ABSTRACT

An abstract of the thesis of Harkirat Singh for the Master of Science in Computer Science presented October 23, 2002.

Title: Analysis of Energy Consumption of TCP Reno, Newreno, and SACK in multi-hop wireless network

In this thesis we compare the energy consumption behavior of three versions of TCP -- Reno, Newreno, and SACK. The experiments were performed on a wireless test-bed where we measured the energy consumed at the sender node. Our results indicate that, in most cases, using total energy consumed as the metric, SACK outperforms Newreno and Reno while Newreno performs better than Reno. The experiments emulated a large set of network conditions including variable round trip times, random loss, bursty loss, and packet reordering. We also estimated the idealized energy for each of the three implementations (i.e., we subtracted out the energy consumed when the sender was idle) and here, surprisingly, we found that in many instances SACK performed poorly compared to the other two implementations. We conclude that if the mobile device has a very low idle power consumption then SACK is not the best implementation to use for bursty or random loss. On the other hand, if the idle power consumption is significant, then SACK is the best choice since it has the lowest overall energy consumption.

Dedication

ੴ ਸਤਿ ਨਾਮੁ ਕਰਤਾ ਪੁਰਖੁ ਨਿਰਭਉ ਨਿਰਵੈਰੁ
ਅਕਾਲ ਮੂਰਤਿ ਅਜੂਨੀ ਸੈਭੰ ਗੁਰ ਪ੍ਰਸਾਦਿ ॥

ਵਿਦਿਆ ਵੀਚਾਰੀ ਤਾਂ ਪਰਉਪਕਾਰੀ ॥

(*Contemplate and reflect upon knowledge,*

and you will become a benefactor to others)

Acknowledgements

I sincerely thank my advisor Dr. Suresh Singh for his continuous support and guidance throughout the course of my thesis and graduate work. It has benefited me immensely in terms of both intellectual and personal growth.

I would like to thank my committee members, Dr. Suresh Singh, Professor Jim Binkley, and Dr. Douglas V. Hall for their invaluable time and feedback. I am thankful to Professor Jim Binkley for his support and guidance with FreeBSD.

I owe a special debt of gratitude to my parents and family. I thank my family members for their constant support and encouragement. It is because of their wishes and love from across the ocean that I have been able to get this far in my life. I thank my wife, Ruby, for her selfless support and love that makes me want to excel. Thanks to all of my friends who have made my stay in Portland a memorable one.

ANALYSIS OF ENERGY CONSUMPTION OF TCP RENO, NEWRENO, AND SACK IN MULTI-HOP WIRELESS NETWORK

by

HARKIRAT SINGH

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

MASTER OF SCIENCE
in
COMPUTER SCIENCE

Portland State University
2002

Table of Contents

List of Tables

List of Figures

1. Introduction

The demand for pervasive computing will accelerate in the near future. Wireless devices equipped with IEEE 802.11b are gaining popularity as the platform of choice for deploying a range of mobile applications. In comparison to Moore's Law the rate of growth of battery capacity is much slower. Studies have shown that the energy capacity has doubled roughly every 30 years [Pow95]. Since these devices operate on battery power alone, it is important to ensure that energy-efficient considerations are incorporated into the design of their hardware and software. Data communication plays a key role in many mobile applications and accounts for a large proportion of the cost in running these applications on handhelds. Users need reliable communication for various applications like e-mail, file transfer, and web browsing and TCP is the dominant reliable transport protocol on top of which all these services run. It is important to understand and characterize the energy cost of wireless communication and use this information in the design of the communications component of these devices.

In this work we focus our attention on the energy efficiency of three variants of TCP for connections running over wireless links. Our goal is to characterize the energy consumption as well as the throughput of three versions of TCP (Reno, Newreno, and SACK) for a variety of wireless network conditions including loss (random as well as bursty), variable round trip times (RTT), and packet reordering. We also investigate the total energy consumption as well as Idealized energy consumption of TCP variants. We used a test-bed of three-hop

wireless network, and each laptop was equipped with Lucent 802.11b 11Mbps DSSS silver card running FreeBSD 4.3. SACK was developed based on RFC 2018. We used HP 34401A multimeter for measurement of current at the sender. We observed the interesting results; SACK consumes the lowest total energy in most of scenarios and has the highest troughput. However, if we discount the energy consumed by the Sender in an idle state (i.e., when the Sender is awaiting ACKs prior to transmitting more packets), SACK appears to have the highest energy cost in many cases. This is due to the fact that SACK introduces additional computational complexity at the Sender, thus resulting in higher energy consumption. This difference is interesting in mobile computation because it points to the need for a careful selection of protocol for the handhelds. If the idle energy for a handheld is very small, then SACK is probably not a good choice for that device. On the other hand, if the handheld has a high idle energy cost, then SACK is a good choice since it completes the data transmission the earliest.

The remainder of thesis is organized as follows. In Chapter 2, we describe Transport Control Protocol (TCP). In Chapter 3, we define the energy model used to compare the energy cost. In Chapter 4 we discuss the current state-of-the-art in energy and other performance studies of energy studies. Chapter 5 describes our experimental hardware and software setup and we discuss the experiment parameters used. The results are presented in Chapter 6, and we discuss the implication of this work in Chapter 7.

2. Review of Transmission Control Protocol

Transmission Control Protocol (TCP) provides a connection oriented reliable byte stream on top of the unreliable datagram service provided by the IP layer [RFC793]. Unlike UDP [RFC768], TCP provides reliability by sending data combined with positive acknowledgements (ACKs) and retransmissions. Each byte in the transmission data stream is numbered, starting at some value and increasing over a period of time. When a packet is formed at the Sender, this packet contains not only the data segment from application layer but also the sequence number of the first byte in the data segment, the (implied) length of the segment, a 16-bit TCP checksum, sender and destination port numbers, and TCP options if any. TCP at the Sender will then initiate a retransmission timer when the packet is sent out. The Receiver will respond with the ACK if the segment arrives in sequence and without scrambled, otherwise the Receiver will respond to the Sender by acknowledging the last received correct packet that was in sequence.

All of the current TCP implementations are based on TCP Tahoe which incorporated algorithms for slow-start, congestion avoidance, fast retransmit, and modifications to the formula for estimating round-trip times (RTT) (see [Ste94] for more details). TCP Reno, Newreno, and SACK are essentially similar to Tahoe but with a modified fast retransmit algorithm that includes fast recovery as well. A brief explanation of the mechanism of TCP Reno, Newreno and SACK follows.

2.1 TCP Reno

TCP Reno [Jac88] is widely deployed in the Internet. Once the three-way handshake between the Sender and Receiver is over the TCP Sender transmits packets to the Receiver, and the Receiver acknowledges these packet with ACKs. TCP is a connection-oriented protocol, and a state must be maintained at the Sender and the Receiver. Some of the state variables of TCP Control Block are:

- snd_nxt: the sequence number of the next data byte to send
- snd_una: the sequence number of the oldest unacknowledged data byte
- cwnd: the congestion window size
- ssthresh: the slow start threshold (max value is 65535, without option)
- mss: maximum segment size
- th_ack: ACK from receiver

TCP Reno uses slow start and congestion avoidance algorithm for effectively throttling the network.

2.2 Slow Start

This algorithm [Jac88] is used at the beginning of a new connection and after a retransmission.

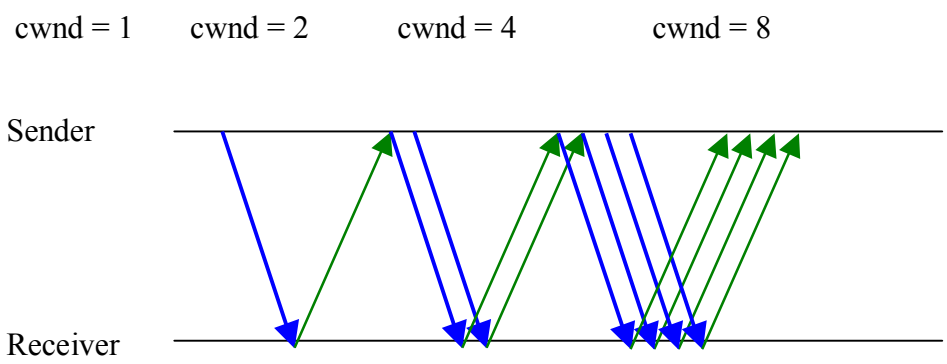

Figure 1: TCP Slow Start

The cwnd is initialized to one segment and each time the Sender receives an ACK it increases cwnd by one segment. This algorithm provides an exponential increase in the growth of cwnd, when cwnd reaches ssthresh slow start algorithm stops and the congestion avoidance operates.

2.3 Congestion Avoidance

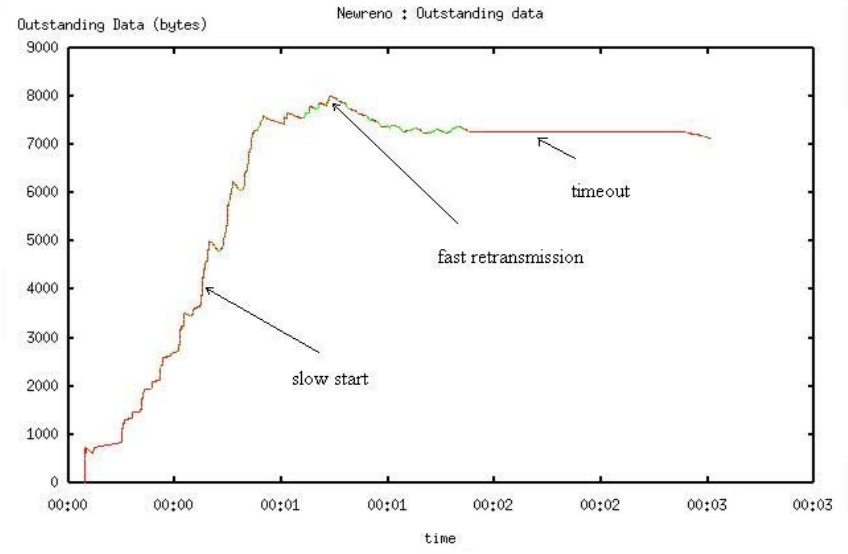

Figure 2: Evolution of Congestion Window

Congestion Avoidance throttles the network by increasing cwnd more conservatively each time a new ACK arrives.

$$cwnd = cwnd + \frac{1}{cwnd}$$

2.4 Fast Retransmit and Fast Recovery

TCP Sender detects loss by tripple duplicate ACKs or timeout. When a TCP Receiver misses a packet (due to reordering or loss) on a connection but receives several packets that are later in sequence, it generates an ACK for the missing packet.

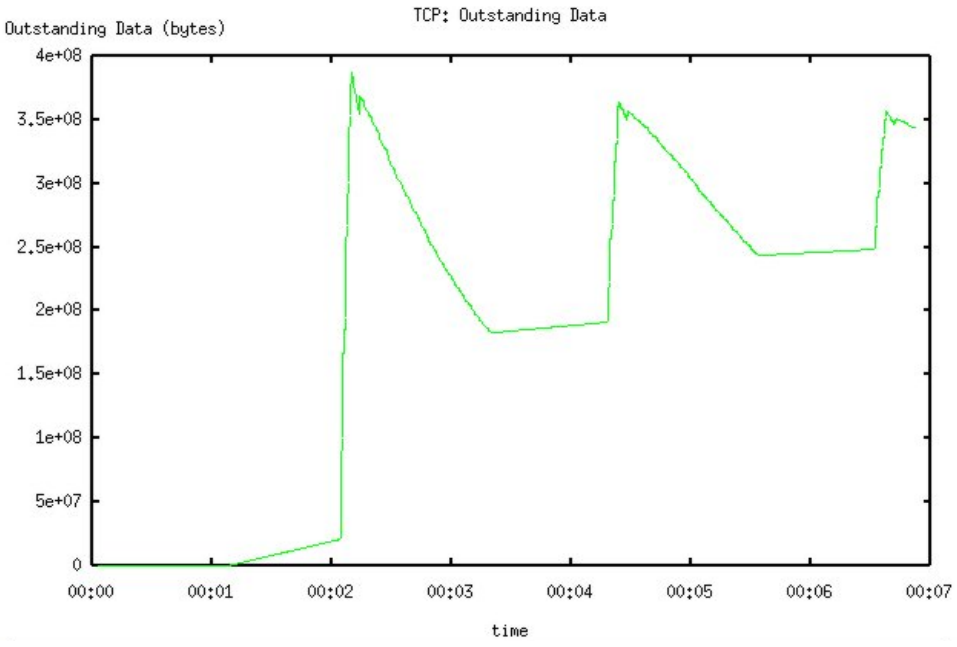

Figure 3: Evolution of Congestion Window

When the Sender sees more than a threshold (three) number of duplicate ACKs, it infers that the packet is lost. The Sender performs the following action:

- ssthresh = min (cwnd, receiver advertised window) / 2
- snd_nxt = th_ack, cwnd = 1
- Retransmit the missing segment and cwnd = ssthresh + 3

Each time another duplicate ACK arrives, the Sender increments cwnd by the segment size and transmit a packet (if allowed by the new value of cwnd).

When the next ACK acknowledges all the intermediate segments sent between the lost segment and the receipt of the third duplicate ACK, cwnd is set to ssthresh (step 1), which is termed "deflating" the window. Figure 3 shows three different events of fast-retransmit and fast-recovery, and one of the events is at 2.00 sec, when after receipt of three duplicate ACKs TCP Sender reduces its cwnd to half.

2.5 Retransmission Timeout

TCP maintains an estimate of the round trip time (RTT), i.e., the time it takes for the segment to travel from the Sender to the Receiver plus the time it takes for the ACK (and/ or any data) to travel from the Receiver to the Sender. The variable RTO (Retransmit Time Out) maintains the value of the time to wait for an ACK after sending a segment before timing out and retransmitting the segment. If the RTO estimate is much lower than the actual round-trip time of the connection,

segments will be retransmitted unnecessarily before the actual segment or its corresponding ACK has propagated through the network. If the round-trip estimation is too high, time-outs will be longer than necessary thus the actual throughput will be low because even if a segment gets lost, the Sender will not retransmit until the timer goes off.

The RTO is set by taking into account both the mean round-trip time (RTT) between the Sender and the Receiver, and the variation in it. In most modern implementations of TCP, RTO = mean RTT + 4 * mean deviation in RTT.

Whenever a TCP Sender has outstanding data that has not yet been acknowledged, it sets a timer known as the retransmission timer. If there is unacknowledged data at the time the timer expires, then the oldest packet among these is assumed to have been lost. After timeout the connection enters into another slow start phase.

2.6 TCP New Reno

When multiple packets have been dropped from a single window of data the Fast Retransmit and Fast Recovery algorithms do not recover fast [FF96, Hoe96]. In this case, if the SACK option is available, the TCP Sender has the information to make intelligent decisions about which packets to retransmit and which packets not to retransmit during Fast Recovery.

In the case of multiple packets dropped from a single window of data, the first new information available to the Sender comes when the Sender receives an acknowledgement for the retransmitted packet (that is the packet retransmitted

when Fast Retransmit was first entered). If there had been a single packet drop, then the acknowledgement for this packet will acknowledge all of the packets transmitted before Fast Retransmit was entered (in the absence of reordering). However, when there were multiple packet drops, then the acknowledgement for the retransmitted packet will acknowledge some but not all of the packets transmitted before the Fast Retransmit. We call this packet a partial acknowledgment [Hoe96].

Newreno [FH99] modifies the fast retransmit and fast recovery algorithm such that the Sender does not wait for a retransmission timeout and continues to retransmit lost segments every time it receives a partial ACK. The details of the Newreno fast retransmit and recovery algorithm are as follows:

1. When the third duplicate ACK is received and the Sender is not already in the Fast Recovery procedure, set ssthresh to no more than the value given by the equation below. (This is equation 3 from [RFC 2581]).

$$ssthresh = \max\left(\frac{\min\left(cwnd, \textit{receiver advertised window}\right)}{2}, \quad 2 * MSS\right)$$

Record the highest sequence number transmitted in the variable "recover".

