Computational Energy Cost of TCP

Bokyung Wang
Telecommunications System Division
SAMSUNG Electronics Co., Ltd
Suwon, S. Korea
Email: bk.wang@samsung.com

Suresh Singh
Department of Computer Science
Portland State University
Portland, OR 97207
Email: singh@cs.pdx.edu

***Abstract*— In this paper[1,2,3], we present results from a detailed energy measurement study of TCP. We focus on the node-level cost of the TCP protocol and obtain a breakdown of the energy cost of different TCP functions. We analyze the energy consumption of TCP on two platforms (laptop and iPAQ) and three operating systems (FreeBSD 4.2, 5 and Linux 2.4.7). Our results show 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, ∼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 analyze the 15% TCP processing cost and show that the cost of computing checksums accounts for 20 – 30% of TCP processing cost. Finally, we determine 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 achieved.**

I. 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 has looked at the energy consumption of various wireless interface cards [1], [2], the impact of ARQ (Automatic Repeat reQuest) protocols on overall energy consumption [3], [4], the effect of channel conditions on energy consumption [5], [6], and comparative energy and throughput-based studies of different TCP flavors [7]. 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.

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 book keeping 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: (1) it enables us to develop techniques to reduce the computational energy cost of TCP; (2) 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; (3) our node-level energy models can be incorporated into simulators such as NS-2 to obtain overall energy cost of TCP connections (NS-2 currently only includes the radio energy cost); (4) 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 III presents an overview of TCP processing and describes the key tasks that are energy consuming; section IV details the methodology we followed in our study to estimate the various energy costs; results of our measurement are described in section V; 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 VI; finally, section VII presents our conclusions.

II. Related Work

[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 V-B and V-C) where we draw comparisons with our 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. Following this early work, researchers have developed a variety of techniques to make TCP efficient and fast including: using zero-copy, header prediction, interrupt suppression (where the processor is only interrupted when several packets have been received), using jumbo frames, and integrating checksum computation with copy; and in hardware, techniques such as, checksum offloading, offloading some common path TCP/IP functionality to the

[1]An early version of this paper appeared as a Poster at the ACM SIGMETRICS'03 Conference.

[2]This research was supported by NSF grant ANI-0196043.

[3]This work was done when Bokyung Wang was at Portland State University.

NIC (Network Interface Card), packet aggregation on the NIC card, etc. (see, for example, [9], [10], [11], [12], [13], [14], [15], [16], [17]).

[7] 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 but does not attempt an energy breakdown of different TCP functions. Thus, in this current paper we address a relatively new problem that complements previous work which examined the impact of the wireless environment on TCP's cost.

III. Energy consumption in TCP

For our study we used a laptop and an iPAQ. These devices were equipped with a 802.11b PCMCIA card to provide network connectivity. We used this wireless set up because we wanted to make our results relevant to the mobile computing domain where energy efficiency is desired.

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 [18] connection. In order to determine the computational energy cost of this connection, we identify regions A – B, C – D, etc. where the current draw spikes. The integral of the current draw for these regions (indicated by the box between A – B, for example) gives us the total computational energy cost. Note that this value *does not* include the radio energy costs or the idle energy draw of the laptop. The energy value obtained is thus the pure hardware/software cost of running TCP at the node.

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 [14], 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 per packet (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 kernel-to-NIC 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[4,5]

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. Using this breakdown, the total computational cost of a TCP session, in which D bytes of data are transmitted, can be written as:

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

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. We also note that a similar equation can be written for the receiver.

IV. Methodology

A. 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" 11Mbps 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. *The reasons for turning off power management are:*

- If power management is turned on then the spikes in current draw could result either from TCP code execution alone, or from some artifact of power management (e.g., spinning up the disk), or from a combination of the two. Thus, extracting the actual current draw for TCP alone would be very difficult.

