AN ABSTRACT OF THE THESIS OF

Sandeep Agarwal for the degree of Master of Science in Electrical and Computer Engineering presented on April18, 2000. Title: Conserving Energy in TCP for Mobile Ad-Hoc Networks.

Redacted for Privacy

Abstract Approved:___ _____

Suresh P. Singh

The widespread use of TCP as a transport layer protocol for mobile ad-hoc networks and wireline networks has motivated the need to make its implementation highly power efficient specially with respect to ad hoc radio networks. Over the past years many researchers have developed energy efficient protocols for mobile ad-hoc networks. This thesis deals with the various modifications and fine-tunings in the TCP code which, when applied, help in conserving battery power at nodes by saving on the software overhead at the mobile nodes. The various modifications proposed have been tested with actual experiments done on two laptops with Lucent WaveLan wireless cards. The results obtained from the experiments indicate that with certain modifications made in the implementation of TCP code, significant savings in power can be achieved along with an increase in the overall efficiency of TCP over wireless links. Finally, a discussion of how other modifications can be researched and tested as the hardware for mobile systems change is proposed.

Conserving Energy in TCP for Mobile Ad-Hoc Networks

by

Sandeep Agarwal

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented April 18, 2000
Commencement June 2000

Master of Science thesis of Sandeep Agarwal presented on April18, 2000

APPROVED:

Redacted for Privacy

Major Professor, representing Electrical and Computer Engineering

Redacted for Privacy

Chair of Department of Electrical and Computer Engineering

Redacted for Privacy

Dean of Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature authorizes release of my thesis to any reader upon request.

Redacted for Privacy

Sandeep Agarwal, Author

Acknowledgements

I wish to thank my major professor, Dr. Suresh P. Singh, for his help and guidance in developing the ideas in this thesis. I also wish to thank the other members of my committee for giving so generously of their time. Finally I would like to thank the members of the faculty of Electrical and Computer Engineering for their help and support during the duration of my research.

TABLE OF CONTENTS

TABLE OF CONTENTS(Continued)

LIST OF FIGURES

LIST OF ABBREVIATIONS

(TCP)	Transmission Control Protocol
(IP)	Internet Protocol
(MAC)	Medium Access Control
(CSMA/CA)	Carrier Sense Multiple Access with Collision Avoidance
(RF)	Radio Frequency
(CPU)	Central processing Unit
(I/O)	Input/Output
(ACK)	Acknowledgement
(BER)	Bit Error Rate
(T/TCP)	Transactions/Transmission Control Protocol
(RTT)	Round Trip time
(SPT)	Server Processing Time
(UDP)	User Datagram Protocol
(FIFO)	First In First Out
(RISC)	Reduced Instruction Set Computer
(CISC)	Complex instruction Set Computer
(LAN)	Local Area Network
(LFN)	Long Fat Network
(WAN)	Wide Area Network
(RTO)	Retransmission Time Out
(MSL)	Maximum Segment Lifetime
(PAWS)	Protection Against Wrapped Sequence Numbers
(SYN)	Synchronize Sequence Numbers flag
(SACK)	Selective Acknowledgement
(MTU)	Message Transmission Unit
(MSS)	Maximum Segment Size

LIST OF ABBREVIATIONS (Continued)

(FDDI)	Fiber Distributed Data Interface
(IEEE)	Institute of Electrical and Electronics Engineers
(TTL)	Time To Live
(FIN)	Finish flag
(RAM)	Random Access Memory
(PCMCIA)	Personal Computer Memory Card International Association
(FCC)	Federal Communications Commission
(RTS)	Request To Send
(CTS)	Clear To Send
(LCD)	Liquid Chromium Display
(WEP)	Wired Equivalent Privacy
(PHY)	Physical
(DSSS)	Direct Sequence Spread Spectrum
(FHSS)	Frequency Hopping Spread Sequence
(GHz)	Giga-Hertz
(AP)	Access Point
(ELFN)	Explicit Link Failure Notification
(TCP-F)	Transmission Control Protocol - Feedback
(BSS)	Basic Service Set
(ESS)	Extended Service Set
(ESSID)	Extended Service Set Identifier
(ISM)	Industrial, Scientific and Medical

Conserving Energy in TCP for Mobile Ad-Hoc Networks

1. INTRODUCTION

As we continue to use computers to enhance life in our society, the need to network computers for data sharing and access to the Internet becomes increasingly important. Various different network topologies have emerged over the past years and during the recent years the wireless local area network has gained increasing importance. At the physical layer this network uses radio frequency (RF) to communicate. At the routing layer IP is still predominantly used and at the transport layer it is mainly TCP that is used.

Ad hoc networks are multi-hop wireless networks where all the nodes cooperatively maintain network connectivity. This means that, all the mobile nodes in the networked area act as routers and transmit data packets received from a sender on to the receiver. The range of the networked area is effectively increased as compared to the transmitting range of a single radio. These types of networks are useful in any situation where temporary network connectivity is needed at short notice, and no suitable infrastructure for wireline networks is available (or can be established at such short notice).

For instance, consider the problem of establishing a temporary wireless network in a huge region where an emergency search operation is on. An ad hoc network here would enable the rescue/search agents in the field to retrieve maps and weather forecasts from the Internet (assuming that one or more of the nodes of the ad hoc network are connected to the Internet). It would allow them to exchange photographs and similar data with other team members who would also be using laptops. Other examples of such ad hoc networks include internetworking

participants in a meeting hall or building to enable them to exchange data, battlesite networks, etc.

Nodes in an ad hoc network need to remain on battery power for extended periods of time. These nodes need to be energy-conserving so that battery life is maximized. Battery life imposes a severe constraint on the deployment and large scale use of mobile computing technology in the future, and has prompted several researchers to develop approaches for conserving power on mobile computers. Several technologies are being developed to achieve these goals by targeting specific components and optimizing their energy consumption. A significant amount of power is consumed by the display, by spinning disks, by the CPU, by I/O devices and by the transceiver radio. Hence the motivation for low-power displays [13], algorithms to reduce power consumption of disk drives [14,15,16], and low power I/O devices [17]. These, along with the development of low-power CPUs (such as those used in laptops and other hand-held devices) and high capacity batteries have all contributed to overall energy savings in the mobile nodes in ad hoc networks.

Recently some researchers have begun studying the problem of reducing power consumption during wireless data communication. Reducing power consumption during file transfers between transmitting and receiving nodes is clearly an important goal because battery life is not expected to increase significantly in the coming years. In an ad hoc network, it is even more important to reduce power consumption because these networks are typically established in mission critical environments (such as disaster relief).

At the transport layer of the communication stack the TCP protocol is most widely used. The implementation of this protocol can be fine-tuned to give significant improvements in power consumption at the mobile nodes.

Since the MAC protocol used for wireless communications is IEEE 802.11, a brief explanation of the protocol is also provided.

1.1 Overview of TCP

TCP provides a connection oriented, reliable, byte stream service. The application data is broken into what TCP considers the best-sized segments to send. When TCP sends a segment it maintains a timer, waiting for the receiving end to acknowledge reception of the segment. If the acknowledgement isn't received in time, the segment is retransmitted. To determine when to retransmit a segment, TCP dynamically estimates the round-trip time by measuring the time it has taken for earlier segments to be acknowledged. Segments are uniquely numbered to identify them. When TCP receives data from the other end of the connection, it sends an acknowledgement. This acknowledgement is not sent immediately, but is normally delayed for a fraction of a second. TCP maintains a checksum on its header and data. This is an end to end checksum whose purpose is to detect any modification of the data in transit. If a segment arrives with an invalid checksum, TCP discards it and doesn't acknowledge receiving it. It expects the sender to time out and retransmit. TCP segments can arrive out of order since it does not assume reliability from its underlying layers. A receiving TCP re-sequences the data if necessary, passing the received data in the correct order to the application. A receiving TCP discards duplicate data. TCP also provides flow control. Each end of a TCP connection has a finite amount of buffer space. A receiving TCP allows the other end to only send as much data as the receiver has buffers for. This prevents a fast host from taking all the buffers on a slower host. There is a limit, known as the window size on the number of unacknowledged segments that may be outstanding at any time, to bound the amount of buffering of unacknowledged segments that must be done at the sender and receiver.

1.2 Overview of the MAC Layer Protocol - 802.11b

Two widespread standards today underpin much of the commercial 2.4 GHz wireless LAN market. They are the IEEE 802.11 standard and the OpenAir 2.4 standard. The IEEE 802.11 specification is a wireless LAN standard developed by the IEEE (Institute of Electrical and Electronic Engineering) committee in order to specify an "over the air" interface between a wireless client and a base station or Access Point, as well as among wireless clients. First conceived back in 1990, the standard has evolved from various draft versions (Drafts 1 through 6), with approval of the final draft on June 26, 1997.

Like the IEEE 802.3 Ethernet and 802.5 Token Ring standards, the IEEE 802.11 specification addresses both the Physical (PHY) and Media Access Control (MAC) layers. At the PHY layer, IEEE 802.11 defines three physical characteristics for wireless local area networks: diffused infrared, direct sequence spread spectrum (DSSS), and frequency hopping spread spectrum (FHSS). While the infrared PHY operates at the base-band, the other two radio-based PHYs operate at the 2.4 GHz band. This latter frequency band is part of what is known to be the ISM band, a global band primarily set aside for industrial, scientific and medical use, but can be used for operating wireless LAN devices without the need for end-user licenses. In order for wireless devices to be interoperable they have to conform to the same PHY standard. All three PHYs specify support for 1, 2 and 11 Mbps data rate.

The 802.11 MAC layer, supported by an underlying PHY layer, is concerned primarily with the rules for accessing the wireless medium. Two network architectures are defined: the Infrastructure Network and the Ad Hoc Network. An Infrastructure Network is an architecture for providing communication between wireless clients and wired network resources. The transition of data from the wireless to the wired medium is via an Access Point. The coverage area is defined by an Access Point (AP) and its associated wireless clients, and together all the devices form a Basic Service Set.

An Ad Hoc network is an architecture that is used to support mutual communication among wireless clients. Typically created spontaneously, an ad hoc network does not support access to wired networks, and does not need an AP to be part of the network.

The primary services provided by the MAC layer are as follows:

- **Data transfer**

  Wireless clients use a Collision Sense Multiple Access with Collision Avoidance (CSMA/CA) algorithm as the media access scheme.

- **Association**

  This service enables the establishment of wireless links between wireless clients and APs in Infrastructure Networks.

- **Re-association**

  This takes place in addition to association when a wireless client moves from one Basic Service Set (BSS) to another. Two adjoining Basic Service Sets form an Extended Service Set (ESS) if they are defined by a common ESSID.

  If a common ESSID is defined, a wireless client can roam from one area to another. Although re-association is specified in 802.11, the mechanism that allows AP-to-AP coordination to handle roaming is not specified.

- **Authentication**

  Authentication is the process of proving a client identity, and in IEEE 802.11, this process takes place prior to a wireless client associating with an AP. By default, IEEE 802.11 devices operate in an Open System, where essentially any wireless client can associate with an AP without the checking of credentials. True authentication is possible with the use of the 802.11 option known as Wired Equivalent Privacy or WEP, where a Shared Key is configured into the AP and its wireless clients. Only those devices with a valid Shared Key will be allowed to be associated to the AP.

- **Privacy**

  By default, data is transferred "in the clear"; any 802.11-compliant device

can potentially eavesdrop PHY 802.11 traffic that is within range. The WEP option encrypts data before it is sent wirelessly, using a 40-bit encryption algorithm known as RC4. The same Shared Key used in authentication is used to encrypt or decrypt the data; thus only wireless clients with the exact Shared Key can correctly decipher the data.

- **Power management**

  IEEE 802.11 defines two power modes, an Active Mode, where a wireless client is powered to transmit and receive, and Power Save mode, where a client is not able to transmit or receive, but consumes less power. Actual power consumption is not defined and is dependent upon the implementation.

Standardization and interoperability among devices utilizing the same PHY is the intent of the IEEE 802.11 specification. (At the physical level, the three modulation schemes are incompatible with each other, so an infrared wireless client will not synchronize to a DSSS Access Point, for example). However, even among devices with the same PHY, a few key ingredients necessary to achieve multi-vendor interoperability are absent in the ratified standard.

1. **AP-to-AP coordination for roaming**

   The standard does not specify the handoff mechanism to allow clients to roam from one AP to another.

2. **Data frame mapping**

   The standard does not state how an Access Point addresses data framing between the wired and the wireless media.

3. **Conformance test suite**

   There is no conformance test suite specified to verify that a device is compliant with the IEEE 802.11 specification.

