From the transport layer's point of view, the path to the destination through a lossy wireless link simply appears to be a slow link without losses and TCP simply adjusts

accordingly.

Arguments against link-layer retransmissions [22] claim that reliable links incur unnecessary overhead due to the required transmission of acknowledgments in the reverse path (and therefore incur unnecessary power consumption). 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 [13]. However, a TCP session consists of two opposing streams: data flowing in the forward direction and ACKs flowing in the reverse, and therefore an efficient link-layer protocol can surely take advantage of these opposing flows by piggybacking link-layer ACKs with transport layer ACKs.

The rest of this paper is organized as follows. Section 2 discusses in more detail prior approaches to improving TCP over wireless networks. Section 3 describes TULIP and gives an example of its operation. Section 4 describes the implementation of TULIP used in our experiments. Section 5 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 when TULIP or Snoop [7] is used. The simulation experiments assume a simple configuration with a base station and a single wireless host roaming around the base station in order to compare TULIP's performance against the performance of the Snoop protocol [7] using the same type of experiments for which very good performance-improvement results have been reported. 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. 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. Section 6 gives our conclusion.

2. Related Work

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 [3] 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. Another approach demonstrating the validity of link-layer solutions shows analytically how to achieve improved throughput by insuring the buffer at the interface to the wireless connection is sufficient [12]. A simple stop-and-wait protocol is used over the wireless link to quickly retransmit packets before TCP discovers the loss.

DeSimone *et al.* [13] 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 (the most popular value for TCP implementations), but rather an ideal case in which a timeout occurs at the estimated round-trip time value. Balakrishnan *et al.* [6] 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 [4]. The TCP running at the base station buffers and falsely acknowledges packets to the source that have not yet been acknowledged by the receiver. The drawback to this approach is that it violates the semantics of TCP, and cannot therefore be easily deployed in the Internet. Another similar approach, M-TCP

[10], splits the TCP connection, but preserves TCP semantics. M-TCP aims to improve throughput for connections which exhibit long periods of disconnection. M-TCP is not a complete solution and the authors state the algorithm requires a good link-layer protocol to recover losses on the wireless link in conjunction with their protocol to handle the periods of disconnection.

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 [7], which is tailored to the case in which mobile hosts are attached to the Internet through a wireless link to a base station. The Snoop module runs above IP at the base station and is responsible for retransmitting lost packets and suppressing duplicate TCP acknowledgments by sniffing all packets entering an interface before they are passed on to IP. The Snoop protocol performs retransmissions of TCP packets when it detects two duplicate acknowledgment for any packet Snoop has seen previously and stored in its buffer. In addition, Snoop maintains two retransmission timers. The first timer is similar to the TCP timer in that one packet per windowful of data is timed to obtain a round-trip time estimate over the link. 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 [7]; 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 guesses about the pattern of losses. As we show in Section 5, 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.

It has also been suggested that TCP-SACK [14] can be used to improve TCP performance over wireless links [6]. TCP-SACK provides for end-to-end selective acknowledgment (SACKs) of received TCP segments; such SACKs are independent of the link-level acknowledgments used by TULIP. While TCP-SACK would certainly expedite the discovery of lost packets on wireless segments if no underlying link-level recovery were used; however, the algorithm would still respond adversely to lost segments on the wireless link and incorrectly interpret them as a sign of congestion. In addition, for connections with long end-to-end delays, 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.

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 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. TULIP uses a simple selective repeat retransmission strategy, similar to that of many prior link protocols [8], coupled with a packet interleaving[1] strategy to make efficient use of half-duplex links[2].

3.1. Service Provided

Although all link-level packets could be delivered reliably, such an 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 in-order delivery of reliable link-level packets (RLP) to the receiver, without duplication, and within a finite time. The unreliable service consists of a single transmission by the source of an unreliable link-level packet (ULP). In TULIP, TCP data packets are encapsulated in RLPs, while TCP acknowledgments (TACKs) with no data, UDP packets, and link-level acknowledgments (LACKs) are encapsulated in ULPs. TCP acknowledgment packets are transmitted with unreliable service, because there is generally more

[1] Packet interleaving is not to be confused with bit interleaving.
[2] All COTS radios today are half duplex.

than one packet in flight at a time and subsequent cumulative acknowledgments supersede the information in a lost 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 from the receiver. In contrast to prior reliable link protocols, in TULIP, the sender does not stop and wait to receive a correct LACK after either every packet (as in the traditional Stop-and-wait ARQ strategy [8]) or after an entire window of packets (*e.g.*, AIRMAIL [3]). Instead, for the case of unidirectional traffic, the sender simply allows enough time between the transmission of two data packets for the receiver to send either a LACK or a LACK together with a data packet (which could contain a TACK if one was available). This leads to the interleaving of packets from the two ends of the logical link on a turn-taking basis, as depicted in Figure 1. In this figure, a connection between nodes A and B is assumed to exist; packets from A to B or from B to A are labeled A→B and B→A respectively, and the signals between TCP/IP, TULIP and the underlying MAC protocol are indicated. Unidirectional traffic is depicted on the left, and bidirectional transfer of data (discussed in greater detail in Section 3.3.1) is shown on the right half of the figure. The relative positions of LACKs and TCP packets was used to illustrate that LACKs are not controlled or triggered by TACKs. The interleaving of data packets 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].