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

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

- Since power management is turned off in our studies, we know that the increase in current draw (A – B, C – D, etc. in Figure 1) correspond to activity relating to TCP computation (we confirmed this by correlating power measurements with tcpdump output).
- The actual power consumed by TCP is determined by *subtracting* the idle power consumption with power management turned off (area between B and C Figure 1) from the total power consumed during a ttcp run (in other words, we only measure the deltas above the idle).

Thus, the high power consumption when power management is off does not affect our results.

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 [2], 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 [2], [1], [7] in which, the 802.11b PCMCIA card is attached to a Sycard PCCextend 140A CardBus Extender [19] that in turn attaches to the PCMCIA slot in the laptop or iPAQ. This extender 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 or iPAQ communicated with a fixed access point and acted as the sender or receiver in various experiments. The other end-point of the connection was a fixed PC. 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)
Laptop	1905	1709	1220	15.08
HP iPAQ	655	630	517	5.05
WaveLAN	263	181	159	5.046

B. 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 1. 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 4).

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 because the resolution of our multimeter is 1 msec). In Figure 1, this means that we sum the current in regions A–B and C–D and subtract out the node's idle current draw ($\sim$1.25 Amps) *over this time period*. For the sake of illustration, let us only consider region A–B. The sum of the current draw (after subtracting the idle current draw) here is 17137.6μAmpere-Second (this is indicated by the box in Figure 1). 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 3 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. Furthermore, there were no disk access costs since we parked the disk. In order to compute the computational energy as described above, we wrote scripts that automatically performed the above algorithm on very large data sets.

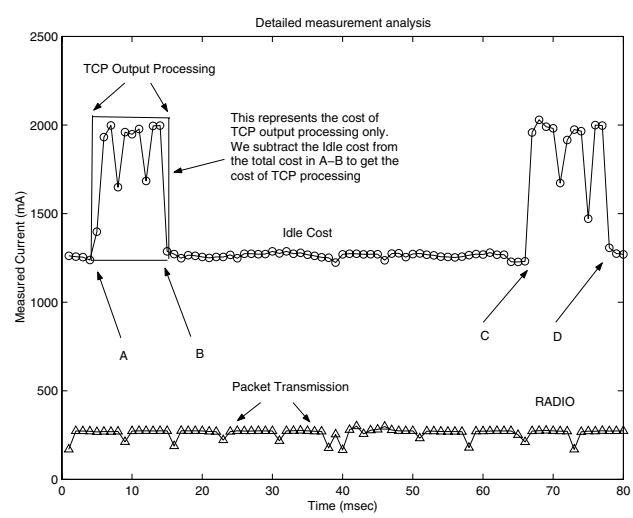

Fig. 1. Example of determining computational energy cost.

C. 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 [20] and tcpdump [18]. The information collected from these tools was used to identify the radio events and appropriately subtract the corresponding radio energy cost from the overall system cost. The parameters we used for the various measurement experiments are illustrated below.

Paramaters	*Values*
TCP window size	16K
RTS/CTS	OFF
MTU Size	168, 296, 552, 960, 1500 bytes
Packet Loss	(1) No default setting (2) 3%, 5%, 7%, 10% using Dummynet (section V-C)

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 [21] at a local node in the connection path and dropped packets from the ttcp connection.

V. 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 [22], in the case of low throughput the idle cost would dominate all others).
- In sections V-B and V-C we provide a breakdown of the total computational energy cost into the major components discussed in section III.
- Finally, in section V-D 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.

A. 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[6,7]. 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 2). *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[8] 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.
- 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 V-B.
- 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 3 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 V-B.

Figure 4 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

[6]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.

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

[8]We were unable to set the MTU to 168 bytes for the iPAQ and thus the graphs for the iPAQ do not show this data point.

		Laptop BSDv4.2	Laptop BSDv5	iPAQ Linux
Sender	Radio Cost	32%	29%	69%
	Compute Cost	68%	71%	31%
Receiver	Radio Cost	27%	24%	62%
	Compute Cost	73%	76%	38%

TABLE I

RADIO COST AND COMPUTATIONAL COST AS A PERCENTAGE OF TOTAL COST.

