Analysis of TCP's Computational Energy Cost for Mobile Computing

Bokyung Wang and Suresh Singh
Department of Computer Science
Portland State University
Portland, OR 97207
Accepted to ACM SIGMETRICS 2003 as a Poster
Full paper submitted for publication to J. Wireless Networks

Abstract

In this paper we present results from a detailed measurement study of TCP (Transmission Control Protocol) running over a wireless link. Our primary goal was on obtaining a breakdown of the *computational energy* cost of TCP at the sender and receiver (excluding radio energy costs) as a first step in developing techniques to reduce this cost in actual systems. We analyzed the energy consumption of TCP in FreeBSD 4.2 and FreeBSD 5 running on a wireless laptop and Linux 2.4.7 running on a wireless HP iPAQ 3630 PocketPC. Our initial results showed that 60 - 70% of the energy cost (for transmission or reception) is accounted for by the Kernel – NIC (Network Interface Card) copy operation. Of the remainder, $\sim$15% is accounted for in the copy operation from user space to kernel space with the remaining 15% being accounted for by TCP processing costs. We then further analyzed the TCP processing cost and determined the cost of computing checksums accounts for 20 – 30% of TCP processing cost. Finally, we determined the processing costs of two primary TCP functions – timeouts and triple duplicate ACKs. Putting all these costs together, we present techniques whereby energy savings of between 20% – 30% in the computational cost of TCP can be acheived.

1 Introduction

Communication underlies many applications run by users of mobile devices and thus there is a need to better understand how and where energy is consumed in the communications pipeline. Prior work by various researchers (discussed in section 2) has looked at the energy consumption of various wireless interface cards [12, 13], the impact of ARQ (Automatic Repeat reQuest) protocols on overall energy consumption, the effect of channel conditions on energy consumption, and comparative energy and throughpout-based studies of different TCP flavors. While all of these studies have contributed a great deal to understanding the impact of the wireless link on the overall end-to-end energy consumption, none of these studies has looked at the *computational energy* cost of running the TCP protocol itself. In this paper we analyze the computational energy cost of the TCP protocol in detail and use this analysis to develop approaches by which the node-level cost of running TCP can be reduced. Interestingly, our results show we that approaches for optimizing TCP for high-speed network operation, can, in some cases, result in lower energy consumption in the wireless domain as well!

The computational cost of TCP includes the cost of various copy operations, the cost of computing checksums, the cost of responding to timeouts and triple duplicate ACKs, and other bookeeping costs. Our goal in this study is to estimate the energy (in Joules) taken by each of these operations, and, in order to ensure that our results are robust, we performed these measurements on three different systems. The benefit of our study is fourfold: first, it enables us to develop techniques to reduce the computational energy cost of TCP; second, other researchers working on developing throughput models for TCP can use the results of our study to simultaneously develop energy models since we have characterized the cost of the primary TCP functions; third, our node-level energy models can be incorporated into simulators such as NS-2 [1] to obtain overall energy cost of TCP connections (NS-2 currently only includes the radio energy cost); finally, the methodology we have developed can be used by others to evaluate the TCP energy cost in their system of choice.

The remainder of the paper is organized as follows. In the next section we discuss related work; section 3 presents an overview of TCP processing and describes the key tasks that are energy consuming; section 4 details the methodology we followed in our study to estimate the various energy costs; results of our measurement are described in section 5; we

use the results of our measurements to develop techniques to reduce the overall computational energy cost of TCP and a discussion of this is presented in section 6; finally, section 7 presents our conclusions.

2 Related Work

The TCP protocol has been analyzed in the past decade by a large number of researchers for a variety of different reasons. Of this large body of prior work, however, we focus on two specific sub-areas that are directly relevant to the work reported in this paper: work that has analyzed the energy consumption of TCP (or other protocols) for wireless networks and work that has examined the performance of TCP at the node level (where the goal was to enable end-systems to operate at gigabit speeds). While our paper focuses on the wireless domain, we see that many of our results are parallel to those obtained for making TCP operate at high data rates.

[8] is one of the earliest papers that examined the processing overhead of TCP. The goal of this work was to see if TCP processing was indeed a bottleneck in end-to-end connection throughput. The authors counted the number of instructions executed in TCP and IP at the sender as well as at the receiver when TCP follows the "normal path" of execution. In addition to instruction counts, they also provided a breakdown of the processing cost in terms of time consumed. We refer to specific results of [8] later in our paper (sections 5.2 and 5.3) where we draw comparisons with out own measurements. Among the results of [8] is the conclusion that TCP processing cost is not high, contrary to popular belief in 1989, and is not the bottleneck to achieving high data rates.

In order to get TCP to operate at gigabit speeds, researchers have developed methods in software as well as in hardware. We do not provide a comprehensive discussion of the various results here because our focus in this paper is on wireless networks. However, it is useful to list some of the main techniques developed because *our results indicate that some of these techniques developed for high-speed TCP can also result in energy savings in the wireless domain where TCP processing speed is not as important!* The techniques developed by the high-speed TCP community (for example [21, 5, 22, 3, 17, 15, 10, 9, 7]) include, on the software side, using zero-copy, header prediction, interrupt supression (where the processor is only interrupted when several packets have been received), using jumbo frames, and integrating checksum computation with copy. In hardware, the techniques suggested include, checksum offloading, offloading some common path TCP/IP functionality to the NIC (Network Interface Card), packet aggregation on the NIC card, etc. The goal of all these optimizations is to ensure that applications can run at network speeds and not be hampered by TCP's processing.

In the wireless domain, there have been several papers dealing with the problem of TCP's energy consumption over wireless links where energy consumption is modeled as the ratio of the number of successful transmissions to the total number of transmissions. Most of these papers focus on the impact of the link layer on the energy efficiency of TCP. [29] considers the effect of ARQ strategies on energy consumption. The key idea is to suspend packet transmission when channel conditions worsen and probing the channel state prior to packet transmission. When channel conditions improve, packet transmission is resumed. [30] analyzes the energy consumption performance of various versions of TCP for bulk data transfer in an environment where channel errors are correlated. The paper only considers one-hop wireless links with zero propagation delays. [19] also considers the effect of ARQ, FEC (Forward Error Correction) and a combination of the two on energy consumed in ad hoc networks. The paper uses the Maisie simulation language and Glomosim to perform the study. The underlying theme in these papers is that reducing the number of unnecessary retransmissions implies better energy efficiency.

In the recent past there have been simulation-based studies [31, 11] of throughput and energy consumption of TCP connections in wireless networks. These studies investigate the tradeoff of radio transmit power on the throughput of TCP. [31] concludes that increased trasmitted power (RF power) is not always good as their study shows that increased transmitted power results in higher TCP throughput up to a point (because of better signal to interference ratio), after which an increment of transmitted power actually leads to worse performance due to greater interference. [11] examines the effect of transmission range on the energy as well as throughput of TCP in multi-hop networks. They model energy as a sum of the idle energy consumed for the duration of the connection and the sum of the hop-by-hop transmit energy consumed (path loss is the primary contributor to this energy cost). The throughput is modeled as being inversely proportional to the product of the packet error rate and the square root of the round trip time. The authors then show that decreasing the transmission range will result in lower overall energy consumed but will decrease throughput. However, decreasing the transmit energy beyond a certain point will result in an increase in energy because the throughput falls to very low levels and the idle energy cost dominates. While interesting, this paper appears to make several simplistic assumptions about the radio environment, such as: ignoring the relationship between packet loss rates and decreased transmit power,

energy consumed by receiver nodes (since each transmission is often heard by several neighbors), and the possibility of radios entering sleep states to minimize idle power consumption. [27] compares the energy and throughput-efficiency of TCP error control strategies for three implementations of TCP Tahoe, Reno, and New Reno. They implemented the three versions of TCP using the x-kernel protocol framework and their focus was to study heterogenous wired/wireless environments. Finally, [26] provides an energy and throughput comparison of TCP reno, newreno, and SACK for multihop wireless networks. This paper reports on the results of actual energy measurements and the primary conclusion is that TCP newreno work best in wireless environments.

Unlike the papers discussed above, the focus of our work has been on examining the *computational energy cost of TCP*. This cost is affected by the software as well as by the hardware of the mobile node. We explore this relationship in detail and then develop approaches that would result in a reduction of the node-level TCP energy cost. Our work can thus be thought of as complimentary to the above referenced wireless papers in that while we look at the node-level cost of running TCP, the other papers examine the impact of the propagation environment on the total end-to-end cost of TCP.

3 Energy consumption in TCP

The computational energy cost of a TCP session established by a wireless device can be viewed as the total energy cost of the session minus the cost of the radio and the idle energy cost of the connection (i.e., when the node is idle awaiting ACKs or data segments). Consider the example illustrated in Figure 1 which provides plots of current draw versus time. The upper curve is the total system current draw for a laptop equipped with a 802.11b radio card and the bottom plot is the simultaneous current draw for the 802.11b card alone. In this plot, the laptop is acting as a sender for a ttcp [28] connection. In order to determine the computational energy cost of this connection, we first delete all portions of the system current plot in which the system is idle (this corresponds to the areas in the upper plot marked by *A* – not all areas are marked for clarity). We are then left with a plot consisting only of those periods that show TCP related activity (i.e., the regions marked *C*). From these activity periods we subtract out the radio energy cost (the regions in the lower plot marked as *B*). This final data is summed over the lifetime of the connection (the units are Ampere-second) and multiplied with the voltage to give us the total computational energy cost of the connection (we provide an example of this calculation in section 4.2). Thus, we see that this procedure appropriately eliminates the radio cost (transmit/receive/idle/sleep) and the idle system cost from the measurement. The energy value obtained is thus the pure hardware/software cost of running TCP at the node.