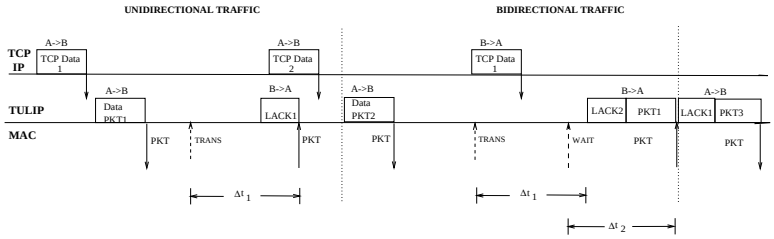


Figure 1. Packet interleaving over half-duplex link in TULIP from the perspective of source A.

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 and is accomplished with two signals: TRANS and WAIT. First, once TULIP passes TCP data packet #1 to the MAC layer, TULIP requires that the MAC layer notify TULIP that the transmission attempt for the packet has started. This is done by the signal TRANS. After receiving TRANS, TULIP starts a timer to wait for a minimum of Δt_1 seconds (or the arrival of a LACK) before sending the next packet. The timeout Δt_1 is given by the following equation:

$$\Delta t_1 = t_{PKT} + 2\tau + t_{ACK} + 2t_{TR} + 2t_c + t_p \quad (1)$$

where t_{PKT} is the time to transmit a data packet, t_{ACK} is the time to transmit a LACK (which could contain data), τ is the propagation delay over the wireless link (one each for the data packet and LACK), 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 LACKs, the MAC layer must inform TULIP of the need to wait longer than Δt_1. Because we are interested in contention-oriented MAC protocols (*e.g.*, CSMA, DFWMAC [1], FAMA [15]), the MAC is assumed to pass a WAIT signal, specifying the additional amount of time (Δt_2 in Figure 1) that a node sending packets needs to hold down beyond Δt_1, before it sends the next packet [3]. This procedure of interleaving allows the two sources to be self-clocking during the bidirectional transfer of data over the link.

Flow control and error recovery are managed by the sender through a sliding window of size W. Data packets are assigned sequence numbers modulo $2W$ and sender and receiver maintain a buffer of W packets to ensure correct operation [8]. 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

[3] In the case of TDMA this time could be passed with the TRANS signal. 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.

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.

```
Definition of Terms
    ACK = received pkt is an ACK
    WAIT = 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
    S = {SN_min, ..., SN_max}
    W = window size

Initialization
    Initialize SN_min and SN_max to 0

[This procedure is called when the sender receives
a signal or LACK packet from the MAC layer]
procedure receive_from_MAC ( incoming_pkt or signal )
    begin
        switch packet or signal type
            LACK:
                cancel Timer T_1
                process_received_ack()
                if(data to send)
                    send_packet()
            WAIT:
                cancel Timer T_1
                set timer T_2 = Δt_2(RTS.data_length)
            TRANS:
                set timer T_1 = Δt_1(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

procedure process_received_ack ( incoming_pkt )
    begin
        if pkt.CumAck ∈ S
            SN_min = (pkt.CumAck + 1) mod 2W
        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) AND |S| < W
                        SN_max = (SN_max + 1) mod 2W
                        send packet with sn = SN_max
                    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 2. Complete Sender Algorithm

The Sender transmits data packets according to the algorithm shown in Figure 2. A packet transmission is triggered by a timer that expires every Δt_1 (see Figure 5) or by the reception of a LACK. 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, 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 most recent transmission whenever a correct LACK covering the packet fails to arrive. Instead, if the transmission window is not full, the sender transmits the next incoming packet. This allows

new data to be processed at the receiver in the event LACKs are lost. 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, the sender retransmits the oldest unacknowledged packet in its window to force a LACK back from the receiver.

The algorithm followed at the receiver is shown in Figure 3. The receiver buffers all out-of-order data packets it receives and upon reception of a correct data packet, passes all in-order data packets to the network layer. The receiver sends a LACK every time it receives a data packet. To improve throughput, LACKs in TULIP consist of both a sequence number and a bit vector of length $W-1$. The sequence number is a cumulative acknowledgment (CumACK) that specifies the sequence number of the highest in-order data packet received. 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 [26]). Starting with the sequence number specified in the CumACK, the bit vector notifies the sender about the successful or unsuccessful reception of up to $W-1$ additional data packets. A 0 in the vector indicates that a packet with the corresponding sequence number has not been received correctly; a 1 indicates otherwise.

TULIP uses a connection establishment strategy that is much the same as those reported in the literature for selective repeat protocols [8].

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. Our implementation of TULIP runs on top of FAMA-NCS [15] and includes a MAC-acceleration feature, depicted in Figure 4(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 there is a data packet waiting to be returned and the packet payload is 40 bytes or less (which is large enough to carry a TCP ACK packet), then this packet is piggybacked 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). If there is no available data packet, then the TULIP ACK is transmitted immediately. On the other hand, if the packet is larger than 40 bytes, FAMA is instructed to immediately send an RTS