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).

- 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 (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 III). These results are shown in Table I. 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. This is probably due to the more energy efficient StrongARM processor used in the iPAQs. 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.

B. 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 III. 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 V-C.

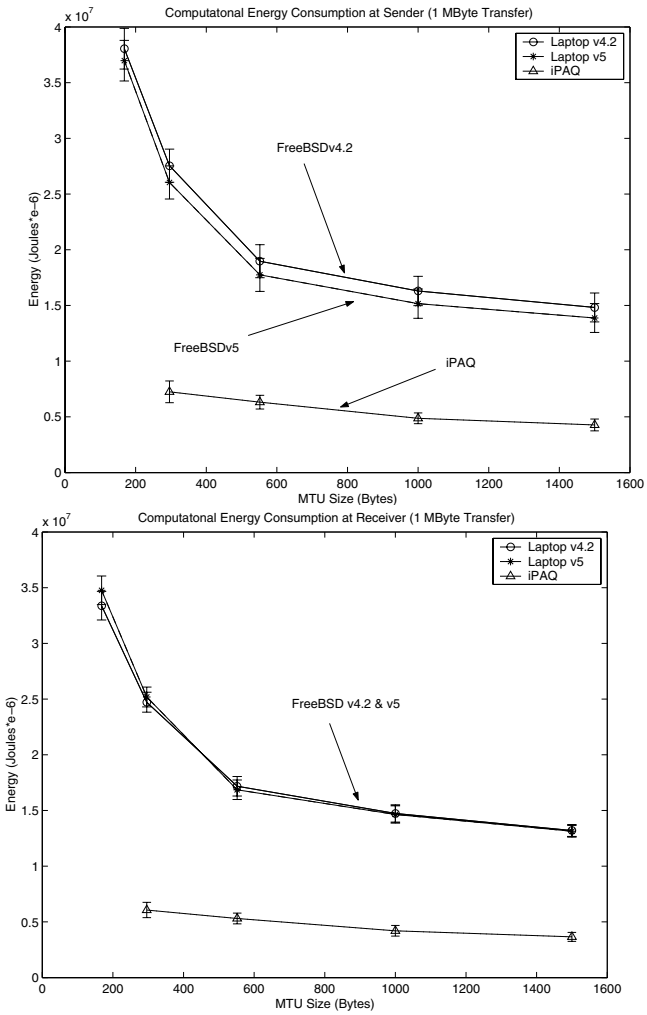

Fig. 2. Total computational energy costs.

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) *in memory*. 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. We subtracted the UDP checksum cost from these values.

Figure 5 shows the relative costs (in %) at the sender of the two copy operations and TCP processing (TD, TO, ACK, etc.) 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 – 80% of the total cost incurred at the sender. To better understand

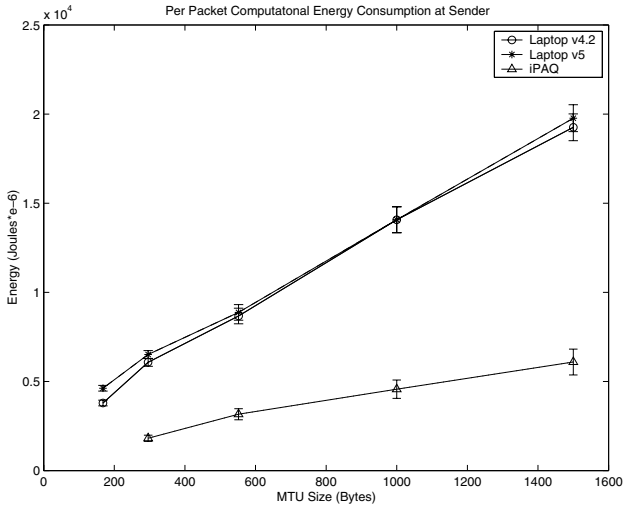

Fig. 3. Per packet cost at the sender.