2. Retransmit the lost segment and set cwnd to ssthresh plus 3*MSS. This artificially "inflates" the congestion window by the number of segments (three) that have left the network and which the Receiver has buffered.

3. For each additional duplicate ACK received, increment cwnd by one. This artificially inflates the congestion window in order to reflect the additional segment that has left the network.

4. Transmit a segment, if allowed by the new value of cwnd and the Receiver's advertised window.

5. When an ACK arrives that acknowledges new data, this ACK could be the acknowledgment elicited by the retransmission from step 2, or elicited by a later retransmission.

6. If this ACK acknowledges all of the data up to and including "recover", then the ACK acknowledges all the intermediate segments sent between the original transmission of the lost segment and the receipt of the third duplicate ACK. Set cwnd to ssthresh, where ssthresh is the value set in step 1 (this is termed "deflating" the window), and exit fast recover algorithm.

2.7 TCP Sack

Multiple packet losses from a window of data can have a catastrophic effect on TCP throughput [FF96, Hoe96]. TCP [RFC793] uses a cumulative acknowledgment scheme in which received segments that are not at the left edge of the receive window are not acknowledged. This forces the Sender to either wait a round-trip time to find out about each lost packet, or to unnecessarily retransmit segments that have been correctly received [FF96]. With the cumulative acknowledgment scheme, multiple dropped segments generally cause TCP to lose its ACK-based clock, reducing overall throughput.

Selective Acknowledgment (SACK) [RFC2018] is a strategy that corrects aforesaid behavior in the face of multiple dropped segments. With selective acknowledgments, the data Receiver can inform the Sender about all segments that have arrived successfully, so that the Sender needs to retransmit only the segments that have actually been lost.

2.7.1 SACK Permitted Option

SACK option, which is known as SACK Permitted Option, needs to be negotiated during connection establishment phase. This two bytes option must not be sent on a non-SYN segment. The first byte of this option contains the kind which is four, and the second byte the length in bytes which is two.

The SACK option is to be sent by the data Receiver to the Sender to inform the Sender of the non-contiguous blocks of data received and help in case retransmission. When missing segments are received, the data Receiver acknowledges the data normally by advancing the left window edge in the Acknowledgement Number Field of the TCP header. The SACK option does not change the meaning of the Acknowledgement Number field.

The first byte of this option contains the kind of the option which is five. The second byte contains the length of the option which is variable according to the number of blocks reported. Each contiguous block of data queued at the data receiver is defined in the SACK option by two 32-bit unsigned integers in network byte order.

- Left Edge of Block: This is the first sequence number of this block.
- Right Edge of Block: This is the sequence number immediately following the last sequence number of this block.

The maximum number of SACK blocks reported by receiver is limited by the length of the option field which is 40 bytes. Hence maximum number of blocks is 4 (8 * n + 2 bytes). It is expected that the SACK option generally be used in conjunction with other options (as Timestamps which is two bytes long) that limits the number of SACK blocks to three.

2.7.2 Data Receiver behavior

If the Receiver has received a SACK permitted option on the SYN segment for this connection it should generate a SACK option on each ACK that does not acknowledge the highest sequence number in the Receiver's queue. If not, the Receiver must not generate a SACK option in any case. Thus every duplicate ACK should bear a SACK option.

The first SACK block must specify the contiguous block of data containing the segment which triggered this ACK, unless that ACK advances the acknowledgement field in the header. This assures that the ACK with the SACK option reflects the most recent change in the Receiver's queue.

The data Receiver should include as many distinct blocks as possible in the SACK option. Note that the maximum available option space may not be sufficient to report all blocks present in the Receiver's queue.

The SACK option SHOULD be filled out by repeating the most recently reported SACK blocks (based on first SACK blocks in previous SACK options) that are not subsets of a SACK block already included in the SACK option being constructed. This assures that in normal operation, any segment remaining part of a non-contiguous block of data held by the data Receiver is reported in at least three successive SACK options, even for large-window TCP implementations. After the first SACK block, the following SACK blocks in the SACK option may be listed in arbitrary order. However, SACK blocks must not report any old data that is no longer actually held by the Receiver if it has reneged.

2.7.3 Data Sender behavior

When receiving an ACK containing a SACK option, the data Sender should record the selective acknowledgement for the future reference. The data Sender is assumed to have a retransmission queue that contains the segments that have been transmitted but not yet acknowledged, refereed as holes. On receipt of SACK blocks TCP Sender will update the hole information at the data Receiver.

2.7.4 Congestion control and retransmission strategy

We describe here the behavior of TCP-SACK during fast retransmit and fast recovery. This is explained based on Sally Floyd's algorithm using pipe variable which is an estimation of the amount of data in the network.

1. When the third duplicate acknowledgement is received, set ssthresh to one-half of the current window. Record the highest sequence number transmitted in the variable "recover".

2. Retransmit the missing segment. Initialize pipe to cwnd minus three the segment size and set cwnd to ssthresh

3. Each time another duplicate ACK is received decrement pipe by one segment size. If pipe is less than cwnd retransmit a segment that has not been SACKed and increment pipe by one segment size. If all non-SACKed segments have already been retransmitted, send new data.

If a partial ACK (an ACK for new data but not as much as for the “recover”) is received decrement pipe by two segment size and perform retransmission as stated above. When the “recover” ACK is received TCP returns to normal behavior and performs congestion avoidance.

It is important to note that the pipe variable is decremented by one segment size when a duplicate ACK is received because it means that a segment left the network. The pipe variable is decremented by two segments size when a partial ACK is received because it means that the segment that triggered this ACK has left the network and that the next expected segment by receiver has been lost (which means two segments out of the network).

2.8 D-SACK

There has been a extension to the SACK known as D-SACK [RFC 2883]. The use of D-SACK does not require separate negotiation between a TCP Sender

and Receiver that has already negotiated SACK capability. When D-SACK is used, the first block of the SACK option should be a D-SACK block specifying the sequence number of the duplicate segment that triggers the acknowledgement.

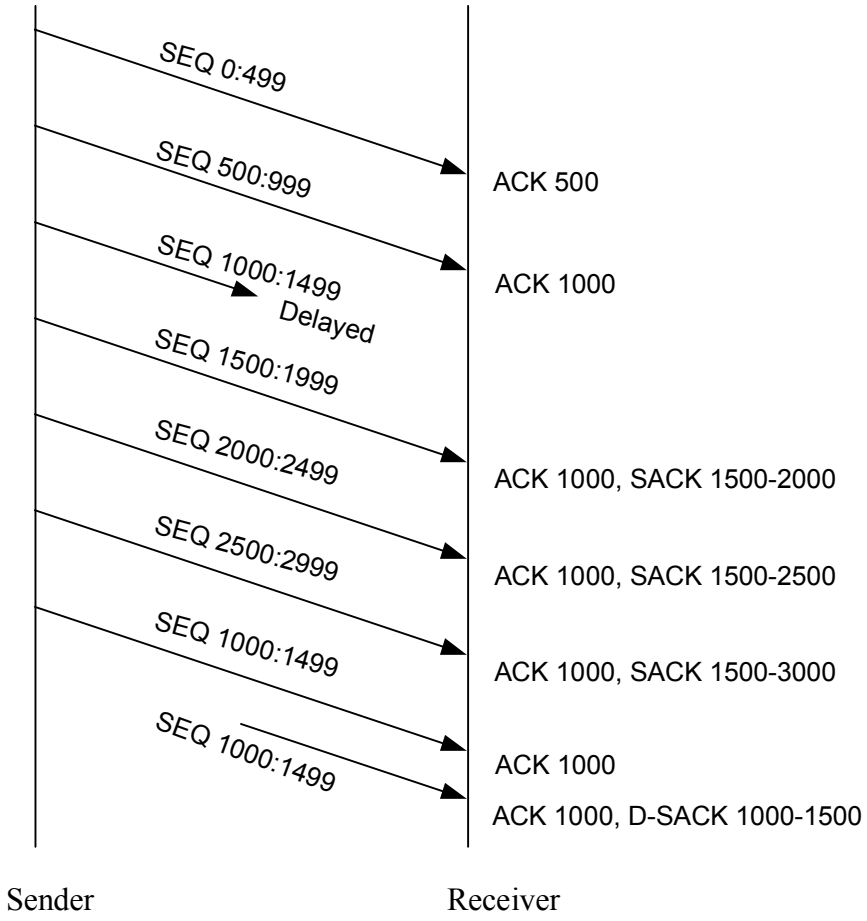

Figure 4: D-SACK

If the packets are reordered in the network such that a segment arrives more than 3 packets out of order, TCP's fast retransmit algorithm will retransmit the out-of-order packet. Without the use of D-SACK, the sender will unnecessary perform

false fast retransmission and fast recovery. However, the use of D-SACK allows the Sender to detect some cases when a a Fast Retransmit was due to packet reordering instead of a packet loss.

Without the use of D-SACK, the Sender would only know that the first transmission of the segment 1000-1499 was delayed in the network, or that either one of the data segments or the final ACK was duplicated in the network. Thus the use of D-SACK allows the Sender to more reliably infer that the first transmission of the segment 1000-1499 was not dropped. This allows TCP sender to “undoing” the reduction in the congestion window.

3. Ad Hoc Network and Energy Model

In this section we will explain the ad hoc network and its key characteristics. We also investigate the performance of TCP under these characteristics. Afterwards, we explain Energy model for TCP.

3.1 Wireless Network

The 802.11 standard [Spec99] define two modes of operation of wireless network: infrastructure (BSS) and ad hoc network (IBSS). In infrastructure mode (Figure 5), the wireless network consists of at least one access point and a set of mobile nodes equipped with wireless network interface (NI) card.

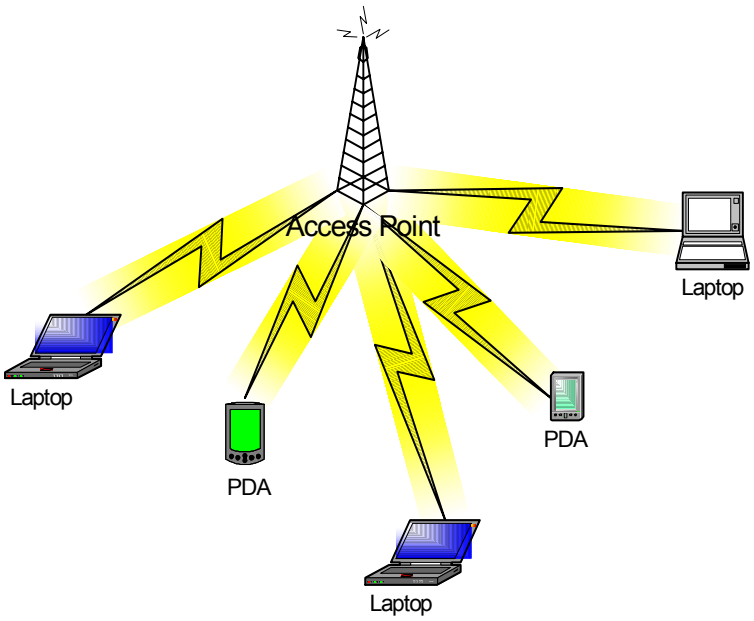

Figure 5: Infrastructure Mode

This configuration is called a Basic Service Set (BSS). An Extended Service Set (ESS) is a set of two or more BSSs forming a single sub-network. Since most corporate WLANs require access to the wired LAN for services (file servers, printers, Internet links) they will operate in infrastructure mode.

Ad hoc mode (also called peer-to-peer mode or an Independent Basic Service Set, or IBSS) is simply a set of 802.11 mobile nodes that communicate directly with one another without using an access point or any connection to a wired network (Figure 6).

This mode is useful for quickly and easily setting up a wireless network where a wireless infrastructure does not exist or is not required for services, such as a conference room, convention center, or airport, or where access to the wired network is barred such as disaster relief or battlefield.

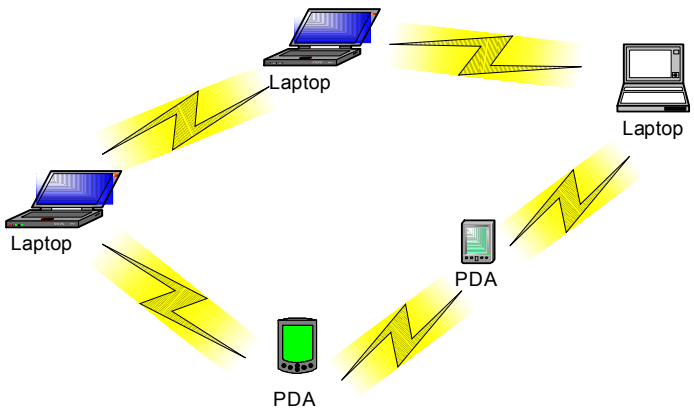

Figure 6: Ad hoc Mode

We will focus our discussion in the context of ad hoc (IBSS) network as this network is more flexible in terms of set up, mobility etc.. Further, it has different power requirement as compare to BSS [FN01].

The throughput of TCP connection is very low in ad hoc network due to the fact that mobile nodes communicate over relatively low bandwidth links which are prone to high propagation loss, burst error, fading, multi-user interference and frequent topological change. The three primary key factors for this are:

3.1.1 Routing Failure

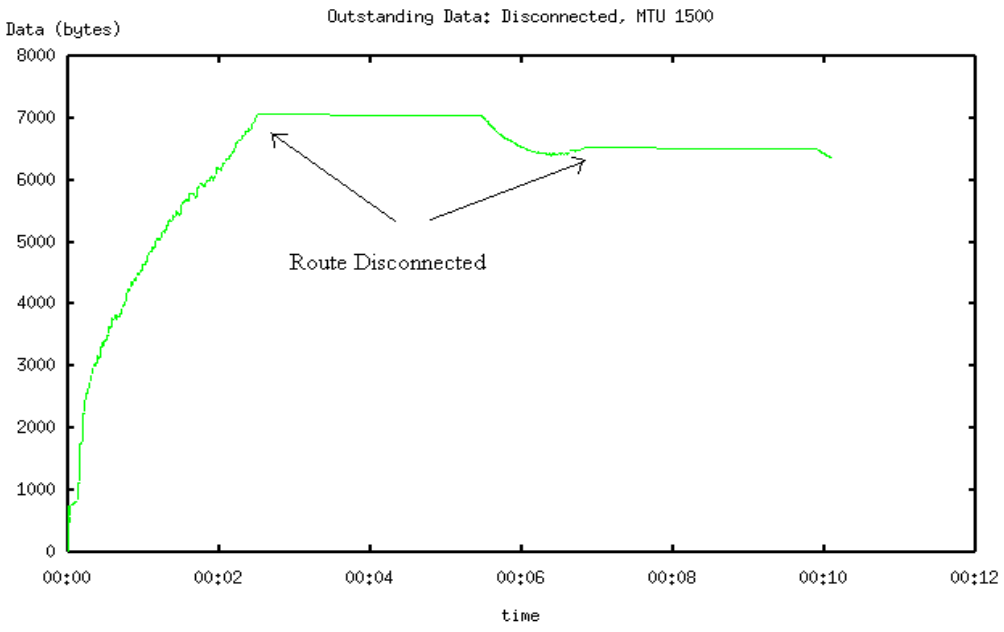

Figure 7: Evolution of CWND in the event of Route Disconnection

A related problem pointed out in [HV99] is that the underlying routing protocol (DSR JM96] in this case) may find invalid routes due to the cache reply mechanism incorporated into the protocol.

This causes further delays in finding a route and can lead to serial timeouts at the Sender, which can become very long. Finally, it is possible that even if a route is found, the TCP Sender may not send data because it is waiting for a timeout to retransmit data. By the time the Sender times out, the route may no longer be valid! Thus the TCP throughput will be very small.

