Improving TCP Performance with Bandwidth Estimation and Selective Negative Acknowledgment in Wireless Networks

Rung-Shiang Cheng and Hui-Tang Lin

Abstract: **This paper investigates the performance of the transmission control protocol (TCP) transport protocol over IEEE 802.11 infrastructure based wireless networks. A wireless link is generally characterized by high transmission errors, random interference and a varying latency. The erratic packet losses usually lead to a curbing of the flow of segments on the TCP connection and thus limit TCP's performance. This paper examines the impact of the lossy nature of IEEE 802.11 wireless networks on the TCP performance and proposes a scheme to improve the performance of TCP over wireless links. A negative acknowledgment scheme, selective negative acknowledgment (SNACK), is applied on TCP over wireless networks and a series of ns-2 simulations are performed to compare its performance against that of other TCP schemes. The simulation results confirm that SNACK and its proposed enhancement SNACK-S, which incorporates a bandwidth estimation model at the sender, outperform conventional TCP implementations in 802.11 wireless networks.**

Index Terms: **Congestion control, IEEE 802.11, selective negative acknowledge ment (SNACK), transmission control protocol (TCP).**

I. INTRODUCTION

Mobile, high-speed wireless networks have attracted considerable interest in recent years due to the proliferation of mobile computing devices and their ease of deployment. New standards for wireless local area networks (WLANs) have greatly improved data transmission speeds and have prompted the development of high-speed mobile data communication services which are expected to make possible a wide range of new applications over the coming years.

IEEE 802.11 has emerged as the standard of choice for local wireless data networks nowadays. 802.11b provides transmission speeds of up to 11 Mbps with fall-back rates of 5.5 Mbps, 2 Mbps and 1 Mbps, respectively. To meet the expected growth in bandwidth demand in the future, the channel speeds of the 802.11 protocols continue to increase. For example, 802.11a and 802.11g both support 54 Mbps, while enhanced versions operate at speeds up to 108 Mbps.

To support the requirement for reliable transmission, a significant amount of today's Internet traffic, including WWW, FTP, E-mail, and remote access traffic, is carried by the TCP transport protocol. This protocol was originally designed for wired, relatively reliable propagation networks, and its design makes numerous assumptions typical of such wired environments [1], [2]. However, a wireless link is generally characterized by an unpredictable bit-error rate and varying latencies. The propagation latency, limited bandwidth, and error-prone links of wireless environments limit the application performance. Hence, wireless environments pose formidable challenges when attempting to provide reliable, end-to-end data transmission for transport protocols such as TCP.

When packets are lost in networks for reasons other than congestion, the invoked congestion control routines curb the flow of segments on the TCP connection, and therefore reduce the TCP end-to-end throughput. Several proposals to resolve this problem have been reported in recent years [3]–[7]. Basically, the proposed methods either modify the TCP mechanisms themselves or modify the medium access control (MAC) protocol to enable TCP to differentiate between transmission errors and network congestion. Although these approaches are effective in some cases, they produce less satisfactory results in others.

A recently proposed development called congestion coherence [8] proposes TCP source and receiver to be modified to correlate packet losses with the return packets with the explicit congestion notification (ECN) [9] mechanism. This scheme takes advantage of the temporal coherence of ECN marks to find the causes of packet losses. However, it requires ECN to be fully supported in all routers in the wired network.

This study aims to enhance the TCP performance by modifying only the TCP transport layer protocol; leaving the functionality of the MAC protocol and the intervening network unchanged. A negative acknowledgment with bitmap scheme, named selective negative acknowledgment (SNACK), is introduced for TCP congestion control in wireless environments. Ns-2 simulations are performed to investigate the performance of TCP using various recovery schemes over an 802.11 infrastructure based environment. In the simulations, the *Gilbert-Elliot* burst error model is applied to mimic the behavior of link-level channel error.

The remainder of this paper is organized as follows. Section II provides a brief description of the IEEE 802.11 standard and the TCP protocol. Section III describes various schemes presented in the literature designed to improve the performance of TCP over wireless links. Section IV presents the SNACK scheme and its proposed enhancement, SNACK-S. Section V describes the results of the performance evaluation simulations. Finally, Section VI presents some brief conclusions.

II. IEEE 802.11 BACKGROUND

Due to differences in the physical layer, wireless devices and networks possess distinct characteristics which differentiate

Manuscript received May 18, 2006; approved for publication by Luciano Lenzini, Division II Editor, July 4, 2007.

R.-S. Cheng is with the Department of Electrical Engineering, National Cheng Kung University, Taiwan, email: chengrs@nsda.ee.ncku.edu.tw.

H.-T. Lin is with the Department of Electrical Engineering and Institute of Computer Communication Engineering, National Cheng Kung University, Taiwan, email: htlin@mail.ncku.edu.tw.

them from wired networks and elements. Depending on the network formation method and the network architecture, two operational modes can be identified for 802.11 technologies, i.e. *infrastructure-based* and *infrastructureless* (or ad hoc). The former provides wireless access and the range extension of preconstructed communication infrastructures. In such networks, the wireless stations connect to wireless access points (APs), which function as a bridge for every packet sent or received by the wireless station. Conversely, in ad hoc networks, temporary networks are formed by an arbitrary set of independent nodes within a limited area and capable of self-configuration without the need of stationary infrastructure networks.

The IEEE 802.11 standard specifies both the physical layer and the MAC layer [10]. The 802.11 MAC protocol defines two different service types, namely a contention-based distributed coordination function (DCF) and a contention-free point coordination function (PCF). The DCF service operates strictly alone in ad hoc networks, but can operate either alone or with the PCF service in infrastructure-based networks. DCF is the basic access method for 802.11 and is based on the *carrier sense multiple access-collision avoidance* (CSMA/CA) scheme [11]. DCF comprises both a basic access method and an optional virtual carrier sensing method based on RTS/CTS exchanges.

In 802.11 DCF, priority access to the wireless medium is controlled by applying an inter-frame space (IFS) time between the frame transmissions. The station may proceed with its transmission if the medium is sensed to be idle for an interval larger than the distributed inter-frame space (DIFS). If the medium is currently busy, or becomes busy during this interval, the transmitter defers the frame transmission until it detects a DIFS. At this point, the transmitter selects a random interval, referred to as the *backoff time*, to determine the moment at which to commence transmission. The backoff time is an integer number of slots, uniformly chosen from the interval (0, $CW-1$), where CW is the backoff window, also referred to as the *contention window*. The backoff number counts down slot-by-slot, and when it reaches zero, the frame is transmitted.

Due to the nature of wireless signal propagation, stations in the network are unable to detect a collision simply by listening to their own transmissions. Therefore, an immediate positive acknowledgment (ACK) technique is employed to confirm the successful reception of a frame. Specifically, having received a frame, the receiver waits for a short inter-frame space (SIFS) and then transmits a positive MAC acknowledgment to the transmitter, confirming that the frame has been correctly received. The SIFS is deliberately assigned a shorter interval than the DIFS in order to assign the receiving station a higher priority than any other stations waiting to make a transmission. The positive ACK is only transmitted if the frame is received correctly. Hence, if the transmitter does not receive an ACK, it assumes that the data frame must have been lost and therefore schedules a retransmission. (For the details of the basic operations involved in 802.11 DCF, please refer to [12]).