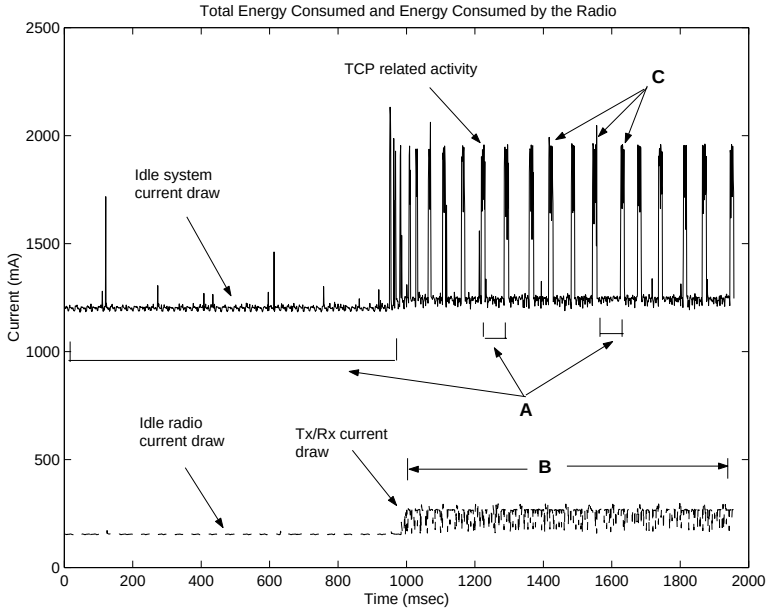

Figure 1: Sample plot of current draw for a ttcp connection.

Our goal in this paper is to decompose this computational energy cost further so as to better understand which TCP/IP

functions cost more energy. To this end, we view the *TCP computational energy cost* as being composed of the following primary components:

- The cost of moving data from the user space into kernel space that we denote *user-to-kernel copy*. Note that this cost can be eliminated by using zero-copy [24, 6], if available.
- The cost of copying packets to the network interface card that we denote as *kernel-to-NIC copy*.
- The cost of processing in the TCP/IP protocol stack (*TCP Processing Cost*) which includes:
  - The cost of computing the checksum (at sender and receiver),
  - The cost of ACKs (at sender and receiver),
  - The cost of responding to timeout events (TO) (at the sender this is the processing cost plus NIC-to-kernel copy cost of the retransmitted packet),
  - The cost of responding to triple duplicate ACKs (TD) (at the sender this is the processing cost plus the NIC-to-kernel copy cost of the retransmitted packet), and
  - Other processing costs such as window maintenance (at sender on receiving ACKs), estimate round trip time (at sender), obtaining the TCB (Transmission Control Block), interrupt handling, and timer maintenance [1,2]

While it is possible to break down some of these costs even further, we believe that this level of granularity is sufficient for most modeling purposes[3]. Using this breakdown, the total computational cost of a TCP session, in which D bytes of data are transmitted, can be written as:

$$\begin{aligned} E(D, \text{MTU Size}) \quad &= \textit{user-to-kernel copy} \text{ for } D \text{ bytes } + \text{ checksum cost for } D/\text{MTU packets} \\ &+ \text{ACK cost} + \textit{kernel} - \textit{NIC copy} \text{ cost for } D/\text{MTU} \\ &\text{packets of size MTU} + \text{ Number of TOs} \times \text{TO copy and processing cost} \\ &+ \text{Number of TDs} \times \text{TD copy and processing cost} + \text{ Other processing costs} \end{aligned} \tag{1}$$

We have written the total energy cost E as a function of the MTU (Maximum Transmission Unit) size because different MTU sizes result in different total energy costs.

4 Methodology

4.1 Experimental Setup

In order to ensure that our measurements of TCP's computational energy cost are widely applicable, we performed the measurements on three different platforms:

- Toshiba Satellite 2805-S201 laptop with a 650MHz P3 Celeron processor, 256MB RAM running FreeBSD 4.2,
- The same laptop running FreeBSD 5.0, and
- A Compaq iPAQ H3630 PocketPC with a StrongARM processor running Linux 2.4.7.

The *same* network card was used for each of the three platforms above – a 2.4GHz Lucent 802.11b WaveLAN "Silver" card. Finally, power management was turned off in the computers as well as the 802.11 card for our experiments. The display was also off and the x-server was not running.

Energy consumption was determined by measuring the input voltage and current draw using two Agilent 34401A digital multimeters that have a resoultion of one millisecond. We did not use batteries in the devices because, as noted in

[1] In this paper we chose not to analyze TCP options such as SACK (Selective ACK) but rather concentrate on the core TCP implementation alone.

[2] We have not isolated the cost of interrupt processing but rather folded it into the various processing costs.

[3] Typical TCP modeling studies (see [20, 25]) proceed by obtaining the probability of TD and TO events as a function of packet loss rates and then using these values to obtain throughput as a function of packet loss probability, round trip time, window size, etc. The primary factor affecting throughput is the presence of TO and TD events since these result in a reduction in the congestion window size. Given this, we believe that the level of granularity we use to characterize energy consumption is adequate for modeling purposes.

[13], avoiding the use of batteries allows for a more steady voltage to be supplied to the device. Since our goal was to measure the computational cost of TCP alone, we needed to separately but simultaneously measure the current draw of the laptop/iPAQ and the current draw of the 802.11b PCMCIA card (as illustrated in Figure 1). To do this, we followed the methodology described in [13, 12, 26] in which, the 802.11b PCMCIA card is attached to a Sycard PCCextend 140A CardBus Extender [16] that in turn attaches to the PCMCIA slot in the laptop or iPAQ. This extender is a breakout box which allows us to directly measure the current and voltage draw of the 802.11b card. Our final experimental set up was as follows: the laptop's (or iPAQ's) power supply is measured by one Agilent 34401A multimeter while the 802.11b's power supply is measured by attaching a second Agilent 34401A multimeter to the appropriate terminals on the Sycard CardBus extender. Both these multimeters are triggered simultaneously by a second laptop that also collects data from these multimeters. The average current and voltage measurements are given below:

	Measured Average Current (mA)			Voltage
	Transmit	Receive	Idle	(V/DC)
Toshiba Laptop	1905	1709	1220	15.08
HP iPAQ	655	630	517	5.05
WaveLAN	263	181	159	5.046

4.2 Sample Measurement

In order to measure the computational energy alone, we need to identify and then subtract out the energy consumed by the radio as well as the idle energy consumed by the laptop/iPAQ. We illustrate this process via the example shown in Figure 2. In this figure we plot the measured simultaneous current draw for the laptop and the WaveLAN card (MTU 1500 bytes). We make the following observations:

- The radio continues transmitting long after the packets have been copied to its input buffer (the TCP processing and copy operation corresponds to the region between A–B and C–D in the figure). This is because the packets are copied at a much higher speed of 20MBytes/sec over the PCMCIA bus as compared with the radio transmission speed.
- Interestingly, we see that the radio transmission speed is far below specifications and is approximately 1.2Mbits/sec (see also Figure 5).

The computational energy cost of TCP is calculated as follows:

- For all measurements of system current draw above idle, we compute the total current drawn assuming that the current draw is constant over 1 msec intervals (we needed to do this becasue the resolution of our multimeter is 1 msec). In Figure 2, this means that we sum the current in regions A–B and C–D. For the sake of illustration, let us only consider region A–B. The sum of the current draw here is 17137.6μAmpere-Second. The energy draw is therefore 0.258 Joules (after multiplying with the voltage drawn).
- We next compute the radio energy drawn for this same period. In our example, the radio energy drawn in the period A–B is 0.012 Joules (note that the radio card runs at 5V rather than 15V).
- The computational energy cost is then the difference of the above two values. In our example, this gives us a value of 0.246 Joules. Our experiments used a send buffer size of 16KBytes or approximately 11 packets. Thus, the per packet computational energy in this example is 0.022 Joules (see Figure 4 where we plot the packet cost).

Note that by following the above procedure we discount the idle energy draw of the system as well as the radio cost. In order to compute the computational energy as described above, we wrote scripts that automatically performed the above algorithm on very large data sets.

4.3 Parameter Selection

We used ttcp (the buffer size for ttcp was set at 8192 bytes) to measure the energy consumption for transmitting 1 MByte of data. We simultaneously ran tcptrace [18] and tcpdump [28]. The information collected from these tools was used to identify the radio events (*B* in Figure 1) and appropriately subtract the corresponding radio energy cost from the overall system cost (*C* in Figure 1). The parameters we used for the various measurement experiments are illustrated below.