```
Initialization
    CumACK = −1
    BitVector = {0, ··· , 0}

[this procedure is called when a pkt is received]
procedure process_incoming_pkt ( incoming_pkt.sn )
    begin
        if incoming_pkt.sn ∈ {CumACK + 1, ··· , CumACK + W}
            if incoming_pkt.sn = (CumACK+1) mod 2W
                release to network layer
                release any other in sequence packets
                for each packet released
                    shift left BitVector
                CumACK ← LastReleased.sn
            else if packet not in buffer
                accept packet into buffer
                set corresponding bit in Bit Vector
            else
                drop packet
                return
            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
        else
            drop packet
        end if
    end process_incoming_pkt
```

Figure 3. Complete Receiver Algorithm

to reserve the channel for a long packet, as shown in Figure 4(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.

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.

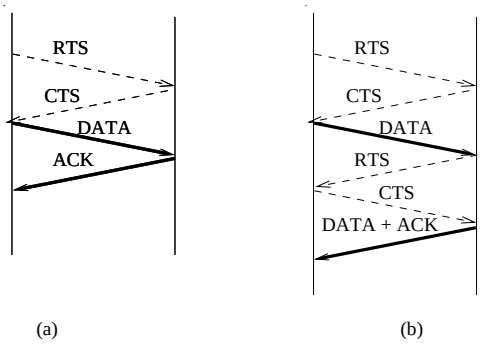


Figure 4. (a)**MAC Acceleration:** FAMA transmits TULIP data packet and returns ACK without another RTS/CTS exchange. Returning TULIP ACK may contain a TCP ACK. (b)FAMA exchange with large ACK packet (encapsulated data packet) requires another RTS/CTS exchange.

- **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 5 with the label **Unidirectional Traffic**, is straightforward: The link-layer passes a data 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 transmission timer T_1 equal to the value Δt_1 calculated by Equation 1. Note that with MAC Acceleration, an additional RTS/CTS exchange is not needed for the receiver to transmit the corresponding ACK. 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 2, selects the next packet for transmission. It is shown in Figure 10(a) of our simulation results (Section 5), that with no errors on the link TULIP provides higher throughput than Snoop precisely because of this 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 5 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 WAIT signal to the link-layer when the MAC layer receives an RTS. 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.

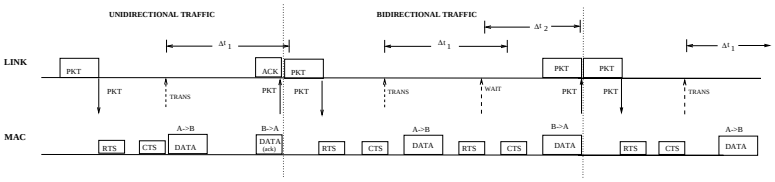


Figure 5. Unidirectional and bidirectional Traffic from the perspective of Node A in a logical link with Node B.

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 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 at A 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. Transmission Example for Unidirectional Traffic

A sample transmission session is given in Figure 6 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.

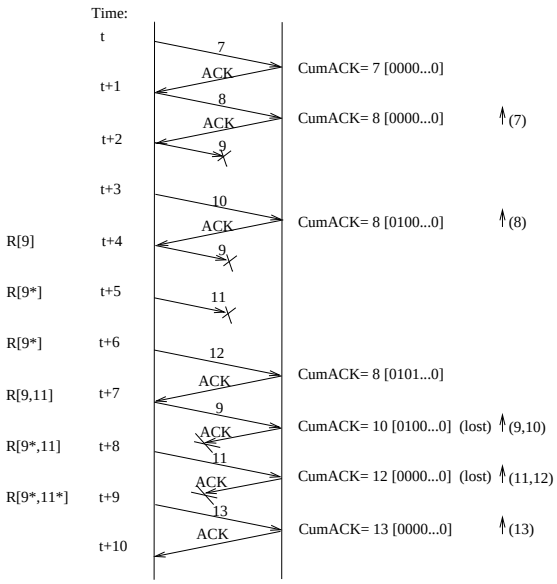


Figure 6. Example of Transmission. Window Size = 8.

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 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 $t+9$ the current retransmission list has been completed so a new packet (#13) is selected for transmission. When its acknowledgment is received at time $t+10$ the bit vector indicates that all packets have been received correctly.

4. Implementation

We have implemented TULIP and Snoop [7] in the C++ Protocol Toolkit (CPT) [9][4]. Wireless nodes in our simulation run the protocol stack shown in Figure 7. 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) [17], and in hosts attached to the WINGs. The IEEE 802.11 specifications are used at the physical layer to emulate 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 also be altered through scripts. Either TULIP or a dummy link-layer without retransmissions is used at Layer 2. Our implementation of TULIP runs on top of FAMA-NCS [15] with the MAC acceleration feature described in Section 3. TULIP in turn interacts with IP [23] and the wireless Internet routing protocol (WIRP) [21] for packet forwarding. The only higher-level protocol requesting reliable link service