To alleviate the hidden-station problem, 802.11 DCF also provides an optional channel access method using a virtual carrier sensing mechanism based on the use of two special control frames, namely, request-to-send (RTS) and clear-to-send (CTS). Before transmitting a frame, the transmitter transmits an RTS frame to ask the receiver. Once the receiver receives this RTS frame, it waits for the specified SIFS interval and then sends a CTS frame to the transmitter. The neighbors of both the transmitter and the receiver overhear these frames and consider the medium to be reserved for the duration of the transmission. Although this mechanism can reduce collisions, which helps to combat hidden-terminal problem, RTS/CTS exchange introduces delay, and consumes channel resources. For this reason, this mechanism is only used to reserve the channel for the transmission of a long DATA frame.

In unicasting, the 802.11 MAC layer retransmits a packet a maximum of n times before discarding it. However, even with the retransmission mechanism in the MAC layer, packets may still be discarded without being handed over to the transport layer due to spurious interference or channel collisions [5], [13]. Since TCP regards packet loss as a sign of network congestion, it reacts by reducing the size of its congestion window, which in turn leads to a reduction of the TCP performance [14]. This issue is particularly serious in wireless networks with large bandwidth delays. In such networks, recovery from the multiple packet losses which may occur with a large congestion window can lead to considerable delays, and this further degrades the TCP performance.

III. RELATED STUDIES OF TCP OVER 802.11 WIRELESS LINKS

TCP is a reliable end-to-end acknowledgment-clocking window based protocol. TCP controls the sending rate using a congestion window parameter. The manner in which TCP adjusts the congestion window depends on the size of the current congestion window size relative to the prescribed slow start threshold. When congestion window is less than slow start threshold, TCP is said to be in its slow start phase and the size of the congestion window is increased exponentially. However, when the threshold limit is reached, TCP enters the congestion avoidance phase and increases the congestion window linearly [2]. The TCP sender updates the size of the congestion window for each ACK received in accordance with the following function:

(1) Receipt of new (non-repeated) ACK:
If $W < W_t$, set $W = W + 1$ (Slow start phase), else set $W = 1 + 1/W$ (Congestion avoidancephase).

(2) Receipt of duplicate ACK:
Increment duplicate ACK count for segment being ACKed. When duplicate ACK count exceeds specified threshold, retransmit "next expected" packet; set $W_t = W/2$, then set $W = W_t$.

(3) Upon timer expiry:
Set $W_t = W/2$, then set $W = 1$; recover lost packets from slow start phase.

Parameters W and W_t are referred to as the current congestion window size and the slow start threshold, respectively. Because a TCP sender increases its window size each time a new ACK is received from the receiver and decreases the congestion size by a factor of two when the path is congested (i.e.,

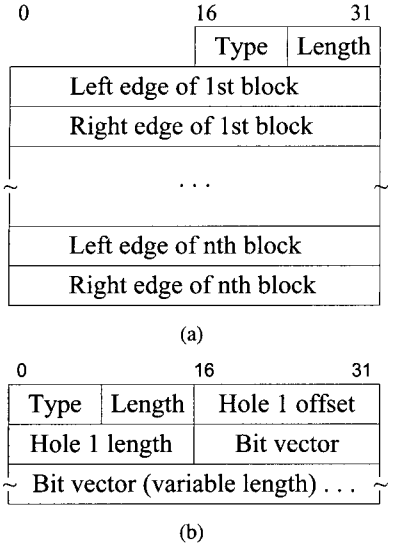

Fig. 1. Structure of SACK and SNACK options: (a) SACK, (b)SNACK.

packet loss occurs), TCP is commonly referred to as an additive-increase and multiplicative-decrease (AIMD) algorithm.

TCP Tahoe [1], Reno [2], NewReno [15], and SACK [16] are some of the commonly deployed variants of window based congestion control technique. These traditional TCP congestion control/avoidance techniques are proven to be effective in the wired networks of Internet. But this is not true in wireless networks, where packet loss and delay become more serious due to congestion as well as time varying nature of the wireless channel. This section summarizes various mechanisms reported in the literature for improving the performance of TCP over wireless links.

A. Split-Connection Schemes

One method for dealing with erratic errors on a wireless link is to split the wireless portion of the network from the conventional TCP connection. The indirect-TCP scheme (commonly referred to as I-TCP) in [4] is one such scheme. I-TCP splits an end-to-end TCP flow into two separate TCP connections, i.e., a regular TCP connection over the wired network and a wireless TCP connection over the wireless link. Under this approach, any corrupted packets are retransmitted directly through base stations on the wireless part of the path such that the wired connection is unaware of the wireless losses. In this way, the transport layer is isolated from the erratic behavior of the wireless link. However, a major drawback of this split-connection approach is that it fails to preserve the TCP end-to-end semantics because an ACK originating from the base station may reach the sender side before the corresponding data packet reaches its destination.

Balakrishnan *et al.* [3], [17] proposed the use of a snoop agent to confine the retransmissions to the wireless part of the path. In their approach, the TCP packets were sniffed on a per-connection basis such that the corrupted packets could be transparently retransmitted without breaking the TCP end-to-end semantics. Under this approach, if a packet is deemed to be lost on the wireless link, the packet is retransmitted by the agent from its local cache and the end-to-end semantics are maintained by suppressing the ACKs until the retransmitted packet has been successfully received at its destination.

However, this local-retransmission approach requires the base station to maintain state information and to cache unacknowledged packets for every TCP connection passing through it. Furthermore, performing retransmissions locally may affect the round-trip time (RTT) estimation by the TCP source, and this in turn can impair the ability of TCP to detect congestion losses.

B. Link-Layer Schemes

As discussed earlier, the principal cause of TCP performance degradation over wireless links is the inability of TCP to determine whether a packet was lost as a result of packet corruption or as a consequence of congestion. Accordingly, the explicit loss notification (ELN) scheme [18] has been developed to provide TCP with the ability to differentiate between corruption and congestion losses, thus allowing the sender to react appropriately in each case. Under the ELN scheme, the base station keeps track of all the TCP segments which arrive over the wireless link, but does not cache any of them. Whenever the base station detects a non-congestion loss, it sets the ELN bit in the subsequent TCP header and propagates the segment to the receiver, which then echoes it back to the TCP sender. When it receives the ELN notification, the TCP sender at the wireless host retransmits the lost packet without invoking congestion control. However, the ELN mechanism does not take ACK losses into account. Furthermore, implementing ELN requires not only modification of the transport layer of the sender-receiver-pair, but also the use of an ELN agent at the intermediate base station.

C. End-to-End Schemes

When multiple packets are lost in the same transmission window, conventional TCP schemes can only infer the first packet to have been lost from the duplicate ACKs received. After retransmitting the lost packet, the sender must then wait to receive new duplicate ACKs before it can detect the next lost packet. As a result, conventional TCP (e.g., Tahoe, Reno, and NewReno) can only recover from a single loss event per RTT. However, wireless links may frequently corrupt multiple packets per window, leading to a high-error recovery delay, and particularly over long delay paths [14].