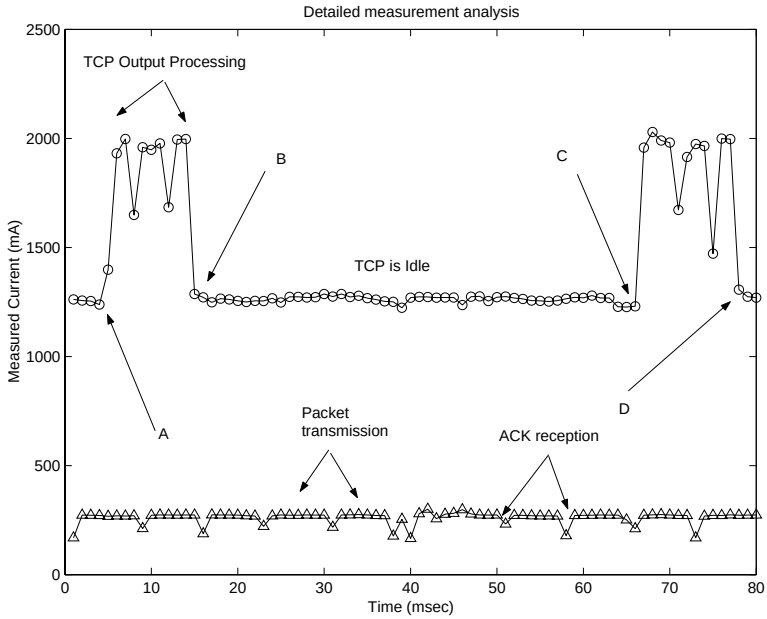

Figure 2: Example of determining computational energy cost.

Paramaters	*Values*
TCP window size	16K
RTS/CTS	OFF
MTU Size	168, 296, 552, 960, 1500 bytes
Packet Loss	No default setting (for most experiments) 3%, 5%, 7%, 10% using Dummynet (for experiments in section 5.3

In order to determine the energy consumed by various functions of TCP, we needed to set up TCP connections from our test devices via a wireless LAN to some host. We used an access point that was connected to the local 100Mbps ethernet backbone. For some studies (such as those where we studied the cost of TD and TO processing) we needed to carefully control the packet loss rates. To do this, we ran Dummynet [23] at a local node in the connection path and dropped packets from the ttcp connection.

5 Results & Analysis

We have divided the measurement results into three sections for reasons of clarity:

- In the first section we discuss the total computational and radio energy costs and provide a comparison between the three systems. The radio energy costs include the transmission and reception costs only (i.e., we discount the idle radio energy cost because, as noted in [27], in the case of low throughput the idle cost would dominate all others).
- In sections 5.2 and 5.3 we provide a breakdown of the total computational energy cost into the major components discussed in section 3.
- Finally, in section 5.4 we validate our results in a new set of experiments in which we compare predicted energy cost and the actual energy measured for these connections.

We would like to note that for all the plotted data in the following sections we show 95% confidence intervals that were obtained by combining data from between 50 and 100 repetitions of each experiment.

5.1 Comparison of computational energy costs between the three systems

In this first set of experiments, we set up TCP connections with a local host and the three systems were placed in the exact same physical location in the laboratory and were close to the access point[4,5]. One megabyte of data was sent from or to each system using ttcp. The first study compares the computational energy cost of the three systems (acting either as a sender or as a receiver) as a function of MTU size (see Figure 3). We note that *we did not observe any losses* and thus the computational cost we measured does not include the TD and TO costs.

- In general, we can see that the iPAQ[6] is indeed far more energy efficient than the laptop (with either version of FreeBSD). This is to be expected because the iPAQ uses a low-power StrongARM processor and is optimized for low energy operation. It is important to note that these energy costs represent only the computational energy cost and do not include the idle energy cost which, in the case of the laptop, consists of the cost of the fan, hard disk, CD, screen, etc.
- The impact of increasing the MTU size in all cases is to reduce the computational energy cost because the cost of operations such as copy and checksum computation are amortized over more bytes. It is also noteworthy that the impact is higher in the case of the laptop. These behaviors are explained in more detail in section 5.2.
- In comparing the two versions of FreeBSD, we note that there is no difference in energy cost when the laptop is a receiver. However, when it is a sender, we see that version 5 is more energy efficient (however, since the confidence intervals of one include the means of the other we cannot make a very strong statement about this difference).
- In all three cases, we note that the computational cost of sending is greater than that of receiving. This is expected because the sender does perform more bookeeping tasks such as window management, RTO computation, etc.. However, the three systems show a variance in behavior. The sender in FreeBSDv4.2 consumes 10% more energy than the receiver whereas the sender in FreeBSDv5 consumes only 5% more energy. It is difficult to pinpoint the reason for this difference in behavior because the TCP/IP code for version 5 is very different from version 4.2 (including the fact the version 5 implements TCP newreno while version 4.2 implements reno). Finally, we note that the iPAQ sender consumes 15% more energy than an iPAQ receiver.
- In Figure 4 we plot the *per packet* computational energy cost (excluding the radio transmission/reception cost) as a function of MTU size. As expected, larger packets do cost more to process and there is a linear increase in cost as a function of packet size. The increase is, however, steeper for the case of the laptop. We return to a discussion of this in section 5.2.

Figure 5 plots the throughput of the three systems as a function of MTU size. We see that larger MTU sizes result in higher throughputs.

- We note that FreeBSDv5 has a higher throughput than FreeBSDv4.2. The reason for this is that the device driver for FreeBSDv5 copies data more efficiently to the card than the older driver for version 4.2 (since there were no losses, the TCP flavor, reno vs newreno, has a smaller impact on this difference).
- We observe that the throughput of the iPAQ was the lowest of the three even though the TCP implementation in the iPAQ is Newreno. The reason for this behavior is that the 802.11b device driver in the iPAQ copies data more slowly than the device driver in the laptop (possibly due to a poorer driver implementation but most probably due to the difference in processor speed – 650MHz for the laptop versus 200MHz for the iPAQ).

A last comparison we performed between the three systems was a measurement of the relative costs of the radio (i.e., the transmit and receive costs only) and the computational cost of TCP (as defined in section 3). These results are shown in Figure 6. As we can see, in the case of the laptop, the computational cost accounts for the majority of the total cost of the ttcp connection. The figures are reversed for the case of the iPAQ where the computational costs are smaller than the radio costs. Finally, note that the radio costs account for a smaller percentage in the case of a receiver. This is because the receive current draw of the 802.11b card is smaller than the transmit current draw.

[4] Also note that the same 802.11b card was used in each of the three systems and, furthermore, the same laptop was used for two cases representing FreeBSDv4.2 and FreeBSDv5 – we simply booted with the appropriate kernel to perform the measurements.

[5] In the studies reported in this section, i.e., section 5, we did not use the zero-copy option in FreeBSDv5. We use it later in section 6.

[6] We were unable to set the segment size to 128 bytes for the iPAQ and thus the graphs for the iPAQ do not show this data point.

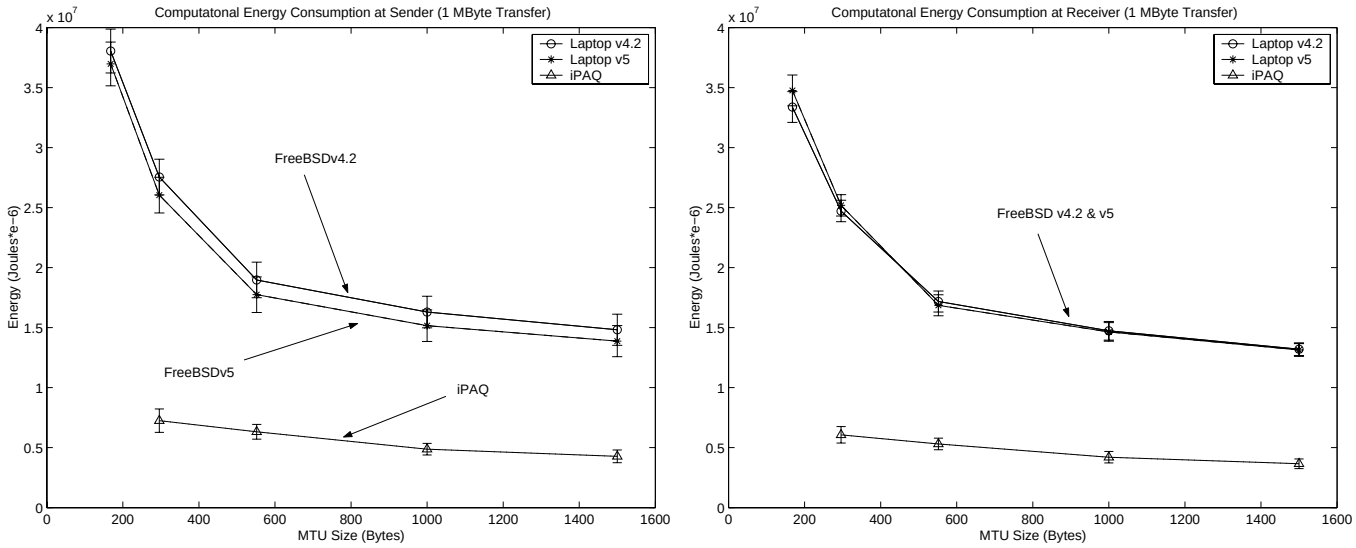

Figure 3: Total computational energy costs.