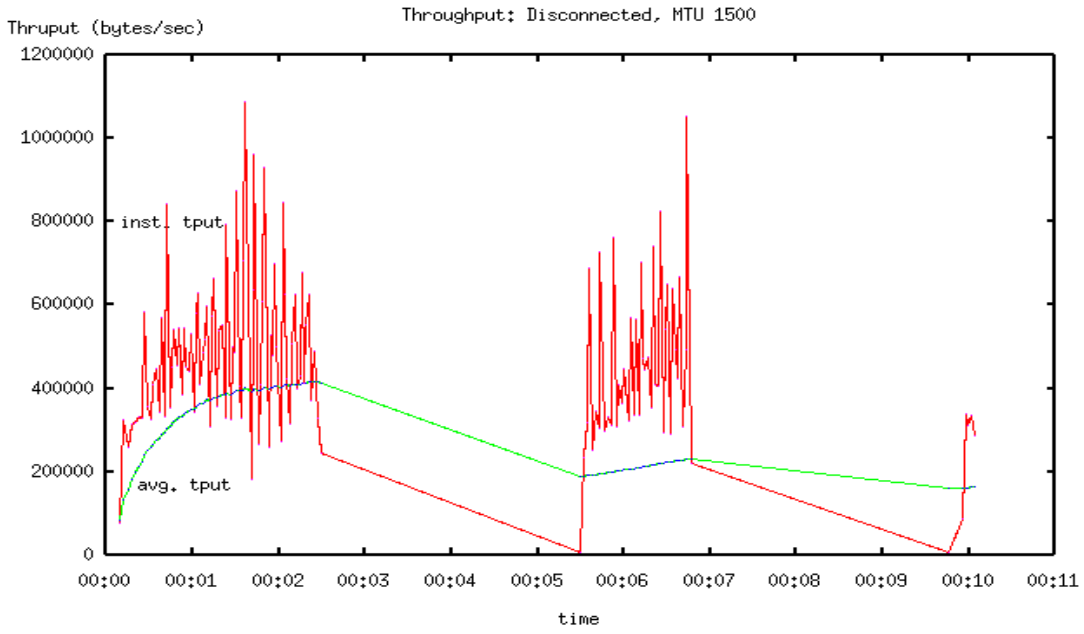

Figure 8: Throughput in the event of Route Disconnection

In Figure 7 route failure occurs at time 2.00 and 5.00 sec, the TCP Sender remains idle for some periods of time, many retransmissions take place when there is no route, the Sender will timeout and invoke slow start and congestion window is

small. The throughput of the connection degrades, the corresponding time periods in Figure 8 have throughput almost zero.

3.1.2 Out-of-order packets

Due to route changes during the lifetime of a TCP connection or due to multi-path routing, it is possible for packets and/ or ACKs to arrive out-of-order [BPS99]. This can cause the Sender to receive triple duplicates which in turn results in the Sender retransmitting the offending packet and shrinking its congestion window by a half. Thus, the overall energy consumption increases due to reduced throughput as well as due to unnecessary retransmissions.

3.1.3 Routing Congestion

Routing failure can cause network congestion if there are several active connections [MSB00]. This is because each route failure forces the routing protocol to find new routes and as a consequence, the number of control packets in node buffers can increase significantly. Similarly, TCP Senders will timeout and retransmit packets that may already be present in the buffers of intermediate hops. These two factors taken together can result in congestion at one or more nodes in the network which in turn results in lowered TCP throughput and higher energy consumption.

TCP's design is not optimized for the three network conditions discussed above that are endemic to ad hoc networks. Therefore, in each case, TCP

misinterprets the network state resulting in poor throughput and excessive packet retransmissions.

Clearly standard[1] TCP implementation does not have mechanism to react to aforesaid ad hoc network characteristic which causes for unnecessary retransmission of packets, longer idle connection time and lower throughput.

3.2 Energy Consumption in TCP

Consider the case where a node in an ad hoc network needs to transmit *B* bytes of data reliably. It transmits a window of packets and waits to receive ACKs. Upon reception of ACKs, it moves its window and transmits more packets.

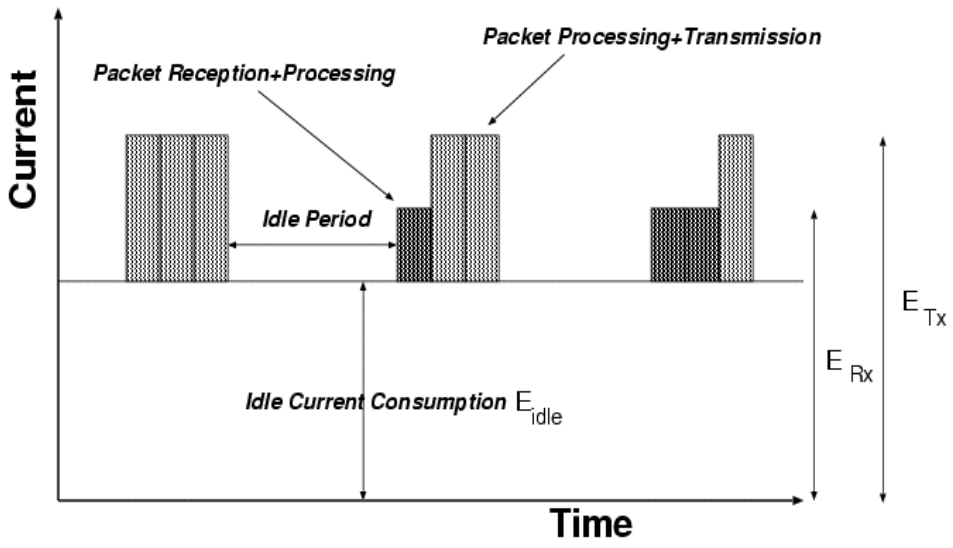

Figure 9: Total Energy Consumption

[1] We mean Reno, Newreno and SACK as standard TCP implementation

Figure 9 shows the evolution in time of a Sender where we plot the current drawn by the Sender (assuming a fixed voltage) as a function of time. When the Sender is idle, it draws a fixed amount of current[2]. When a packet is to be transmitted, there is some processing energy consumed[3] in addition to the transmission energy. Likewise, when a packet is received, there is energy consumed to receive the packet (by the interface card) plus the processing energy needed to process the received packet.

We make a distinction between the *total energy E* consumed and the *idealized energy* E_I. The total energy refers to the total system energy, from the start of data transmission to the end that is consumed by the system. The idealized energy refers to the total energy minus the energy consumed in the idle periods. The reason this distinction is interesting is that E_I depends on the protocol processing, transmission, and reception costs only, whereas E depends on E_I as well as the *throughput* of the connection. As we will show in Chapter 6, for some cases SACK has a lower E (than Reno and Newreno) but a higher E_I due to the additional computation involved. Finally, it is interesting to note that as the idle power consumption is minimized by improved power-management in hardware, E will asymptotically approach E_I.

---

[2] We are considering the idle energy used by the sender as a whole, i.e., the interface card, the processor, the memory and any other devices that are powered on.

[3] The main source of energy consumption is the copy operation (user space to kernel space and then to the interface card).

3.2.1 Energy and Throughput Relation

We can write a simple expression for the total energy consumed by a node to transmit B bytes of data reliably as follows:

$$E = P_{idle}\left(t_{total} - t_{Tx} - t_{Rx} \right) + P_{Tx} t_{Tx} + P_{Rx} t_{Rx}$$

Where P_{idle} is the idle power consumed by the Sender, t_{total} is the total time needed to complete the transmission of B bytes, t_{Tx} and t_{Rx} are the time spent in transmitting and receiving packets, and P_{Tx} and P_{Rx} are the corresponding power consumed at the Sender for packet transmission and reception.

The first term in the above expression denotes the *idle energy cost* at the Sender. This is the energy consumed by the Sender while it awaits reception of ACKs from the Receiver or timeout events. Thus, if the channel has a low bit rate, or if the losses are high, the Sender is likely to spend a large amount of time in the idle state consuming energy. The second term in the expression denotes the energy expended for packet transmission. For a given amount of data B, the value of this term will primarily depend on the number of transmissions and the MTU size (Maximum Transmission Unit or packet size) used. Larger MTU sizes will require fewer packet transmissions but a higher loss rate will result in a larger penalty since a larger amount of data will need to be retransmitted (recall that packet transmission cost is a sum of at least two memory copy operations, a checksum calculation, and, eventually, the actual transmission). Finally, the third term in the

expression for E is the packet reception cost. If we consider the case when the data transmission is one-way only (i.e., the Sender only receives ACKs), then the reception cost is actually quite small (ACKs are small packets and since they contain no data, there is a much smaller copy cost). We can therefore drop this third term from the expression for E to simplify it and obtain:

$$E = P_{idle} \left(t_{total} - t_{Tx} \right) + P_{Tx} t_{Tx}$$

Notice that we have also dropped t_{Rx} from the first term because the total time spent in receiving and processing ACKs is quite small.

If we assume that the *average* connection throughput is τ bytes/sec and the transmission speed is r bytes/sec we can write,

$$\begin{aligned} E &= P_{idle} \left(B/\tau - B/r \right) + P_{Tx} B/r \\ &= (B/\tau) P_{idle} + (B/r) (P_{Tx} - P_{idle}) \\ &\propto 1/\tau \end{aligned}$$

Thus, we see that the total energy consumed is *inversely* proportional to the average throughput achieved by the connection.

4. Related Work

Ad hoc networks are collections of mobile nodes (laptop, PDA, cell phone etc.) which communicate with each other in the absence of any networking infrastructure. These mobile nodes need to be light in weight. Furthermore, they are battery operated with finite energy resources, hence they need to be energy conserving so that battery life can be increased. Even though battery technology is improving continuously and hardware is improving rapidly in terms of power consumption, battery life and battery weight are issues that are the motivation for researchers to reduce overall power consumption. In this section we will summarize energy studies and power saving techniques proposed at different layers of the mobile node.

4.1 Energy Efficient Hardware

In the past several studies have showed that CPU, Memory, liquid crystal display (LCD) and wireless network interface (NI) are the predominant source of power consumption in a laptop [Lor95]. Therefore, it is important to reduce their energy consumption. Today's hardware is based on CMOS technology. There are two main components that establish the amount of power dissipated in a CMOS circuit [WE93]. These are:

- Static dissipation: Due to leakage current or other current drawn continuously from the power supply.

- Dynamic dissipation: Due to switching transient current and charging and discharging of load capacitances

4.1.1 Dynamic Power

Dynamic power dissipation is the largest component in the power consumption. It can be expressed as:

$$P_d = C_L * [V_{DD}]^2 * f_p \quad (1)$$

where P*d* is the power in watts, V_{DD} is supply voltage, C_L is load capacitance and f_p is repetition frequency. Dynamic power can be reduced by reducing one or all of these factors. However, they each have their side effects. Lower Voltage will reduce power (quadratic) at the expense of speed so time taken to complete a task will increase. Load capacitance can be reduced by using smaller transistor and routing capacitance.

It is possible to disable the clock to the portion of the hardware which is not active; this reduces unwanted transitions and therefore reduces Dynamic Power consumption. Specifically, a mobile node spends a small amount of its time sending and receiving traffic, so idle power consumption is a significant contribution to overall power consumption (refer to Table 1). These results are presented by [FN01].

	MEASURED	SPECIFICATION
Sleep Mode	10 mA	10 mA
Idle Mode	156 mA	n/a
Receive Mode	190 mA	180 mA
Transmit Mode	284 mA	280 mA
Power Supply	4.74 V	5 V

Table 1: Lucent IEEE 802.11 WaveLAN PC Card Characteristics

Wireless Network interface, which is a significant consumer of power [SK97], can be turned off in the idle state. However, we will see in later sections that this approach is not so straightforward.

We can rewrite the equation 1 in terms of time as:

$$P_d * t \quad = \quad C_L * [V_{DD}]^2 \qquad (2)$$

Power-delay product gives the energy consumption of a particular task; hence with lower clock frequency it will take a longer time to finish a task. The Compaq Itsy energy study [FBA+0] has similar findings for Pocket PCs. It recommends that for compute-intensive applications with no voltage switching, application should be run at as fast a clock frequency as possible until completed. The system then should be placed into a low-powered mode.

System architecture power can be reduced by low power displays [HW96], power efficient disk drives algorithms [DKM+94], CPU scheduling with voltage scaling [BB96; GCW95; Wei93], low power I/O devices, etc..

4.2 Physical Layer Energy Consumption

In recent years, the idea of wireless micro-sensor networks has garnered a great deal of attention by researchers. A distributed, ad-hoc wireless network consists of hundreds to several thousands of small sensor nodes scattered throughout an area of interest. Proposed applications for wireless micro-sensor networks are unique hence wireless micro-sensor systems will have different challenges and design constraints than existing wireless networks (e.g. cellular networks and wireless LANSs). Ambient conditions are time varying in wireless micro-sensor networks, thus the system should be able to adapt to these varying conditions. In addition to these challenges, the energy consumption of the underlying hardware is also of paramount importance. [SCI+01] proposes that it is possible to take advantage of hooks and knobs in the physical layer to build more energy-efficient protocols and algorithms. It recommends a physical layer driven approach to protocol and algorithm design for wireless sensor networks. If protocol designers treat the physical layer as a black box, then protocols can be detrimental to energy consumption.

4.3 MAC Layer Energy Optimization

We will first explain the mechanism of IEEE 802.11 afterwards we will explain energy saving techniques and Power aware MAC protocols.

The 802.11 standard [Spec99] define two modes of operation for a wireless network interface: infrastructure mode (BSS) and ad hoc mode. In BSS mode there

exists at least one Access Point (AP), which is responsible for moderating traffic between hosts. In Ad hoc mode each mobile node is in the transmission range of other nodes, they communicate directly with each other on peer-to-peer level.

The 802.11 Media Access Control (MAC) is very similar in concept to 802.3, in that it is designed to support multiple users on a shared medium by having the sender senses the medium before accessing it. The WLAN is half-duplex, i.e. it cannot listen while transmitting, and therefore, collision detection is not possible. To account for this problem 802.11 MAC layer standard [Spec99] has defined a basic method which is Carrier Sense Multiple Access / Collision Avoidance (CSMA/CA). There are two other methods which are DCF (Distributed Co-ordinate Function) RTS/CTS, and PCF (Point Co-ordinate Function).

4.3.1 CSMA /CA

A mobile node wishes to transfer senses the medium, and, if it is idle. If so, the mobile node waits an additional, randomly selected period of time and then transmits if the medium is still free. However, if it is busy, each mobile node may wait until transmission stops and enters into random *backoff* period. This prevents multiple nodes seizing the medium immediately after completion of preceding transmission. To ensures reliability each data packet is followed by an ACK.

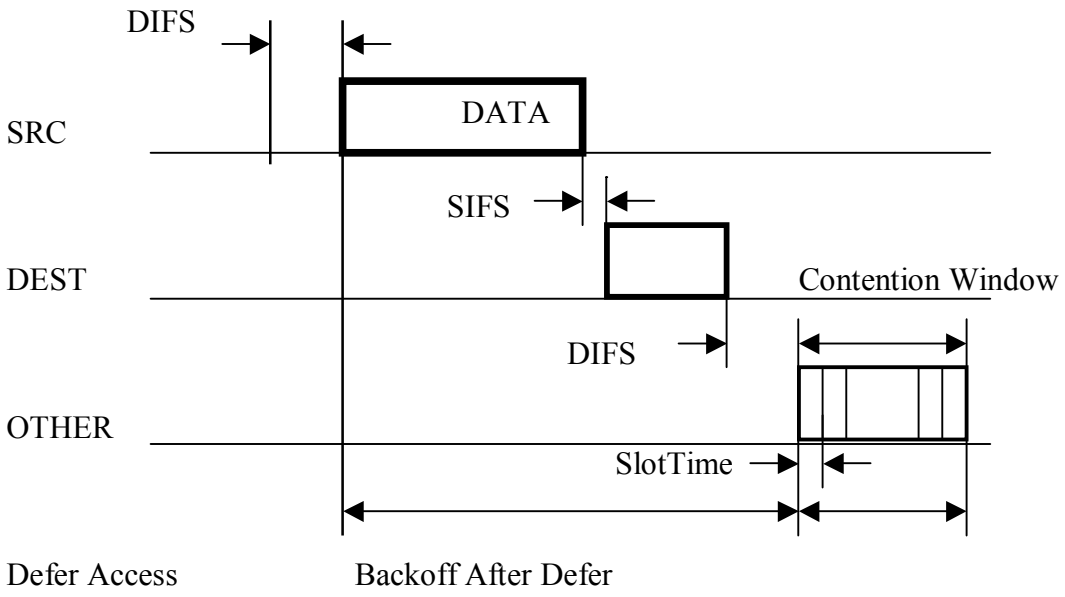

Figure 10: Carrier-Sense Multiple Access (CSMA)

The period between completion of packet transmission and start of ACK frame is one Short Inter Frame Space (SIFS).