One approach for dealing with multiple segment losses is for the data-receiver to inform the sender of the segments which it has received. Selective acknowledgment (SACK) [16] is one such scheme. In SACK, the data-receiver provides the sender with an image of the set of non-contiguous data blocks currently residing in its buffer by applying a so-called SACK option in the TCP header of the ACK packets. As shown in Fig. 1(a), the first block in a SACK option identifies the most-recently-received data block, while the remaining blocks repeat the most recently reported SACK blocks. Using this information, the sender can then retransmit only the segments which have been lost.

SACK has been proposed as a complement to the traditional cumulative ACK scheme. A SACK option which specifies n blocks will have a total length of $8 \times n + 2$ bytes (see Fig. 1(a)). However, since the maximum length of the TCP header options

is 40 bytes [1], when other options, e.g., Timestamp, are also used, the maximum number of SACK blocks which can be signaled in a single SACK option is reduced to just three. Consequently, a single ACK with the SACK option cannot provide the complete status of the receiver buffer if the number of blocks required to do so is greater than three. Therefore, the sender is obliged to examine multiple successive ACKs in order to acquire a complete image of the receiver buffer. However, as described in [19], if a number of successive ACK packets are dropped in the network, the sender may be unaware that the receiver has received a packet, and may therefore execute redundant retransmissions. This problem is particularly acute in error-prone wireless environments.

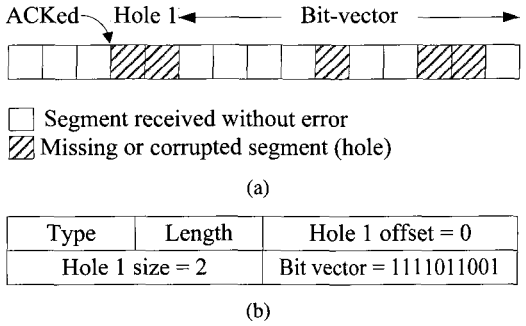

Fig. 2. SNACK example: (a) Receiver side buffer, (b) SNACK segment when bit-vector is used.

IV. THE SNACK SCHEME AND PROPOSED ERROR RECOVERY PROCEDURE

A. SNACK Basis

The SNACK [20], [21] scheme is a transport layer acknowledgment scheme which integrates the respective capabilities of SACK and negative acknowledgment (NAK). SNACK is similar to the solution standardized as the cumulative ACK with bitmap in IEEE 802.16 [22] (similar to the selective repeat ARQ with partial bitmap (SRPB) in HIPERLAN/2 [23]). The basic retransmission mechanism of SNACK is kept in TCP ACK, but packet loss information is constructed with bitmaps, so that the overhead of acknowledgments can be reduced.

As in the SACK scheme, under SNACK, the receiver informs the sender of the segments which it has not received. Unlike SACK, however, SNACK is capable of specifying a large number of holes in a bit-efficient manner. Therefore, the receiver can provide the sender with a complete image of the receiver buffer within a single ACK simply by specifying a list of the segments which are missing. Hence, the sender will not inadvertently retransmit segments which have already been successfully cached in the receiver buffer. As in SACK, SNACK uses the option field in the TCP header to convey the SNACK information. Fig. 1(b) illustrates the structure of a SNACK option. The *hole 1 offset* field specifies the starting location, while the size of the first hole is indicated in the *hole 1 length* field. Both the offset and the length values are expressed in terms of the number of maximum segment size (MSS) units. The optional variable-length *bit-vector* reports zero or more additional holes, which are also expressed in terms of MSS-sized blocks. The *bit-vector* maps the sequence space of the receiver buffer beyond the end of the block specified by the first hole. Each zero in the *bit-vector* signifies missing data in the corresponding MSS-sized block of the receiver buffer.

As described above, the receiver uses the *bit-vector* field to inform the sender of the MSS-sized blocks which have been received and those which have not. Fig. 2(a) illustrates a hypothetical out-of-sequence queue formed at the receiver buffer. In this figure, *ACKed* is the last consecutive ACK up to which all of the data has been correctly received.

When out-of-order segments arrive at the TCP receiver, it scans its received buffer, invokes SNACK options, and then sends these options out on the outgoing ACK segment. Importantly, as shown in Fig. 2(b), a SNACK option only occupies a small number of bytes in the TCP header when specifying the *bit-vector*. This is particularly advantageous when the out-of-sequence queue is large or when large windows are operated over long delay paths.

B. Proposed Error Recovery Procedure

Upon receipt of a SNACK option from the receiver, the sender checks the *bit-vector* and immediately retransmits any segments necessary to fill the holes. In the proposed implementation, whenever the lost packets indicated by SNACK are retransmitted, the sender sets the expected recovery sequence number to the highest sequence number sent so far in order to keep track of the recovery segment in the error recovery phase. The error recovery phase (which includes fast retransmission and fast recovery) is triggered when the sender receives a SNACK and terminates when it receives acknowledgment that all of the transmitted data has been received. The sender waits for triple duplicate ACKs before it reconfigures the congestion window, slow start threshold, and then enters the fast recovery phase.

During the fast recovery phase, the sender increases its congestion window size by one segment every time it receives a duplicate ACK. Thus, the TCP sender can transmit a new packet to fill the pipe from the source to the destination if it is possible to do so under the new window size. The sender checks for any retransmission losses by examining the *bit-vector* of the SNACK option. If the SNACK option indicates that the packet beyond the expected recovery sequence number has been cached in the receiver buffer, the sender retransmits the first hole indicated in the SNACK option and updates the retransmission timer since the previously retransmitted packet may have been lost again.

The receiver acknowledges a correctly received segment whether or not it arrives in the correct sequence. Out-of-order packets are buffered until the missing or corrupted segments are received. By inspecting the SNACK option, the sender can identify any holes in the receiver buffer and can therefore verify whether or not the retransmitted segments have been successfully received. For example, in Fig. 3(a), by examining the ACKed sequence number and the history information provided by the SNACK option, the sender can perceive that the retransmitted segment 1 has not yet been received and must therefore be retransmitted immediately.

If the returned SNACK for the previous retransmitted packet

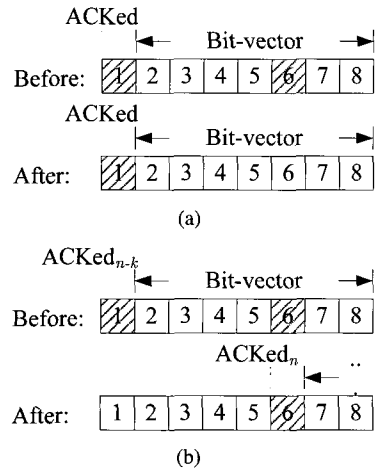

Fig. 3. NACK bit-vector example: (a) Retransmitted segment 6 received but segment 1 not received, (b) retransmitted segment 1 received but segment 6 not received.

acknowledges a new ACK sequence number (as illustrated in Fig. 3(b)), this ACK is deemed to be a partial ACK because not all of the transmitted packets have been acknowledged. In this event, the sender remains in the fast recovery phase until all of the outstanding data packets up to the expected recovery sequence number have been on ACKed.