5.2 Computational energy cost breakdown

In this section and the next we decompose the total computational cost of TCP into its components as discussed in section 3. Recall that the computational energy can be decomposed into three primary categories: user-to-kernel copy cost, kernel-to-NIC copy cost, and TCP processing cost. The two copy costs are similar at the sender and at the receiver. However, the TCP processing cost is somewhat different. At the sender the TCP processing cost includes checksum, ACK processing, congestion window updates, timeout (TO) processing, triple duplicate (TD) processing, timer processing, RTO computation, and other OS related costs. At the receiver, TCP processing includes ACK generation, checksum computation, data sequencing, receive window management, etc. In this section we decompose the TCP computational cost into the *copy costs* (user-to-kernel space, kernel-to-NIC) and the *TCP processing cost* (i.e., TCP computational cost minus the two copy costs). We further decompose the TCP processing costs in section 5.3.

Data sent through a TCP socket is first queued in the socket send buffer and it is then copied into kernel space for further processing. In order to determine the cost of performing this copy (that we call user-to-kernel space copy), we implemented a kernel program using the *copyin()* and *copyout()* functions. We then ran this program and copied large files (which were composed of random data). We measured the current draw while this program was running and used this measurement to obtain the user-to-kernel copy cost. To determine the copy cost from the kernel to the NIC card, we used UDP for data transmission. Unlike TCP, data sent through a UDP socket goes all the way down to the network interface and the socket send buffer is not used.

Figure 7 shows the relative costs (in %) at the sender of the two copy operations and TCP processing for the three systems. As the figure shows:

- By far the most expensive operation is the Kernel – NIC copy cost. This accounts for between 60 – 75% of the total cost incurred at the sender. To better understand these costs, note that the PCMCIA card is not very efficient and it takes a significant amount of time to copy the data from memory to the 802.11b card. The device driver calls a copy operation for each packet resulting in an interrupt and context switch since the PCMCIA 2.1 specification does not allow DMA (Direct Memory Access) or bus mastering[7].

- As a percentage of the total cost, we note that the TCP processing cost increases with a decrease in MTU size. This is explained by the fact that each packet results in the same code being executed on the output (Table 1 shows the

[7]It is noteworthy that the new CardBus cards do allow DMA and bus mastering and they run at a lower voltage (3.3V rather than 5V) and allowing 32-bit operation. Thus, we expect the overall energy cost of the Kernel – NIC copy cost to be lower for these cards. However, device drivers were not available for these CardBus cards and we were unable to test them for this paper. Furthermore, there are currently very few vendors who manufacture CardBus cards for 802.11b (we could only identify one vendor – D-Link). Most of the CardBus cards are targeted at the 802.11a standard where greater transfer speeds are required to match the 54Mbps transmission speed of the radio.

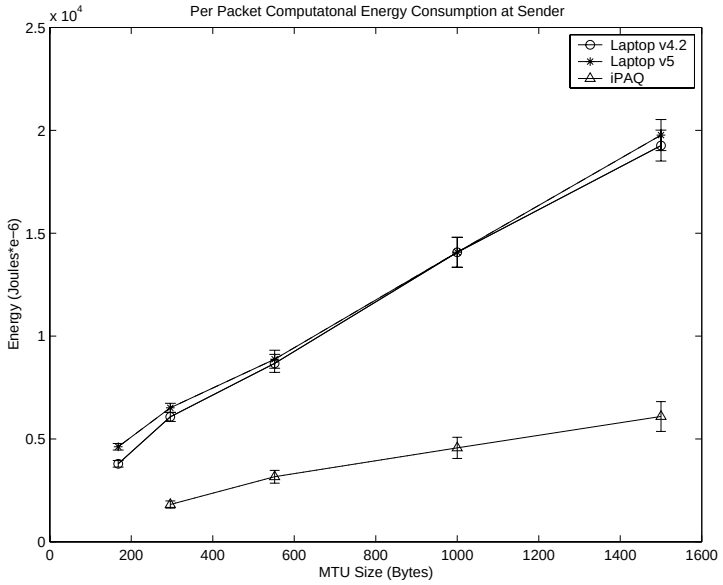

Figure 4: Per packet cost at the sender.

TCP processing cost per packet at the sender as a function of MTU size, as we can see, the processing cost is fairly independent of the MTU size). Thus, smaller MTUs result in more packets sent with the corresponding increase in overall TCP processing cost and hence a larger share of the overall cost. If we view the data in Table 1 together with Figure 4 in which we showed a linear increase in per packet cost as a function of MTU size, we can conclude that while the TCP processing cost is independent of MTU size, the cost of copying packets depends on the MTU size and is the primary contributor to total computational cost of TCP.

- Figure 8 plots the same data for the receiver. We note a similar trend as that at the sender with the kernel-to-NIC copy being the dominant factor. However, we note that in general the TCP processing cost is a smaller percentage here than at the sender. This is understandable because the receiver does less TCP processing than the sender.
- It is interesting to compare the relative costs for our three systems with the results reported in [8] where the authors report on the *time consumed* in the TCP stack: user – kernel copy, NIC-to-kernel copy, and processing costs for 1460 byte packets. The table below summarizes the costs of our studies and [8] for the case of 1460 byte segments:

Relative costs at sender

	NIC-to-kernel copy	kernel-to-user copy	TCP processing + OS	*Metric*
FreeBSDv4.2	78%	16%	6%	*Energy*
FreeBSDv5	77%	15%	8%	”
iPAQ	72%	15%	13%	”
[8]	40%	17%	43%	*Time*

As we can see, the relative cost of the kernel-to-user copy operation appears to be relatively unchanged. However, in the three systems we studied, the relative cost of the NIC-to-kernel copy is far greater than in [8]. We can conclude that over the past decade while processors have gotten very efficient, the bus architecture has not kept pace and the implementation of TCP has gotten more efficient (using some techniques described in [8]). Thus, the relative cost of TCP processing (as a fraction of total cost of a TCP connection) today is far smaller than a decade ago.

5.3 Breakdown of TCP Processing: *Checksum, TO, TD, ACK*

As noted at the start of the previous section, the TCP processing cost at the sender and at the receiver is made up of several components. We obtained a breakdown of this processing cost at the sender and receiver into three major categories:

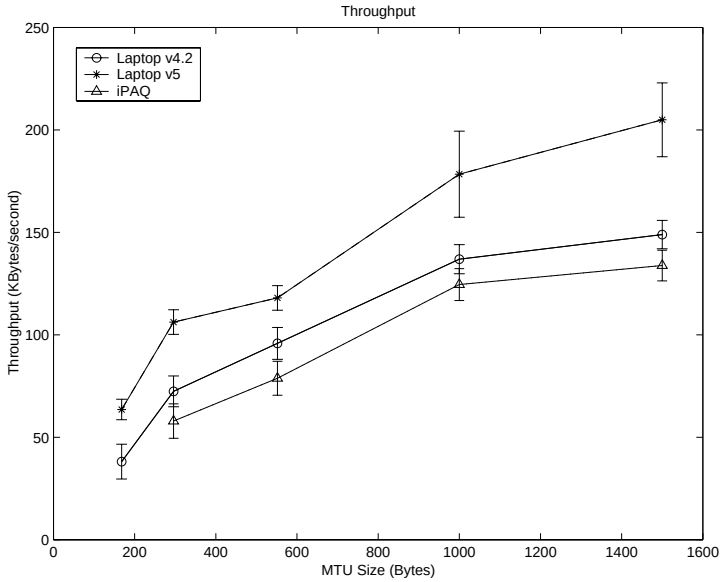

Figure 5: Throughput for the three systems.

checksum, ACK, and "other", where the "other" piece corresponds to congestion window management, RTO computation, TCP (Transmission Control Block) lookup, etc. We could not decompose the "other" piece further because the resolution of our method was not good enough to give statistically significant data. We note, however, that we did measure the TO and TD energy cost at the sender.

5.3.1 Checksum cost

The checksum cost was measured directly by extracting the checksum code from the implementation in the kernel and measuring the current draw of the laptop/iPAQ when running this code repeatedly. Table 1 provides the result of our measurement. In the table we provide the checksum cost as a percentage of the packet processing cost (this cost excludes the copy costs as well as any radio cost). In the case of the laptop, we see that the checksum computation accounts for approximately 30% of the packet processing cost whereas in the iPAQ it accounts for between 17 – 20% only. In view of the fact that the iPAQ runs a more power efifcient processor at a slower clock speed, this result is not surprising. Interestingly, the cost appears to be the same for the two versions of the FreeBSD kernel thus indicating use of the same algorithm. [8] also reports on the cost of checksum processing for 1460 byte packets. The relative cost of checksum processing (in terms of time) as a fraction of TCP processing cost is 35% – this cost is very similar to the relative checksum cost for the two FreeBSD versions.

5.3.2 Determining TO, TD and ACK Costs

In this section, we focus our attention on characterizing the processing cost of TCP's retransmission mechanism and on determining the cost of processing ACKs at the sender[8]. Directly measuring the cost of ACKs (processing plus copy cost from the NIC to the kernel) is difficult because the actual amount of ACK code is very small (less than ten lines). Thus, we measured the ACK cost by measuring the cost of duplicate ACKs that occur in the event of lost packets. We therefore folded the ACK cost measurement in with the measurement of TO and TD events.