Transmission other than ACKs must wait at least a DCF Inter Frame Space (DIFS) before transmitting data. If a transmitter sense a busy medium, it determines a random *backoff* period by setting an internal timer. After medium becomes idle, mobile node wishes to transmit waits a DIFS plus an integer multiple (min value of 0 and max of 255) of Slot Time. Upon expiration of DIFS, the timer begins to decrement. If the timer reaches zero, the node may begin transmission. However, if the channel is seized by other node before the timer reaches zero, the timer value is retained at decremented value for subsequent transmission this ensure fairness.

RTS/CTS is used to counter the problem of “hidden terminal”. Time-bounded data application such as video and voice are supported by PCF.

4.3.2 Power Saving Mode

A wireless network interface consumes significant amount of total power even when it is in an idle mode. The 802.11 standard describes a power saving mode enabling them to go into *awake* and *doze* mode, in former mode mobile node is continuously drawing power, however, in *doze* mode the radio is dozing. Transition from the *doze* state to the *awake* state results in additional energy consumption [HS00].

4.3.2.1 BSS Power Saving

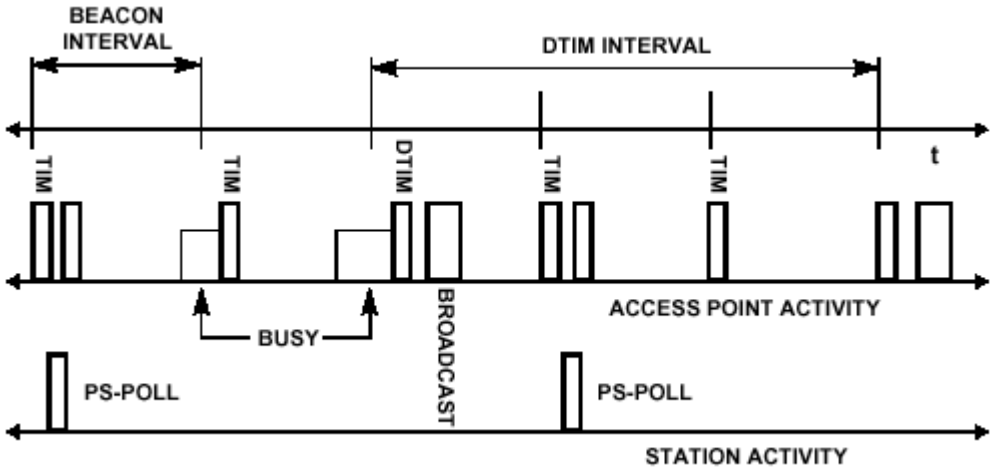

Figure 11: BSS Power management

Mobile nodes communicate thru Access Point (AP). Once every beacon period the AP sends a beacon (also used for time synchronization) indicating if AP has a data queued for a dozing node. This Traffic indication map (TIM) is broadcasted as part of timing synchronization beacons [ZP99]. Mobile node wakes up to listen to these beacons. If a node has a data queued as indicated by TIM it returns to *awake* mode and alerts AP by sending PS-Poll message that it is ready to receive data. Buffered broadcast / multicast message is indicated by DTIM to awaken all mobile nodes and alert them for forthcoming traffic. AP then transmits the queued message without PS-Poll message.

4.3.2.2 Ad-Hoc Power Saving

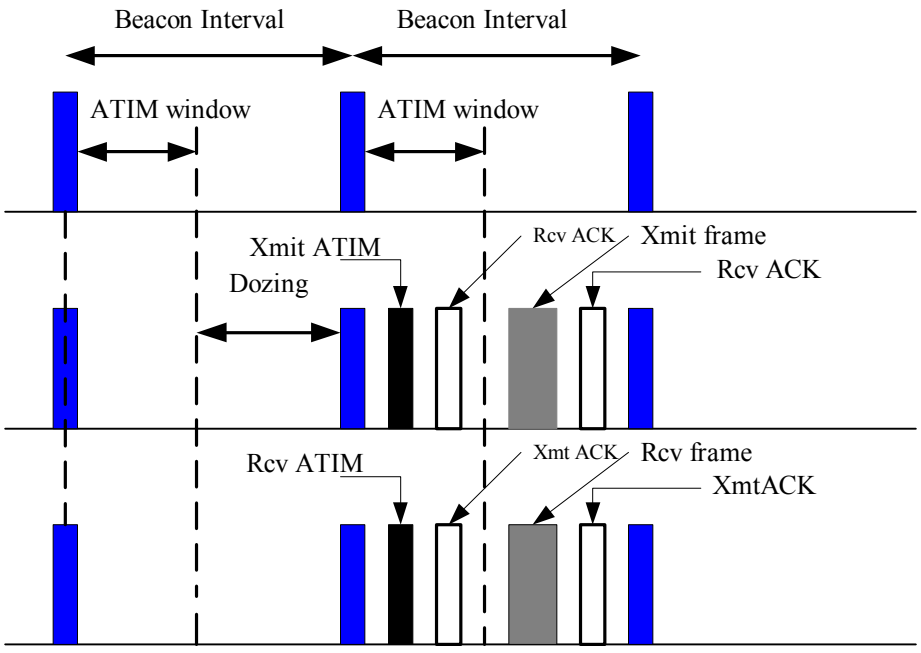

Figure 12: DCF Power management

The 802.11 ad hoc power saving mode uses periodic beacons to synchronize nodes in the network. Beacon packets contain timestamp that synchronizes node's clock. A beacon period starts with an ad hoc traffic indication message window (ATIM Window), and each node stays awake for an ATIM window interval. Transmission of ATIM frames is performed using the CSMA/CA or DCF mechanism [ZP99]. A node that receives an ATIM replies by ATIM-ACK, sender and receiver nodes remain awake during entire beacon interval.

The beacon period and ATIM window size considerably affects throughput and energy consumption [WESW98]. If the ATIM window size chosen to be too small, there may not be enough time to advertise all buffered packets. If ATIM window is too large, there may not be enough room for transmitting data as data transmission starts after the end of ATIM window.

4.4 Energy aware MAC Protocols

We can divide these protocols into three categories: first, which modifies Link layer to reduce effective number of packet transmission and better error control models, second, where new Power aware MAC protocols are developed for ad hoc networks and lastly which deals with enhancement of 802.11 power aware mechanism.

4.4.1 Link Layer

Automatic Repeat Request (ARQ), Forward error correction (FEC), and hybrid of these two also exist. Another possibility is to allow the error control

strategy as a function of the channel conditions. Wireless links experience widely varying channel conditions, therefore, it seems likely that any one of the schemes is not likely to be optimal in terms of energy efficiency at all the time.

[LSS99] shows that with ARQ alone, energy consumed is directly proportional to mean number of retransmission (MNoR), result shows that in good channel condition MNoR is negligible, whereas for poor channels, it reaches the maximum allowed value and flattens out there. In FEC alone, energy is proportional to computational cost of producing the redundancy. Adaptive frame size can have a dramatic impact on the behavior of the ARQ protocol. Large packet size will be optimal in good channel as this minimizes overhead. However, in a noisy channel, a smaller packet will be preferable despite the excess overhead, since the packet loss rate of the large packets will dominate.

[ZR97] found that Classis ARQ strategies lead to a considerable waste of energy, due to large number of transmissions. They propose to time packet transmissions (and retransmissions) in a manner that maximizes energy efficiency. In native ARQ modes, classic ARQ protocols, such as Go-Back-N or Selective Repeat, recover errors by retransmitting packets regardless of the state of the channel. Thus, in bad channel condition it may lead to depletion of the energy. Subsequently, when the channel becomes good again the depletion in the energy may make transmissions more prone to errors resulting in potentially more transmissions, leading to a catastrophic loss of energy.

The key idea is when channel conditions deteriorate (upon receipt of NACK or garbled feed-back), the transmitter enters a probing mode, and sends a short probing packet every so often. When channel conditions improve, the transmitter switches back to the normal mode and restarts transmission from the point it was interrupted. The potential gain of this protocol depends on how rapidly fading occurs relative to round trip delay time on the link. It was shown that accepting a moderate throughput reduction may make it possible to significantly reduce the energy consumption of the system, relative to classic ARQ protocols.

[HSB00] have developed a MAC protocol (E^2MaC) suitable for wireless multimedia traffic. Traffic over the wireless link is scheduled by the base station based on the QoS requirements of the connection and on the current channel conditions. The scheduler tries to avoid only non-time critical traffic during bad channel periods, thereby not affecting traffic with demanding QoS.

4.4.2 Ad Hoc Energy Efficient MAC Protocols

PAMAS [SR98] is designed for ad hoc protocols with energy conservation as the primary goal. It uses separate channel for data and signaling. A mobile node with a packet to transmit sends RTS and enters AwaitCTS state. If CTS does arrive it starts transmission over data channel and receiving node transmits *busy tone* over control channel to ensure that neighbors do not seize the channel for data transmission.

Nodes who are not participating in ongoing transmission can turn off their radio to ensure that power is not depleted in overhearing. A node on wakeup performs binary probe to determine the longest remaining transfer for which it can go back to sleep again before attempting to seize the channel.

SPAN, [CJBM01] an energy-efficient scheme, elects a group of "coordinators" which are changed periodically. The coordinators stay awake and forward traffic for active connections. Non–coordinators follow standard IEEE 802.11 power saving scheme. Nodes, typically coordinators, buffer the packets for *dozing* nodes and announces them during ATIM window. In SPAN network, packet routed via non-coordinate nodes are rare, therefore, it proposes new advertised traffic window, following standard ATIM window. During this advertised traffic window, the announced packets and the packets for the coordinator can be transmitted. After this window, only the packets for the coordinator can be transmitted and other nodes (non-coordinators) can go to *doze* state if they do not have traffic to send or receive.

DPSM [JV02] is motivated form the fact that fixed ATIM window leads to lower throughput and higher energy consumption in various scenarios [WESW98]. It proposes an adaptive scheme to dynamic choose a suitable ATIM window size.

4.5 Energy efficient Routing Protocols

[SWR98] proposes Power Aware routing where links are assigned weights as a function of energy consumed when transmitted over the link, lower residual energy may correspond to higher cost. The goal is to minimize energy consumed

per packet, load balancing among cut-set nodes and maximize duration before a node fails due to energy depletion (network partition). [CT00] Proposes energy conserving routing, which selects the paths based on corresponding power level so that the time until the batteries of the nodes drain-out is maximized.

[RH00] proposes a method for selecting transmitting power level to adjust the network topology to maximize the network lifetime and reduces spatial interferences. [WLBW01] describes a topology maintenance algorithm using similar underlying radio support, their algorithm guarantees connectedness using directional information.

[RM99] Describes position-based distributed network protocol optimized for minimum energy consumption in mobile wireless networks that support peer-to-peer communications. A simple local optimization scheme executed at each node guarantees strong connectivity of the entire network and attains the global minimum energy solution for stationary networks. A position-based algorithm is proposed to set up and maintain a minimum energy network between users that are randomly deployed over an area and are allowed to move with random velocities.

4.6 Transport Layer

[SK97] compares Energy for Send, Receive and Idle period for two transport protocols, TCP Reno and Modified UDP. Researchers found that the dominant cost in the energy usage of a transport protocol is the time that the transfer takes to complete, not the number of packets send or receive by a particular transport protocol. Results conform that *idle* time makes major contribution to final

energy cost. At higher loss TCP mistakes packet losses for congestion and reduces transmission rate which increases total energy cost.

[FN01] investigated the per packet energy cost for Lucent WaveLAN Silver (11 Mbps) card in sleep mode, idle mode, receive mode and transmit mode. It was observed that while operating in ad hoc mode idle power consumption is significant because hosts required to maintain their network interface in idle mode to main connectivity of the ad hoc network. Experiments were performed on IBM ThinkPad running FreeBSD4.0 and equipped with Lucent 802.11 WaveLAN "Silver" card (11 Mbps). The comparison of idle power consumption is given in Table 2.

	Ad hoc Mode	BSS Mode
Idle Power (mWatts)	843	66

Table 2: Lucent WaveLAN PC Card Power Consumption

[TBGP00] Compared the energy and throughput-efficiency of TCP error control strategies of three implementations of TCP Tahoe, Reno, and New Reno. They believe that estimation of additional energy expenditure cannot be based solely on the byte overhead due to retransmission, since overall connection time might have been extended while waiting or while attempting only a moderate rate of transmission. On the other hand, the estimation cannot be based solely on the overall connection time either, since the distinct operations performed during that

time (e.g. transmission vs. idle) consume different levels of energy. Nevertheless, the potential for energy saving can be gauged from the combination of time and byte overhead savings achieved. Based on these metrics, one can estimate lower bounds for energy consumption.

The three versions of TCP were implemented using the x-kernel protocol framework. Their focus was to study heterogeneous wired/wireless environments. In order to simulate error conditions, a virtual protocol VDELDROP was configured between TCP and IP. It has two states consists of Markov chain. All of the experiments were undertaken using 5 Mbytes data transmission. Connection time, data packet overhead (retransmission etc.) were taken into consideration as significant factor for energy expenditure as well as throughput.

Under low loss (0.01 to 1%) conditions Reno outperforms other two versions of TCP as it gets sufficient time to expend the window to maximum, yet, unlike Taheo, when an error occurs it does not always follow Slow Start, but sometimes performs Fast Recovery instead. Tahoe performs distinctly better when errors are intensive (20 %) and persistent because large windows of data could continue to be transmitted for a period of time despite the prevailing error phase. New Reno can be the protocol of choice only for environments with relatively infrequent and short/random errors.

[ZR99] analyzes the energy consumption performance of various versions of TCP for bulk data transfer in an environment where channel errors are correlated. It is good to throttle data flow in the event of loss and congestion,

energy efficiency is considered as the ratio of number of successful transmissions over total number of transmissions. Analytical approach is based on a Markov/renewal reward approach. Only single transmitter-receiver pair running TCP on a 1.5 Mbps dedicated link with zero propagation delay and feedback were considered.

Efficient usage of energy is achieved when a scheme stops transmitting when channel conditions become adverse and resumes the transmission when they improve. This is exactly how window adaptation algorithm of TCP works. Tahoe performs better than TCP Reno when correlated fading is more. Brief summary of some of the research work dealing with performance analysis of variants of TCP where throughput was the metrics of comparison is as follows:

[FF96] shows that the SACK algorithm performs better than several non-sack based recovery algorithms when 1--4 segments are lost from a window of data. [AHKO97] compared Reno, Newreno, and SACK for communication over satellite links. They used two Intel machines with NetBSD1.1, two CISCO routers and ACTS VSAT connected to a satellite, and a hardware emulator for dropping packets. They dropped 1,2,3 and 4 packets for different experiments, used two data sizes (200KB and 5MB) and experimented with three bit error rates of 10e-5,10e-6 and 10e-7. For 1,2,3 and 4 packet loss the performance was similar to [FF96]. For losses of 10e-7 and 10e-6, they saw the same results as ours, and for the high loss case 10e-5, they show that all of the TCP variants performed poorly and SACK was similar to Reno and Newreno. [HK99] evaluated the performance of Reno,

Newreno, and SACK for a satellite network. Experiments were carried out in the simulator ns. They used a RTT in the range of 100 -- 600 msec. The paper shows that using partial ACKs to trigger retransmissions in conjunction with SACK, improves performance when compared with TCP using fast retransmit/fast recovery alone. Specifically, SACK-Newreno is better than SACK-Reno which is in turn better than Reno.