2. RELATED WORK

The design of efficient TCP for error-prone wireless links has received a lot of attention by many researchers – but most of the solutions that have been proposed deal with the alleviation of the poor end-to-end performance shown by unmodified TCP implementation. The aim in this thesis is to fine-tune the implementation of TCP (modified or unmodified) so as to get power efficiency at the nodes. This can be achieved by reducing the TCP protocol processing (software overhead) required at the nodes with no loss in the overall end-to-end performance. Along with the proposed fine-tunings in this thesis, the other methods suggested by various authors should be employed to conserve energy in transporting data. In fact the proposed fine tunings in this thesis work best only when the other features of previous research are also incorporated into TCP.

The research work in this field deals with modifying TCP to make it a reliable protocol which can differentiate between motion-related and congestion-related packet losses, and suggest how to adapt these protocols to perform better in mobile cellular environments. A clear distinction has to be made between wireless cellular networks and wireless ad hoc networks. Wireless cellular networks have the problem of handoffs. Handoffs can cause excessive delay if they occur during a TCP connection. This affects the throughput adversely. The handoff problem is non-existent in mobile ad-hoc networks. So the problem of having excessive delays due to handoffs during a TCP connection is not present in an ad-hoc environment. However, in ad hoc networks there is a problem of link failures due to mobility. This problem is addressed in [24, 28, 34] and the suggested methods of ELFN, TCP-F, etc. can be used to overcome it.

The types of solutions suggested up to now for improving the TCP throughput for mobile cellular networks basically fall into three categories:

a) end-to-end protocols, where loss recovery is performed by the sender

b) link-layer protocols, that provide local reliability by retransmissions
c) split-connection protocols, that break the end-to-end connection into two parts at the base station.

Some of the solutions that have been developed by various researchers for wireless cellular networks are discussed below briefly.

[1] explores the performance of reliable data communications in mobile computing environments. Motion across wireless cell boundaries causes increased delays and packet losses while the network learns how to route data to a host's new location. TCP interprets these delays and losses as signs of network congestion. It consequently throttles its transmission, further degrading performance. In this paper the authors propose an end-to-end fast retransmission scheme that can reduce these pauses. The fast retransmission is done at the TCP layer. The need to differentiate between motion-related and congestion-related packet losses is made clear in this paper.

[2] proposes Indirect TCP for mobile hosts which can tackle mobility and wireless related performance problems without compromising backward compatibility with TCP used over the wired network. Indirect TCP utilizes the support of Mobility Support Routers to provide transport layer communication between mobile hosts and those on the fixed network. In this solution the connection is broken into two logical connections – one over the wired part and the other over the wireless part. Loss over the wired part is treated as congestion related loss whereas loss over the wireless part is treated as motion related loss. However TCP semantics are not maintained.

[3] discusses the problems that frequently plague mobile networks such as – high bit error rate (BER), frequent disconnections of the mobile user, and low wireless bandwidth that may change dynamically. The authors propose a protocol that addresses this problem in TCP and increases its throughput performance while maintaining end-to-end TCP semantics.

[26] proposes a solution which does not break the semantics of TCP. This solution works by making several modifications to the network layer code in the base station. A snooping agent is added that observes and caches TCP segments going out to the mobile host, and ACKs coming back from it. This snooping agent does local retransmissions to the mobile host and also suppresses duplicate ACKs being sent to the TCP sender. However, if the wireless link is very lossy, the TCP sender may time out waiting for an ACK, and invoke the congestion control algorithm. Thus it is advisable to have a conservative value for the RTO.

In [25] the authors conclude that a Link Layer scheme that is TCP aware gives the best performance results. In this thesis we see that IEEE 802.11 MAC layer is a reliable link layer scheme which provides good performance results. These results can be further improved with some modifications in the TCP layer. However, 802.11 is not TCP aware and hence some modifications can be done to make it TCP aware. The snoop protocol [25, 26] is the best suited for this job. The goal in this thesis is to assume reliable performance from the Link Layer and then modify the TCP layer to make it perform better over such a Link Layer. In theory TCP should be independent of the technology of the underlying layer – but in practice it does matter what layer TCP is operating over. The performance of TCP greatly depends on the underlying layer and ignoring this can lead to a TCP implementation that is logically correct but has horrendous performance. Thus the modifications suggested in this thesis work best with a Link Layer protocol which is similar to the snoop protocol or is at least a reliable, TCP aware protocol. Due to certain timer interactions between the Link Layer and the TCP layer, the TCP sender is not fully shielded from the wireless losses. The fast retransmissions by the TCP sender (due to duplicate ACKs from the receiver), in spite of the Link Layer retransmissions also add to this problem. This causes the performance of TCP to diminish – hence it is advisable to have a Link Layer that is reliable, free from timer interactions with the TCP layer and also TCP aware (it should suppress duplicate ACKs). This will ensure that competing and redundant retransmissions are avoided. As described in [29], a reliable Link Layer with a "mild" backoff

strategy would be helpful to prevent capture of the channel by certain nodes which occurs during bulk data transfers. Some suggestions for queue scheduling, per queue transmission scheduling and congestion control within the MAC layer are also made in [29]. For all the above categories of solutions we can incorporate the fine-tunings suggested in this thesis to increase the power efficiency at the nodes. As an extension to the work in this thesis the IEEE 802.11 MAC layer can be modified to make it TCP aware, suppress duplicate ACKs, have a mild backoff strategy, and have per queue transmission scheduling and congestion control. This will improve the performance of TCP greatly, and at the same time reduce the software overhead at the nodes, thus increasing their longevity.

Another characteristic of wireless links is that the latency and bandwidth is variable [27, 30] and this causes certain problems in the exact calculation of the round trip time. Using TCP Timestamps option is not very helpful since the RTT can vary on a per packet basis hence it is always better to use the previous more conservative approach, *i.e. srtt + 4*mdev*. Other solutions to this problem are proposed in [27]. The significance of the RTT at the TCP layer is anyway reduced if we have a reliable Link Layer as described above. In [27] it is mentioned that due to asymmetry in the characteristics of the channel it is sometimes better to decrease the frequency of ACKs from the receiver to the sender. This idea can be exploited and we can have a decreased frequency of ACKs during the entire connection, so as to save protocol processing at the nodes and thus save energy. The side effects of this Stretch ACK Violation [32] phenomenon are that the sender becomes burstier, there is a slowdown in the window growth, a decrease in the effectiveness of the fast retransmit algorithm, and it may cause needless retransmission timeouts in lossy environments, as it increases the possibility that an entire window of ACKs is lost. However, the sender becoming burstier is not a big problem if we have a Link Layer protocol as described above and in [29]. A slow down in the window growth is not a problem for slow wireless links and is only a problem for high-speed links [32] - a simple solution for this is proposed in [27]. The other problems of the fast retransmit algorithm losing its effectiveness and increase in the number of

retransmissions are also not pronounced when we have a reliable Link Layer as described above. It should be noted that even TCP ACKs are transmitted reliably by the Link Layer.

Another line of argument that some researchers have proposed to reduce the protocol over head of TCP is to do away with TCP altogether. A much simpler and liberal type of protocol is instead employed which is called Transaction TCP. T/TCP has the best parts of both TCP and UDP combined. It provides sufficient reliability with minimum protocol overhead for data transactions. With the availability of T/TCP the choice of an application designer is not restricted to TCP or UDP only.

The main ideas behind the motivation of T/TCP are:

1. The overhead of connection establishment and connection termination should be avoided. When possible, send one request packet and receive one reply packet.
2. The latency should be reduced to RTT plus SPT, where RTT is the round trip time and SPT is the server processing time to handle the request.
3. The server should detect duplicate requests and not replay the transaction when a duplicate request arrives. (Avoiding the replay means the server does not process the request again. The server sends back the saved reply corresponding to that request.)

Today the choice an application designer has is either TCP or UDP. TCP provides too many features for transactions, and UDP doesn't provide enough. As a result, usually the application is built using UDP (to avoid the overhead of TCP connections) but many of the desirable features (dynamic timeout and retransmission, congestion avoidance, etc) are placed into the application, where they are reinvented over and over again. A better solution is to provide a transport layer that provides efficient handling of transactions. The definition of T/TCP is described in detail in [4] and [5].

However, it must be kept in mind that many applications still require TCP, which is widely used, and hence the motivation to reduce the processing overhead of TCP so as to conserve power at the nodes. The TCP protocol processing at the nodes is highly power consuming; this area needs to be researched so as to come up with very efficient TCP implementations for the existing and future mobile hardware. A part of the power consumed in data transport is the power consumed by the transmitter or receiver, but a lot of power is also consumed in processing the code of the TCP stack at the nodes. This can be fine-tuned for better power performance with no significant loss in the overall performance efficiency of end-to-end TCP. While trying to fine-tune TCP it must be kept in mind that it has proved quite hard to find parts of TCP that could be eliminated without compromising the protocol's capabilities.

3. TECHNIQUES FOR GOING FAST

A lot of research has been done in the area of improving TCP protocol implementations, and a number of techniques have been developed as a result. Some researchers have seriously investigated the various performance limitations of TCP and come up with better implementations of the protocol code. Some of the techniques that have been already implemented are reviewed below.

3.1 Operating Systems Overhead

Operating systems have to undergo a huge amount of overhead for a single context switch, since the whole state of the previous context has to be saved before the context can be switched. It would be in the best interests of energy and throughput efficiency to reduce the number of context switches. One way of doing this is to minimize the number of interrupts thrown at the operating system – since an interrupt always requires a context switch from the current context to the context of the interrupt service routine. This can be done by suppressing transmission interrupts and by receiving multiple packets from the receive-FIFO in response to a single interrupt. A certain number of packets can be coalesced at the receiver (interface card on the receiving node) before interrupting the operating system. This would, however, result in some throughput loss; but when implemented properly, this throughput loss can be negligible.

Memory management - given the disparity in memory and processor speeds (this huge gap is projected to widen even further in the future), copying data from one piece of memory to another is one of the slowest operations a processor can be asked to do. Clearly, minimizing the number of copies as data is passed up/down the stack is a power saving feature. In fact it is suggested that transmitting and receiving data should consist of no more than a single copy. Designing the stack in

such a way so as to have a single copy only as the data moves through the stack is the main aim – this will greatly help in reducing the processing overhead, and thus help in power saving.

3.2 Better Table Lookup Techniques

The TCP protocol architecture is such that there are several cases where a piece of information has to be looked up in a table. For instance TCP must find the connection block for each segment received. In the general case, each of these lookups has a worst case cost of $O(\log_k n)$, where n is the number of protocol blocks in the table, and k is some base indicating the fraction of the blocks that can be eliminated on average by each comparison. Lookups represent a very large fraction of the cost of protocol processing, and finding ways to minimize lookup costs is important to increase performance.

Two obvious ways to try to reduce lookup costs are:

a) use caches of frequently used information to minimize the number of expensive lookups
b) find lookup algorithms with very good average running times.

An effective and efficient cache is one in which the hit rate is maximized while the costs of searching and maintaining the cache is minimized. It is very fortunate that computer data networks exhibit precisely the kind of traffic patterns that are likely to make caches effective, and studies strongly suggest that caches of just one control block may actually achieve very high hit rates. It has been shown in [6, 7] that one-back (having only one entry) caches have reported significant cache hit rates and performance improvements. It has been shown in [8] that a cache consisting of 20 route entries is likely to yield a 90% hit rate. To get hit rates beyond this the cache size has to be greatly increased and is not advisable. A 90%

hit rate means that out of ten table lookups only one of them would effectively be very expensive, since only one would result in a cache miss.

It is described in [9] that the most effective table lookup scheme is hashing using open chaining, where the head of each hashed link list keeps a cache of the last accessed block. Hence if the hashing function is good we will get a very good algorithm for table-lookup supplemented with the benefits of caching.

3.3 Reducing Checksum Costs

The first and foremost step to optimize a checksum algorithm is to try to do the sum using the hosts machine's native word size (to optimize memory accesses) and native byte order (to minimize byte swapping costs). The TCP checksum is a sixteen-bit one's complement sum over the whole segment (data and header - with odd lengths padded by a zero byte). This sum can be done independent of byte order as shown below.

Consider the sequence of hex bytes:

Ox50, Ox51, Ox52, Ox53, Ox54, Ox55

which are added as sixteen-bit words into a sixteen-bit sum:

Ox5051 + Ox5253 + Ox5455 = Ox575A

where + is one's complement addition.