The proposed modified error recovery procedure described above alleviates the multiple-packet-loss problem. Hence, when retransmitted packets are lost repeatedly as a result of transmission errors, SNACK has a higher success rate in retransmitting the lost packets without retransmission timeouts occurring. The ability of this modified error recovery procedure to respond robustly to errors is of particular importance in wireless environments, which are characterized by a relatively high corruption probability of the packets and their ACKs.

C. Enhancing the SNACK Startup Procedure

Burst random losses in wireless links usually result in a small window size, and hence only relatively few data packets can be delivered per round-trip. Under such conditions, the TCP source will spend excessive time in attaining an acceptable bandwidth usage, and consequently the link utilization will be degraded. As discussed in [24], [25], this induces a considerable performance cost, particularly for high-speed networks with long delay paths.

Since channel errors on wireless links usually result in a reduction of the TCP window size, and a corresponding reduction in the transmission rate, a more appropriate growth strategy must be considered for wireless environments. Accordingly, this study develops an enhanced SNACK scheme (with the proposed error recovery procedure implementation), referred to as SNACK-S, to accelerate the growth rate of the TCP window in the slow start phase or following the fast recovery phase in order to improve the TCP performance.

Let W denote the congestion window size, t denote the measured minimal round-trip delay time, and W_t be the slow start threshold. Without loss of generality, the expected transmission rate, λ, at the TCP source can be related to the window size by

$$\lambda = \frac{W}{t}. \tag{1}$$

Furthermore, let μ denote the transmission rate of the bottleneck on the path from the source to the destination (unknown to the transport protocol). If the queue gradually accumulates at the bottleneck, the transmission rate of the TCP source is given by

$$\lambda = \mu + \nu \tag{2}$$

where ν denotes the excess transmission rate in the measured RTT.

During a round-trip interval, the correlation between the packet in-flight along the connection path and the window size during a measured round-trip period is given by

$$W = \mu t + (b + q) \tag{3}$$

where b denotes the observed buffer at the end of the previous RTT (i.e., the number of packets waiting for service in the buffer of bottleneck), and q denotes the additional queue built-up at the bottleneck contributed by this connection during this round-trip period. The observed buffer occupancy [26] can be given by: $b = (RTT - t)\,W/RTT$, where W/RTT can be deemed as the "actual" transmission rate of the TCP source.

In order to quickly utilize the available bandwidth at startup phase without causing buffer overflow at the bottleneck, it is necessary to estimate the bandwidth at the bottleneck. By combining (1), (2), and (3), the "expected bottleneck bandwidt" at the end of each RTT period can be given by

$$\mu = \lambda - \nu = \lambda - (b + q)\frac{1}{t}. \tag{4}$$

As described in [27], to estimate the available bandwidth correctly, the TCP source must observe its own link utilization for a time longer than that actually required to transmit the entire packet cluster. Hence, in the startup phase, the estimated bandwidth per RTT interval is given by

$$\begin{aligned} \sigma &= \rho\,\mu + (1 - \rho)\,\hat{\mu} \\ &= \rho\left(\lambda - (b + q)\frac{1}{t}\right) + (1 - \rho)\,\hat{\mu} \end{aligned} \tag{5}$$

where $\hat{\mu} = r/\delta$, $\delta = s/r$, $s = W/t_W$, and $r = (W' - 1)/t_{AW'}$. The parameter, $\rho = t_W/RTT$, indicates the ratio of the sender's transmission time t_W (i.e., the time required to transmit W packets of congestion window size) to its measured round-trip delay RTT (as illustrated in Fig. 4). Thus, the value of ρ varies, depending on the bandwidth utilization of the sender in the previous round-trip. The instantaneous source transmission rate relative to the ACK packet arrival rate is given by $\delta = s/r$. Furthermore, the "measured bandwidth", r, is obtained by dividing the number of acknowledged packets by the sum of ACK inter-arrival times. To prevent the estimated bandwidth increasing too fast to overflow the buffer at the bottleneck particularly in slow start phase, the measured bandwidth r is divided by δ to decouple temporary burst in next round.

In (5), the estimated bandwidth varies in accordance with the bandwidth utilization of the sender in the previous round-trip

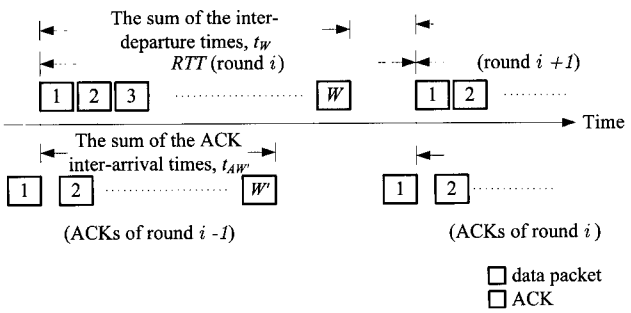

Fig. 4. A sender view of packets sent and received during a RTT period.

(indicated by the parameter ρ). At the beginning of the startup procedure, less information is gathered from the network (by incoming ACKs), thus the estimated bandwidth is dominated by the measured bandwidth. However, when the sender bandwidth utilization achieves higher, the expected bandwidth dominates.

The sender then estimates the available bandwidth on the basis of the aggregated information generated along the connection path during each RTT in the startup phase. Based on the estimated bandwidth, the size of the threshold window during the slow start phase is given by

$$W_t = \sigma t. \tag{6}$$

When a packet loss event occurs, the long-interval estimated bandwidth, φ, is given by

$$\varphi = \frac{L}{T} \tag{7}$$

where T is the period between two packet loss events and L are the lengths of the packets sent during this extended period. Based on the estimated bandwidth, when triple duplicate ACKs are received, the values of W_t and W are updated as follows:

$$W_t = W/2 \tag{8}$$

$$W = \max\{W_t, \varphi t\}. \tag{9}$$

This approach avoids the blind-halving of the TCP transmission rate following packet loss and improves the ability of TCP to maintain its self-clocking mechanism during the fast recovery phase. If $W_t < \varphi t$, the setting of W_t is too conservative and unused bandwidth is available along the path. The window size is therefore adjusted in accordance with the estimated bandwidth to increase the transmission rate to fully exploit the unused bandwidth.

The bandwidth estimation scheme is used to adjust the TCP congestion window size and the slow start threshold adaptively during the startup phase, which is either the initial slow start phase or the startup period following the fast recovery phase. By equipping the SNACK sender with the estimation mechanism provided in SNACK-S, the need for per-flow support in the routers is avoided and the available bandwidth in large bandwidth-delay networks can be fully exploited.

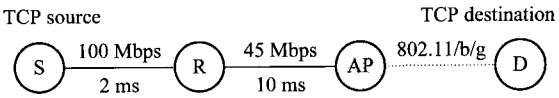

Fig. 5. Simple simulation model.