[BHZ98] performed experiments in which two PCs running FreeBSD were connected by a SUN UltraSparc acting as a router that inserted link delays. The link delay was 25 msec and bandwidth was limited to 2 Mbps. The buffer space at the sender and receiver was set to 16KB. They studied two types of losses -- random uniform loss (0% to 9%) and bursty loss where three packets were dropped randomly. In the random uniform loss case, they observed that if the loss probability is low then both Reno and SACK behave similarly. Likewise, if the loss probability is high, these two protocols again behave similarly because retransmitted packets at high loss probabilities will be lost causing SACK to timeout which is a normal case with Reno. However, if the loss probability is between 2% and 4% then SACK outperforms Reno. In the burst loss case, SACK improves throughput by a significant amount (60% to 70%). This is because for Reno the loss of three isolated packets or three consecutive packets results in the same behavior while SACK recovers quickly from a bursty loss and does not cause timeouts.

In the recent past there have been simulation based studies on the issues RF power and TCP throughput [BA+02] [ZRM02]. [ZRM02] concludes that increased transmitted power (RF power) is not always good as their study shows that increased transmitted power results in higher TCP throughput up to a breakpoint (because of better SIR), after which an increment of transmitted power actually leads to worse performance due to greater interference. However, we did not modify the WaveLAN card's transmission speed & power in our study

4.7 Application Layer

[FS99] used energy profiling of the application to dynamically adapt their quality to conserve energy when it is scare. An attribute called *fidelity*, captures the notion of data degradation in Odyssey. For example, a client playing full-color video data from a server could switch back to black and white video when bandwidth drops, rather than suffering lost frames. Energy used by video player, speech recognizer, map viewer, and Web browser were considered. It was found that hardware-centric power management combined with Dynamic *fidelity* results in maximum energy conservation in most of the cases.

[FBA+0] investigate the energy consumption of low-power StrongARM SA-1100 microprocessor based Itsy Pocket Computer. It recommends that for compute-intensive applications with no voltage switching, application should be run at as fast clock frequency as possible until completed, and then the system should be placed into a low-powered mode. They evaluate the energy and power impact of several tradeoffs in the design of a JVM (Java Virtual Machine) for Itsy

Pocket PC. Single JVM to run multiple applications can reduce as much as 25% energy usage. Preloading Java classes can reduce start up time without impacting energy consumption. JIT (Just-In-time) compilation can provide significant energy saving for Pocket PC.

4.8 Summary

In this chapter we summarize various practices adopted by research community during past few years to reduce the energy consumption of nodes in mobile ad hoc networks. We saw that these techniques can be broadly divided into three categories: System hardware level optimization, Protocol level, and Application level.

5. Experimental Setup

In this section we will discuss about experimental methodology, afterwards we discuss about the SACK implementation and its validation.

5.1 Experimental Methodology

We first describe our test environment: how the ad hoc network (Figure 13) is realized. Afterwards, we explain workload and factors considered for characterizing TCP Energy, later on we explain how we measured energy.

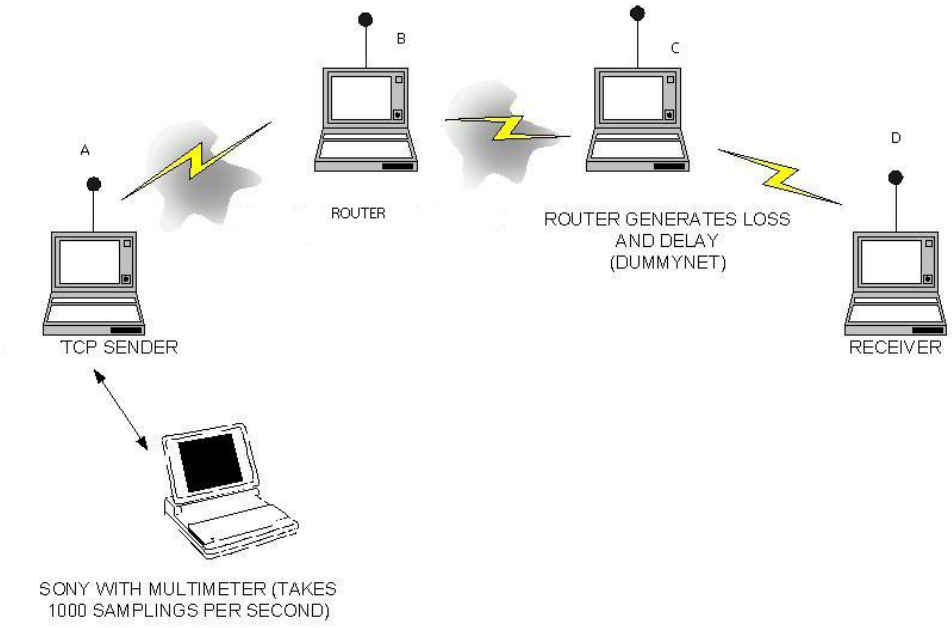

Figure 13: Test-bed setup

Characterizing the energy consumption of TCP in multi-hop wireless networks poses a challenge due to the fact that, on the one hand, to emulate

complex networking scenarios we need to use a simulator like NS-2 [NS01] while, on the other hand, measuring the actual energy consumed requires us to use a real system (the energy models in NS-2 do not consider node costs whereas simulators like Simple Power [Sim01], which simulate node-level energy costs, do not simulate networks of communicating nodes). The approach we settled on was a hybrid one -- we used an actual wireless network (as illustrated in Figure 13 where we could measure the energy consumed by the sender).

5.2 Workload

We used TTCP [PS98] as a workload generator. TTCP is a popular public domain tool for measuring end-to-end throughput by sending bulk data over the network. It achieves high performance by filling a memory buffer with data, then repeatedly transmitting this data. Since there are no I/O operations involved, the traffic transmitter and receiver can operate at true network speeds. TTCP has a simple command line interface which allowed us (among other settings) to specify the receiver window size and the sender's buffer size.

5.3 Test-bed

As shown in Figure 13, we use four laptops each running FreeBSD 4.3. All laptops were equipped with 2.4GHz DSSS Lucent 802.11b 11Mb/s WaveLAN PC cards. The path from the sender to the receiver traverses three hops. We considered a string topology as in A->B->C->D. A is sender and D is receiver, B and C are routers. A and B are on one subnet, B and C are on a separate subnet, and C and D

are on same subnet. For A, B is the default router. B and C have static routes to each other. For D, C is the default router. The WaveLAN interface on B and C was aliased in order to be on two different subnets.

We used Dummynet at node C to simulate a wide range of network conditions. Dummynet [Riz97] works by intercepting communications of the protocol layer under test and simulating the effect of finite queues, bandwidth limitations and communication delays. It is implemented as a filter in the protocol stack of the operating system. Packets that are handed from one network layer to another can be intercepted and passed through objects called pipes. Since no copies of the data are done, and all operations require constant time, the overhead introduced by Dummynet is almost negligible.

We used Dummynet with HZ=1000, a kernel compile option which will give a granularity of 1 ms. Pipes can be configured to simulate the effect of packet loss, reordering, burst error and propagation delays. Each pipe can be configured separately, so that one can assign different attributes like packet loss rates, delays, bandwidth etc..

Delay sets the propagation delay of the pipe, in milliseconds. All delays are with the granularity of 1/HZ seconds. Random packet loss rate (PLR) is a floating-point number between 0 and 1 which causes packets to be dropped at random. This is done to simulate lossy channel. All of the above parameters of the pipe can be configured and changed on the fly using simple IPFW [HB01] commands. Delay

and PLR emulated conditions similar to the real world where TCP connection has to react to sudden change in the wireless channel.

We used separate channel for conducting our test, we also made sure that there is no interference which might lead to error in our results. We used tcpdump to validate no other traffic was present on the channel.

5.4 Energy Measurement

To measure the energy consumed by the sender (laptop A) we connected its' power supply (in series) to HP 34401A multimeter that is controlled by a separate laptop running VeePro software. We used two multimeters – one measured the total system energy while the second measured the radio-level energy alone (Figure 14 shows a sample data trace).

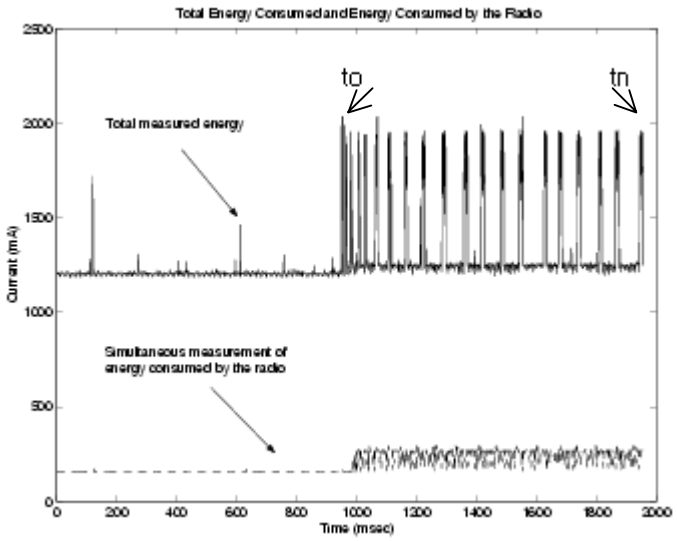

Figure 14: Sample of Measured Data

The multimeter takes 1000 current readings per second and saves them to the laptop running VeePro through a serial cable. Throughout the experiments, the TCP connection between A and D is the only networking activity in the channel. In order to ensure accurate energy measurements, no other computational activity is done on A.

Our sender was a Toshiba Satellite laptop whose idle current draw was 1.2A at a voltage of 15V. Figure shows a sample trace of a simultaneous measurement of the total system energy and the energy draw of the radio card. See [FN01] for a detailed characterization of the radio energy consumption.

We start measuring energy at sender before the actual start of TTCP (we used automated shell script for this) and then we use generated current trace to find out actual value of t_0 and tn. In our experiments t_0-t_n is the interval over which TTCP is running and channel is active.

5.5 Experimental Design

Our goal in this research was to characterize and improve TCP's energy consumption in ad hoc networks. At a high-level, we can classify ad hoc networks as those where nodes are very mobile (e. g., network of vehicles) or those where nodes move slowly, if at all (e. g., peer-to-peer networks or static ad hoc networks like Rooftop). In the former case, mobility is the primary detractor of performance while in the latter case, link-layer loss/ congestion is the primary detractor.

5.5.1 Factors

We divided our study into two parts - the first examined the impact of mobility induced factors and in the second, we studied the impact of loss on the three protocols.

5.5.1.1 Mobile Nodes

To study the impact of mobility, we emulate routing failure and out-of-order packets. As Reno, Newreno, and SACK do not have any mechanism to detect route failure that's why all the three protocols performed poorly. Therefore, we decided to drop route failure case.

5.5.1.2 Static Nodes

Ad hoc network nodes communicate over relatively low bandwidth links which are prone to high propagation loss, burst error, fading and multiuse interference. To study static ad hoc networks, we emulate these conditions by creating random loss and bursty loss at node C using *dummynet*.

The summary of factors considered three different conditions: *random packet loss, bursty loss and packet reordering* is tabulated as follows:

Parameters	Values
Average RTT	15, 40, 70, 100, 130 msec
Packet Loss	1%, 5%, and 10%
MTU Size	512 and 1500 bytes
RTS/CTS	OFF
Protocols	Reno, Newreno, and SACK

Table 3: Experimental parameters for loss case

Parameters	Values
Average RTT	15, 40, 70, 100, 130 msec
Burst Packet Loss	85% loss rate for 1 second every 12 seconds
MTU Size	1500 bytes
RTS/CTS	OFF
Protocols	Reno, Newreno, and SACK

Table 4: Experimental parameters for the burst loss case

Parameters	Values
Average RTT	15, 40, 70, 100, 130 msec
Packet Loss	1%, 5%, and 10%
Reorder Rate	1% and 5% reordering
MTU Size	512 and 1500 bytes
RTS/CTS	OFF
Protocols	Reno, Newreno, and SACK

Table 5: Experimental parameters for the packet reordering case

5.6 Implementation

We implemented SACK in FreeBSD-4.3. We derived our implementation from the SACK implementation in OpenBSD2.9, which itself based on Sally

Floyd's SACK algorithm described in RFC2018. We used *netstat*, *tcpdump*, Shawn Ostrermann's *tcptrace* [Tra02] and our own shell scripts for validating SACK implementation. We modified *tcptrace* [Tra02] to generate more specific graph.

5.7 Validation

We used our test-bed as described in Figure 15 for validation. Node 'C' was configured to generate End-to-end propagation delay and random uniform loss as 40msec and 2% (for data only) respectively.

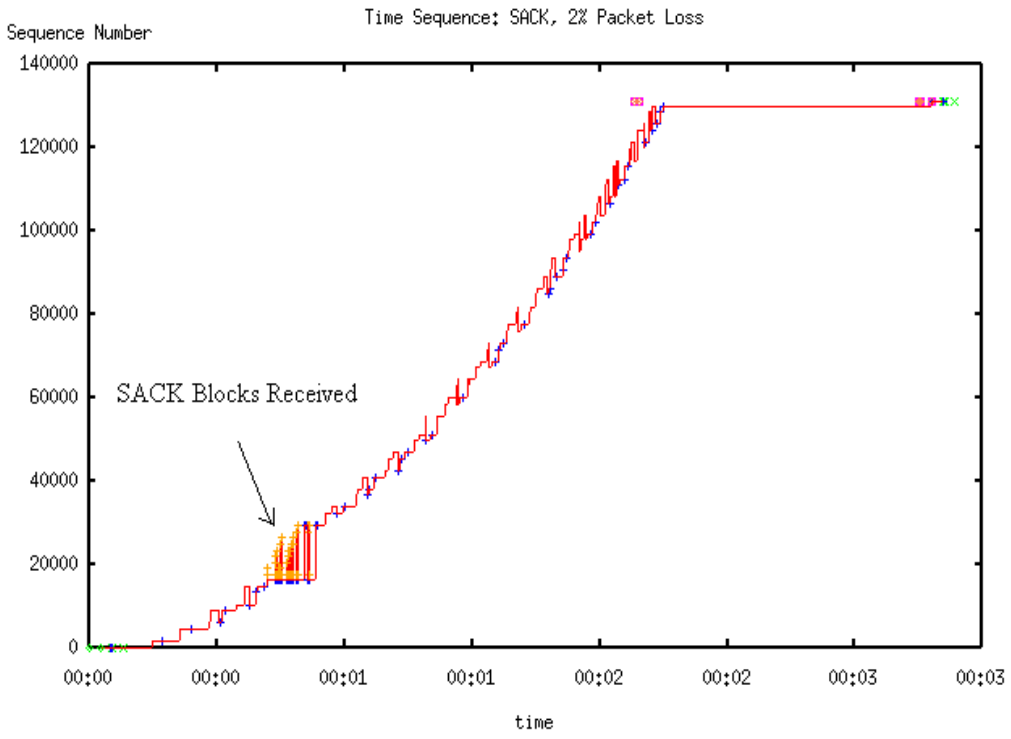

Figure 15: Evolution of ACK received by TCP-SACK Sender

Figure 15 shows the evolution of ACKs received by the TCP-SACK sender. At 0.5 second sender receives consecutive duplicate ACKs each bearing the recent SACK block information from sender. The snapshot of duplicate ACK period is given below in Figure 16.