[4] We thank Rooftop Communications Corporation for donating the toolkit.

from TULIP is TCP, which in our implementation, has been ported from the TCP Reno code contained in the REAL simulator [19].

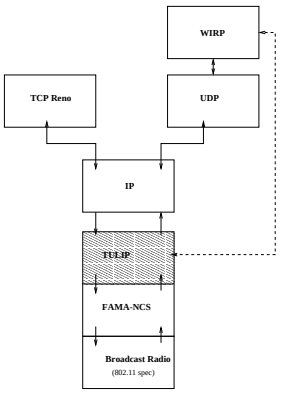


Figure 7. Protocol stack of wireless node.

The code of the Snoop protocol [7] was modified from the on-line FreeBSD implementation[5] 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 overall code was strictly maintained (when possible only data structures specific to the FreeBSD implementation were adapted to 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 $2 * rtt$ to match that of published work [7]. For optimization purposes retransmissions are performed after the receipt of 2 duplicate acknowledgments [5]. In addition, as specified [7], only 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 and is discussed in Section 5.2.

5. Performance

In this section we present the results of a series of simulation experiments used to compare the performance of TULIP, the Snoop protocol [7] and TCP Reno implemented with no underlying link-layer retransmissions (referred to as No LL). For both steady-state and in the early stages of a TCP data transfer,

[5] We thank H.Balakrishnan for providing on-line source code at ftp://daedalus.cs.berkeley.edu/pub/snoop/

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 important because it reveals if a protocol makes unnecessary retransmissions.

Given that the transport protocol is transparent to TULIP, TULIP can improve the performance of any transport protocol that can confuse packet losses in the wireless segment with congestion. We chose to use TCP Reno as the basis for comparison because it is widely used. We did not consider TCP-SACK, because it still confuses packet loss over the wireless link as a sign of congestion, requires modification of the receiver TCP at the mobile host and is not currently widely deployed. The MAC acceleration could not be used with Snoop because Snoop has no access to the MAC layer (the Snoop agent resides above IP); therefore, Snoop cannot instruct the MAC layer to transmit an ACK differently than data. It is precisely because TULIP resides at the link layer that it is able to interact with and control the MAC sub-layer, thereby taking advantage of MAC acceleration.

The test case used in our study is the same used by the authors of Snoop to describe its performance [7]. Although TULIP can operate in ad-hoc networks, our intent in this paper is to compare TULIP's performance against Snoop's, which is one of the best performing TCP improvements for wireless networks reported to date. In this configuration, shown in Figure 8, an Internet source sends data to a wireless host via a base station. Of primary interest is traffic flowing toward the wireless receiver, as is the case of a mobile user downloading files. This type of configuration is also typical of Web traffic; however, this and other interesting simulation scenarios, such as multiple receivers, cannot be shown in this paper, but are planned for our future work. The topology is such that a TCP source transfers 10Mbyte of data from a host located on a wired segment via a base station to a wireless receiver. 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 improvement 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.

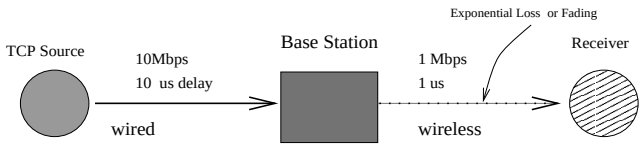


Figure 8. Topology for Experiments 1 - 3