Now compare this result with the sum when the bytes are reversed:

Ox5150 + Ox5352 + Ox5554 = Ox5A57

The sums are the same except that their bytes are reversed. To see why this is always true, note that the carries are the same in both the cases: from bit 15 to bit 0, and from bit 7 to bit 8 (recall that one's complement addition requires that carries be added back into the lowest significant bit). This means that the checksum calculation can be done in any byte order. It is best to do the checksum calculation

as the bytes are stored in memory, i.e. Big Endian or Little Endian – this saves us the cost of unnecessary byte swapping that would be otherwise required.

The TCP checksum can be done using any word size of sixteen bits or greater, depending on the host machine's native word size. For example, consider summing 32-bit quantities. One can simply add the 32-bit numbers using one's complement addition and, when the 32-bit sum has been computed, fold and add the high 16-bits of the 32-bit sum to the low 16-bits and get the 16-bit sum. This saves time wasted in doing 16 bit sums whereas most processors nowadays are capable of doing 32-bit arithmetic operations. If we have 64-bit processors the same approach can be applied, except that in this case, first the 64-bit sum has to be folded to 32-bit and then the 32-bit sum has to be folded to give the 16-bit sum.

Another optimization for checksum calculation is also possible. RISC (nowadays even CISC) processors have a super pipelined architecture. This pipeline is stalled after a memory read or memory write operation due to the nature of the pipeline. After a memory read or write operation there is a loss of one or two (one for memory write and two for memory read) clock cycles during which no new memory accessing instructions can be processed.

As a result, in a copy loop of instructions of the following form:

load [r0], r2 ; load the contents of the memory location pointed to by r0 to r2
store r2, [r1] ; store the contents of r2 into memory location pointed to by r1

there is a space in between the two memory-accessing instructions for two non memory accessing instructions. We can put the checksum instructions (non memory-accessing) into the slots after these instructions to avoid the pipeline stalls:

load [r0], r2 ; load the contents of the memory location pointed to by r0 to r2
add r5, r2, r5 ; add to running checksum in r5
addc r5, #0, r5 ; add carry into r5
store r2, [r1] ; store the contents of r2 into memory location pointed to by r1

Because these two slots would otherwise be unused (unless we have a very smart compiler, which can put other non memory-accessing instructions there and still maintain the logical program flow), this effectively means that performing the checksum comes for free. It makes sense to replace data copy from user space to interface buffers with a combined checksum and copy. As the data is copied the checksum of the data is also calculated at no extra computational cost.

Other suggestions for improving checksum costs are to leave them out completely (at least for LAN traffic), but this suggestion has its obvious drawbacks, which TCP cannot afford to have due to its very characteristics. A second suggestion is to move checksums to the end of the packet, a practice known as trailing checksums or trailers. The advantage of doing this is that, if the checksum is at the end, the sending machine can start sending the packet before the checksum computation is finished. If the checksum is at the start of the packet, the sender cannot release the packet until the checksum has been computed and put in the header. However, trailers have one main disadvantage - they require that the delivery of data to the sending interface be predictable. If the operating system is somehow interrupted as it is passing data to the interface, and the interface is already putting data onto the network, fragmented packets may result. We know for a fact that fragmented packets will cause more software overhead.

3.4 Header Prediction

TCP behavior is highly predictable, and one can take advantage of this predictability by optimizing the frequent path through the TCP code in both the sending and receiving implementations.

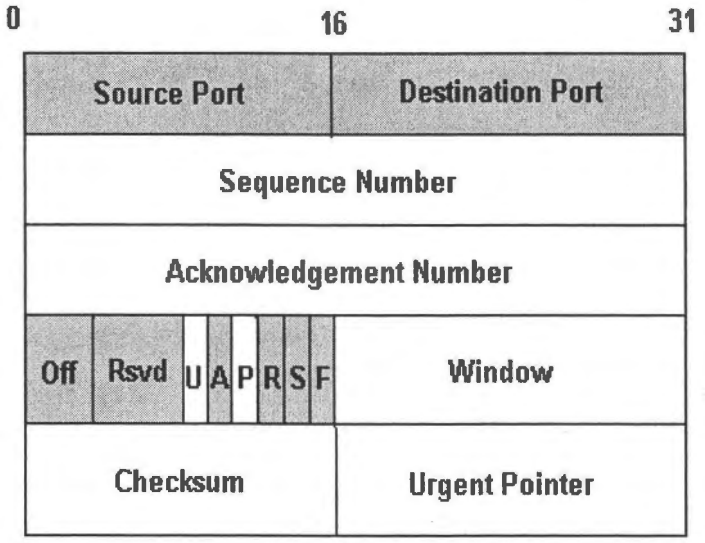

Figure 3.1: TCP Header Fields that Change in an Established Connection

Figure 3.1 shows the TCP header with the fields that do not change shaded. The source and destination ports are set at connection setup, and because TCP connections either always use or never use options, the data offset (Off) remains constant, as do most of the control bits.

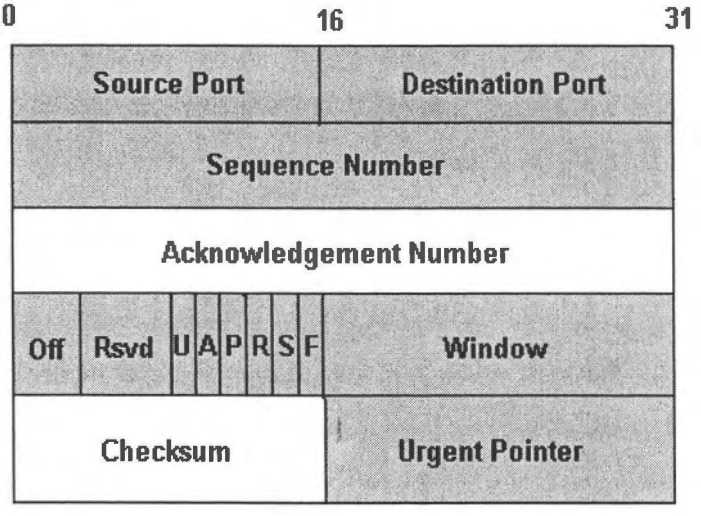

Figure 3.2: TCP Header Fields that Change Unpredictably

Figure 3.2 shows the TCP header with fields that change unpredictably not shaded. In situations in which no segments get lost or re-ordered (which is most of the time), the sequence number changes by the amount of data in the last segment received.

New Sequence Number = Last Segment Sequence Number + Amount of data (bytes) in the Last Segment that was received.

The window size typically does not change, given that the receiving TCP passes the data to the application and is immediately ready to receive new data. The urgent pointer is only used if the urgent bit (U) is on, and it usually is not, and the PUSH bit (P) can be ignored if the receiver passes data up to the application promptly.

These observations led Jacobson in [10] to develop an algorithm for TCP receivers called header prediction. Header prediction looks for segments that fit the profile of the segment the receiver expects to receive next; namely segments that

a) are for connections that have been established,
b) have only the acknowledgement bit (A) and optionally the push bit (P) set,
c) are the expected next segment in the sequence (i.e. the data in this segment starts where the last segment left off),
d) have not changed the window size,
e) are for connections that are not re-transmitting data (no error or segment loss for this segment)
f) are either ACKs for data or new data arriving, but not both (unidirectional flow).

Once the control block has been located (can be done efficiently if caches are used and a good hashing algorithm is used), these above tests require just five simple comparisons. A data packet that meets all the conditions (and most will if there are no errors or loss) then requires very few instructions and can be passed up

to the application. Thus the incremental cost of receiving a TCP segment, after connection lookup and performing the checksum, is very small. Header prediction is an algorithm for the receiving TCP but similar prediction schemes work for optimizing sending also.

On the sending side, an application typically writes its data to some sort of a connection handle, a file descriptor or socket. This connection handle can be designed to map directly to a control block (thus eliminating the control block lookup). Just as the incoming segment can be predicted, so the TCP header of the outgoing segment can be predicted. The sending TCP can keep a template TCP header, whose sequence number is incremented as segments are sent and whose acknowledgement number is updated as segments are received. As a result, sending becomes a matter largely of copying the template header onto the front of the TCP data, filling in the checksum (computed as the segment's data was copied), and sending the segment. However, there are certain difficulties in having header prediction for sending.

4. CURRENT IMPLEMENTATION

All the modifications discussed above have already been implemented in the current implementation of TCP code in Linux. Along with the above modifications currently TCP has certain options and extensions implemented which are only useful for long fat networks i.e. very high-speed gigabit networks. These options cause extra protocol processing both at the receiver and sender. This extra protocol processing is justified for long fat networks (in which the bandwidth-delay product is very high) but not for slower wireless networks. In the case of wireless ad-hoc networks which now have a maximum data speed of 11 Mb/s (and so a low delay-bandwidth product) this extra protocol processing is not justified.

We must keep in mind that for the bandwidth-delay product to be low, the bandwidth of the medium must be low, and the propagation delay for the medium must be low under normal (non – congestion) conditions. For LFN's we see that the bandwidth is high (gigabit) and the delay of the medium is dependent on the distance between the source and the destination (generally this is also very large since gigabit networks are mostly used for WANs). Whereas in ad-hoc networks the bandwidth is low (only 11Mbit/s), the propagation delay of the medium is low since the distance between the source and destination is generally not very large. We can comfortably say that the bandwidth-delay product of mobile ad-hoc networks is much lower than that of LFNs. Also it must be kept in mind that in mobile ad-hoc networks typically there are no handoffs like in cellular wireless networks that can cause long intermittent delays. Th delays caused due to link failures can be dealt with by the use of ELFN, TCP-F [24, 28, 34] etc.

With this knowledge we can say that the features and extensions which are helpful for LFNs might not be as helpful for mobile ad-hoc networks due to the significant characteristic difference in the delay-bandwidth product. With the constraint of power, the extra protocol processing resulting in no extra benefit in

efficiency or throughput, is a problem for wireless ad-hoc nodes. These various options, which are suitable only for gigabit networks and need to be modified for slow wireless networks, are discussed below.

4.1 Timestamp Option

TCP implements reliable data delivery by re-transmitting segments that are not acknowledged within some retransmission timeout (RTO) interval. Accurate dynamic determination of an appropriate RTO is essential to TCP performance. RTO is determined by estimating the mean and variance of the measured round-trip time (RTT), i.e., the time interval between sending a segment and receiving an acknowledgment for it [11]. Many TCP implementations base their RTT measurements upon a sample of only one packet per window. While this yields an adequate approximation to the RTT for small windows (used in mobile ad-hoc systems), it results in an unacceptably poor RTT estimate for LFN which have very large sized windows.

The timestamp option is one in which the sender uses 12 bytes of the TCP options field to place a timestamp in every segment sent (including retransmissions) to the receiver. The receiver echoes this timestamp value in the ACK packet sent to the sender. By using this timestamp option in every packet the sender is able to get a better value of the round trip time (RTT). The sender gets an estimate of the RTT from every ACK received from the receiver by subtracting the echoed timestamp value that it received from the receiver, from the current timestamp.

This option is actually very useful for high-speed connections but has less use for slower wireless connections. In wireless networks one sample of the RTT per window is good enough for an accurate estimate of the RTT. Getting more samples of the timestamp can in fact cause much more oscillations in the estimated value of the RTO. Also, the sender as well as the receiver have to do more

processing in order to attach the timestamp in every packet and to calculate the RTT after it receives the ACK for the particular segment. Hence this option only causes more protocol processing at the nodes with no significant gain in the accuracy of the RTO values calculated. With a reliable Link Layer the significance of a very accurate RTO is also diminished. Also, due to the variations in the propagation delay it is better to have a conservative estimate of the RTO. Removing this option for mobile wireless systems would give us a slight gain, due to the smaller header size and lower processing costs.

PAWS (Protection Against Wrapped Sequence Numbers) uses the same TCP Timestamps option as the RTT mechanism described above, and assumes that every received TCP segment (including data and ACK segments) contains a timestamp whose values are monotone non-decreasing in time. The basic idea is that a segment can be discarded as an old duplicate if it is received with a timestamp, which has a value that is less than the last previous timestamp received on this connection. This is again a very useful option for very high-speed networks. This feature is not required for slower wireless networks, since the wrap around of sequence numbers will not occur within the MSL (Maximum Segment Lifetime) due to the lower transmission speeds of wireless links. It is clear that the sequence number wrap around problem only occurs at gigabit speeds, and will never occur at slow wireless speeds. It should be mentioned here that having the PAWS checking is not much of a load on the nodes but still it would be better not to have them since the case would never occur anyway. However, PAWS checking is the first step in the frequent fast path of the TCP code, and it is best to get rid of it, as it has no purpose for wireless networks. By removing this check we can make the frequent fast path even more efficient.