When packets are lost, the TCP sender recovers from the loss by either the fast retransmit mechanism or via slow-start after a timeout occurs. In order to compute the cost of responding to TD or TO events, we needed to use indirect methods because it is virtually impossible to directly measure the current draw when either of these events occur (the timescale is too small to allow the multimeters, which have a resolution of one millisecond, to capure the data reliably).

[8]Since almost all of the ACK cost is explained by the NIC-to-kernel copy cost, we observed that the ACK cost at the sender was the same as the cost at the receiver (within statistical error).

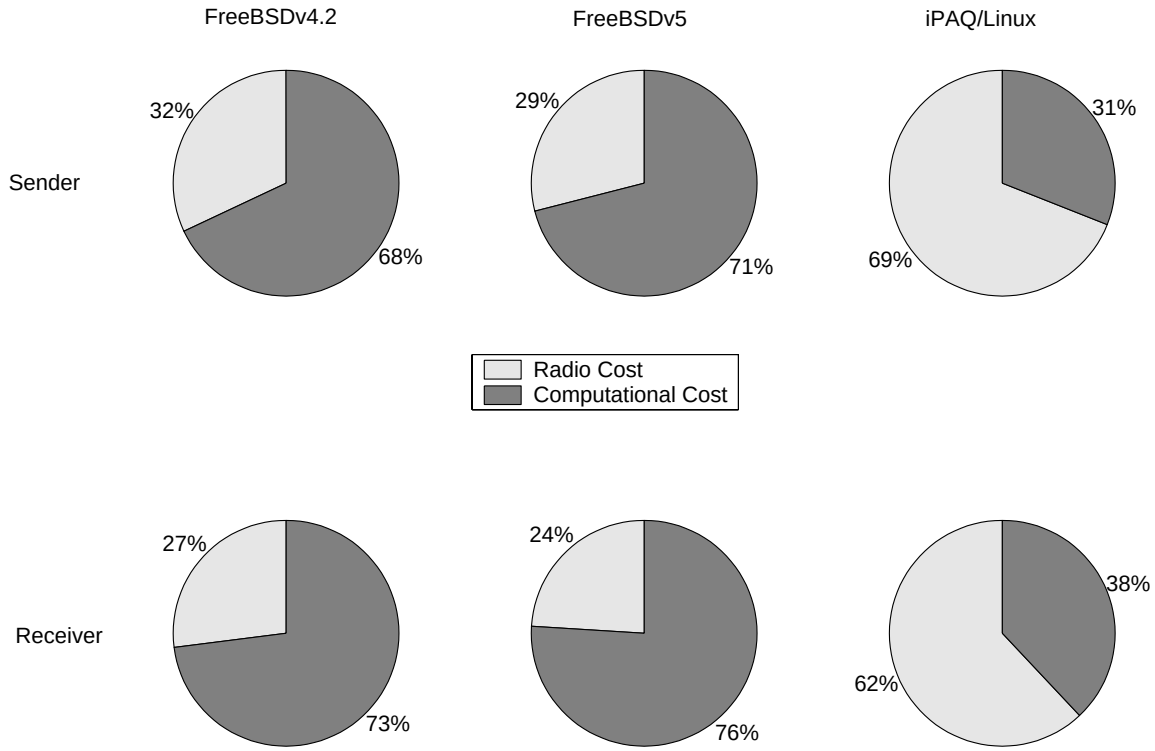

Figure 6: Radio cost and computational cost as a percentage of total cost.

The method we adopted consisted of the following steps:

1. First, we included counters inside the TCP/IP code to count how many TD and TO events occurred during a session. We also separately counted the number of Duplicate ACKs received (excluding triple duplicate ACKs).

2. We ran Dummynet [23] at a node in the path of the connection so that we could enforce losses. This was needed in order to force TO and/or TD events at the sender.

3. Let $E(1MB, MTU)$ denote the computational cost of TCP at the sender for transmitting 1 MByte of data *without loss* (this corresponds to the data discussed in section 5.1) using a given MTU. Let $E^L(1MB, MTU)$ denote the computational energy cost when there were losses created by Dummynet. Then, $E^L(1MB, MTU) - E(1MB, MTU)$ gives us the total retransmission cost for the session. Specifically,

$$E^L(1MB, MTU) - E(1MB, MTU) \quad = \text{TO \& TD Processing Cost} + \text{Duplicate ACK copy \& processing cost} + \text{Kernel} - \text{NIC Copy Cost for Retransmitted Packets}$$

4. We used tcptrace and tcpdump to monitor all packets from the flow and thus determined the actual number of retransmitted packets (say p). In section 5.2 we had measured the kernel-to-NIC copy cost of packets. We thus determine the energy (ϵ) for processing TO, TD, and ACKs as,

$$\epsilon = E^L(1MB, MTU) - E(1MB, MTU) - p * \text{Kernel-to-NIC Copy Cost for Reransmitted Packets}$$

5. Recall that we count the number of times TO, TD, and Duplicate ACKs that occur during a connection. We can thus write,

$$\epsilon = C_{TD} \times \text{TD Cost} + C_{TO} \times \text{TO Cost} + C_{ACK} \times \text{Dup ACK Cost} \quad (2)$$

where C_{TD}, C_{TO}, C_{ACK} denote the value of the counters for each of these events.

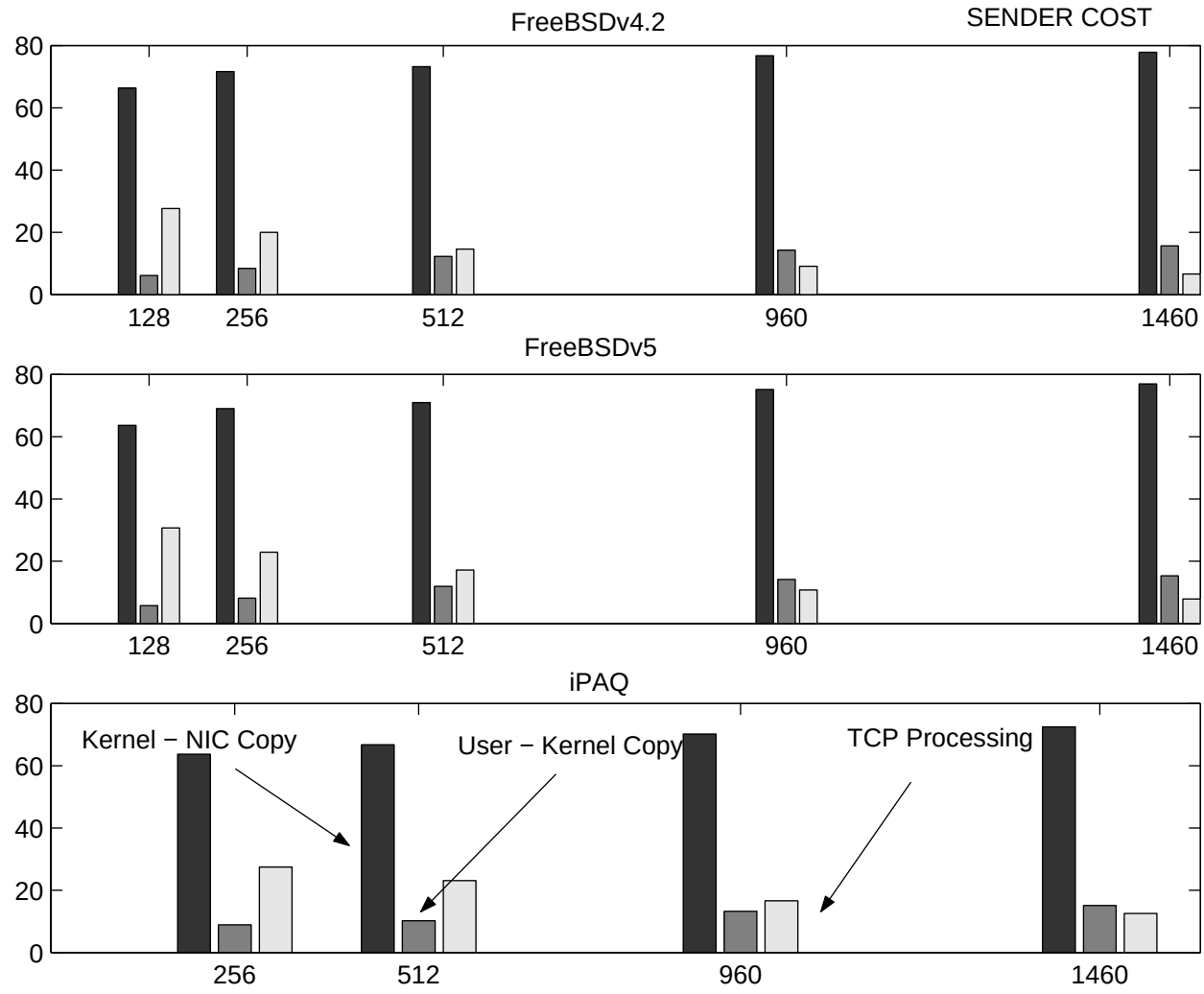

Figure 7: Relative costs at sender for sending 1 MByte data.

6. We ran several experiments with different loss rates and obtain several sets of linear equations (2). These equations are then solved in sets of three to obtain values for the processing cost of TD and TO, and the cost of ACKs. We ran over 100 such experiments in order to obtain enough data to compute 95% confidence intervals.