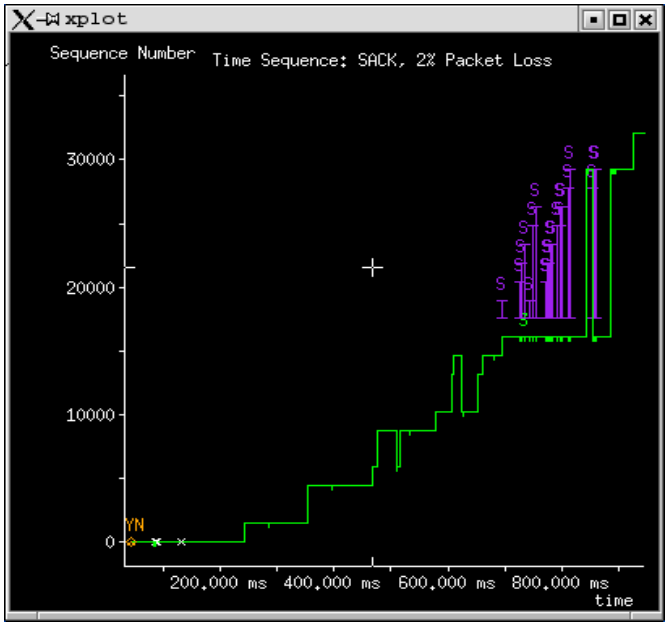

Figure 16: Snapshot Evolution of ACK received by TCP-SACK Sender

Afterwards we analyzed the *tcpdump* file using *tcptrace* the details of the same are as follows:

We use abbreviation as S: Sender, R: Receiver, ACK: acknowledgement. Sequence numbers are unique and numbered starting at some value and increasing over a period of time. However, for simplicity we used Sequence offset. We have also dropped ACK sequence from the Sender to the Receiver as it is same.

S > R SYN 0:0 (0)	MSS (1460) SACKREQ		
R > S SYN 0:0 (0)	ACK (1) MSS (1460) SACKREQ		
S > R ACK (1)	three-way-handshake		
S > R 1:1461 (1460)	cwnd (1)		
R > S ACK (1461)			
S > R 1461:2921 (1460)	cwnd (2)		
S > R 2921:4381 (1460)	cwnd (2)		
R > S ACK (4381)			
S > R 4381:5841 (1460)	cwnd (3)		
S > R 5841:7301 (1460)	cwnd (3)		
S > R 7301:8761 (1460)	cwnd (3)		
R > S ACK (5841)			
R > S ACK (8761)			
S > R 8761:10221 (1460)	cwnd (5)	41136	
S > R 10221:11681 (1460)	cwnd (5)	39676	
S > R 11681:13141 (1460)	cwnd (5)	38216	
S > R 13141:14601 (1460)	cwnd (5)	36756	
S > R 14601:16061 (1460)	cwnd (5)	35296	
R > S ACK (10221)			
R > S ACK (13141)			
R > S ACK (14601)			
S > R 16061:17521 (1460)	cwnd (8)	33836	segment dropped
S > R 17521:18981 (1460)	cwnd (8)	32376	
S > R 18981:20441 (1460)	cwnd (8)	30916	
S > R 20441:21901 (1460)	cwnd (8)	29456	
S > R 21901:23361 (1460)	cwnd (8)	27996	
S > R 23361:24821 (1460)	cwnd (8)	26536	
S > R 24821:26281 (1460)	cwnd (8)	25076	
R > S ACK (16061)	SACK [17521--18981]		Positive ACK
R > S ACK (16061)	SACK [17521--20441]		Dup ACK
R > S ACK (16061)	SACK [17521--21901]		Dup ACK
R > S ACK (16061)	SACK [17521--23361]		Dup ACK

S > R 26281:27741 (1460) cwnd (8)
S > R 27741:29201 (1460) cwnd (9) cwnd = 0

R > S ACK 16061 SACK [17521--24821]
R > S ACK 16061 SACK [17521--26281]

S > R 16061:17521 (1460) cwnd (1) fast-retransmit

R > S ACK 16061 SACK [17521--27741]
R > S ACK 16061 SACK [17521--29201]
R > S ACK 29201

5.7.1 Discussions

SACK Permitted Option gets negotiated during connection establishment phase. This two bytes option is not sent on a non-SYN segment.

Segment (16061:17521) gets dropped and the Receiver uses the SACK option to inform the Sender about the non-contiguous blocks of data received. The first SACK block specifies the contiguous block of data containing the segment which triggered this ACK. The subsequent SACKed blocks contain the most recent information. Once the Sender receives three duplicate-ACKs it invokes fast-retransmit and fast-recovery algorithm and based on hole information it retransmits specific segments.

6. Experimental Results

All experiments were conducted at least ten times and we computed 95% confidence intervals (which are also shown in the figures). Furthermore, in order to get statistically significant results, we transmitted 5M of data for each run (transmitting smaller amounts of data resulted in high measurement error due to the 1msec granularity of the multimeter).

6.1 Metrics

We evaluated Reno, Newreno, and SACK using three metrics:

- *Total Energy/bit E* measured in Joules/bit. This includes the energy consumed while the sender is idle.
- *Idealized Energy/bit* E_I measured in Joules/bit. This measure excludes the idle time energy and thus more closely approximates the cost of the various protocols.
- *Goodput in kbps*

6.2 Random uniform Loss Case

Table 3 outlines the experimental design for this group of experiments. For each case, we transmitted 5MB of data using TTCP. Figures 17, 20, 21, and 22 plot (1) the total energy *E* as a function of different RTTs for two values of the MTU and (2) the goodput as a function of RTT. We can make several observations based on these results:

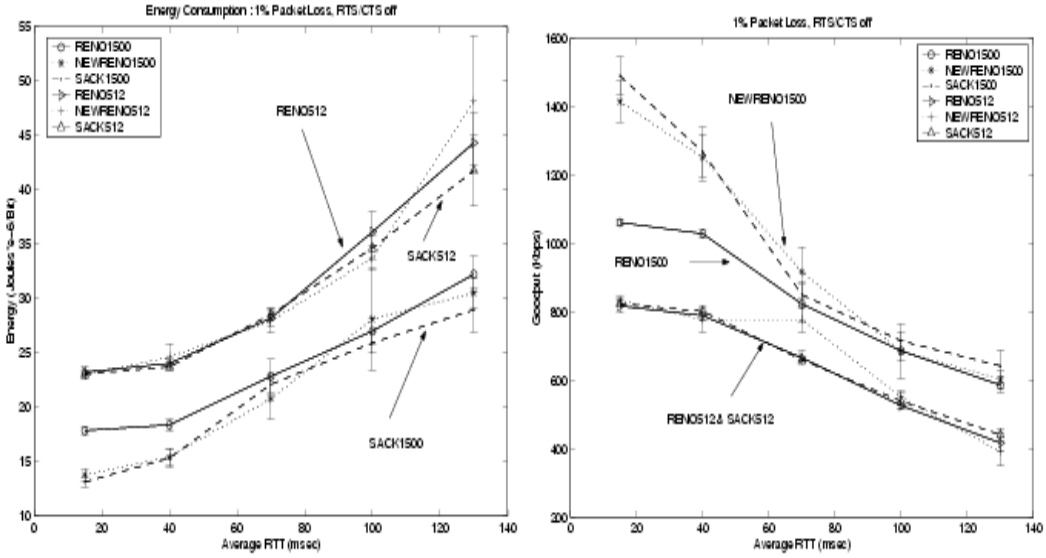

Figure 17: Total Energy E per bit and Goodput for 1% packet loss

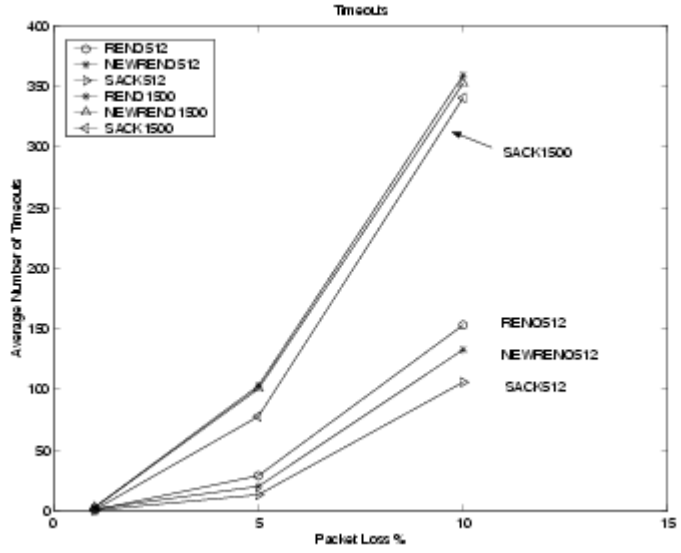

Figure 18: Comparison of the number of timeouts

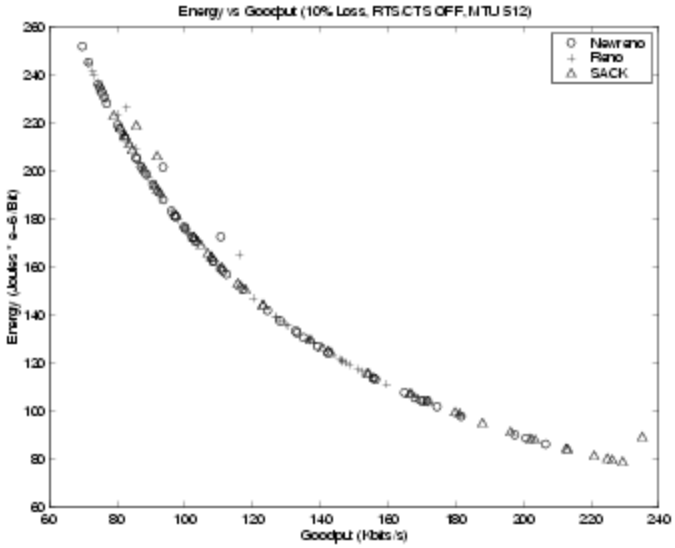

Figure 19: Relationship between energy and throughput (RTS/CTS Off)

- In general, smaller MTUs are better (i.e. consumes less energy at the high losses (see Figure 21, the 10% loss case) whereas larger MTUs are better at low loss (see Figure 17). This observation has been made by several previous researchers when throughput was the metric studied.
- For most cases, SACK consumes the least amount of energy. The reason is that SACK retransmits missing segments earlier than Reno (which waits for a timeout in most cases) and Newreno (which does not know which of the unacked segments is missing at the receiver). Thus, SACK completes transmission of the data sooner resulting in lower overall energy.
- Figures 17, 20, 21, and 22 show that for the 1% and the 5% loss cases, SACK consumes the least amount of energy (for both MTU sizes) while for

the 10% loss case, SACK consumes the most energy at an MTU of 1500. There are two reasons for this:

- When the loss rate is high, with large MTU, the number of segments in a window is small and hence the possibility of receiving 3 dupacks is small. In this case the fast retransmit algorithm is seldom triggered. Thus SACK does not really help improve throughput (see also RFC 3042 which notes the same problem and suggests the use of a "Limited Transmit Algorithm" which we have not implemented).

Figure 18 shows that the number of timeouts with or without SACK at MTU 1500 is about the same. With a smaller MTU, on the other hand, SACKs do help more and therefore we see that at higher packet loss rates, the timeouts for the SACK case are smaller.

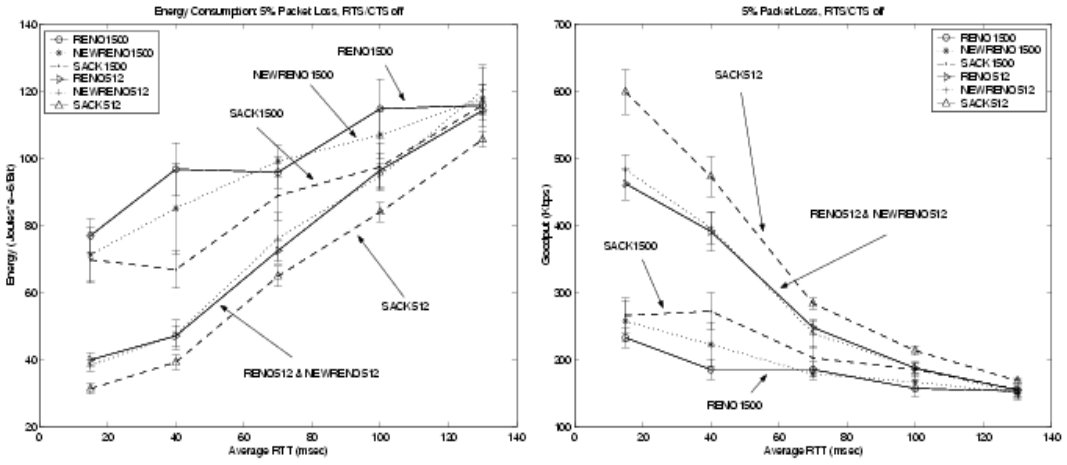

Figure 20: Total Energy E per bit and Goodput for 5% packet loss

- Since the SACK throughput is small, the total energy consumed by SACK will be at least as high as Reno and Newreno. However, SACK adds an additional burden in terms of computational overhead and this results in a higher total energy consumption.

- The energy consumed increases with an increase in RTT. This is clear because the throughput falls with an increase in RTT.
- We plot the total energy per bit as a function of goodput for the 10% loss case and a MTU of 512 bytes in Figure 21. As we can clearly see, as the goodput decreases the energy used increases and there is an inverse relationship between these two quantities as predicted by equation

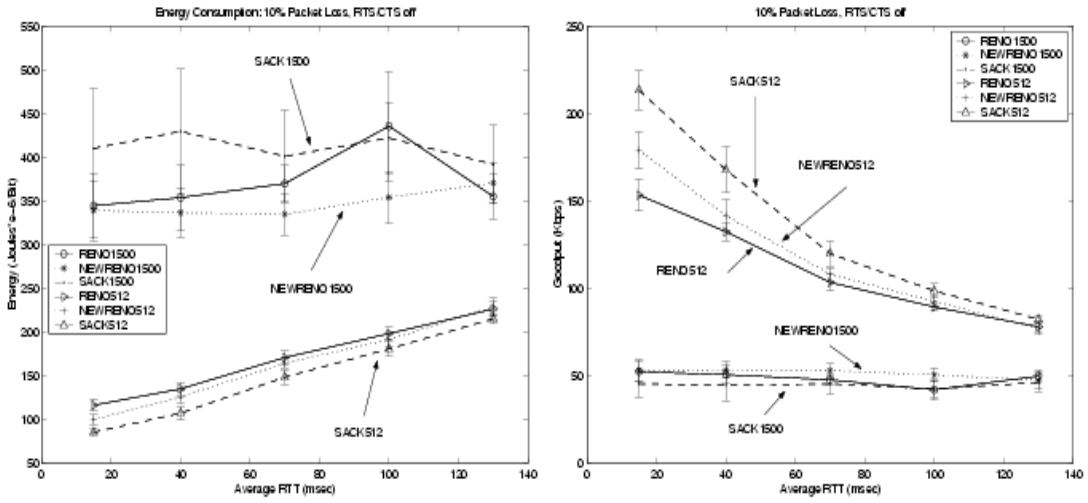

Figure 21: Total Energy E per bit and Goodput for 10% packet loss

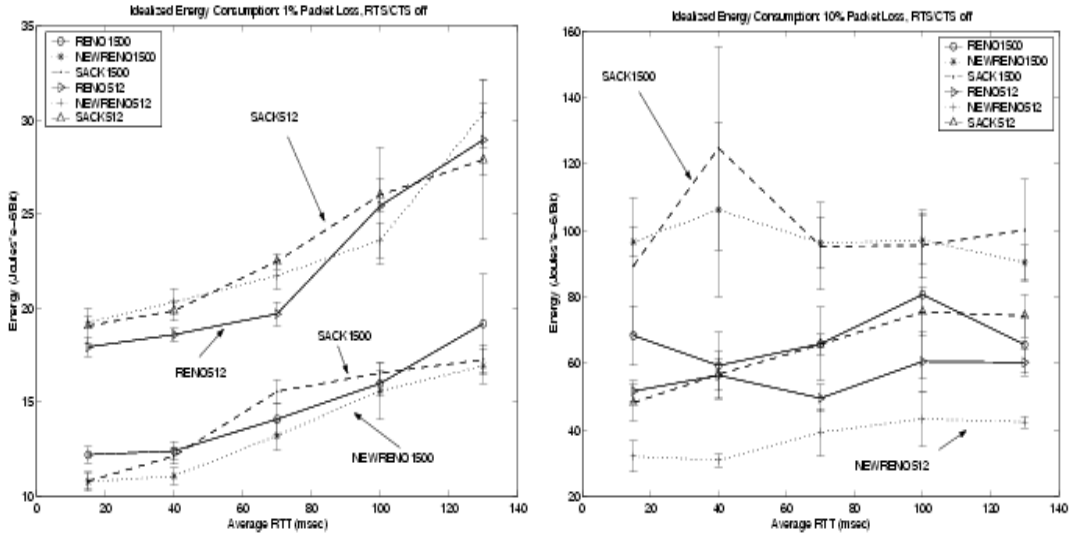

Figure 22: Idealized Energy E_I per bit 1% and 10% packet loss

We take a detour and present a trace of 5% loss case, we used *tcptrace*[4] for generating the graphs. We also tabulate the packet retransmitted, timeout events for three TCP protocols Reno, Newreno, and SACK.