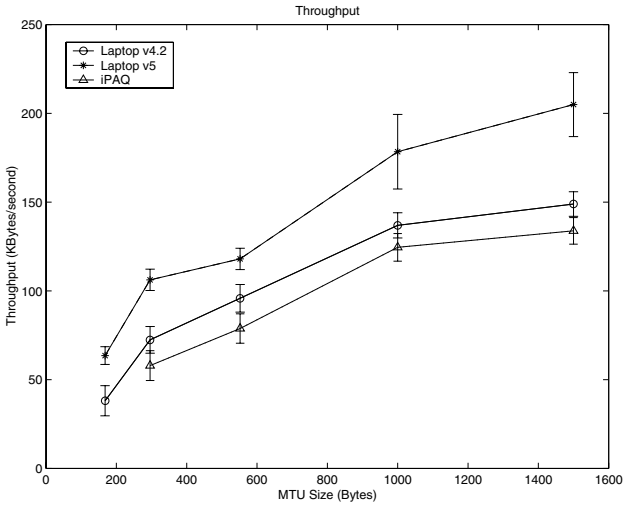

Fig. 4. Throughput for the three systems.

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. We note, however, that CardBus cards do allow DMA and will potentially consume less energy.

- Table II shows the 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.
- As a percentage of the total cost, we note that the *total* TCP processing cost for transmitting 1MByte decreases with an increase in MTU size. This is explained by the fact that each packet results in the same code being executed on the output regardless of 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.
- Figure 5 plots the same data for the receiver. We note a similar trend as that at the sender with the NIC-to-kernel 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:

	BSDv4.2	BSDv5	iPAQ	[8]
NIC-to-Kernel	78%	77%	72%	40%
Kernel-to-user	16%	15%	15%	17%
TCP Proc.	6%	8%	13%	43%
Metric	Energy	Energy	Energy	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] (with the caveat that the metrics used by us and by [8] are different). 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. 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.

C. *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: checksum, ACK, TO, TD, 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. However, we did measure the TO and TD energy cost at the sender.

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 II provides the result of our measurement. 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 efficient 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