The following parameters are fixed for all experiments: the 802.11 specifications [1] are used for the physical layer, the channel capacity from the wired source to the base station is 10Mbps, the wireless transmission rate is 1Mbps, file transfers are 10Mbytes with 1400 byte data packets, TULIP's window size is 8 packets, and the offered traffic is a Poisson source with an average rate of 1Mbps. These specifications are the same as those used in the Snoop experiments [7], except for the radio characteristics (including a lower transmission rate for our experiments).

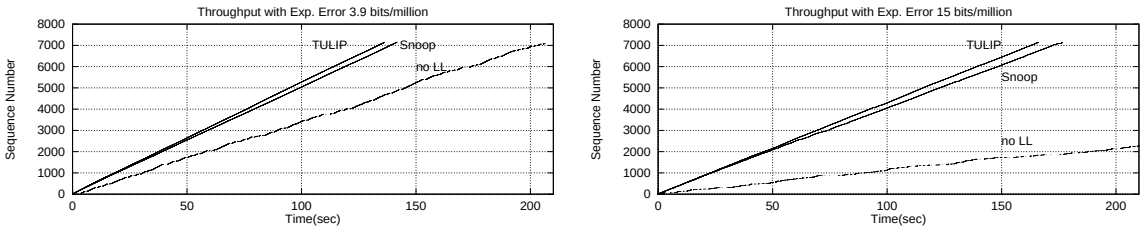
5.1. Experiment One: Low Error Rates

In this experiment the bit error rate is varied from 0 to 15 bits/million and the TCP receiver window is set at 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.

5.1.1. Throughput

Figure 9(a) shows the progression of sequence numbers for a 10Mbyte TCP file transfer with a bit error rate (BER) 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. TULIP performs slightly better than the Snoop protocol 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 returns LACKs from the receiver to the sender with MAC Acceleration as described in Section 3. When the Snoop protocol is running at the network layer, the TCP acknowledgment packets from receiver to sender must of course compete for channel access with the data packets flowing in the forward direction.

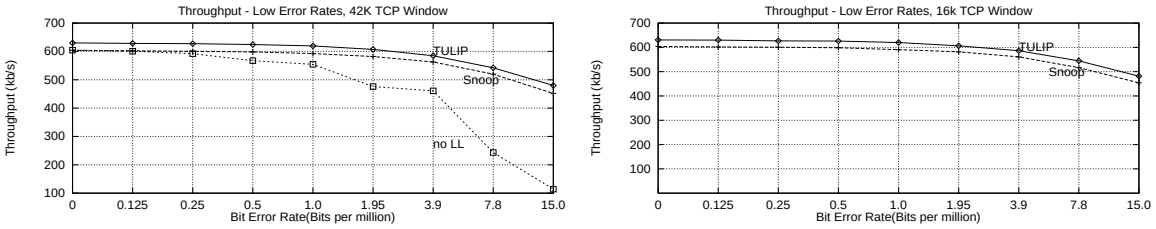
Figure 9(b) shows the progression of sequence numbers with a BER of 15 bits/million, or a packet loss rate of approximately 16%. It is again apparent



(a) (b)

Figure 9. Experiment 1: TCP Sequence number growth. (a) BER = 3.9 bits/million = 1/256Kbytes (b) BER = 15 bits/million = 1/64Kbytes. Receiver window 42 Kbytes.

in this transfer that TCP is suffering considerably due to losses on the wireless link. TULIP and the Snoop protocol, on 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.



(a) (b)

Figure 10. Experiment 1: Average throughput for all three protocols with varying BER. (a) 42k receiver window (b)16k receiver window

Figures 10(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. When the link errors are zero the graphs show that MAC Acceleration provides a 4 – 5% improvement in throughput. 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 BER rises; however, their overall throughput is consistently better than unassisted TCP. The reduction in throughput is due to the numerous retransmissions that occur as error rates increase. 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.

5.1.2. Goodput

Figure 11 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 slightly from ideal goodput as the errors on the link increase. Table 1 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 BER 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 retransmission of 158 redundant packets.

Figure 11. Experiment 1: Goodput for all three protocols with varying packet error rates and a receiver window of 42Kbytes.

5.1.3. A Closer Look at TCP

In this section we investigate why TCP makes so many unnecessary retransmissions. First of all, unless a connection has a large number of outstanding

Comparison of Goodput for TULIP Protocol, Snoop and no LL						
Protocol	BER (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 1
Experiment 1: Ideal and achieved goodput, number of TCP timeouts and redundant packets transmitted over the wireless link for a BER of 15 bits per million and a receiver window of 42Kbytes.

packets and enough ACKs flow back from the receiver, TCP cannot recover from two losses per window without resorting to a timeout. Figure 12(a) shows the sequence number growth and returning 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.