[4] We modified tcptrace to generate specific graphs.

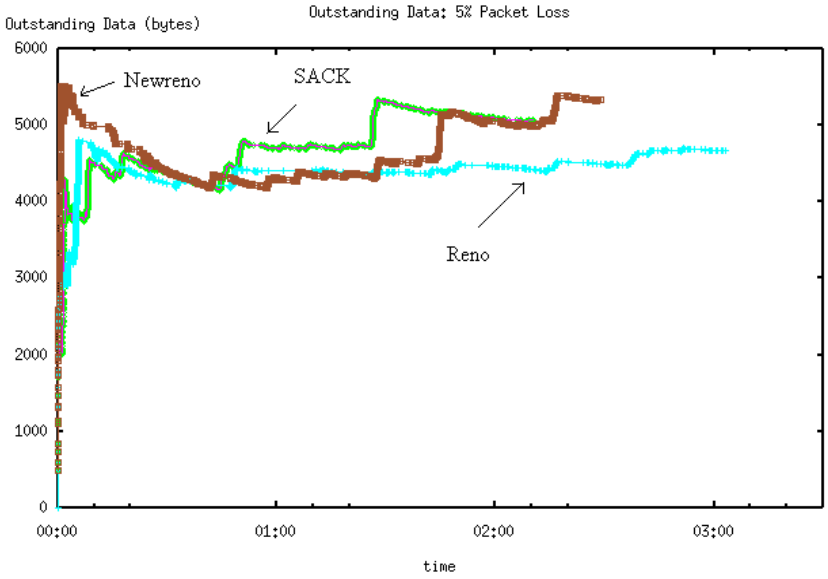

Figure 23: Evolution of CWND, 5% Packet Loss case

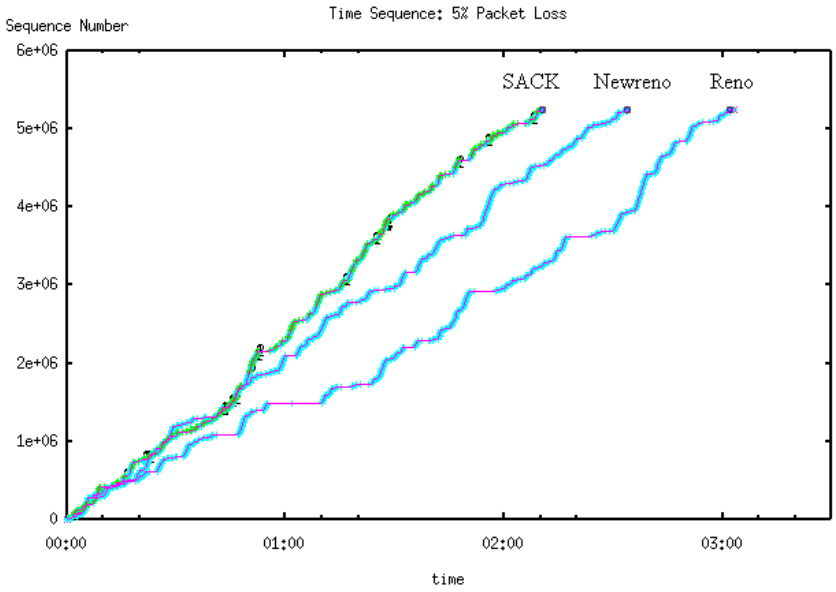

Figure 24: Time-Sequence (ACK), 5% Packet Loss case

	Timeout	Pkts Retransmitted
SACK	69	85
Newreno	87	215
Reno	93	204

Table 6: Timeout and Pkts Retransmitted, 5% Loss case

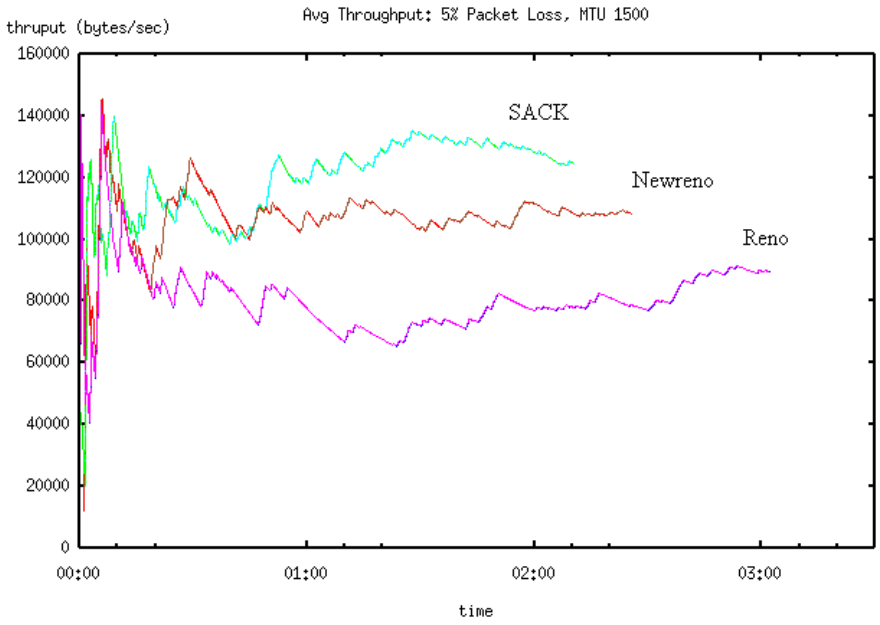

Figure 25: Average Throughput, 5% Packet Loss case

SACK has the highest throughput (Figure 25) it achieves so by maintaining high cwnd (Figure 23) and lower number of timeouts as compare to Reno and Newreno. Reno has the poorest performance because when multiple segments are lost in a window it will be forced to invoke fast-retransmit and recovery algorithm in quick succession (no partial ACK) which will lead to small cwnd. It seldom

successfully comes out of fast-recovery as Reno's "ACK clock" is lost and must wait for retransmission timeout. Figure 24 shows that Reno has longer time periods where no ACK is received by the sender which leads to timeout. [FF96] also found the same behavior of these three protocols.

We return to the discussion of loss case. In Figure 22 we plot the ideal energy E_I consumed for the 1% and 10% loss cases. At the 1% loss case we see that SACK actually consumes more energy than either of Reno or Newreno at both MTU sizes! This is because, as we noted above, SACK has an added computational burden that only becomes visible when we discount the idle energy cost. We see a similar pattern in the 5% and 10% loss cases as well.

	1% Loss	5% Loss	10% Loss
MTU 512	$E^S \approx E^N \approx E^R$ $\tau_S \approx \tau_N \approx \tau_R$ $E_I^R < E_I^N \approx E_I^S$	$E^S < E^N \approx E^R$ $\tau_S > \tau_N \approx \tau_R$ $E_I^N \approx E_I^S \approx E_I^R$	$E^S < E^N \approx E^R$ $\tau_S > \tau_N > \tau_R$ $E_I^N < E_I^R < E_I^S$
MTU 1500	$E^S < E^N < E^R$ $\tau_S \approx \tau_N > \tau_R$ $E_I^N < E_I^R \approx E_I^S$	$E^S < E^N < E^R$ $\tau_S > \tau_N > \tau_R$ $E_I^N \approx E_I^R < E_I^S$	$E^N < E^R < E^S$ $\tau_N > \tau_R > \tau_S$ $E_I^R < E_I^N \approx E_I^S$
R – Reno, N – Newreno, S – SACK			

Table 7: Summary of Total energy, goodput, and idealized energy

Table 7 summarizes the relative performance of the protocols as measured by the total energy, the goodput, and the idealized energy for all of the loss and MTU combinations (we derive these rank orderings based on a comparison of the mean values of energy, goodput, and idealized energy for the different RTTs and MTUs). We note that the total energy and goodput are inversely related. Thus, the

protocol that has the highest goodput also has the lowest total energy. We also note that in all cases except MTU 1500 & 10% packet loss rate S*ACK has the highest goodput and the lowest total energy*. In this one case, Newreno has the lowest energy and highest throughput.

When we look at the idealized energy, on the other hand, we see that *either Newreno or Reno use the lowest idealized energy* in all cases except in the MTU 512 & 5% packet loss case where SACK and Newreno perform equally well. In general, we believe that SACK performs poorly with respect to this metric because of the additional data structures and computation it performs. Newreno and Reno, on the other hand, have very similar computational overhead. The only reason Newreno sometimes performs better in some cases (e.g., MTU 512 & 10% packet loss) is that in fast recovery it retransmits unacked packets before the retransmit timer goes off. Reno, on the other hand, only retransmits one packet (the one that received three duplicate acks) and then retransmits the remaining packets when the retransmit timers go off. Figure 18 plots the number of timeout events for different MTU sizes. As we can see, at a MTU of 512, Newreno has fewer timeouts than Reno and thus performs better by this metric. However, even though SACK has the least number of timeouts, its computational cost is high enough to offset any gain due to its better throughput.

The lesson here is that if the wireless devices can be designed to enter deep power saving modes when there is inactivity (i.e., reduce the idle cost as much as possible), then SACK may not be a good choice for mobile environments. On the

other hand, if the idle power remains high, then SACK definitely results in overall energy savings.

6.3 Burst Loss Case

Figure 26 plots the energy and idealized energy cost for the experimental setup described in Table 4. In the energy plot, we plot both the total energy E as well as the idealized energy (E_I) in the same graph. We see that SACK has the lowest total energy while Newreno has the lowest idealized energy consumption.

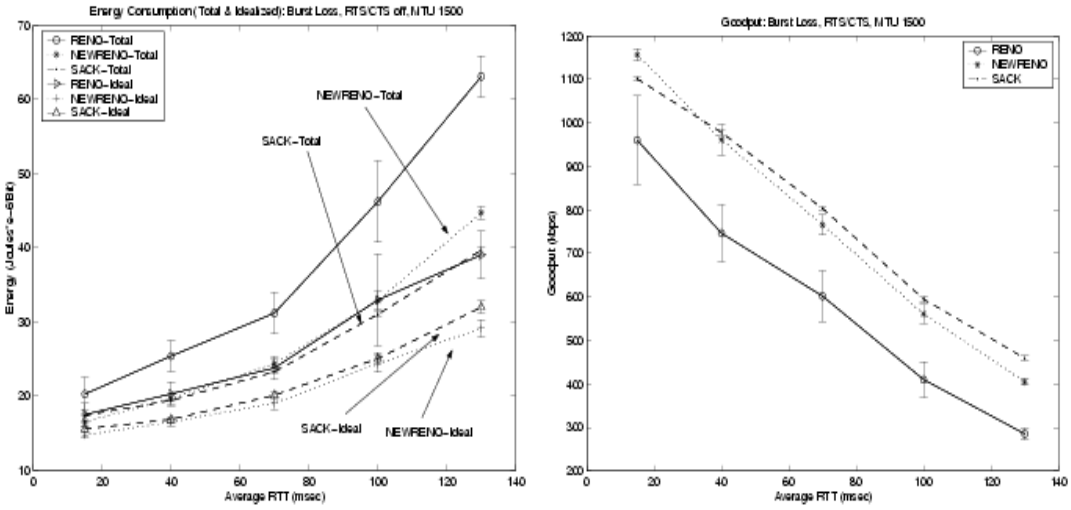

Figure 26: Summary of Energy and throughput for burst loss

The reason for this behavior is that in the case of a bursty loss, Newreno will retransmit lost packets without waiting for the retransmit timers to go off (based on partial ACKs received). SACK, likewise will retransmit these packets as well (as indicated by the SACKs) but it also has the added overhead of maintaining

SACK-related data. This additional cost results in SACK having a higher idealized energy cost even though its goodput is the highest (Figure 26).

6.4 Packet Reordering Case

Figures 27 and 28 plot the total energy and idealized energy for the reorder case (see Table 5). We can summarize the main findings as follows:

- SACK is by far the winner in terms of total energy as well as idealized energy. The goodput of SACK is also the highest.

  Let us first look at the reason for SACK's better goodput (and total energy). When the sender receives three duplicate ACKs (that also contain information about holes in the receiver's buffer), the sender retransmits one segment and then retransmits the segments that corresponded to holes in the receiver's buffer as and when *pipe* is less than CWND. Newreno, on the other hand, sequentially retransmits segments on receipt of partial ACKs. This results in some packets not being retransmitted early enough and we get timeout events. In our experiments, we noted that SACK never had any timeouts for the reorder experiments while both Reno and Newreno had timeout events (Reno more than Newreno).

- The above discussion explains why SACK has a higher goodput and lower total energy than Newreno and Reno. The reason it also has the lowest idealized energy is because (1) there are no timeouts for SACK thus reducing the processing involved, (2) SACK retransmits fewer packets than Reno and Newreno.