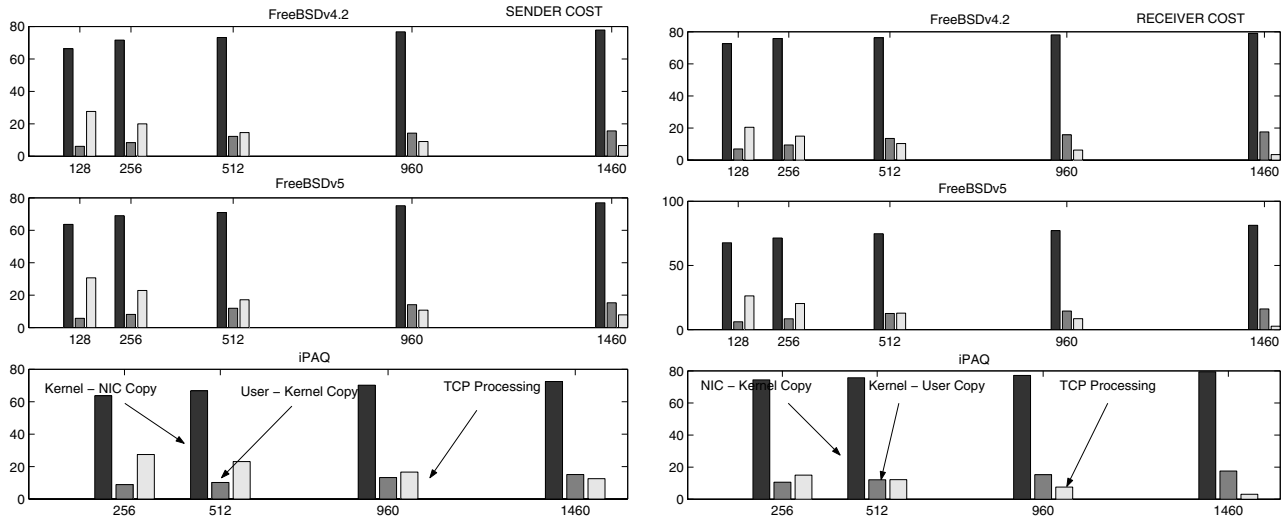

Fig. 5. Relative costs at sender/receiver for sending/receiving 1 MByte data.

MTU	BSDv4.2		BSDv5		iPAQ	
	TCP (μJ)	Cksum (μJ)	TCP (μJ)	Cksum (μJ)	TCP (μJ)	Cksum (μJ)
1500	1,589	514	1,546	512	761	153
1000	1,588	504	1514	502	758	143
552	1,583	463	1,520	462	730	130
296	1,577	444	1,489	443	726	126
168	1,513	424	1,416	422		

TABLE II

SENDER-SIDE TCP PROCESSING COST (TD, TO, ACK, CHECKSUM, ETC.) AND CHECKSUM PROCESSING COST.

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.

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[9]. 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 method we adopted consisted of the following steps:

1) 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 [21] at a node in the path of the connection so that we could enforce losses. Recall that the sender (instrumented laptop/iPAQ) and receiver are local nodes and there are typically few 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 V-A) 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,

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

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 V-B we had measured the kernel-to-NIC copy cost of packets. We thus determine the energy (ϵ) for processing TO, TD, and ACKs as,

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

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

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

[9]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).

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

6) We ran several experiments with different loss rates and obtain several sets of linear equations eqn (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 III. 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 II).
- 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 7 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.

	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	1,417	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	3,015	1,058
	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 IV

PARAMETERS USED IN VALIDATION STUDY.

D. 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 computed the estimated energy as well. The estimated energy was determined by simply using the energy costs derived in sections V-B and V-C:

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

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 IV. Table V 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.

VI. 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 user buffers 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 VI-B.

A. Using Zero Copy

We used zero copy in FreeBSDv5 and re-ran the experiments described in section V above. Figure 6 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.

B. 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. We have investigated two complimentary 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 IV 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

		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 III

SUMMARY OF TO AND TD PROCESSING COSTS AND ACK COSTS.

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 V

SAMPLE VALIDATION RESULTS (MTU 1500 BYTES).