4.2 Window Scale Option

The TCP header uses a 16-bit field to report the receive-window size to the sender. Therefore, the largest window that can be used is 2^{16} = 65K bytes. The window scale extension expands the definition of the TCP window to 32 bits and then uses a scale factor to carry this 32 - bit value in the 16-bit Window field of the TCP header. The scale factor is carried in a new TCP option, called Window Scale. This option is sent only in a SYN segment (a segment with the SYN bit on), hence the window scale is fixed in each direction when a connection is opened. Again it should be noted that very large windows are not necessary for slower wireless connections, hence this option is not required by wireless TCP. However, it does have a purpose in very high-speed connections as the window size will be a limitation in those connections, so larger window sizes might have to be negotiated in the SYN segment if allowed by both the receiver and the sender. Avoiding the window scale option would save us some protocol processing at the receiver and the sender (though this saving in protocol processing would be very little since a simple shifting operation is not very expensive).

4.3 SACK Option

Any packet losses in an LFN can have a catastrophic effect on throughput. This happens because the time taken for the sender to get feedback about the loss is very high due to the high bandwidth-delay product. This effect is exaggerated by the simple cumulative acknowledgment of TCP. Whenever a segment is lost, the transmitting TCP will eventually time out and retransmit the missing segment. However, the sending TCP has no information about segments (after the lost segment) that may have reached the receiver and been queued, because they were not at the left window edge. So the sender may be forced to retransmit these segments (after receiving three duplicate ACKs) unnecessarily unless a new

updated ACK is received, which would send one after receiving the lost packet. In the case of slower wireless connections this is not the case, since many segments after the lost segment would not have been transmitted to the receiver (because the propagation delay is not very high). Hence the number of needless retransmissions would be very small.

If the sender is bursty then the number of segments retransmitted obviously increases. TCP may experience poor performance when multiple packets are lost from one window of data. With the limited information available from cumulative acknowledgments, a TCP sender can only learn about a single lost packet per round trip time. An aggressive sender could choose to retransmit packets early (fast retransmit and fast recovery), but such retransmitted segments may have already been successfully received. A Selective Acknowledgment (SACK) mechanism, combined with a selective repeat retransmission policy, can help to overcome these limitations. The receiving TCP sends back SACK packets to the sender informing the sender of data that has been received. The sender can then retransmit only the missing data segments. This is not useful for wireless links in which the round trip time is not very high as compared to WANs which use high speed links.

If SACK is implemented there is a benefit of not re-transmitting needlessly at all, but the extra protocol processing involved in implementing SACK might overcome this small benefit gained. There is a certain trade off between the amount of power required by the sender to retransmit the packets and the power required to do the SACK protocol processing. SACK protocol processing power is significant; in fact we see in the results of our experiments that there is less power consumed when we turn off the SACK option. In the non–LFN regime, selective acknowledgements reduce the number of packets retransmitted (not a whole lot), but do not otherwise improve performance, making their complexity (and extra power consumed due to this complexity) of questionable value. SACKs are much more important in the LFN regime and are not very helpful in the power constrained slower wireless networks. SACKs are only useful if there are multiple packet losses in a single window.

It must be kept in mind that SACKs are only useful if the physical medium is highly error-prone and there is a general tendency of getting more than a single packet loss in a particular window. But since the wireless MAC protocol standard used is 802.11, which is more reliable as compared to Ethernet, the wireless link appears to be quite error-free to the TCP layer. Therefore we are able to justify not having the SACK option implemented to save on energy at the nodes.

4.4 Header Prediction

"Header prediction" [10] is a high-performance transport protocol implementation technique that is most important for high-speed links. This technique optimizes the code for the most common case, receiving a segment correctly and in order. Using header prediction, the receiver asks the question, "Is this segment the next in sequence?" This question can be answered in much fewer machine instructions than the question, "Is this segment within the window?" Adding header prediction to the timestamp procedure leads to the following sequence for processing an arriving TCP segment – this is also the implementation in Linux:

H1) Check timestamp: this means check to see if the packet is not a delayed packet by checking the timestamp value with the most recent timestamp value received earlier.

H2) Do header prediction: if the segment is next in sequence (checked by using the frequent path code which basically has about five comparisons) and if there are no special conditions requiring additional processing, accept the segment, record its timestamp, and skip H3.

H3) Process the segment normally: (this is the slow path, which would only be taken if there are errors etc). This includes dropping segments that are outside the

window and possibly sending acknowledgments, and queuing in-window, out-of-sequence segments.

In the above algorithm the modification that we can make would be to interchange steps H1 and H2, i.e., to perform the header prediction step H2 first, and perform H1 and H3 only when header prediction fails. This can be done because H2 basically also checks for H1 except for the case when the packet received is exactly the same packet (i.e. next in sequence with the same headers) but from the previous window. This could be a performance improvement, since the timestamp check in step H1 is very unlikely to fail, and it requires interval arithmetic on a finite field, which is a relatively expensive operation [10]. To perform this timestamp check on every single segment is contrary to the philosophy of header prediction and speeding up the frequent path.

However, putting H2 first would create a hazard: a segment from 2^{32} bytes in the past might arrive at exactly the wrong time and be accepted mistakenly by the header-prediction step. The following reasoning has been introduced in [12] to show that the probability of this failure is negligible. If all segments are equally likely to show up as old duplicates, then the probability of an old duplicate exactly matching the left window edge is the maximum segment size (MSS) divided by the size of the sequence space. This ratio must be less than 2^{-16}, since MSS must be less than 2^{16} (MTU is only 2296 bytes at the most for IEEE802.11b); for example, it will be

$(2^{11})/(2^{32}) = 2^{-21}$ for the 802.11 protocol.

However, the older a segment is, the less likely it is to be retained in the network (due to TTL – time to live, and it being rejected by some intermediate router). Under any reasonable model of segment lifetime the probability of an old duplicate arriving exactly at the left window edge must be much smaller than 2^{-16}. The 16-bit TCP checksum also allows a basic unreliability of one part in 2^{16}. A protocol

mechanism whose reliability exceeds the reliability of the TCP checksum should be considered "good enough", i.e., it won't contribute significantly to the overall error rate. From the above reasoning it can be concluded that we can ignore the problem of an old duplicate being accepted by doing header prediction (step H2) before checking for the timestamp (step H1).

But it has been argued that any reasoning based on a probability theory is not a good enough reason for removing the PAWS check. As the mobile hardware improves we can change the relatively expensive PAWS operation into not too expensive an operation and thus be able to have this check as it is. If we do not have the Timestamp option then we do not save anything further by modifying the header prediction algorithm – since the PAWS checking is anyway disabled. At the TCP sender the modification of the header prediction algorithm has very little effect.

4.5 Cumulative and Delayed ACK Implementation

Currently the Linux implementation of TCP is such that for every two full packets received the receiver sends a cumulative ACK (acknowledges both the packets) to the sender. This scheme is useful for very high - speed networks so as to get instant feedback (instant feedback is necessary because the delay-bandwidth product is very high) on the network status and also to get more samples of the RTT (due to the timestamp option). However this scheme does not make good use of the window size advertised by the receiver to the sender. Also it does not make good use of the delayed and cumulative acknowledgements allowed by TCP. The receiver sends an ACK for every two full segments received from the sender and this greatly increases the protocol processing at the receiver. The protocol processing at the sender also increases due to this, as it has to update its status based on these frequent (one in every two full segments) ACKs received from the receiver. If we change the implementation of TCP to instead send delayed ACKs

(say every 500ms) instead of sending ACKs for every two full segments received, then we would receive less number of ACKs for a file transfer (if the file is longer than 2 full packets). This would obviously result in some improvement in the power efficiency of the code. This implementation will cause us to get less samples of the RTT but that will not cause a problem as we expect our RTT estimation to be quite accurate. One may argue here that the response time for small sized file transfers would become longer but this is not the case as explained below. The response time for small sized file transfers will not be affected because the ACK will anyway be received in 500ms.

Let us see why this is the case with an example:

Suppose the server sends a small sized packet (say 500 bytes) and does not want to send any more data. The server will set the FIN flag in the packet header of the TCP data packet. This will cause the receiver to respond immediately (instead of waiting for 500ms) with an ACK.

We see that changing the implementation of the cumulative delayed ACK does not cause an effect in the response time of small data transfers, whereas, it does give us an improvement in the power efficiency of the protocol processing at the nodes. The power savings would be at the receiver and sender – the receiver would save by having to send fewer ACKs, and the sender would save by having to respond to fewer ACKs. The contention and collisions at the MAC layer would be less thus contributing to the power savings. TCP receivers, which implement this type of Delayed Acknowledgement (called Stretch ACK Violation as explained in [32]) behavior will cause TCP senders to generate burstier traffic, which can improve performance in non-congested environments. Generating fewer ACKs increases the amount of time needed by the slow start algorithm to open the congestion window to an appropriate point, which diminishes performance in environments with large bandwidth-delay products (not in wireless ad hoc networks). Finally, generating fewer ACKs may cause needless retransmission timeouts in lossy environments, as it increases the possibility that an entire window of ACKs is lost, forcing a retransmission timeout. The 802.11 MAC protocol takes

care of this and does not allow the number of retransmissions to increase excessively.

Another point to bear in mind is that nowadays, wireless cards with support for IEEE802.11 MAC protocol are widely available. This MAC protocol is more reliable as compared to Ethernet and has link layer acknowledgements and retransmissions. Hence the assumption of having an error prone wireless link is not entirely true from the point of view of the TCP layer. The TCP layer in fact can view the link to be quite error-free and so should be tuned accordingly to get better power and throughput performance.

4.6 MTU Size

As we know Ethernet supports a maximum MTU (message transfer unit) size of 1500 bytes. For wireless transmissions the MAC protocol used is the IEEE802.11b standard. The IEEE802.11b standard defines the maximum MTU size to be 2296 bytes and allows all protocols running above it to use this size as its MTU. Hence the IP layer above 802.11 MAC protocol will not fragment datagrams that it receives from the transport layer which are within 2296 bytes. So, the TCP layer can negotiate a MSS (maximum segment size) of 2296 bytes and send datagrams with a maximum size of 2296 bytes without the fear of having the IP layer fragment them and thus increase processing costs. It is obvious that for data transmissions, which are longer than one segment, it is better to send longer segments – this saves us protocol processing and also saves us the header overhead. This decreases the protocol processing at the nodes although it will increase the cost of retransmissions. The protocol processing required at the nodes for a single segment is the same no matter what the segment size. Hence for large file transmissions using longer segment size reduces the power consumed for the transmission. However, the number of retransmissions increases, since the

probability of a single segment being in error now increases. The probability of a single bit being erroneous in a 2296-byte segment is obviously higher than the probability of a single bit being erroneous in a 1500-byte segment. If we have a single bit error in a segment, that particular segment has to be retransmitted. Going by this logic we can say that the number of retransmissions per segment is going to increase. Also if we have smaller sized segments, then the power consumed in retransmissions would be less since the size of the retransmitted segments would be small. This causes us to limit the maximum size of the MTU accordingly so as to have good power efficiency along with good throughput, good channel utilization and at the same time not have very high retransmission costs. We see that in wireless ad-hoc networks using IEEE802.11b as a MAC protocol the number of errors at the TCP layer is low and so it makes more sense to use larger packets. TCP is designed to find out the path MTU for a certain connection and generally it is seen that the path MTU is fixed to 536 bytes for non-local transmissions. But in an ad-hoc network since all the systems would be using IEEE802.11b protocol we can always use 2296 bytes as MSS even for non-local transmissions. However, it should be noted that for connections to the Internet the MSS negotiated would be 1500 bytes because Ethernet supports a MTU of 1500.

Finally, we observe that most of the TCP options discussed above are important for LFN's and/or very high-speed networks. For low-speed wireless ad-hoc networks, which have a low delay-bandwidth product, it would be a performance optimization to NOT use these above-mentioned options. A TCP user concerned about optimal performance over low-speed wireless paths would consider turning these extensions and options off for low-speed wireless paths.

4. EXPERIMENTS AND VERIFICATION

We ran experiments to see the actual performance results of the above modifications made in the code implementation of TCP in Linux.

The experimental setup consisted of two Samsung SENS 800 laptops with Pentium-90Mhz processors and 24MB RAM. The operating system installed on the laptops was RedHat Linux v6.1. The wireless PCMCIA cards used were Lucent WaveLAN TURBO 11Mb SILVER with 64-bit encryption capability at the hardware level.