V. PERFORMACE EVALUATION

Analytical models for 802.11 MAC operations tend neither to be sufficiently general nor sufficiently detailed to support the evaluation and comparison of such 802.11 wireless networks. Therefore, this study employs a simulation technique to establish the crucial performance characteristics. The numerical results presented in this study are obtained using the ns-2 network simulator [28]. In the simulations, the ns-2 802.11 MAC layer module is extended to generate an error probability for each received frame and the frame may be dropped if the error module determines that a transmission error has occurred. Since errors are generated in both directions of the wireless channel, TCP ACKs may also be dropped.

The sections below examine certain parameters inherent in the 802.11 MAC layer which may affect the TCP performance. The discussions commence with the simple DCF infrastructure-based wireless network shown in Fig. 5, in which the link between AP and D is an 802.11 wireless link with a link capacity of 2, 11, or 54 Mbps.

A. Window Scaling

The TCP header uses a 16-bit field to report the receiver window size. Without the window scaling option (described in RFC 1323 [29]), the standard maximum TCP window size is 64 Kbytes and the TCP throughput is limited by the following expression:

$$Throughput = \frac{window\ size}{RTT}. \tag{10}$$

Consider the case of a TCP connection over an 802.11g wireless link with a 40 ms round-trip delay time. The maximum throughput achieved by the connection is limited at 64K bytes/40 ms $\approx$12.8 Mbps, much lower than the 54 Mbps link bandwidth. Hence, based on the RFC 1323, the ns-2 TCP agent is modified in the simulation to allow arbitrarily large window sizes.

B. Error Model

In general, wireless connections suffer higher error rates and are more complicated than wired connections because the bit signals propagating through the propagation medium are exposed to many more factors which potentially influence their signal quality than signals propagating in an electrical conductor or fiber. As a result, it is difficult to generalize the results of the wired case to the wireless domain.

It has been observed empirically that the loss characteristics of a wireless channel are bursty as a result of various fading effects [30], [31]. Therefore, using a uniform error model to evaluate wireless network protocols is unlikely to produce accurate results. To investigate the bursty effect of wireless transmission errors on the TCP performance, this study employs the *Gilbert-Elliot* error model [32] to characterize fading in the communi-

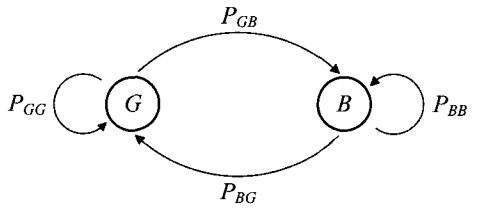

Fig. 6. Two-state Gilbert-Elliott error model.

cation channel of 802.11 wireless links. Fig. 6 presents the state diagram for a two-state Markov model of the *Gilbert-Elliott* channel. In the good state, G, losses occur with a low probability, P_G, whereas in the bad state, B, the channel operates in a fading condition and the loss probability, P_B, is higher. The steady state probabilities of being in states G and B are given by $\pi_G = P_{BG}/(P_{GB} + P_{BG})$ and $\pi_B = P_{GB}/(P_{GB} + P_{BG})$, respectively. The average packet loss rate produced by the *GE* channel is $P = P_G\pi_G + P_B\pi_B$.

C. Analytical Upper Bound of Wireless TCP Performance

Let m denote the data length at the application layer. Without loss of generality, a rough upper bound of the throughput for a TCP connection over an 802.11 one-hop wireless link (i.e., only a sender-receiver couple is active) under the basic access scheme is given by

$$Th_{noRTS/CTS} = m / (2(\theta + DIFS + SIFS + T_{ACK}) + T_{DATA_1} + T_{DATA_2})) \quad (11)$$

where T_{DATA1} is the time required to transmit a MAC frame carrying the TCP data packet, T_{DATA2} is the time required to transmit a MAC frame carrying the TCP layer ACK, and T_{ACK} is the time required to transmit a MAC layer ACK frame and a physical layer header, PHY_{hdr}. Finally, θ is the assumed average backoff time.

If the frame exchange commences under the RTS/CTS access method, the overheads associated with the transmission of the RTS and CTS frames must be added to the denominator of (11). Hence, the approximate upper bound of the TCP throughput under RTS/CTS, $Th_{RTS/CTS}$, is given by

$$Th_{RTS/CTS} = m / (2(\theta + DIFS + 3 \cdot SIFS + T_{RTS} + T_{CTS} + T_{ACK}) + T_{DATA_1} + T_{DATA_2}) \quad (12)$$

In Ethernet, the MTU size is 1500 bytes. Hence, the size of maximum data which can be transmitted from the source to the receiver (DATA_1) and the size of TCP ACK from the receiver to the source (DATA_2) can be calculated respectively as: DATA_1 = PHY_{hdr}(24) + MAC_{hdr}(24) + MAC_Payload_{TCP}(1500)+ FCS (4) = 1552 bytes, DATA_2 = PHY_{hdr}(24) + MAC_{hdr}(24) + MAC_Payload_{TCP}(60) + FCS (4) = 112 bytes.

Table 1 summarizes the approximate upper bound of the TCP throughput over various 802.11 wireless links. In order to simplify the presentation of the proposed approach, the average backoff time is set to $(CW_{\min}/2) \cdot SLOT$. The results reveal that the RTS/CTS/ACK exchange adds a significant overhead to the end-to-end throughput, particularly when TCP runs over a high-speed wireless link.

Table 1. The analytical upper bound of TCP throughput over 802.11 networks.

	Physical bit rate	No RTS/CTS	RTS/CTS
802.11	2 Mbps	1.49 Mbps	1.36 Mbps
802.11b	11 Mbps	5.06 Mbps	3.88 Mbps
802.11g	54 Mbps	14.98 Mbps	7.86 Mbps

Table 2. TCP throughput over 802.11 WLAN (packet size = 1400 bytes).

	Physical bit rate	No RTS/CTS	RTS/CTS
802.11	2 Mbps	1.30 Mbps	1.18 Mbps
802.11b	11 Mbps	4.09 Mbps	2.77 Mbps
802.11g	54 Mbps	12.08 Mbps	6.92 Mbps

D. Effect of Channel Collisions and Transmission Errors

Bit-level error detection and correction schemes are generally located in the physical layer and are isolated from the transport level protocol. Hence, for the sake of simplicity, it is assumed here that the TCP connection is susceptible to a *frame error rate* (FER).

To illustrate the impact of channel collisions on the TCP performance, Table 2 shows the nominal bandwidth and the TCP goodput over a single 802.11b/g wireless link from simulations. In calculating these results, it is assumed that the TCP packets are delivered via unicast with a random *GE* error model in which P_G, P_B, P_{GG}, and P_{BB} are set to 0.001, 0.005, 0.96, and 0.94, respectively. Therefore, if a packet is deemed to be corrupted as a result of channel noise, the MAC layer will continue to retransmit the packet until the retransmission threshold is exhausted. The goodput results in Table 2 indicate the actual data delivered to the application. It is obvious that the high-speed link is more seriously affected by losses. This is because conventional TCP reduces its transmission rate drastically following each loss event and hence it takes far longer to reach the nominal transmission rate supported by the higher speed links.