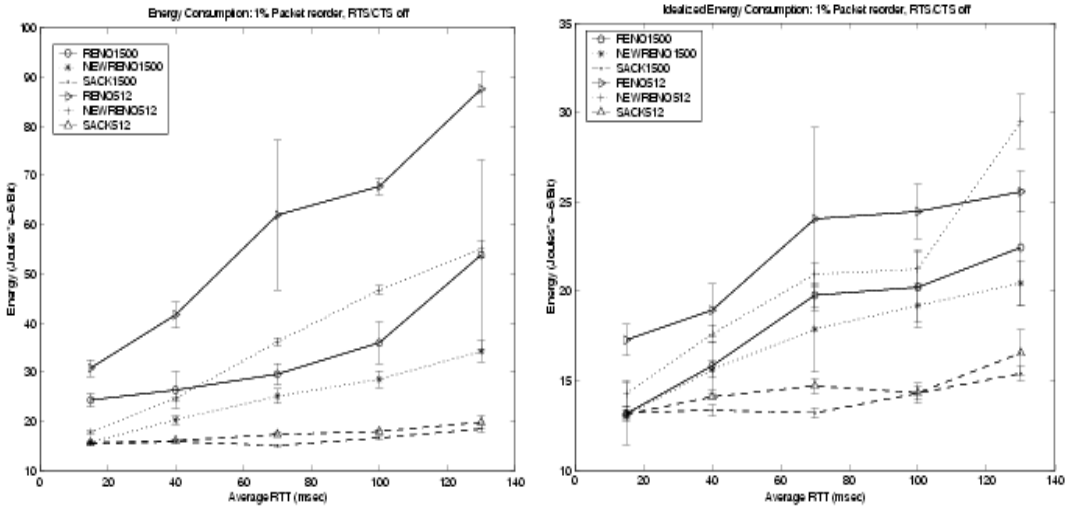

Figure 27: Total energy and idealized energy for 1% packet reordering case

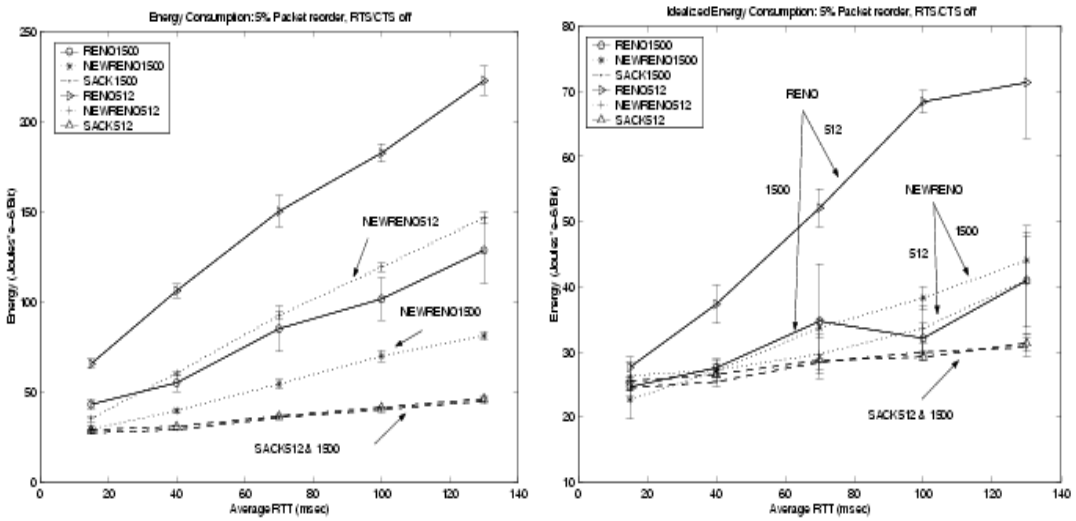

Figure 28: Total energy and idealized energy for 1% packet reordering case

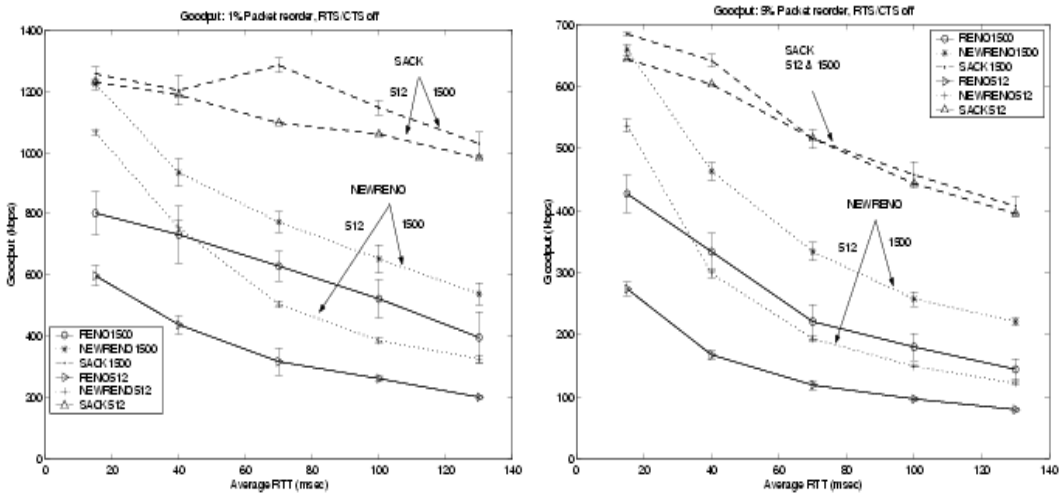

Figure 29: Goodput for reordering case

7. Conclusion

We analyzed the relative performance of TCP Reno, Newreno, and SACK. We measured performance in wireless network with real TCP/IP stack in FreeBSD 4.3. We introduced new metric Idealized energy, which closely approximates the protocol processing energy. We also used other two metrics Total energy and Goodput in analyzing relative behavior of these three protocols. We note the following:

- The total energy consumed (for any protocol) is inversely proportional to the throughput.
- The total energy consumed for SACK is the lowest in almost all cases except for MTU 1500 at a loss rate of 10%. The reason SACK has the lowest energy cost is that it has the highest throughput and the idle energy of our device was high (18W). This high idle energy cost plays a dominating role in the total energy measurements.
- SACK has a poor idealized energy performance due to the fact that SACK implementations require additional data structures and processing. This is the reason that SACK performs poorly for the 10% loss case with 1500 MTU -- here the benefits of using SACK do not come into play since the CWND contains few segments. However, the computational overhead is still present and this causes SACK to have higher total energy consumption.

- In the case of packet reordering, SACK is by far the winner in terms of total energy and idealized energy. This is because SACK has no timeouts for the cases we looked at (both Reno and Newreno have timeouts).
- In the case of bursty losses, SACK has the lowest total energy while Newreno has a slightly lower idealized energy. The reason is again that SACK's computational cost overwhelms the gains due to a higher throughput.

Using the above results we can conclude that when designing a protocol for a mobile device, we must first consider the operating environment (bursty loss or random loss, etc.) and then select the appropriate level of protocol complexity since the computational overhead of a protocol can be significant. In our case, if we extrapolate and imagine a device with very low idle power consumption then it is clear that SACK would be a poor choice for most situations.

The total energy consumed for the SACK is the lowest in almost all cases except for MTU 1500 at the end-to-end loss rate of 10%. The reason SACK has the lowest energy cost is that it has the highest throughput and ideal energy of our device was high. This high idle energy cost plays a dominating role in the total energy consumption.

8. References

[AHKO97] Mark Allman, Chris Hayes, Hans Kruse, and Shawn Ostermann, "TCP Performance Over Satellite Links", In *Proceedings of the 5th International Conference on Telecommunication Systems*, March 1997.

[BA+02] Sorav Bansal et.al., "Energy efficiency and throughput for tcp traffic in multi-hop wireless networks", In *Proceedings of INFOCOM 2002*, New York, NY, 2002.

[BB96] T. D. Burd and R. W. Brodersen, "Processor design for portable systems", *Journal on CLSI Signal Processing,* 13(2/3):203-222, August 1996.

[BHZ98] R. Bruyeron, B. Hemon, and L. Zhang, "Experimentations with TCP Selective Acknowledgment", *ACM Computer Communications Review*, Vol. 28(2), April 1998.

[BPS99] J. Bennett, C. Partridge, and N. Shectman, "Packet Reordering is Not Pathological Network Behavior", *IEEE/ACM Transactions on Networking*, December 1999.

[CJBM01] Benjie Chen, Kyle Jamieson, Hari Balakrishnan, and Robert Morris, "Span: An Energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks", In *Proceedings of MOBICOM 2001*, July 2001

[CT00] J. H. Chang and L. Tassiulas, "Energy conserving routing in wireless ad hoc networks", In *Proceedings of INFOCOM,* Tel-Aviv, Israel, 2000.

[DKM+94] F. Douglis, F. Kaashoek, B. Marsh, R. Caceres, K. Lai and J. Tauber, "Storage Alternatives for Mobile Computers", In *Proceedings of Symposium on Operating Systems Design and Implementation,* OSDI, November 1994.

[FBA+0] J. Flinn, G. Back, J. Anderson, K. Farkas, and D. Grunwald "Quantifying the energy consumption of a pocket computer and a java virtual machine", In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems SIGMETRICS'2000*, June 2000.

[FH99] Floyd, S. and T. Henderson, "The Newreno Modification to TCP's Fast Recovery Algorithm", *RFC 2582*, April 1999.

[FN01] Laura Feeny and Martin Nilsson, "Investigating the Energy Consumption of a Wireless Network Interface in an Ad Hoc Networking Environment", In *Proceedings of INFOCOM,* Anchorage, Alaska, 2001.

[FF96] K. Fall and S. Floyd, "Simulation-based Comparison of Tahoe, Reno, and SACK TCP", *ACM Computer Communications Review*, Vol. 26(3), July 1996, pp. 5 -- 21.

[FS99] Jason Flinn and M. Satyanarayanan, "Energy-aware adaptation for mobile applications", in *Proceedings of the 17th ACM Symposium on Operating Systems Principles,* December, 1999

[FMMP00] S. Floyd, J. Mahdavi, M. Mathis, M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP", *RFC 883*, July 2000

[GCW95] K. Govil, E. Chan, and H. Wasserman, "Comparing algorithm for dynamic speed-setting of a low CPU", in *Proceedings of The First ACM International Conference on Mobile Computing and Networking,* pages 13-25, November 1995.

[HB01] FreeBSD 4.3 handbook**,** http://freebsd.org/handbook/index.html (October 2001)

[HK99] Tim Henderson, Randy Katz, "Transport Protocols for Internet-compatible Satellite networks", *IEEE Journal on Selected Areas of Communications*, February 1999.

[Hoe96] J. Hoe, "Improving the Start-up Behavior of a Congestion Control Scheme for TCP", In *ACM SIGCOMM*, August 1996.

[HS00] Paul J. M. Havinga and Gerard J. M. Smit, "Energy-efficient TDMA medium access control protocol scheduling", In *Proceedings of Asian International Mobile Computing Conference, (AMOC 2000)*, November 2000.

[HSB00] Paul J. M. Havinga and Gerard J. M. Smit, M. Bos "Energy efficient wireless ATM design", *ACM/Baltzer Journal on Mobile Networks and Applications (MONET), Special issue in Wireless Mobile ATM technologies*, Vol. 5, No. 2, 2000.

[HV99] Gavin Holland and Nitin H. Vaidya, "Analysis of TCP performance over mobile ad hoc networks", in *ACM Mobile Computing and Networking (MOBICOM'99)*, pages 219-230, 1999.

[HW96] E.P. Harris and K.W. Warren, "Low Power Technologies: A System Perspective", *3rd International Workshop on Mobile Multimedia Communications, Princeton, NJ,* September 25-27, 1996.

[Jac88] Van Jacobson, "Congestion Avoidance and Control", *ACM SIGCOMM,* August 1998.

[JM96] David B Johnson and David A Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[JV02] Eun-Sun Jung, Nitin H. Vaidya, "An Energy Efficient MAC Protocol for Wireless LANs", In *Proceedings of INFOCOM* 2002.

[LKHA94] K. Li., R. Kumpf, P. Horton and T. Anderson, "A Quantitative Analysis of Disk Drive Power Management in Portable Computers", In *Proceedings 1994 USENIX,* San Francisco, CA, pp. 279-291, 1994.

[Lor95] J. Lorch, "A complete picture of the energy consumption of a portable computer", *Masters Thesis, Computer Science, University of California at Berkeley*, December 1995.

[LSS99] P. Lettieri, C. Schurgers, M. Srivastava, "Adaptive Link Layer Strategies for Energy Efficient Wireless Networking", *Wireless Networks, ACM/Baltzer*, Vol.5, No.5, pp.339-355, 1999.

[MMFR96] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgement Options", *RFC 2018*, October 1996.

[MSB00] Jerey P. Monks, Prasun Sinha, and Vaduvur Bharghavan. "Limitations of tcp-elfn for ad hoc networks", in *Proceedings of MoMuC, Tokyo*, Japan, October 2000

[NS01] NS-2 Network Simulator, http://www.isi.edu/nsnam/ns/(October 15, 2001).

[Pos81] J. Postel, "Transmission Control Protocol", *RFC 793*, September 1981.

[PS98] S. Parker, C. Schmechel, "Some Testing Tools for TCP Implementors", RFC 2398, August 1998

[Pow95] R.A. Powers, "Batteries for Low Power Electronics", In *Proceedings of the IEEE*, pages 687-693, April 1995

[RH00] R. Ramanathan and R. Rosales-Hain, "Topology Control of Multihop Wireless networks using transmit power Adjusment", in *Proceedings of INFOCOM,* Tel-Aviv, Israel, 2000.

[RFC768] J. Postel, "User Datagram Protocol", *RFC 768*, August 1980.

[RFC2018] Mathis, M., Mahdavi, J., Floyd, S. and A. Romanow, "TCP Selective Acknowledgement Options", *RFC 2018*, April 1996.

[RFC2581] Allman, M., Paxson,V. and W. Stevens, "TCP Congestion Control", *RFC 2581*, April 1999.

[RFC2883] S. Floyd et.al., "An Extension to the Selective Acknowledgement (SACK) Option for TCP", *RFC 2883*, July 2000.

[RFC 3042] M. Allman, H. Balakrishnan, "Enhancing TCP's Loss Recovery Using Limited Transmit", *RFC 3042*, January 2001.

[Riz97] L. Rizzo, "Dummynet: a simple approach to the evaluation of network protocols", *ACM Computer Communication Review*, Vol.27, No1, Jan. 1997.

[RM99] V. Rodoplu and T.H. Meng, "Minimum Energy Mobile Wireless Networks", *IEEE Journal on Selected Areas in Communications,* vol. 17, pp. 1333-1344, August 1999.

[SCI+01] E. Shih, S.H. Cho, N. Ickes, R. Min, A. Sinha, A. Wang, A. Chandrakasan, "Physical Layer Driven Protocol and Algorithm Design for Energy-Efficient Wireless Sensor Networks", In *Proceedings of MOBICOM 2001,* Rome Italy, July 2001.

[Sim01] Simple Power, http://www.cse.psu.edu/~mdl/SimplePower.html (October 15, 2001)

[SK97] Mark Stemm and Randy H. Katz, "Measuring and reducing energy consumption of network interfaces in hand-held devices", *IEICE Transactions on Communications, special Issue on Mobile Computing,* vol. E80-B, no. 8, pp. 1125-31, 1997.

[Spec99] Wireless LAN Medium Access Control and Physical layer Specifications, August 1999. IEEE 802.11 Standard (IEEE Computer Society LAN MAN Standard Committee).

[SR98] S. Singh and C. Raghavendra. PAMAS, "Power aware multi-access protocol with signalling for ad hoc networks", In *ACM Computer Communications Review*, 1998.

[Ste94] W. Richard Stevens, "TCP/IP Illustrated, Volume I: The Protocols", Addison Wesley Publishers, 1994.

[SWR98] Suresh Singh, Mike Woo, and C. S. Raghavendra, "Power-aware routing in mobile ad doc networks", in *Proceddings of MOBICOM*, October, 1998.

[TBGP00] V. Tsaoussidis, H. Badr, X. Ge, K. Pentikousis, "Energy/Throughput Tradeoffs of TCP Error Control Strategies", in *Proceedings of the 5th IEEE Symposium on Computers and Communications,* France, July 2000.

[Tra02] Tcptrace http://jarok.cs.ohiou.edu/software/tcptrace/tcptrace.html, October 2002.

[WE93] N. H. E. Weste and K. Eshranghian, "Principles of CMOS VLSI Design: A System Perspective", Addison Wesley Publishers, 1993

[Wei93] M. Weiser, "Some computer science issues in ubiquitous computing", *Communication of the ACM,* 36:74-83, July 1993.

[WESW98] Hagen Woesner, Jean-Pierre Ebert, Morten Schlager, and Adam Wolisz , "Power-saving mechanisms in emerging standards for wireless LAN's: The MAC level perspective", *IEEE Personal Communications*, June, 1998.

[WLBW01] R. Wattenhofer, L. Li, P. Bahl and Y. M. Wang, "Distributed Topology Control for Power Efficient Operation in Multihop Wireless Ad Hoc Networks", In *Proceedings of IEEE INFOCOM,* Anchorage, Alaska, 2001.

[ZP99] Jim Zyren and A. I. Petrick, "Brief Tutorial on IEEE 802.11 Wireless LANs", February 1999, www.intersil.com/data/an/an9/an9829/ an9829.pdf.

[ZR97] M. Zorzi, R.R. Rao, "Error Control and Energy Consumption in Communications for Nomadic Computing", *IEEE Transactions on Computers*, March 1997.

[ZR99] M. Zorzi, R.R. Rao, "Is TCP Energy Efficient?", in Proceedings of *IEEE MoMuC,* November 1999.

[ZRM02] M. Zorzi, M. Rossi and G. Mazzini, "Throughput and energy performance of tcp on a wideband cdma air interface", In *Journal of Wireless Communications and Mobile Computing*, Wiley 2002, 200.