The power characteristics of the WaveLAN cards was the following:

Doze Mode	*10mA*
Receive Mode	*180mA*
Transmit Mode	*280mA*
Power Supply	*5V*

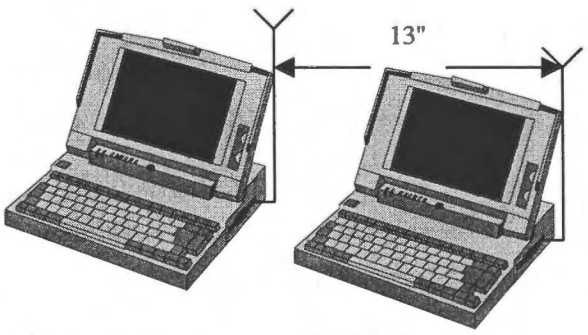

Figure 5.1: Experiment Setup

These wireless PC cards used the IEEE 802.11b standard as the MAC (CSMA/CA) protocol and the transmit range varied depending on the transmit speed. They supported 4 transmit speeds, namely 11Mb/s, 5.5Mb/s, 2Mb/s and 1Mb/s. The R-F Frequency band was 2.4GHz and the number of usable channels was 11 as specified by the FCC.

For these cards we used the driver developed by Andreas Neuhaus and the version of the driver [18] was WaveLAN/IEEE802.11 driver v1.0.3. The driver was available for Linux Kernel v2.x.x. This driver allowed us to modify the segment MTU size used in transmissions. In ad-hoc mode the driver allowed us to setup the speed of transmission and also the channel to be used. For our experiments we selected channel 1 (the default) and also set the speed to its maximum i.e. 11Mb/s. The hardware 64-bit encryption was turned off as we did not want any encryption. The RTS/CTS mechanism of the 802.11 protocol was turned off, as this was not required for our setup.

These two laptops were placed about one inch from each other and the distance between the two wireless cards (antenna's) was about 13 inches as shown in Figure 5.1. For all the experiments one of the laptops was used as a sender and the other as a receiver. The battery for the sender was always fully charged for 2 hours to full capacity with the laptop in off/charge mode. It should be noted that the charge of the battery would be generally restored to its full capacity in about 1.5 hours but the laptop would be kept in the same mode (off/charge) for another 0.5 hours. The receiver was constantly connected to a power source since we wanted to test the power characteristics of the transmitter.

For all experiments the setup of the laptops was changed so that no power saving feature would be on. All the various devices (disk, display, I/O devices etc) of the laptop would be in full power consumption mode as long as the battery lasted. Before the start of the experiments the monitor (LCD) display of the sender would be turned off and the experiment would be started by disconnecting the power connector from the sending node, and the sender would be operating on its fully charged battery. This was done because we wanted to use most of the battery

power for transmitting packets rather than executing other processes in the laptop. To verify that a significant amount of power was consumed for transmission we did a simple experiment. We first drained the battery with no processes running on the laptop. In this condition the sender (it wasn't sending any data) lasted for 223 minutes. Then we configured the sender to continuously send data and this time it lasted for 110 minutes. Thus we concluded that transmission did use up a significant amount of battery power hence any changes in the battery power consumption for transmission would be noticeable.

The actual experiment itself consisted of the sender transmitting a 1MB buffer of data continuously to the receiver. This data when received at the receiver would be discarded and the receiver would be ready immediately for the next packet. Hence the sender was in an endless loop sending data and the receiver would discard the data received and wait for the next segment of data (this ensures that the window size does not change).

Tcpdump with appropriate filter setting was run on the receiver to record the time the sender was alive (the sender's battery would discharge completely and the sender would die), and also the number of bytes the receiver had received from the sender. Each set of experiments was run 5 times and the average was recorded. The deviation among the various runs of the experiments was insignificant within experimental limits (not more than 1 minute for the lifetime of the sender). The readings of the lifetime of the sender were rounded off to the nearest minute.

Some facts that should be kept in mind about the experimental setup are the following:

a) We are using 802.11 as the MAC protocol, which provides link-level acknowledgements and retransmissions so we have a relatively error-free wireless link.
b) We are assuming roaming of the nodes but since it is an ad-hoc environment there are no disconnections and reconnections from base stations – hence no handoffs etc.

c) The experimental setup has only one hop, but a multi-hop ad-hoc network would behave similarly.

5.1 Varying the MTU Size

Four different sizes of MSS were chosen namely 2296, 1500, 1000 and 500 bytes. Figure 5.1a shows the results of the number of bytes transported during the lifetime of the sender. Figure 5.1b shows the number of minutes for which the sender was alive for the different MSS sizes. As we can see the number of bytes transported increases as the MSS size increases. It should also be noted here that even though the lifetime of the sender decreases as the MSS increases, still the total number of bytes transmitted by the sender during its lifetime is greater. This shows that the protocol overhead greatly reduces at the sender as the MSS increases. Going by the same logic we can see that the protocol processing per packet at the receiver would also decrease. We get power savings at both the receiver and sender by increasing the MSS. It would be a good idea to try and find out the maximum MTU for which we get a significant improvement in the energy consumed at the nodes.

5.2 SACK Option

Figures 5.2a and 5.2b show the effect doing away of the SACK option has on the number of bytes transmitted and lifetime of the sender. As we can see in the graph turning the SACK option off causes us to be able to transport more number of bytes. This shows that the protocol processing required by the SACK option in fact causes us to expend more power than the savings achieved by having the SACK option (less number of retransmissions). Again the savings would be of a similar order at the receiver also. In this graph it is seen that the actual savings are

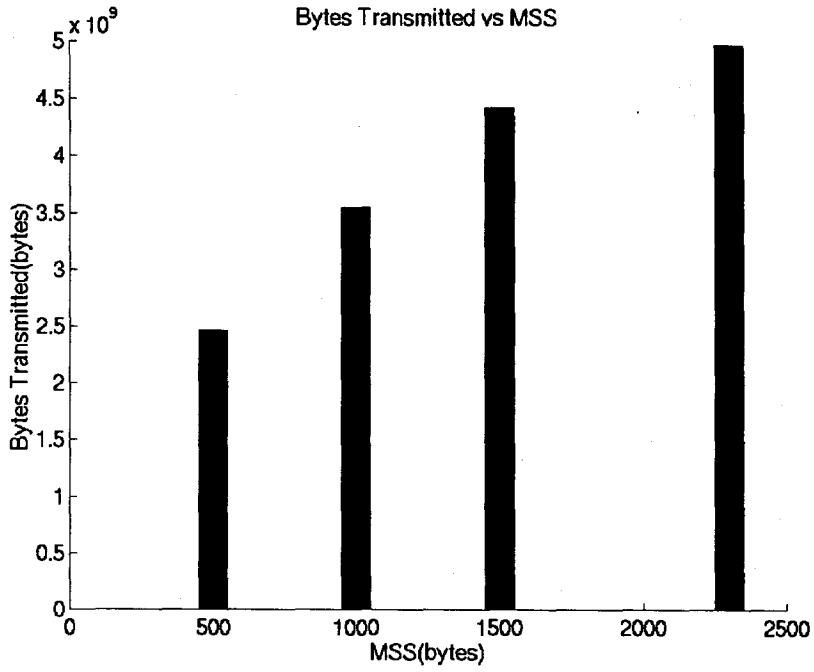

Figure 5.1a: Bytes Transmitted vs MSS - Normal

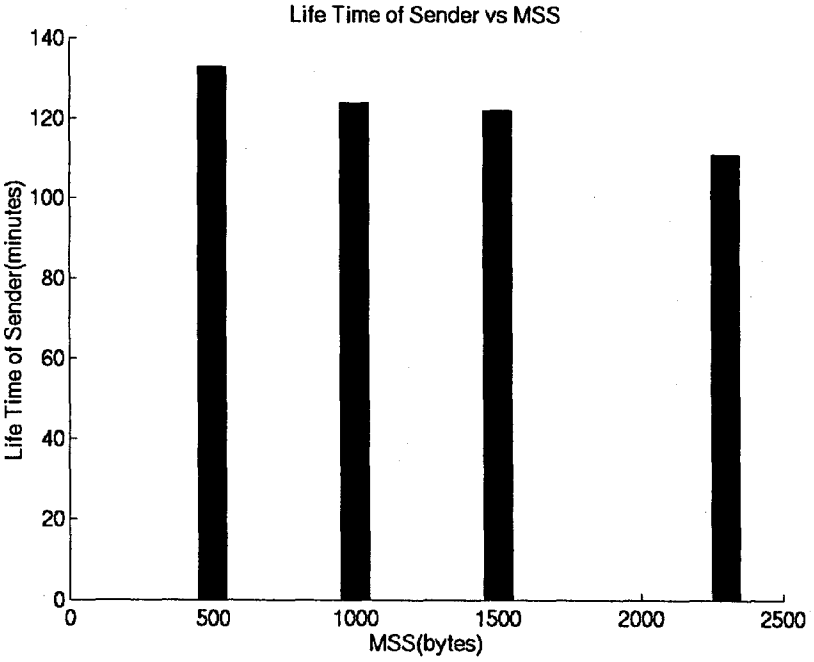

Figure 5.1b: LifeTime of Sender vs MSS - Normal

only for a MSS of 2296. For all other MSSs we get better performance by having the SACK option on. This can be explained by the fact that the SACK option only increases efficiency when there are multiple losses in a single window. As the MSS increases we see that a single window size consists of less number of packets and hence the probability of having multiple packets lost in a single window decreases. This causes SACKs to provide less increase in efficiency as the MSS increases. So for a large MSS it is in fact better to have the SACK option off and save power by not having to undergo the SACK protocol processing instead.

Figure 5.2c shows the comparison between having the SACK option off and having the SACK option on. However, we should keep in mind that given multiple packet loss in a single window (due to congestion etc) it is better to have the SACK option on. In the case of congestion and subsequent packet droppings by the nodes it is always better to have the SACK option on. But the routing layer can control congestion and hence if our sole aim is to save energy at the TCP layer then we can turn the SACK option off. It can be argued that turning the SACK option on or off depends very much on the given situation and traffic patterns. However, we must not forget that the bit error rate is not a very significant factor because the MAC 802.11b will take care of that by doing retransmissions at the MAC layer. Figure 5.2c and 5.2d show the comparison between the cases for which SACK is off and SACK is on. We should note that by turning SACK off we get an increase both in the lifetime and number of bytes transmitted. This shows that the efficiency has not increased.

5.3 Window Scale Option

Figure 5.3 shows the effect turning the window scale option off has on the bytes transmitted by the sender. As explained previously this option does not cost much protocol processing and so the improvement by not having this option is minimal. Since the effect of this option is minimal and it does not cost much power