We summarize the results of these studies in Table 2. We can make the following observations:

- The cost of TO and TD processing is significantly greater than the cost of normal TCP packet processing (see Table 1).
- The ACK cost is fairly high and is almost entirely made up of the cost of copying the ACK from the NIC card to the kernel (Figure 12 shows the relationship between copy costs and copy size). We observed that the ACK cost, as measured using equation (2), is almost the same as the cost of copying the ACK to the kernel from the NIC. Thus, we ignore the the energy draw for the execution of the 9 – 10 instructions dealing with ACK processing.

5.4 Validation

In order to validate our measurements presented in the previous sections, we ran experiments in which 10MByte of data was sent to a remote host at the other side of the country. We measured the total energy consumed by the connection and

	MTU	1500	1000	552	296	168
Laptop (FreeBSDv4.2)	TCP processing (μJ)	1,589	1,588	1,583	1,577	1,513
	Cksum Cost (μJ)	514	504	463	444	424
	%	32.33%	31.73%	29.27%	28.14%	28.05%
Laptop (FreeBSDv5)	TCP processing (μJ)	1,546	1,514	1,520	1,489	1,416
	Cksum Cost (μJ)	512	502	462	443	422
	%	32.13%	33.14%	30.42%	29.72%	29.81%
iPAQ (Linux)	TCP processing (μJ)	761	758	730	726	
	Cksum Cost (μJ)	153	143	130	126	
	%	20.07%	18.85%	17.78%	17.32%	

Table 1: Sender-side TCP processing cost and checksum processing cost.

		MTU 1500	95% CI	MTU 552	95% CI
FreeBSDv4.2 (μJ)	TD Processing Cost	2,406	190	2,402	131
	TO Processing Cost	2,710	33	2,592	123
	ACK Cost (Mainly Copy)	1,209	36	1,171	11
FreeBSDv5 (μJ)	TD Processing Cost	2,380	102	2,379	32
	TO Processing Cost	2,660	92	2,529	52
	ACK Cost (Mainly Copy)	1,188	17	1,164	5
iPAQ (μJ)	TD Processing Cost	1,051	162	1,015	132
	TO Processing Cost	1,302	107	1,161	15
	ACK Cost (Mainly Copy)	176	25	175	17

Table 2: Summary of TO and TD processing costs and ACK Costs.

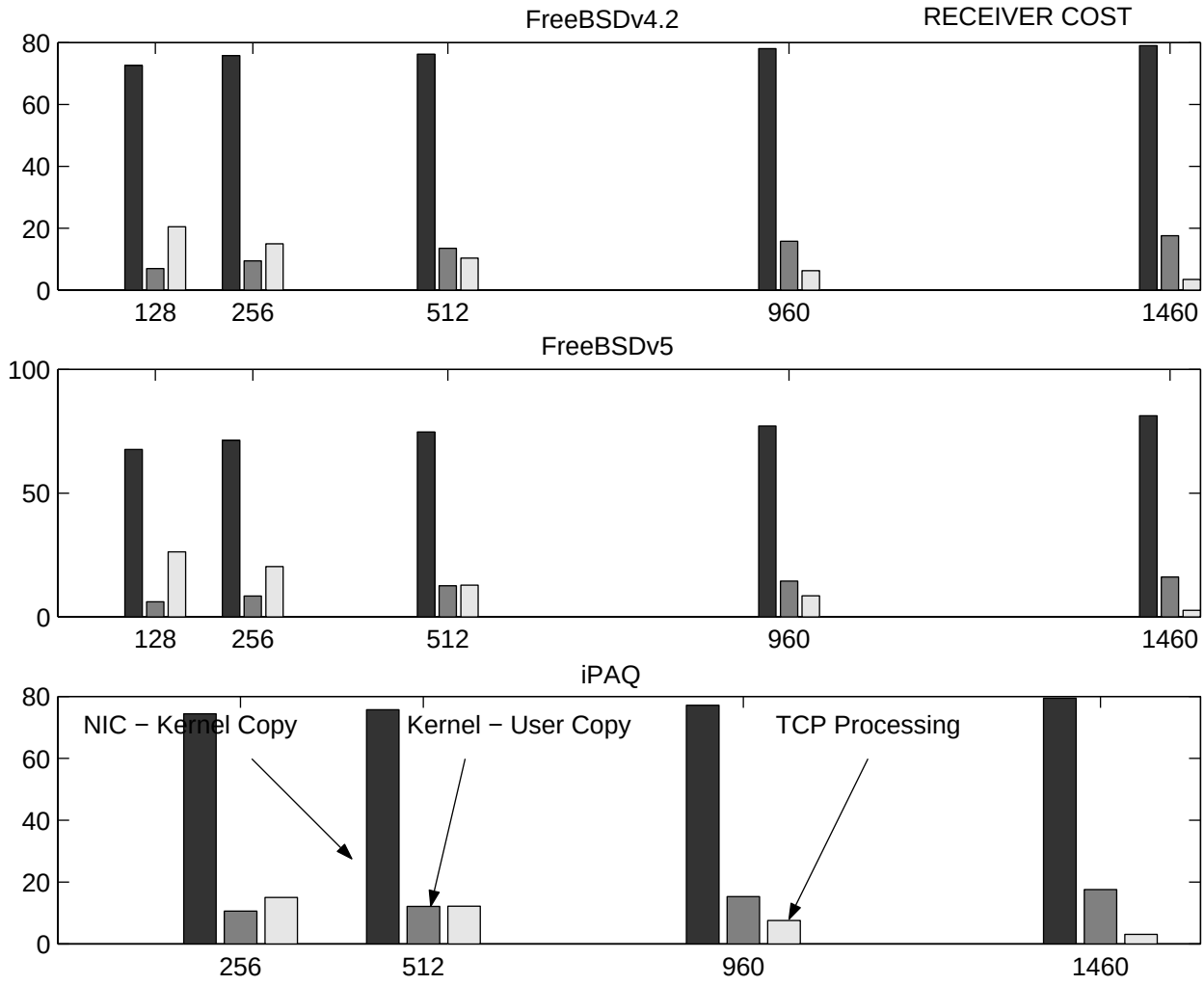

Figure 8: Relative costs at receiver for receiving 1 MByte data.

computed the estimated energy as well. The estimated energy was detetermined by simply using the energy costs derived in sections 5.2 and 5.3:

$$\begin{aligned}\text{Estimated Energy(10MB,MTU)} = \quad & \text{user-to-kernel Copy 10MByte} + n \times \text{Normal TCP Processing Cost} \\ & +(n + C_{TD} + C_{TO}) \times \text{Kernel – NIC Copy for given MTU size} \\ & +C_{TD} \times \text{TD Cost} + C_{TO} \times \text{TO Cost} + \text{Num ACKs} \times \text{ACK Cost}\end{aligned}$$

where n is the number of packets sent (not counting retransmissions) and MTU indicates the MTU size used. For the validation experiments we only used MTU sizes of 1500 and 552. The actual values used for the various costs are summarized in Table 3. Table 4 displays some representative measured and estimated energy values for the case when the MTU was 1500 (The DUP ACK column counts the duplicate ACKs only – the total number of ACKs is the number of duplicate ACKs plus approximately the number of congestion windows transmitted). As we can see, the difference between the estimated values and the actual measured values is very small (less than 0.1%). Similar results were obtained for a MTU of 552.

5.5 Summary

We can summarize the results of this section as follows:

	FreeBSDv4.2	
μJ	MTU 1500	MTU 552
Normal TCP Proc.	1,589	1,583
TD Cost	2,406	2,402
TO Cost	2,710	2,592
ACK Cost	1,209	1,171
Kernel – NIC Copy	16,251	6,744
user-to-kernel Copy	1417	344
	FreeBSDv5	
μJ	MTU 1500	MTU 552
Normal TCP Proc.	1,546	1,520
TD Cost	2,380	2,379
TO Cost	2,660	2,529
ACK Cost	1,188	1,164
Kernel – NIC Copy	15,207	6,293
user-to-kernel Copy	3015	1058
	iPAQ	
μJ	MTU 1500	MTU 552
Normal TCP Proc.	761	726
TD Cost	1,051	1,015
TO Cost	1,302	1,161
ACK Cost	176	175
Kernel – NIC Copy	4,413	2,109
user-to-kernel Copy	916	321

Table 3: Parameters used in validation study.

TO	TD	Dup ACK	Total Computed (μJ)	Total Measured (μJ)	Difference %
FreeBSDv4.2					
58	12	455	112,916,459	112,830,704	0.08%
223	493	1416	121,109,506	121,747,456	0.52%
47	8	412	112,685,425	112,719,544	0.03%
11	1	110	111,783,380	111,793,760	0.01%
135	78	629	114,759,417	115,298,112	0.47%
FreeBSDv5					
45	10	371	112,473,503	112,624,048	0.13%
29	11	233	112,046,029	112,107,487	0.05%
168	125	1,303	117,798,661	118,682,406	0.74%
372	48	2,348	121,252,172	122,318,069	0.87%
39	17	288	112,394,444	112,304,637	0.08%
iPAQ					
6	4	119	39,532,618	39,534,225	0.005%
45	75	702	39,857,329	39,890,940	0.08%
14	77	719	39,793,196	39,828,573	0.09%
16	76	700	39,792,627	39,813,518	0.05%
20	10	287	39,604,594	39,616,089	0.03%

Table 4: Sample validation results (MTU 1500 Bytes).