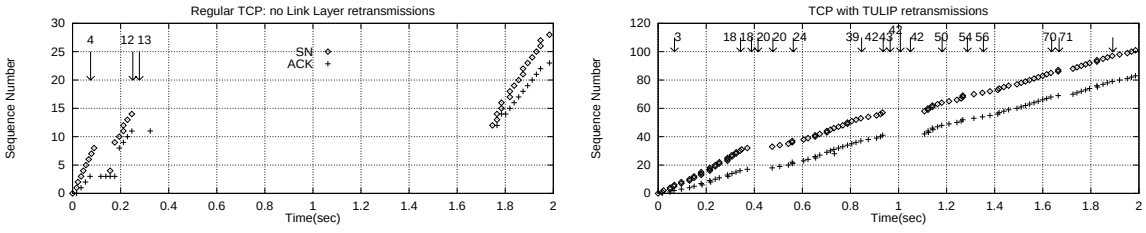


Figure 12. Experiment 1: Sequence number and ACKs at the source for the first 2 seconds of the TCP transfer. Packets dropped on the channel are shown with arrows. BER = 15 bits per million. (a) no link retransmissions (b) TULIP protocol

In contrast, as depicted in Figure 12(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 12(a), we see that TULIP's retransmissions improve the situation considerably: TULIP suffers from 17 packet losses (compared to only 3 losses in Figure 12(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.

5.1.4. TCP Round-trip Time

Figure 13 shows the estimated RTT and deviation calculated by TCP throughout the session, the actual measured RTT and the retransmission timeout value for TCP with a channel BER of 15bits/million and TULIP retransmissions. TCP's retransmission timer [6] generally produces a timeout value with plenty of slack for TULIP to quickly retransmit packets. It is interesting to note in the plot that the variation in RTT is generally within the 500ms granularity of the TCP timer in our implementation. Arguments against link-layer approaches [13] 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! 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.[7]

5.1.5. Packet Delay

Figure 14(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.

[6] The TCP timeout value in our implementation is calculated as the estimated round-trip time plus four times its deviation and has a granularity of 500ms.

[7] 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.

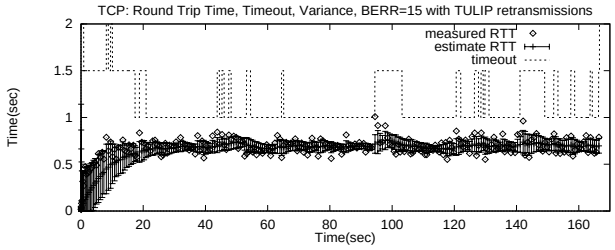
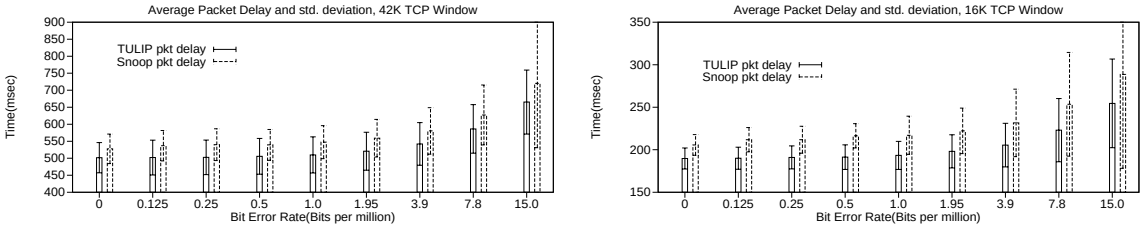


Figure 13. Experiment 1: RTT measurement and estimate for TULIP with a BER of 15 bits/million with receiver window of 42Kbytes.

This becomes apparent as the error rate increases and Snoop's variance jumps up at a BER of 15 bits per million when it has trouble with scattered losses within a window and with the occasional loss of retransmitted packets.



(a) (b)

Figure 14. Experiment 1: Average packet delay and std. deviation for TULIP and Snoop protocols. (a)Receiver Window is 42Kbytes. (b)Receiver Window is 16Kbytes.

Figure 14(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 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.

5.2. 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 [16][20] 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%).

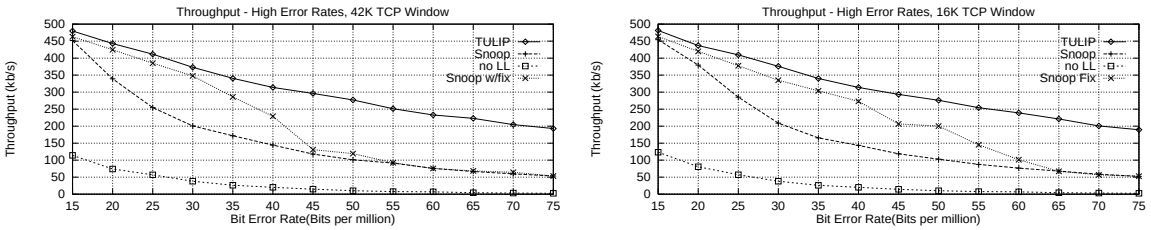
This section shows two different plots for the Snoop protocol. After running simulations with high error rates, Snoop appeared to perform poorly. After examining the source code, we discovered that when many losses occurred in succession, causing Snoop to retransmit multiple lost packets per window, 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.

5.2.1. Throughput

TCP's throughput is shown in Figures 15(a) and 15(b) for receiver window sizes of 42Kbytes and 16Kbytes, respectively. These graphs clearly show that, as the bit error rate increases, all three protocols show a reduced turn in throughput. With a BER of 15 bits/million the TCP connection with no link-layer retransmissions (labeled no LL) has degraded significantly and comes to a near standstill for error rates above 30 bits/million. After a BER of 30 bits/million Snoop's throughput drops off quickly and diverges from TULIP's throughput, which decreases slowly as error rates increase. These error rates are characterized by many multiple losses per window of data. TULIP can easily recover from these episodes and 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 show no appreciable difference in throughput for different window sizes.

5.2.2. Delay