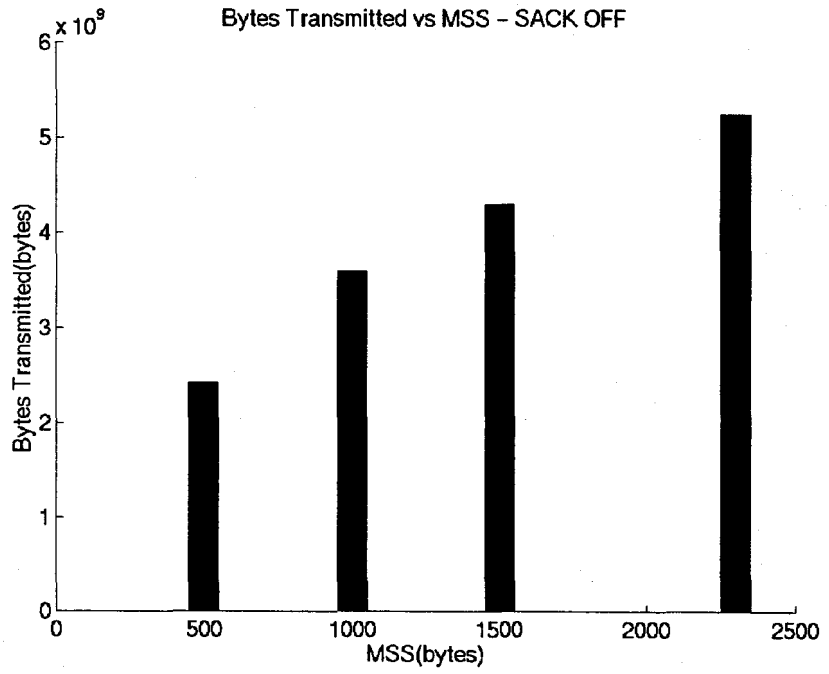

Figure 5.2a: Bytes Transmitted vs MSS – SACK OFF

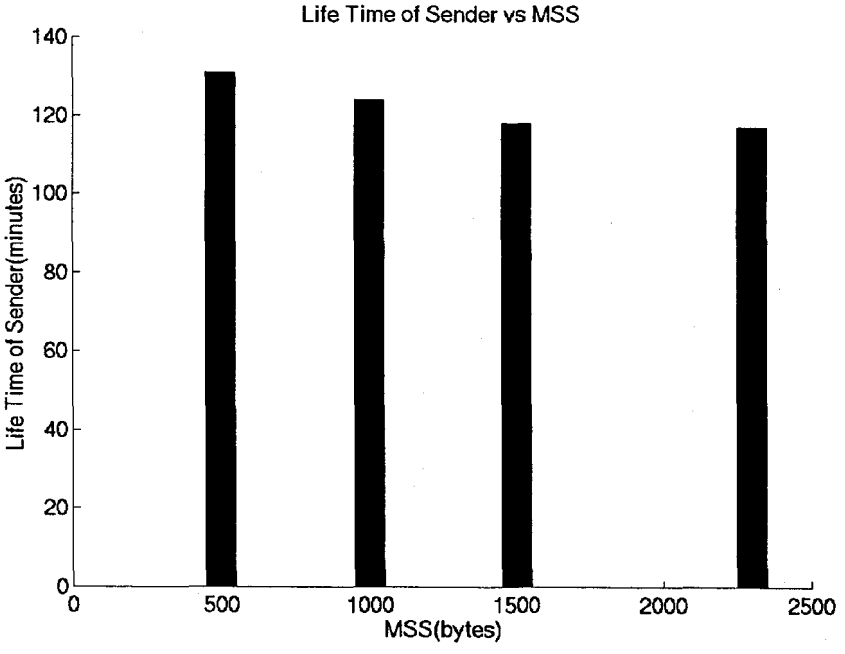

Figure 5.2b: LifeTime of Sender vs MSS – SACK OFF

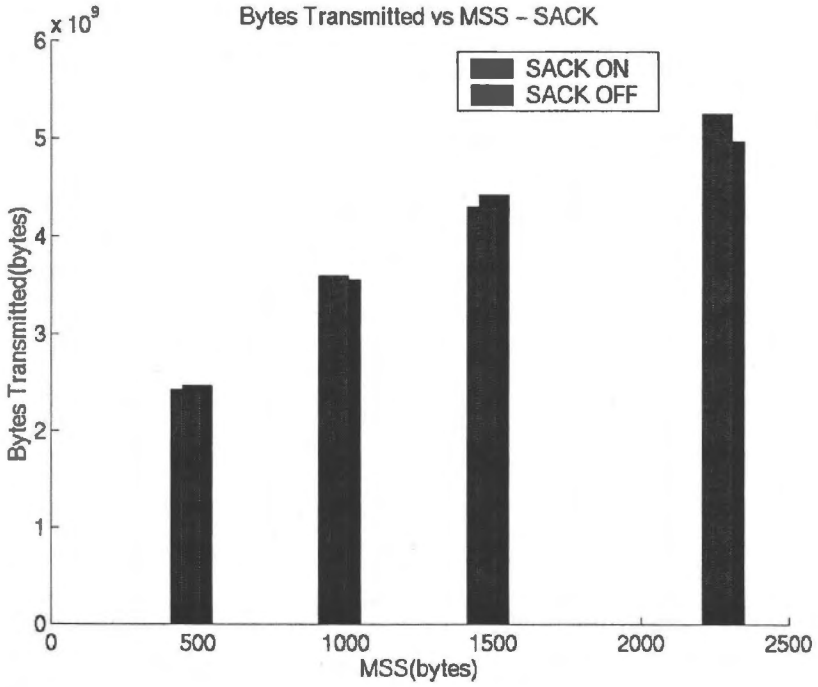

Figure 5.2c: Bytes Transmitted vs MSS - SACK

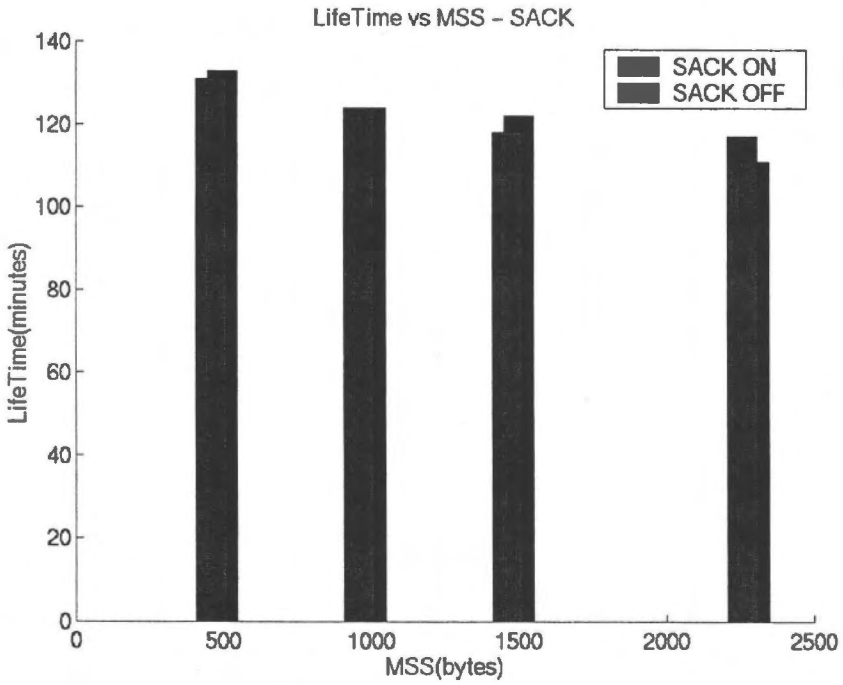

Figure 5.2d: LifeTime of Sender vs MSS - SACK

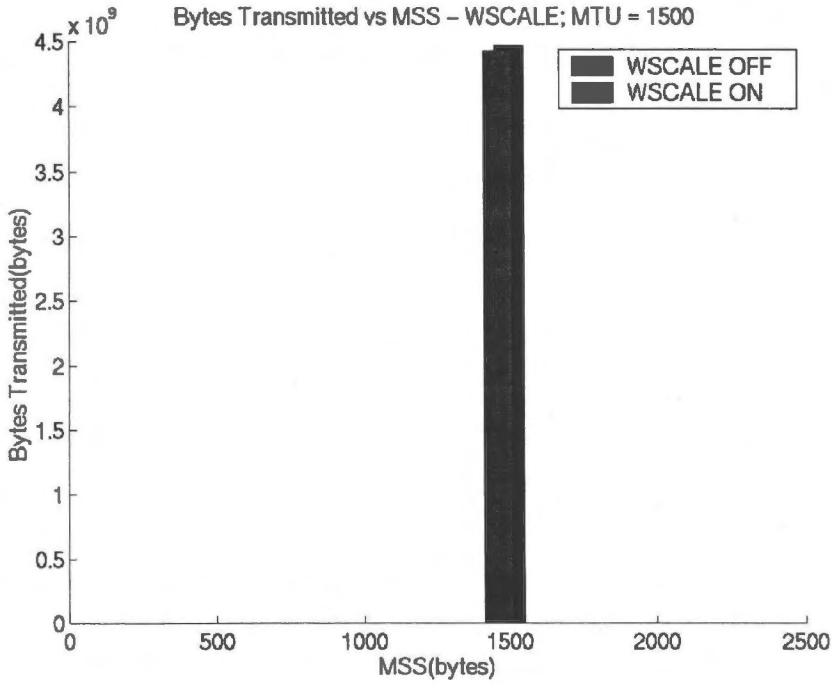

Figure 5.3: Bytes Transmitted vs MSS - WSCALE

in not having this option hence it is better to have this option. Also larger window sizes can help in reducing congestion etc which has not been taken into consideration in our experiments. The lifetime of the sender for both the cases is almost similar also.

5.4 Time Stamp Option

Figure 5.4 shows the effect turning the time stamp option off has on the number of bytes transmitted by the sender. In this we see that the effect of turning this option off is quite significant. But this significant effect can be explained by the fact that turning this option off in fact causes the frequent fast path to be much faster. This is because if there is no time stamp then there is no processing for PAWS checking which is in the fast path. This causes a significant saving.

However when combined with modified header prediction in which the PAWS checking is anyway done away with we will not get as much improvement by having this option off. Still there will be an improvement due to the saving in the header and also due to the calculation of the RTT for every ACK received. The header prediction modification is actually not required since PAWS checking will not be done anyway if the timestamp option is off. The best saving is achieved by having the timestamp option off. The lifetime of the sender with the TimeStamp option off is 115 minutes and with the option it is 111minutes. Hence we see that there is some energy conserved by having this option off.

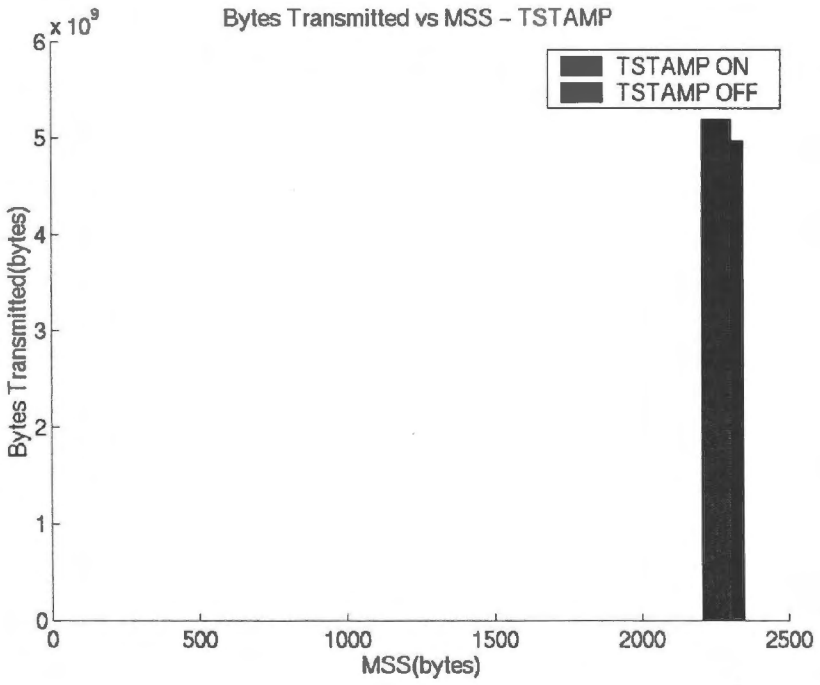

Figure 5.4: Bytes Transmitted vs MSS - TSTAMP

5.5 Header Prediction Modification

Figure 5.5 shows the effect of header prediction modification on the number of bytes transmitted by the sender. As mentioned before the saving is due to the fact that the PAWS checking is moved out of the frequent fast path. As explained it is better to have the time stamp option off and at the same time not modify the header prediction so as to get more savings in the energy consumed at the node. Actually header prediction modification will cause most of its effect only at the receiver and will not affect the power savings as much at the sender. With normal header prediction the lifetime of the sender is 111 minutes whereas with modified header prediction it is 114 minutes. Again we see that there is some energy conserved by having a modified header prediction algorithm.

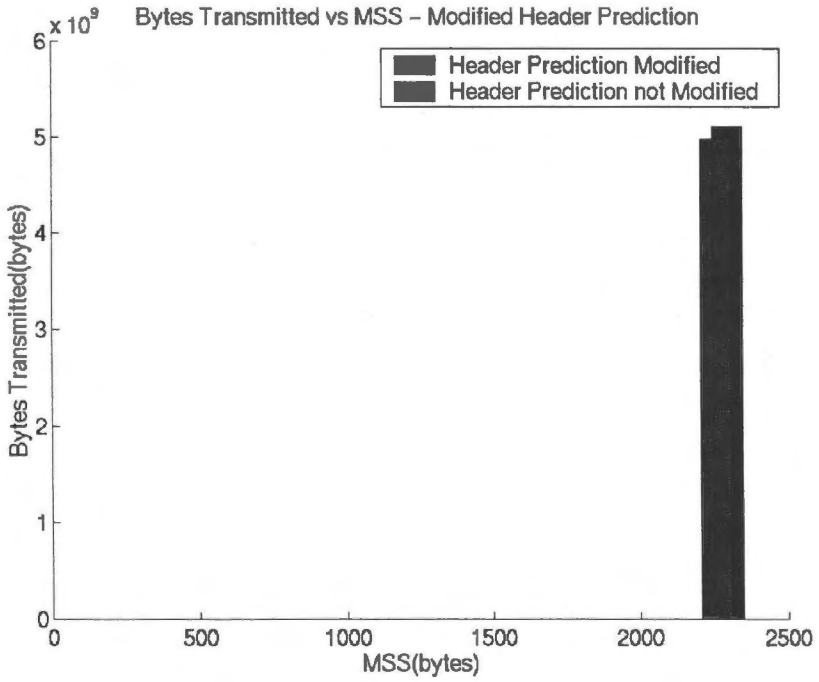

Figure 5.5: Bytes Transmitted vs MSS – Header Prediction

5.6 Delayed ACK Implementation

Figure 5.6 shows the effect of implementation of delayed ACK based on time on the number of bytes transmitted by the sender. This saving is due to the fact that less number of ACKs are sent by the receiver to the sender and less number of ACKs have to be received and processed by the sender. Hence the savings are at both the receiver and the sender. We see actually that without the modification one ACK is sent for every two packets received but with the modification there is a delayed ACK every 500ms if there are no errors in transmission. The lifetime of the sender in both the cases was 106 minutes. This tells us that the efficiency increases when we reduce the frequency of ACKs from the receiver. A similar effect is achieved at the reciever.