- The prmary contributor to TCP's computational cost is the kernel – NIC copy operation accounting for almost 70% of the total cost.
- The TD, TO, and ACK processing costs can be determined by using an equation based approach. The cost of TD and TO processing is approximately 60% greater than the normal processing cost of TCP (on a per packet basis).
- The implementation of the device driver for the NIC card makes a significant difference in the kernel-to-NIC copy cost as well as in the overall throughput seen.

6 Techniques to reduce the energy cost of TCP

In order to reduce the computational cost of TCP without modifying the architecture of the node, we examine two avenues: eliminating the user-to-kernel copy cost and minimizing the kernel-to-NIC copy cost. The first cost can be eliminated by employing *zero copy* where the user data is copied directly from the socket layer to the NIC. Minimizing the kernel-to-NIC copy involves putting more functionality in the NIC card. We discuss the impact of implementing zero copy in the next section and we describe two approaches for minimizing the kernel-to-NIC copy cost in section 6.2.

6.1 Using Zero Copy

We used zero copy in FreeBSDv5 and re-ran the experiments described in section 5 above. Figure 9 plots the total computational energy cost (equation (1)) at the sender and receiver. As we can see, using zero copy does reduce the total energy cost at both the sender and at the receiver. As indicated in the figure, this reduction is as much as 14% when using the maximum MTU size. Figure 10 plots the throughput with and without zero copy. Interestingly, we see a significant improvement in throughput when using zero copy. Thus, using zero copy not only reduces the computational energy but it also increases throughput significantly.

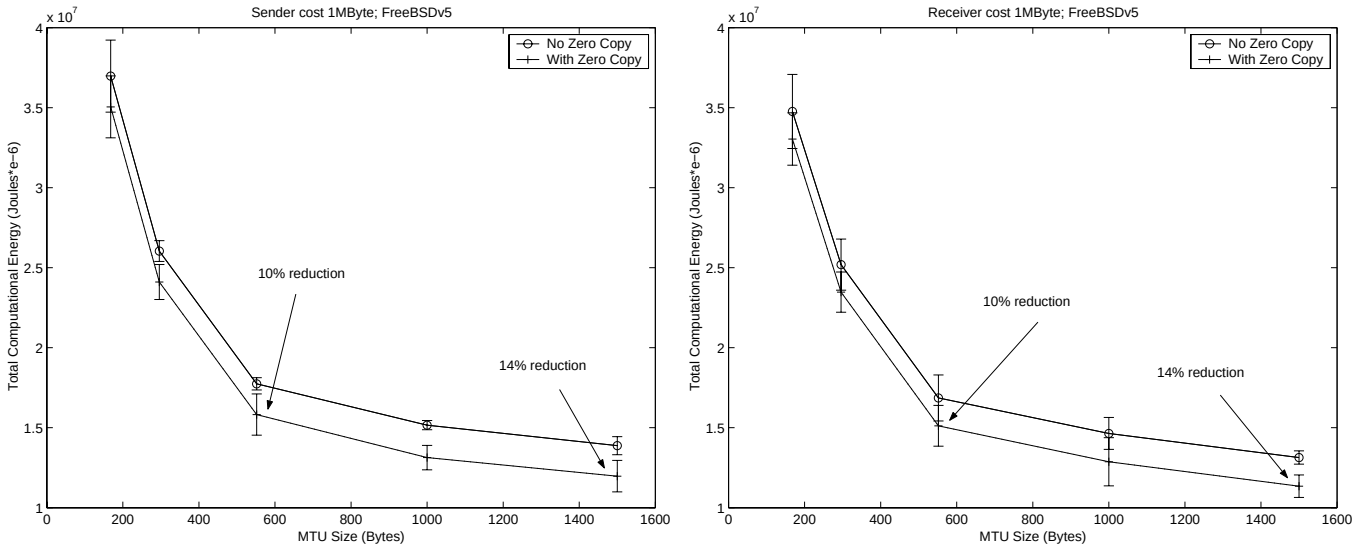

Figure 9: Total computational energy costs with and without zero copy.

6.2 Reducing the kernel-to-NIC copy cost

The kernel–NIC copy cost is by far the largest contributor to TCP's computational energy cost thus we need to develop mechanisms to reduce this cost within the context of the given hardware[9]. We have investigated two complimentary

[9] As noted earlier, using a CardBus card instead of a PCMCIA card will reduce the cost to some extent due to the lower voltage used (3.3V), the wider bus (32-bit), and DMA.

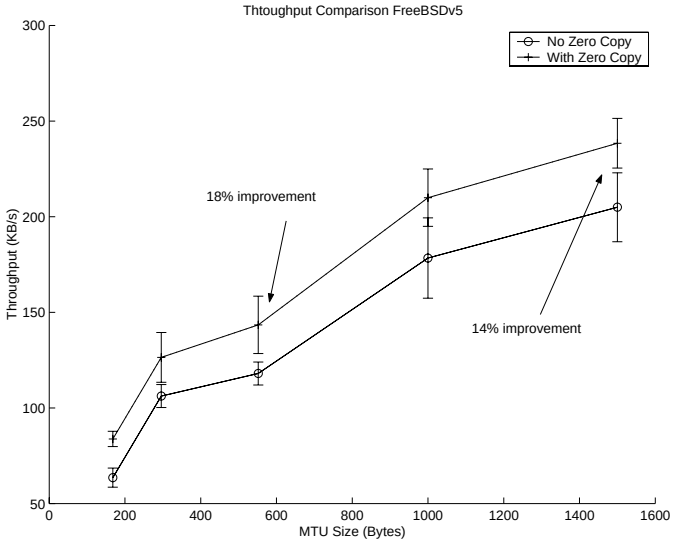

Figure 10: Throughput with and without zero copy.

mechanisms that will enable us to reduce this cost:

1. Maintain TCP's send buffer on the NIC card itself, and
2. Maximize the data transfer size from the kernel to the NIC card.

If we maintain TCP's send buffer on the NIC card, the benefit we get is that we save significant amount of energy on all retransmissions since the packets will not need to be copied from the kernel to the NIC card again. Refer to Table 3 where we give a breakdown of the cost of processing TO and TD events as well as the copy costs. To take an example, if we are running FreeBSDv5, in the case of a TO, the total cost of retransmitting the packet is $2,660 + 15,207\mu$J. However, if the packet is already present in the NIC card, the cost of this TO will only be $2,660\mu$J. Thus, we see significant savings when there are many retransmitted packets as can happen in wireless environments. Note, however, that the kernel needs to have a way to tell the NIC card to retransmit a specific packet from the send buffer. Also, if the host has several open TCP connections, a separate buffer will need to be maintained for each connection.

We ran experiments in which an intermediate node was configured to drop packets using Dummynet. Three loss rates were used – 1%, 5%, and 10% and we counted the number of TO and TD events for each run. We used the data in Table 3 to then estimate the extent of energy savings possible *if* the send buffer was kept at the NIC card. Figure 11 plots the energy savings possible (the righthand plot) for the FreeBSDv4.2 system. As we can see from the left-hand plot, at a low loss rate, the number of TO and TD events is small (for all MTU sizes) and thus the energy savings are approximately 1%. However, at higher losses, the energy savings are more significant – 4% at a 5% loss rate and 10% at a 10% loss rate.

The second mechanism we looked at for reducing the cost of kernel – NIC copy costs has to do with the data copy size. Figure 12 plots the kernel-to-NIC copy cost for copying 1MByte of data as a function of the individual data chunks copied. As is clear from this figure, it is better to copy data to the NIC card in larger chunks since this reduces the total number of interrupts and context switches. In fact, the difference between using 1460 bytes versus the smallest size is *2X* for FreeBSD and *1.4X* for the iPAQ. Thus, in order to reduce the cost of the kernel-to-NIC copy, it is best to use the largest data size possible for this copy operation[10]. However, there is a problem with this approach. Typically, each write to the NIC card corresponds to one packet. Thus, the NIC card automatically knows about packet boundaries. If, however, we copy several packets at once (say the whole send buffer) then we need to have some way of informing the NIC card about packet boundaries. In order to implement the two mechanisms described in this section, the device drivers need to be modified in order to allow send buffers to be maintained on the card and a way for the kernel to inform the card of packet boundaries and which packets to retransmit.

[10]We did not experiment with even larger data copy sizes. However, the expectation is that the energy needed will be smaller. The only restriction on maximizing the data copy size is the hardware limitation on how long a device can capture the bus.