Figure 16(a) shows the average end-to-end packet delay and standard deviation with a receiver window of 42Kbytes for Snoop and TCP. The TCP transfers with no link layer retransmissions exhibit a lower average delay for bit error rates up to 60 bits/million; however, considering the corresponding deviation, the delay is larger than the delay seen by the Snoop protocol. Because the session without underlying retransmissions operates with a small congestion window (and hence

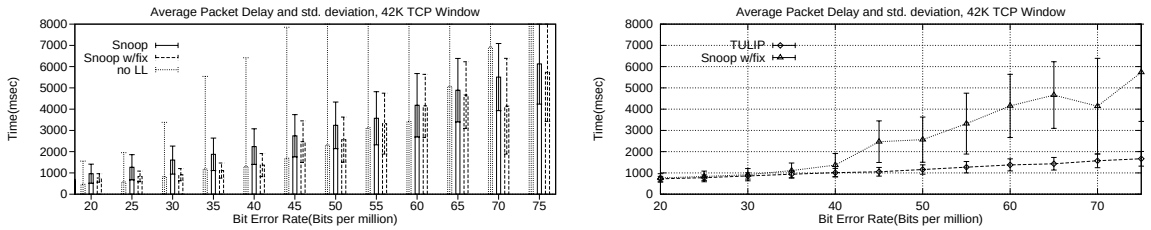


(a) (b)

Figure 15. Average throughput for all three protocols with high error rates. (a) 42Kbyte window (b) 16Kbyte window.

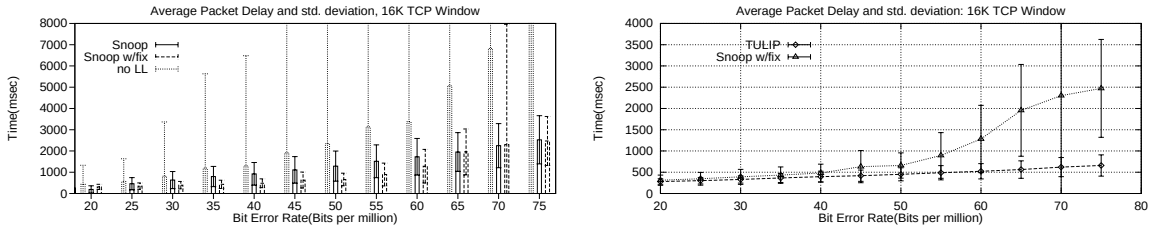
very little queuing at the base station) due to many TCP timeouts, packets are either delivered with very low delay or they suffer from high delay when they are dropped and undergo retransmission. The plot shows that the delay and deviation for Snoop with the fix is significantly lowered. Because the performance of Snoop with the fix is better, it is compared to TULIP in Figure 16(b). This plot shows that the delays and deviation with the Snoop protocol are significantly larger than with TULIP once error rates exceed 35 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. Snoop must rely heavily on its timer and cumulative acknowledgments 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. In addition, if retransmitted packets need to be again 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.

When the receiver's window is reduced to 16Kbytes, as shown in Figures 17(a) and (b) we notice that the end-to-end delays are greatly reduced. As error rates increase Snoop suffers from a very high standard deviation and an average packet delay of more than twice that of TULIP. These results again support our recommendation of a smaller receiver window size.



(a) (b)

Figure 16. Experiment 2:Average packet end-to-end delay and std. deviation with high error rates and 42Kbyte receiver window. (a) Snoop, Snoop w/fix, and No LL (b) Snoop w/fix and TULIP



(a) (b)

Figure 17. Experiment 2:Average end-to-end packet delay and std. deviation with high error rates and 16Kbyte receiver window. (a) Snoop, Snoop w/fix and No LL (b) Snoop w/fix and TULIP

5.3. 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. 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.

5.3.1. Experiment 5a: Uniform Distribution of Burst Losses

In the method used by Balakrishnan et. al. [6], 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 2. 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 same would the case 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 reach the receiver in error, even though the corresponding RTS and CTS packets did not.

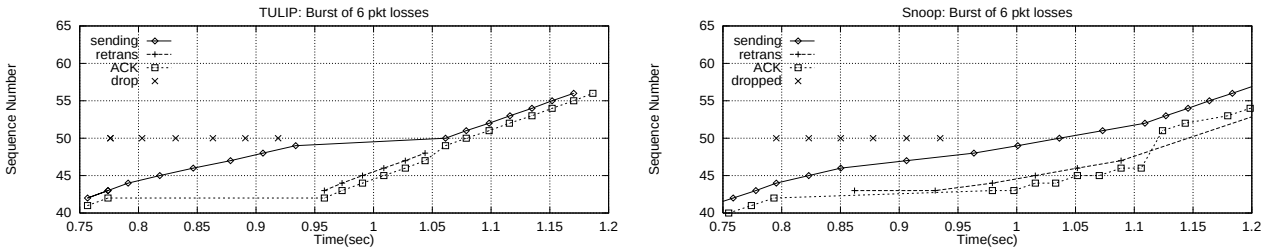
Bursts Distributed every 64Kbytes					
Burst Size #packets	TULIP Throughput(Kbps)	Snoop Throughput(Kbps)	Δ (Kbps)	TULIP Delay $\pm$ dev.(ms)	Snoop Delay $\pm$ dev.(ms)
2	587.3	562.6	24.7	540$\pm$56	582$\pm$60
4	550.0	527.6	22.4	579$\pm$74	621$\pm$84
6	516.1	496.4	19.7	618$\pm$98	660$\pm$114

Table 2
Experiment 5a: 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 18 (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 $t = 0.87sec$ and $t = 0.93$ for the first loss; thereafter, it must interpret the values of returned cumulative acknowledgments to guess if additional errors have occurred. It should also be noted that in this transfer, Snoop only suffers from 4 original packet losses (and 2 retransmissions), whereas 6 original packets are dropped for TULIP.

5.3.2. Experiment 5b: Markov Model of Burst Losses