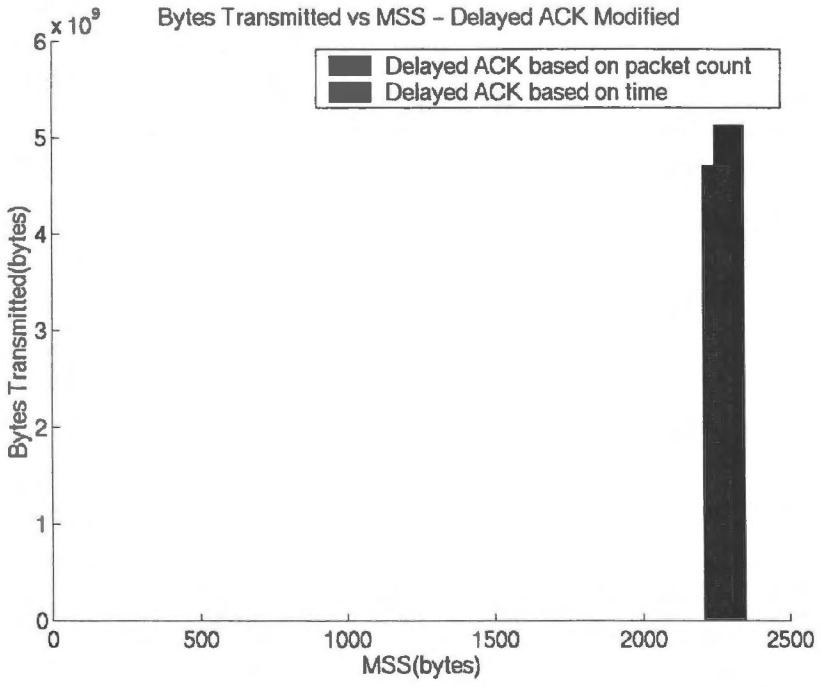

Figure 5.6: Bytes Transmitted vs MSS – Delayed ACK Modified

5.7 Enabled RTS/CTS

Figure 5.7a shows the comparison between no changes made to the current Linux TCP code and with all the significant optimizations incorporated. From the above we see that the optimizations that should be incorporated are MSS increased to 2296, SACK option off, header prediction not modified, timestamp option off, and delayed ACK implemented based only on time. This graph shows that the energy efficiency achieved is almost about 29%. The lifetime of the sender with no changes to TCP is 122 minutes and for the case with all changes it is 113 minutes. So we see that although the lifetime of the sender decreases with all the changes still the number of bytes transferred increases, which means that the efficiency increases greatly.

Figure 5.7b is the same as Figure 5.7a with RTS/CTS enabled. As we know that in the MAC 802.11 protocol there is a provision for media reservation by using initial handshaking. For all the above experiments as mentioned we had this feature turned off but for this experiment we had this featured turned on. For a packet of size greater than 1000 bytes there would be an initial RTS/CTS handshake performed between the two MAC layers to ensure that the media is reserved. By choosing the size as 1000 bytes for mandatory handshaking we ensure that the initial handshaking is required only for data packets and not for ACKs. This figure shows us that there is higher saving in energy with the RTS/CTS feature on. This can be partially attributed to the larger MSS which results in less number of RTS/CTS packets exchanged between the sender's and receiver's MAC layers. There is some gain because of the less number of errors caused due to the fact that the medium is reserved before and data is transmitted. In this the lifetime of the sender is 113 minutes for the case when all changes are made to TCP and it is 114 minutes when no changes are made. This tells us that enabling RTS/CTS increases the lifetime of the sender relatively. The difference in the lifetime of the sender between the two cases is much less when RTS/CTS is enabled which tells us that

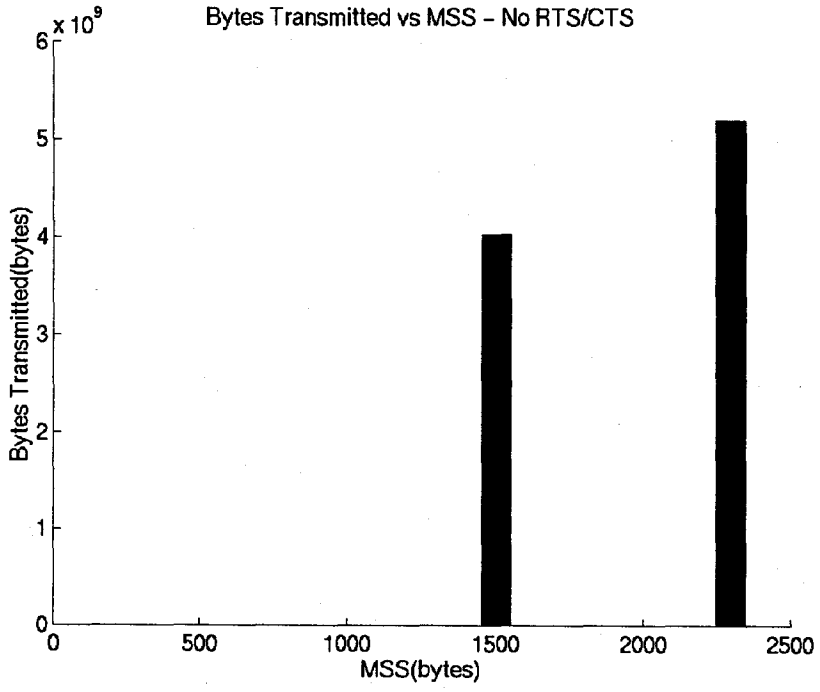

Figure 5.7a: Bytes Transmitted vs MSS - Best

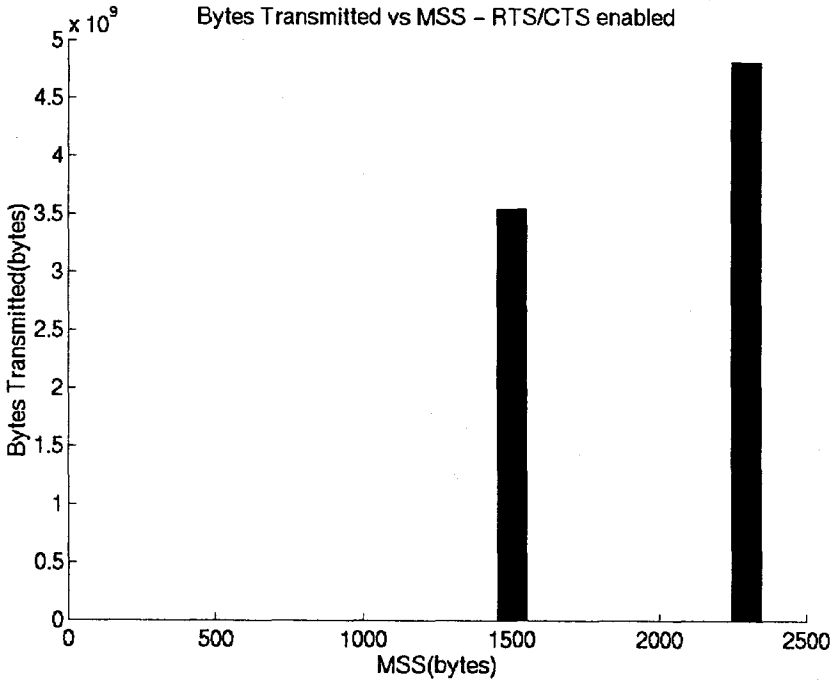

Figure 5.7b: Bytes Transmitted vs MSS – Best with RTS

the efficiency does not increase as much as it does for the case when RTS/CTS is disabled.

5.8 Distance

For this experiment we increased the distance between the sender and receiver to 10 meters. However within experimental limits (not more than 1 minute for the lifetime of the sender) we found the results to be similar to the results obtained when the distance between the sender and receiver was 1 inch.

5.9 Energy Saving at the Receiver

As mentioned we also expect a significant saving of energy at the receiver due to the modifications made. Hence this time we test for the savings on the receiver. Figure 5.9 shows the comparison between the bytes received when all the optimizations were turned on and with none of the optimizations enabled. In this experiment we kept the MTU=2296 for both the cases to see the effect of the other optimizations on the receiver and also enabled RTS/CTS handshaking. The lifetime of the receiver was 121 minutes with no changes and 120 minutes with all the changes.

5.10 Errors at the TCP layer

We measured the total number of retransmissions at the TCP layer of the sender during its lifetime. This gave us a measure of the quality of the wireless link. The number of retransmissions for an MTU of 1500 was found to be about 50. This

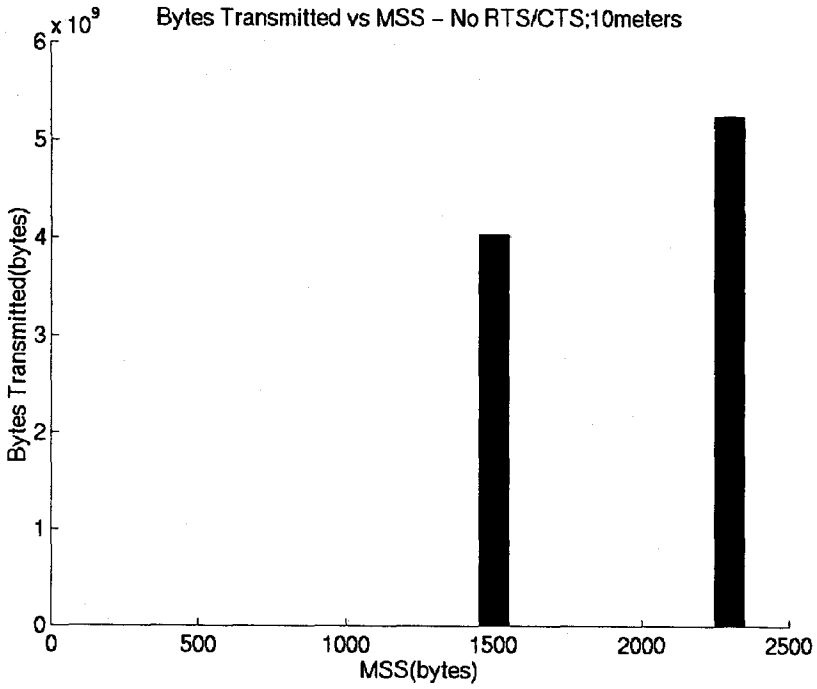

Figure 5.8: Bytes Transmited vs MSS – Distance

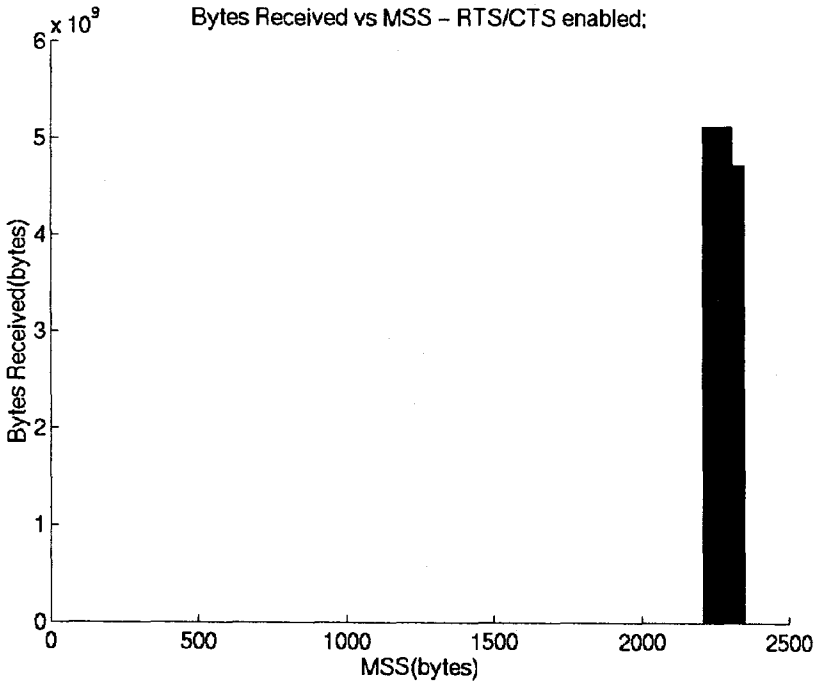

Figure 5.9: Bytes Received vs MSS – RTS and MTU = 2296

is quite a high error rate considering the fact that the MAC protocol is considered to be reliable. However the number of retransmissions is generally high because of the timer interactions between the TCP and the MAC layers and also because of the competing and redundant retransmissions by the TCP Layer since duplicate ACKs are not suppressed by the MAC layer.