In a wireless link, a packet may either be corrupted as a result of channel noise or because it exhausted the retransmission threshold and was therefore discarded by the MAC layer. In general, discarded packets continue to be retransmitted in the MAC layer until the retransmission threshold has been reached. In addition, with large windows, a TCP flow can suffer losses as a result of excessive MAC layer collisions.

Fig. 7 plots the variation of the goodput with the FER, P_B. As shown in this figure, packets transmitted on wireless channels are frequently subject to burst errors, which cause back-to-back losses. When the channel is in the bad state, most transmission attempts fail. Therefore, the MAC layer discards the frame, resulting in the loss of the corresponding packet in the TCP protocol. The effective throughput (goodput) of all TCP schemes suffers in a wireless environment subject to random losses. As shown in Fig. 7, TCP Tahoe and Reno have the poorest performance since they always perform a timeout retransmission if the packet loss cannot be recovered by fast retransmission. However, the fast recovery algorithms employed by NewReno and SACK reduce the probability of coarse-grain timeouts due

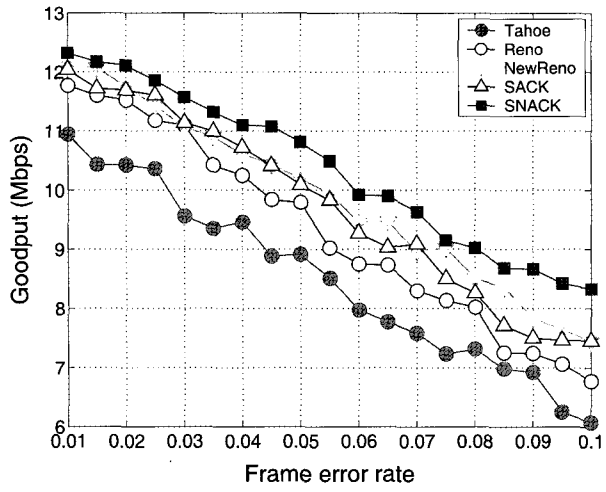

Fig. 7. TCP goodput with different FER.

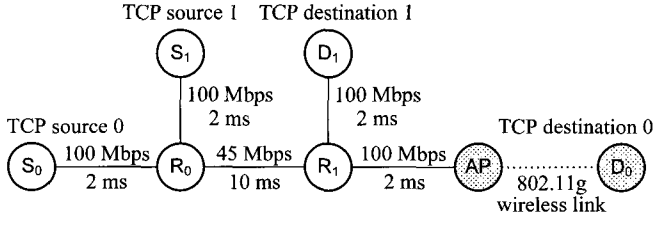

Fig. 8. Simulation model with different latencies.

to their improved response to multiple packet losses (NewReno remains in fast recovery until all of the data outstanding have been acknowledged), and hence the performances of these two TCP schemes are improved slightly relative to those of Tahoe and Reno.

In environments prone to significant losses, if a number of successive ACK packets are dropped in the network, the remaining ACK packets may be insufficient to trigger retransmission. Furthermore, when the sender transmits packets over a fading wireless link, the transmitted packets (particularly the retransmitted packets) are more likely to be corrupted. In this situation, the only way for the sender to detect a dropped packet is to wait until a timeout occurs. Therefore, TCP may perform less well under SACK than under NewReno in a wireless environment. Although SACK allows the receiver to inform the sender of the segments which have been received, the SNACK option provides more detailed information than SACK. Therefore, in SNACK, the sender can retransmit multiple lost packets, and hence the occurrence of retransmission timeouts is reduced. As a result, SNACK outperforms the other TCP flavors even in the case of a higher MAC-layer drop probability.

E. Performance Comparison of Wired and Wireless Connections

The following paragraphs examine the TCP performance for the case where the connections have different receiving conditions. As shown in Fig. 8, the TCP connection (TCP_0) between S_0 and D_0 crosses a lossy wireless link and therefore has a longer RTT than the TCP connection (TCP_1) between S_1 and D_1. In the simulations, the buffers between R_0 and R_1, and between R_1 and AP, are both set to 32K bytes.

Table 3. Interaction of TCP Reno with other TCP flavors (no link-level error).

TCP_0/TCP_1	Goodput (Mbps)
Reno/Reno	9.66/24.50
SACK/Reno	13.88/22.30
SNACK/Reno	14.42/22.31

Table 4. Interaction of TCP Reno with other TCP flavors (with link-level error).

TCP_0/TCP_1	Goodput (Mbps)
Reno/Reno	7.08/27.66
SACK/Reno	8.45/26.71
SNACK/Reno	9.03/26.15

E.1 Link Emulator - No Link-level Error

The first set of simulations investigates the interactions of two TCP connections, one of which crosses a long latency wireless link with no link level transmission corruption. In this set of simulations, the first flow (TCP_0) applies different TCP flavors, while the second flow (TCP_1) uses Reno. The present study compares all the TCP flavors with Reno because Reno is currently widely deployed. As shown in Table 3, TCP_0 has a lower goodput value. This is because a longer RTT slows the arrival rate of ACKs at the sender, and hence limits the rate at which data can be sent.

Table 3 shows that SACK and SNACK both increase the total throughput because they allow the receiver to identify and request the immediate retransmission of corrupted packets. Both SACK and SNACK can efficiently avoid timeouts due to their ability to immediately retransmit lost packets. The results show that SNACK obtains a better performance than SACK because it has an improved response to packet loss.

E.2 Link Emulator - Link-level Error

In the following simulations, packet errors are introduced into the wireless link using the two-state Markov error model. The packet corruption rates, P_G and P_B, are set to 0.0005 and 0.001, respectively, and P_{GG}, P_{GB}, P_{BB}, and P_{BG} are set to 0.9, 0.1, 0.85, and 0.15, respectively.

Table 4 shows that the packet error rate has a significant influence on the end-to-end performance. The results show a goodput increases in the shorter RTT connection (i.e., TCP_1) because the shorter RTT connection is more aggressive. The longer RTT connection (i.e., TCP_0) achieves lower goodput because burst random losses (caused by link-level errors at the wireless link) generally lead to a reduction in the window size and thus dramatically reduce the performance of TCP connections over wireless networks (comparing to the respective ones in Table 3).

Figs. 9, 10, and 11 show the corresponding window dynamics. In this case, Reno (TCP_0) suffers multiple coarse-grain timeouts due to the inability to react to multiple packet loss. Multi-

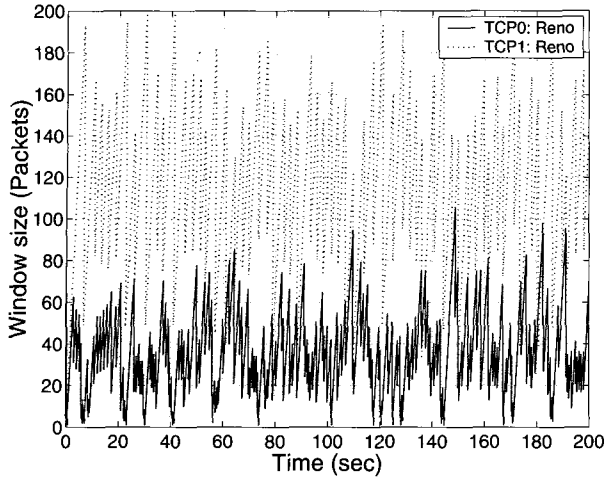

Fig. 9. Congestion window dynamic variation.