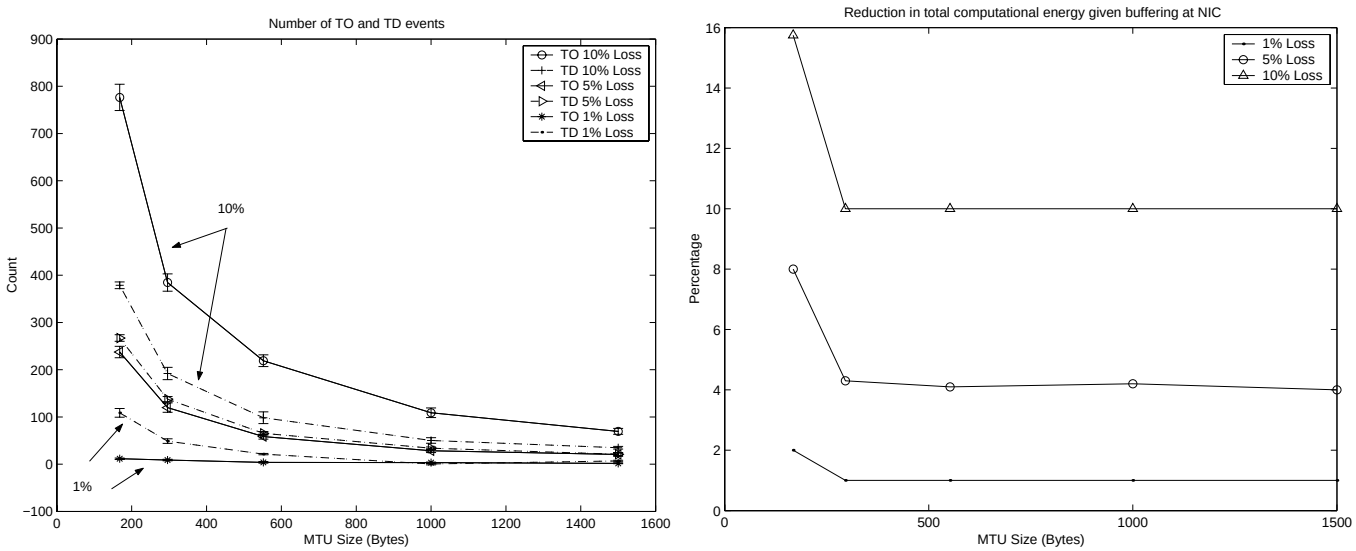

Figure 11: Reduction in total computational energy with NIC-based window.

6.3 Discussion

In this section we have descibed three mechanisms that can be used to reduce the cost of TCP connections. If we put these together, we can estimate the overall energy savings possible. Let us say that the optimum MTU size to use (given channel error rates, etc.) is 552 bytes. Then, we save 10% using zero copy, another 4 – 10% by maintaining the send buffer on the NIC card, and, if we copy data in large chunks (say 1460 byte chunks) then we save 15% (of the kernel – NIC copy cost) or 10% of the total cost. Thus, we can reduce the overall cost at the sender by between 24% and 30% for an MTU size of 552.

It is interesting to note that the techniques we describe above for reducing energy consumption are similar to the ones described by various authors for making TCP support high-speed networking!

7 Conclusions

We developed measurement techniques to allow us to indivudually measure the energy cost of separate logical operations performed in TCP/IP at the sender and receiver. We then used these measurements to develop approaches in software that will allow us to reduce the cost of TCP processing by between 20% and 30% in most cases. These approaches provide us with valuable information on how to modify device drivers and the architecture of NIC cards for mobile computing. Finally, our measurements can be used to enhance simulators such as NS-2 to include node-level energy models. This, in combination with existing radio energy models, will give researchers the capability to study energy consumption in arbitrary wireless networks.

References

[1] Ns-2: http://www.isi.edu/nsnam/ns/, 2002.

[2] Mark Allman, Chris Hayes, Hans Kruse, and Shawn Ostermann. Tcp performance over satellite links. In *5th International conference on telecommunication systems*.

[3] D. C. ANderson, J. S. Chase, S. Gadde, A. J. Gallatin, K. G. Yokum, and M. J. Feeley. Cheating the i/o bottleneck: Network storage with trapeze/myrinet. In *Proceedings 1998 USENIX Annual Technical Conference*, pages 143 – 154, June 1998.

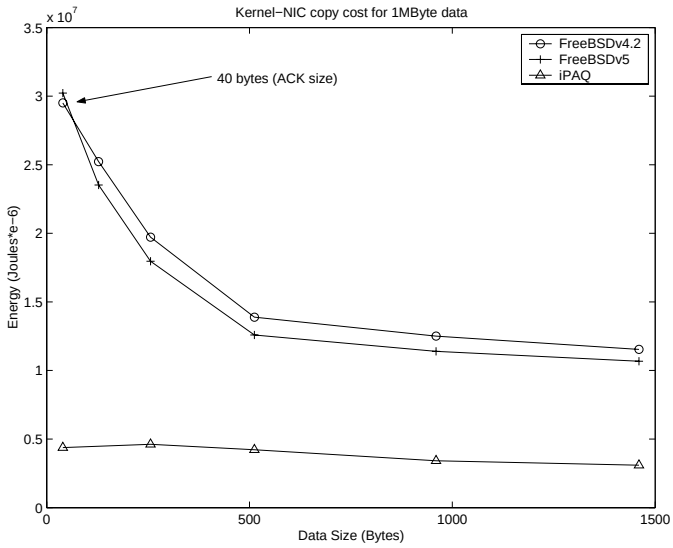

Figure 12: Kernel–NIC copy as a function of copy size.

[4] R. Bruyeron, B. Hemon, and L. Zhang. Experimentations with tcp selective acknowledgement. *ACM Computer Communications Review*, 1998.

[5] J. Chase, A. Gallatin, and K. Yocum. End system optimizations for high-speed TCP. *IEEE Communications Magazine*, 39(4):68–74, 2001.

[6] H. K. Jerry Chu. Zero-copy TCP in solaris. In *USENIX Annual Technical Conference*, pages 253–264, 1996.

[7] David Clark and David Tennenhouse. Architectural considerations for a new generation of protocols. In *ACM SIGCOMM'90*, September 1990.

[8] David D. Clark, Van Jacobson, John Romkey, and Howard Salwen. An analysis of tcp processing overhead. *IEEE Communications Magazine*, 27(6), June 1989.

[9] Zubin D. Dittia, Guru M. Parulkar, and Jerome R. Cox. The apic approach to high performance network interface design: Protected dma and other techniques. In *IEEE INFOCOM'97*, 1997.

[10] P. Druschel, M. Abbot, M. Pagels, and L. L. Peterson. Network subsystem design. *IEEE Network*, 7(4):8 – 17, July 1993.

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

[12] Laura Feeny and Martin Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *Proceedings INFOCOM 2001, Anchorage, Alaska*, 2001.

[13] Paul Gauthier, Daishi Harada, and Mark Stemm. Reducing power consumption for the next generation of pdas: It's in the network interface. In *Proceedings of MoMuC'96*.

[14] Tim Henderson and Randy Katz. Transport protocols for internet-compatible satellite networks. In *IEEE Journal on Selected Areas of Communications*, February 1999.

[15] Hsiao-Keng and Jerry Chu. Zero-copy tcp in solaris. In *USENIX 1996 Annual Technical Conference*, January 1996.

[16] http://www.sycard.com. Sycard technologies, pccextend 140 cardbus extender, July 1996.

[17] K. Kleinpaste, P. Steenkiste, and B. Zill. Software support for outboard buffering and checksumming. In *ACM SIGCOMM'95*, August 1995.

[18] Shawn Osterman. tcptrace: Tcp dump file analysis tool. http://masaka.cs.ohiou.edu/software/, 2002.

[19] M. Srivastava P. Lettieri, C. Schurgers. Adaptive link layer strategies for energy efficient wireless networking. In *Wireless Networks*, volume 5, pages 339 – 355, 1999.

[20] Jitedra Padhye, Victor Firoiu, Don Towsley, and Jim Krusoe. Modeling TCP throughput: A simple model and its empirical validation. In *ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 303–314, Vancouver, CA, 1998.

[21] Craig Partridge. *Gigabit Networking*. Addisson-Wesley, 1993.

[22] M. Rangarajan, A. Bohra, K. Banerjee, E. V. Carrera, R. Bianchini, and L. Iftode. Tcp servers: Offloading tcp processing in internet servers. design, implementation, and performance. Technical Report dcs-tr-481, Rutgers University, Department of Computer Science, March 2002.

[23] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1), January 1997.

[24] P. Shivam, P. Wyckoff, and D. Panda. Emp: Zero-copy os-bypass nic-driven gigabit ethernet message passing. In *SC2001*.

[25] B. Sikdar, S. Kalyanaraman, and K. S. Vastola. An integrated model for the latency and steady-state throughput of tcp connections. *Performance Evaluation*, 2001.

[26] H. Singh and S. Singh. Energy consumption of tcp reno, newreno, and sack in multi-hop wireless networks. In *ACM SIGMETRICS 2002*, June 15 - 19 2002.

[27] V. Tsaoussidis, H. Badr, X. Ge, and K. Pentikousis. Energy/throughput tradeoffs of tcp error control strategies. In *In Proceedings of the 5th IEEE Symposium on Computers and Communications, France*, July 2000.

[28] Gary R. Wright and W. Richard Stevens. *TCP/IP Illustrated, Volume I (The Protocols)*. Addison Wesley, 1994.

[29] M. Zorzi and R.R. Rao. Error control and energy consumption in communications for nomadic computing. In *IEEE Transactions on Computers*, March 1997.

[30] M. Zorzi and R.R. Rao. Is tcp energy efficient? In *Proceedings IEEE MoMuC*, November 1999.

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