5.11 Interfering Traffic Case

To verify our results for the case of interfering traffic we ran some experiments. A total of 8 sets of experiments were run. Table 5.1 shows the results that we obtained. In this table continuous traffic means that there was another node continuously sending data to the receiver. Intermittent traffic was the case when the interfering node would send about 5MB of data every 5 minutes. In this we had all the modifications made to TCP and we used an MTU of 1500 bytes. RTS on means that all the three nodes had the setting for RTS/CTS handshaking to be on for packet exchanges in excess of 1000 bytes. From the table we can see that for the case of continuous traffic the lifetime of the sender was generally high but the bytes transmitted was generally low. Also there is a significant difference between the SACK on and SACK off case when RTS/CTS is off. This shows that a huge amount of power is expended in executing the code for SACK and hence the total number of bytes transmitted is lower during the lifetime of the sender. This is not the case with intermittent load though – this tells us that when we have no initial handshaking and the traffic load is high it is better not to have the SACK option on – since a lot of power would be consumed in going through the SACK code. Again it should be noted that depending on the traffic conditions a decision about having the SACK option on or off has to be made. Other than that the table shows results that are intuitive.

	Time(mins)	Bytes
RTS on; SACK OFF,Continuous Traffic	128	2970847921
RTS on; SACK ON, Continuous Traffic	132	3058387131
RTS off; SACK OFF,Continuous Traffic	124	3811377004
RTS off; SACK ON, Continuous Traffic	117	3647866576
RTS off; SACK OFF,Intermittent Traffic	106	4183867981
RTS off; SACK ON, Intermittent Traffic	109	4185749921
RTS on; SACK OFF,Intermittent Traffic	111	3646355620
RTS on; SACK ON, Intermittent Traffic	112	3631339254

Table 5.1: Interfering Traffic Case

6. CONCLUSIONS AND FUTURE WORK

As can be seen by the results of the experiments, we do get quite a significant improvement in the energy efficiency of TCP by undertaking certain modifications in the way the protocol is implemented. First of all the MSS negotiated should be the maximum allowed by the path from the sender to the receiver and should not be 576 bytes for non-local connections. It should preferably be 2296 bytes since the MAC 802.11 protocol supports this size and could indeed be larger if the MAC 802.11 protocol supported it. Another improvement is by changing the implementation of delayed ACK to be dependent on time rather than the packet count of 2 packets. This makes more effective use of the window and thus causes power savings. The other improvement is by turning the timestamp option off – this causes saving by removing the PAWS checking from the frequent fast path of the TCP code. The SACK option is questionable and highly dependent on the traffic patterns and the number of nodes present etc. For our experiments we see that turning the SACK option off does give us an improvement but this might not be the case if there were variations in traffic which caused packets to be dropped. Hence it is always better to experiment with the SACK option before deciding whether to turn it on/off given the situation it is being used in. It must be noted that for connections to the Internet the MSS negotiated by TCP would be 1500 as the Internet mostly consists of Ethernet based interfaces which only allow a MTU of 1500 bytes. However, for ad hoc transmissions the MSS negotiated would be 2296 and thus the energy savings would be greater.

From the above discussion we can summarize the following:

MSS – maximum power saving.

TimeStamp Option – significant power saving.

Header Prediction Modification – significant power saving (comparable to the TimeStamp Option case).

Delayed ACKS (500ms) – significant power saving.

SACK Option – significant power saving but might not be the case depending on traffic and load on the network.

WindowScale Option – no power saving.

Data compression can also be considered as a method of saving energy. If we have a good compression algorithm and very fast low energy consuming processors then maybe sending compressed data would cause us to save some power. We did run experiments in which the sender sent compressed data to the receiver but the results we got were negative – i.e. more battery power was consumed in compressing the data and sending it rather than sending uncompressed data directly. This loss could be attributed to the high consumption of battery power in data compression since the processor we used was 90Mhz Pentium. Hence as the hard ware of our mobile systems changes data compression can also be considered a good option. In a newer version of the WaveLan driver there is provision for WEP encryption which can be turned on to see the effect of encryption on the power consumption. Also there is a power save mode of the WaveLan card which can cause some savings in power when the card is idle and not transmitting or receiving data. The effect of the above modifications in the TCP layer can be verified for more than one hop. Also various traffic patterns can be tested to see the effects of congestion etc.

There are two possible areas of future work. One is at the MAC layer. Some modifications can be made in the 802.11 layer itself in order to ensure that it transmits an error free and in sequence data to the next hop. The interactions between the MAC Layer and TCP Layer timers must be reduced. The duplicate ACKs received by the sender for which local retransmissions have been done, can be suppressed by the MAC layer. These will cause a decrease in the number of competing and redundant retransmissions by the TCP sender. The backoff strategy at the MAC layer can be made less aggressive eg: MACAW. Congestion control can be added to the MAC layer. Selective queue scheduling can also be

implemented at the MAC layer to avoid congestion and capture of the channel by a certain node. Another area that can be researched is the support of higher MTU by the MAC layer – this will at least help in energy conservation for transfers over purely mobile environments with similar hardware.

The other area where future work is possible is the TCP layer. Here the ELFN scheme can be implemented to take care of the link failures due to mobility. Route failure and re-establishment packets can also be used for this. Some steps can be taken to reduce the side effects of the Stretch ACK Violation. The TCP sender can be limited in its burstiness depending on the traffic patterns. The increase in the congestion window can be dependent on the bytes acknowledged instead of the number of ACKs received. Also the implementation of Stretch ACK Violation can be dynamic and we can send an ACK every certain number (more than 2) of packets depending on the traffic. Another area of TCP that can be experimented with is the fast retransmit and duplicate ACK algorithm. Here we can try to suppress TCP sender fast retransmit when we know that the underlying MAC layer will take care of the packet losses.

BIBLIOGRAPHY

1. Ramon Caceres and Liviu Iftode, "Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments", IEEE Journal in Selected Areas in Communications Vol.13, No.5, June 1995 (also in Mobile Computing, Eds. T. Imielinski and H. Korth, Kluwer Academic Publishers, Boston, 1997).

2. Ajay V. Bakre and B. R. Badrinath, "Indirect Transport Layer Protocols for Mobile Wireless Environment", Mobile Computing, Eds. T. Imielinski and H. Korth, Kluwer Academic Publishers, Boston, 1997.

3. Kevin Brown and Suresh Singh, "M-TCP: TCP for Mobile Cellular Networks", ACM Computer Communications Review, Oct. 1997, pp. 19-43.

4. R. T. Braden, "Extending TCP for Transactions – Concepts", RFC 1379, 38 pages (Nov.) 1992.

5. R. T. Braden, "Extending TCP for Transactions-Functional Specification", Internet Draft, 32 pages (Dec.) 1992.

6. V. Jacobson, "Tutorial Notes from ACM SIGCOMM '90", ACM, Philadelphia, Sept. 1990.

7. C. Partridge and S. Pink, "A Faster UDP," IEEE/ACM Trans. On Networking, Vol. 1, No. 4, August 1993.

8. D. C. Feldmeier, "Multiplexing Issues in Communications System design," Proc. ACM SIGCOMM '90, Philadelphia, Sept. 1990, pp.209-219.

9. P. McKenny and K. Dove, "Efficient Demultiplexing of Incoming TCP Packets," Proc. ACM SIGCOMM '92, Baltimore, 17-20 August 1992, pp. 269-279.

10. V. Jacobson, "4BSD Header Prediction," ACM Computer Communication Review, Vol. 20, No. 1, April 1990, pp. 13-15.

11. V. Jacobson, "Congestion Avoidance and Control", SIGCOMM '88, Stanford, CA., August 1988.

12. V. Jacobson, R. Braden, and L. Zhang, "TCP Extension for High-Speed Paths", RFC-1185, LBL and USC/Information Sciences Institute, October 1990.

13. 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.

14. F. Douglis, F. Kaashoek, B. Marsh, R. Caceres, K Lai and J. Tauber, "Storage Alternatives for Mobile Computers", ACM SIGCOMM '97, Cannes, France, Sept. 14-18, 1997.

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

16. S. Zdonik, M. Franklin, R. Alonso and S. Acharya, "Are "disks in the air" just pie in the sky?", IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA, pp.12-19, December 1994.

17. A. Chandrakasan, T. Simon, J. Goodman and W. Rabiner, "Signal Processing for an ultra low power Wireless Video Camera", 3rd International Workshop on Mobile Multimedia Communications, Princeton, NJ, September 25-27, 1996.

18. andy's Homepage – Linux – WaveLAN/IEEE802.11 driver, http://www.fasta.fh-dortmund.de/users/andy/wvlan/

19. W. Richard Stevens, "TCP/IP Illustrated", Volume 1, The Protocols, Addison-Wesley, 1994.

20. Craig Partridge, "Gigabit Networking", Addison-Wesley, 1993.

21. H. M. Chaskar, T. V. Lakshman and U. Madhow, "TCP Over Wireless with Link Level Error Control: Analysis and Design Methodology", IEEE/ACM Transactions on Networking, Volume 7, October 1999.

22. Bikram S. Bakshi, P. Krishna, N. H. Vaidya, and D. K. Pradhan, "Improving Performance of TCP over Wireless Networks", 17th International Conference on Distributed Computing Systems, Baltimore, May 1997.

23. Cellular Digital Packet Data System Specification: Release 1.0, CDPD Forum Inc 1995.

24. Gavin Holland and Nitin Vaidya, "Analysis of TCP Performance over Mobile Ad Hoc Networks", Fifth Annual International Conference on Mobile Computing and Networking (MOBICOM), Seattle, August 1999.

25. H. Balakrishnan, Venkata N. Padmanabhan, S. Seshan and Randy H. Katz, "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links", ACM SIGCOMM 1996.

26. H. Balakrishnan, S. Seshan and Randy H. Katz, " Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks," Proc. ACM Mobile Computing and Networking Conf,. ACM, pp 2-11, 1995.

27. H. Balakrishnan, V. N. Padmanabhan and R. H. Katz, "The Effects of Asymmetry on TCP Performance," Proceedings of the IEEE Mobicom'97, pp 77-89. Sept 1997.

28. K. Chandran, S. Raghunathan, S. Venkatesan and R. Prakash, "A Feedback based scheme for improving TCP Performance in ad-hoc wireless networks", in Proceedings of International Conference on Distributed Computing Ssytems, Amsterdam, May 26-29, 1998.

29. Mario Gerla, K. Tang and R. Bagrodia, "TCP Performance in Wireless Multi-hop Networks",in Proceedings of IEEE WMCSA'99, Feb 1999.

30. T. V. Lakshman, U. Madhow, and B. Suter, "Window based Error Recovery and Flow Control with a Slow Acknowledgement Channel: A study of TCP/IP Performance", In Proc. Infocom'97, April 1997.

31. L. Zhang, S. Shenker, and D. D. Clark, "Observations and Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic", In Proc. ACM SIGCOMM'91, pages 133-147, 1991.

32. V. Paxson, M. Allman, S. Dawson, W. Fenner, J. Grinner, I. Heavens, K. Lahey, J. Semke and B. Volz, "Known TCP Implementation Problems", RFC 2525.

33. Andrew S. Tanenbaum, "Computer Networks", Third Edition, 1997, Prentice-Hall, Inc., Upper Saddle River, New Jersey.

34. J. Liu and S. Singh, "ATCP: TCP for Mobile Ad Hoc Networks", INFOCOM'00 (submitted).