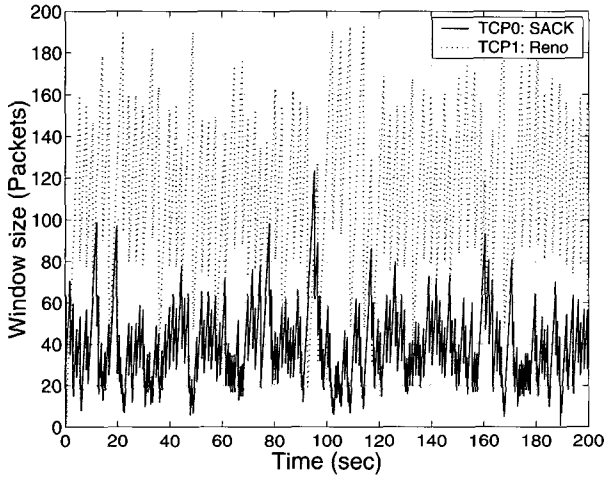

Fig. 10. Congestion window dynamic variation.

ple losses within the same window may cause TCP to resort to timeout-driven retransmissions. This results in severe window oscillations and leads to a significant performance degradation. Of the various schemes presented in Figs. 10 and 11, it is apparent that SNACK provides the best results, i.e., its window size is approximately 20 packets larger than that of SACK. The superior performance arises because SNACK allows TCP to alleviate the impact of wireless transmission errors and therefore the number of unnecessary window reductions is substantially reduced.

E.3 Comparison between SNACK and SNACK-S

It is well known that the performance of TCP flows is affected by the RTT. Fig. 12 shows the TCP goodput as a function of the round-trip latency for TCP Reno, Reno-S (Reno with the proposed bandwidth estimation mechanism), SNACK, and SNACK-S, respectively. In this case, P_G, P_B, P_{GG}, and P_{BB} are set to 0.001, 0.005, 0.96, and 0.94, respectively. It can be seen that TCP becomes increasingly less efficient as the latency increases because the TCP sources reduce their transmission rates as the ACK packets start to return more slowly. As shown

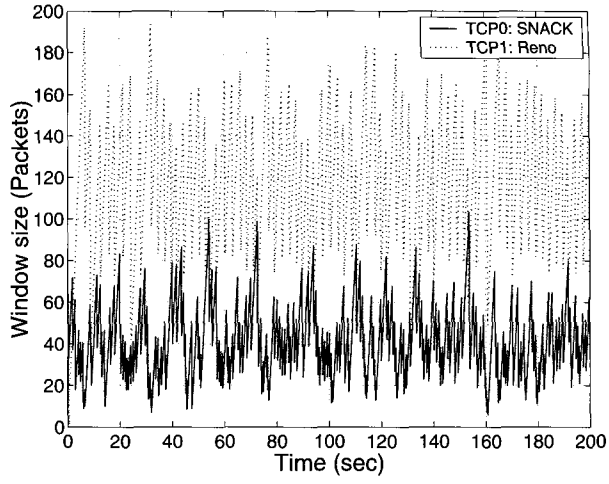

Fig. 11. Congestion window dynamic variation.

in Fig. 12, SNACK-S increases the TCP goodput to a level higher than that of conventional TCP due to the efficiency of its startup procedure. SNACK-S is better suited for long delay, high BER channels, and is therefore a more appropriate choice than SNACK for wireless environments.

Fig. 13 shows the variation of the SNACK-S goodput with the FER, P_B. It is observed that the throughput achieved by SNACK-S is greater than that of the other TCP connections. The reason for this is that the proposed bandwidth estimation algorithm starts each cycle by filling the bit pipe to its capacity, whereas conventional TCP simply applies the default threshold value and blindly halves the window size in the event of congestion.

From the discussions above, it is clear that SNACK provides an effective scheme for quickly retransmitting lost packets, thereby eliminating timeout and long idle time periods. Compared with TCP Reno and SACK, SNACK significantly improves the TCP end-to-end performance over wireless environments. The enhanced scheme, SNACK-S, further improves the TCP end-to-end performance based on the proposed bandwidth estimation mechanism and allows TCP to achieve an acceptable level of bandwidth utilization. Since the applied SNACK-based error recovery procedure and the bandwidth estimation mechanism operate at the TCP level, it provides a viable incremental deployment for enhancing TCP performance without MAC layer involvement.

VI. CONCLUSION

This study has applied an end-to-end scheme based on SNACK to improve the performance of TCP over wireless channels. The SNACK scheme recovers from packet loss events in an effective manner and is ideally suited for long delay, high error probability wireless links. Therefore, it is a suitable candidate for wireless environments.

Burst random losses in a wireless link usually result in a small window size, and hence the number of data packets which can be delivered per round-trip is limited. This study has developed an enhanced SNACK scheme, designated SNACK-S, to enhance

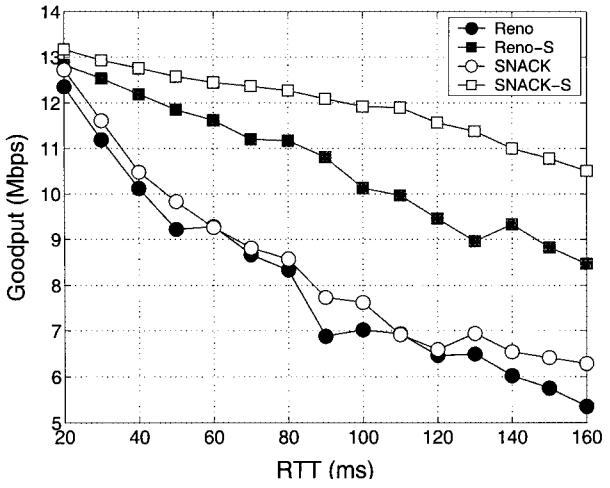

Fig. 12. TCP goodput with different RTT.

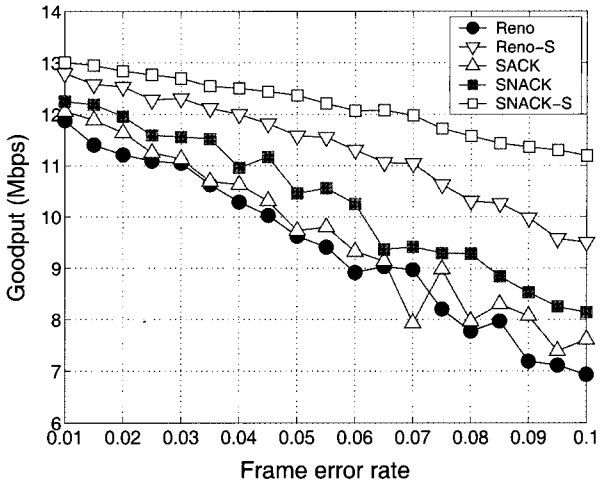

Fig. 13. TCP goodput with different FER.

the TCP performance in wireless links. Equipping the SNACK sender with a bandwidth estimation mechanism, SNACK-S has no need for per-flow support in the routers and can scale to the available bandwidth in large bandwidth-delay networks. Hence, SNACK-S greatly enhances the performance of the TCP over 802.11 infrastructure-based wireless networks.