The method to simulate channel fading consists of a two-state Markov model taken directly from the work by Lettieri *et al.*[22], which is also discussed by Wang *et al.* [30] and Swarts *et al.* [27]. Briefly, 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 error 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

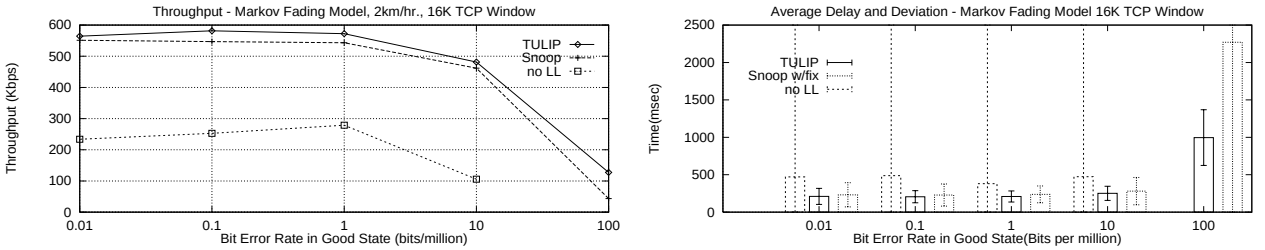


(a) (b)

Figure 18. Experiment 5a: Burst loss of 6 packets. (a)TULIP (b)Snoop

state should occur. Thus, it is possible to change states many times within one packet.

We have performed this experiment for a pedestrian speed of 2km/hr and the results are presented in Figures 19(a) and (b) as the BER in the *Good* state is varied from 0.01 to 100 bits/million and the loss probability in the *Bad* state held constant at 50%. 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 19(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 19(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 tighter bound on the deviation.



(a) (b)

Figure 19. Experiment 5b: TULIP and Snoop protocols during Markov Fading Model. Loss probability in bad state is 50% and BER in good state is varied. Pedestrian speed 2km/hr. (a) Throughput (b) End-to-end delay and standard deviation

6. 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, TULIP provides significantly improved end-to-end delay and delay variation. In addition, reducing the size of the receiver's advertised window can help to alleviate the queuing delay. We have examined 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 but 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.

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 with any current or future version of TCP (*e.g.*, TCP-SACK), 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, by controlling the MAC layer, TULIP conserves wireless bandwidth by piggybacking TCP ACKs with link-layer ACKs and returning them immediately across the channel through MAC Acceleration.

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] 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.

[4] 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.

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

[6] 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.

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

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

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

[10] Kevin Brown and Suresh Singh. M-TCP: TCP for mobile cellular networks. In *Computer Communication Review*, volume 27 No. 5, pages 19 – 43, Oct., 1997.

[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] H. Chaskar, T.V. Lakshman, and U. Madhow. On the design of interfaces for TCP/IP over wireless. In *MILCOM '96 Conference Proceedings*, volume 1 No. 3, pages 199–203, Oct. 1996.

[13] A. DeSimone, 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.

[14] 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.

[15] 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.

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

[17] 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.

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

[19] S. Keshav. Real: a network simulator. Technical report, University of California, Berkeley, Berkeley, California, 1988. Technical Report 88/472.

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

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

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

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

[24] J.B. Postel. Transmission control protocol. Technical report, SRI Network Information Center, September 1981. RFC 793.

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

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

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

[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.