REFERENCES

[1] J. Postel, "Transmission control protocol," RFC 793, Sept. 1981.

[2] V. Paxson, M. Allman, and W. Stevens, "TCP congestion control," RFC 2581, Apr. 1999.

[3] H. Balakrishnan *et al.*, "A comparison of mechanisms for improving TCP performance over wireless links," *IEEE/ACM Trans. Networking*, vol. 5, no. 6, pp. 756–769, 1997.

[4] A. V. Bakre and B. R. Badrinath, "Implementation and performance evaluation of indirect TCP," *IEEE Trans. Computers*, vol. 46, no. 3, pp. 260–278, 1997.

[5] H. Wu *et al.*, "Performance of reliable transport protocol over IEEE 802.11 wireless LAN: Analysis and enhancement," in *Proc. IEEE INFOCOM2002*, 2002, pp. 599–607.

[6] G. Huston, "TCP in a wireless world," *IEEE Internet Computing*, vol. 5, no. 2, pp. 82–84, 2001.

[7] A. A. Hanbali, E. Altman, and P. Nain, "A survey of TCP over ad hoc networks," *IEEE Commun. Surveys & Tutorials*, vol. 3, pp. 22–36, 2005.

[8] C. Liu and R. Jain, "Approaches of wireless TCP enhancement and a new proposal based on congestion coherence," in *Proc. 36th HICSS2003*, Jan. 2003.

[9] S. Floyd, "TCP and explicit congestion notification," *ACM Computer Commun. Review*, vol. 24, no. 5, pp. 10–23, 1994.

[10] IEEE Standard 802.11, "Wireless LAN medium access control (MAC) and physical layer (PHY) specifications," Aug. 1999.

[11] P. Karn, "MACA: A new channel access method for packet radio," in *Proc. ARRL/CRRL Amatuer Radio 9th Computer Networking Conf.*, 1990.

[12] J. Kim *et al.*, "IEEE 802.11 wireless local area networks," *IEEE Commun.*, vol. 35, no. 9, pp. 116–126, 1997.

[13] S. Sharma, "Analysis of 802.11b MAC: A QoS, fairness, and performance perspective," *CoRR cs.NI/0411017*, 2004.

[14] M. Hassan and R. Jain, *High Performance TCP/IP networking: Concepts, Issues, and Solutions*. Prentice-Hall, 2003, pp. 161–163.

[15] S. Floyd and T. Henderson, "The newReno modification to TCP's fast recovery algorithm," RFC 2582, Apr. 1999.

[16] M. Mathis *et al.*, "TCP selective acknowledgement options," RFC 2018, Apr. 1996.

[17] H. Balakrishnan *et al.*, "Improving TCP/IP performance over wireless networks," in *Proc. 1st ACM MOBICOM'95*, Nov. 1995.

[18] H. Balakrishnan and R. H. Katz, "Explicit loss notification and wireless web performance," in *Proc. IEEE GLOBECOM'98*, (Sydney, Australia), Nov. 1998.

[19] S. Floyd, Jan. 1996, Issues of TCP with SACK, Tech. Rep., [Online] Available: http://www-nrg.ee.lbl.gov/floyd/

[20] R. C. Durst, G. J. Miller, and E. J. Travis, "TCP extensions for space communications," in *Proc. ACM MOBICOM'96*, Nov. 1996, pp. 15–26.

[21] SCPS Transport Protocol (SCPS-TP), [Online] Available: http://www.scps.org/scps/

[22] IEEE Standard 802.16-2004, "Part 16: Air interface for fixed broadband wireless access system," Oct. 2004.

[23] H. Li *et al.*, "Automatic repeat request (ARQ) mechanism in HIPERLAN/2," in *Proc. IEEE VTC*, (Tokyo, Japan), 2000, pp. 2093–2097.

[24] R. S. Cheng *et al.*, "Improving the ramping up behavior of TCP slow start," in *Proc. 19th AINA2005*, pp. 807–812, Mar. 2005.

[25] R. S Cheng and H. T. Lin, "TCP selective negative acknowledgment over IEEE 802.11 wireless networks," in *Proc. ICNS2006*, (Silicon Valley, USA), July 2006.

[26] J. R. Chen and Y. C. Chen, "Vegas plus: Improving the service fairness," *IEEE Commun. Lett.*, vol. 4, no. 5, pp. 176–178, 2000.

[27] A. Capone, L. Fratta, and F. Martignon, "Bandwidth estimation schemes for TCP over wireless networks," *IEEE Trans. Mobile Computing*, vol. 3, no. 2, pp. 129–143, 2004.

[28] Network Simulator, NS-2, [Online] Available: http://www.isi.edu/nsnam/ns/

[29] V. Jacobson, R. Braden, and D. Borman, "TCP extensions for high performance," RFC 1323, May 1992.

[30] D. A. Eckhardt and P. Steenkiste, "Measurement and analysis of the error characteristics of an in-building wireless network," in *Proc. ACM SIGCOMM'96*, (California, USA), Aug. 1996, pp. 243–254.

[31] G. T. Nguyen, R. Katz, and B. Noble, "A trace-based approach for modeling wireless channel behavior," in *Proc. Winter Simulation Conf.*, pp. 597–604, Dec. 1996.

[32] E. N. Gilbert, "Capacity of a burst-noise channel," *Bell Syst. Tech. J.*, vol. 39, pp. 1253–1266, 1960.

Rung-Shiang Cheng received the B.S. degree from the Department of Electronic Engineering, National Chin Yi Institute of Technology, Taiwan, and M.S. degree from the Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan, where he is currently working for his Ph.D. degree in the Department of Electrical Engineering, National Cheng Kung University. His current research interests include network performance analysis, wireless communication, and mesh networks.

Hui-Tang Lin received B.S. degree in Control Engineering from National Chiao-Tung University, Taiwan, in 1989, the M.S. and the Ph.D. degrees both in Electrical Engineering from Michigan State University, East Lansing, MI, in 1992 and 1998, respectively. He is currently an assistant professor at the Electrical Engineering Department and the Computer and Communication Institute of National Cheng-Kung University, Taiwan. Prior to joining the EE department of National Cheng-Kung University, he was a member of Technical Staff (MTS) at Agere Systems (former Lucent Microelectronics), where he worked on the design and implementation of high-speed networking ICs. Before he followed the company's spin-off decision to Agere Systems, he was a MTS at Advanced Technologies division of Bell Laboratories in Lucent Technologies, where he worked on an industry-leading silicon solution for ATM and IP switching and port processing. His research interests include QoS of highspeed networks, optical networks, switch architecture, wireless networks, sensor networks, and network security.