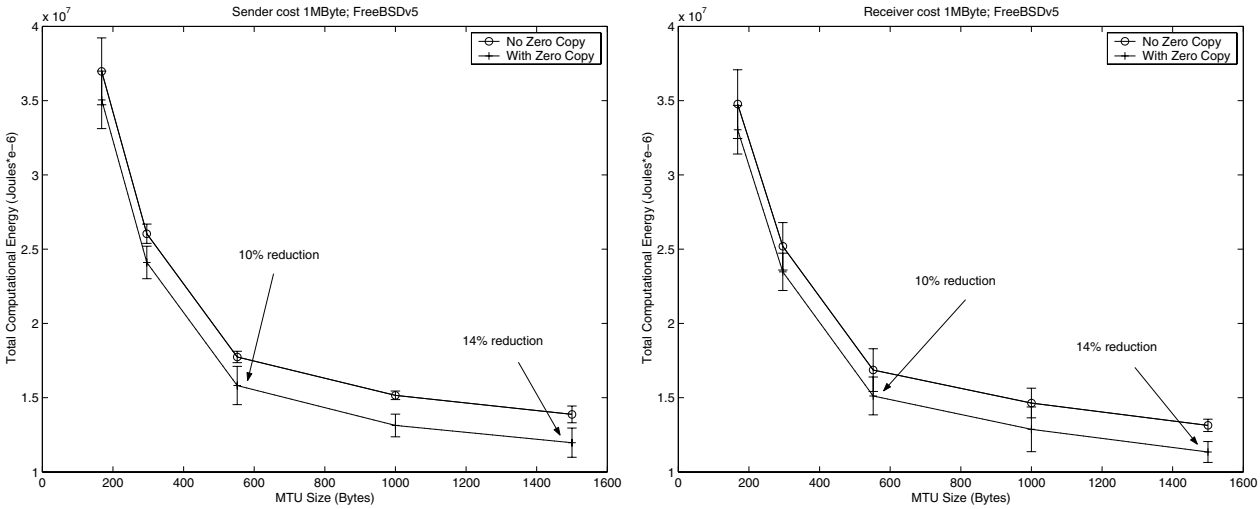

Fig. 6. Total computational energy costs with and without zero copy (at sender).

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.

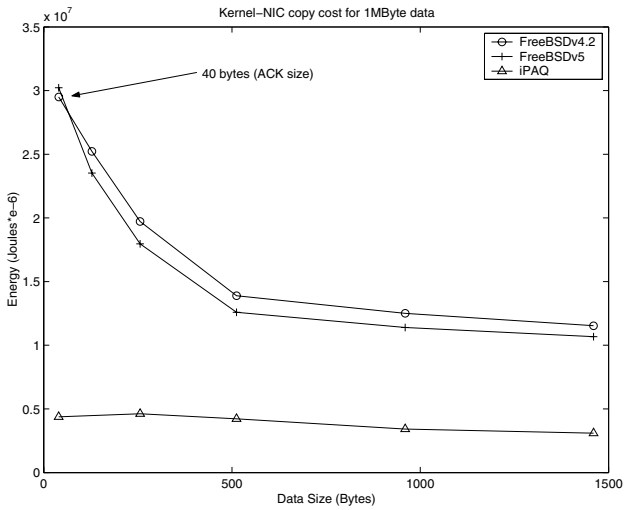

Fig. 7. Kernel–NIC copy as a function of copy size.

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 IV to then estimate the extent of energy savings possible *if* the send buffer was kept at the NIC card. Figure 8 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 7 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.

C. Discussion

In this section we have described 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[11]. 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.

VII. Conclusions

We developed measurement techniques to allow us to individually measure the energy cost of separate functions performed in TCP 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] L. Feeny and M. Nilsson, "Investigating the energy consumption of a wireless network interface in an ad hoc networking environment," in *Proceedings INFOCOM 2001, Anchorage, Alaska*, 2001.

[2] P. Gauthier, D. Harada, and M. Stemm, "Reducing power consumption for the next generation of pdas: It's in the network interface," in *Proceedings of MoMuC'96*.

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

[4] M. S. P. Lettieri, C. Schurgers, "Adaptive link layer strategies for energy efficient wireless networking," in *Wireless Networks*, vol. 5, no. 5, 1999, pp. 339 – 355.

[5] 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.

[6] S. B. et.al, "Energy efficiency and throughput for tcp traffic in multi-hop wireless networks," in *Proceedings INFOCOM 2002, New York, NY*, 2002.

[7] 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.

[8] J. R. David D. Clark, Van Jacobson and H. Salwen, "An analysis of tcp processing overhead," in *IEEE Communications*, vol. 27, no. 6, June 1989.

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

[10] A. G. J. Chase and K. Yocum, "End system optimizations for high-speed tcp," *IEEE Communications Magazine*, vol. 39, no. 4, pp. 68–74, April 2001.

[11] K. B. E. V. R. B. L. I. Murali Rangarajan, Aniruddha Bohra and W. Zwaenepoel, "Offloading tcp/ip processing in internet servers. design, implementation, and performance,," Rutgers University, Department of Computer Science, Tech. Rep. DCS-TR-481, March 2002.

[11]We note that, if the error rate is small, then the maximum MTU size would be used which means that the only time we would save energy is for retransmissions.

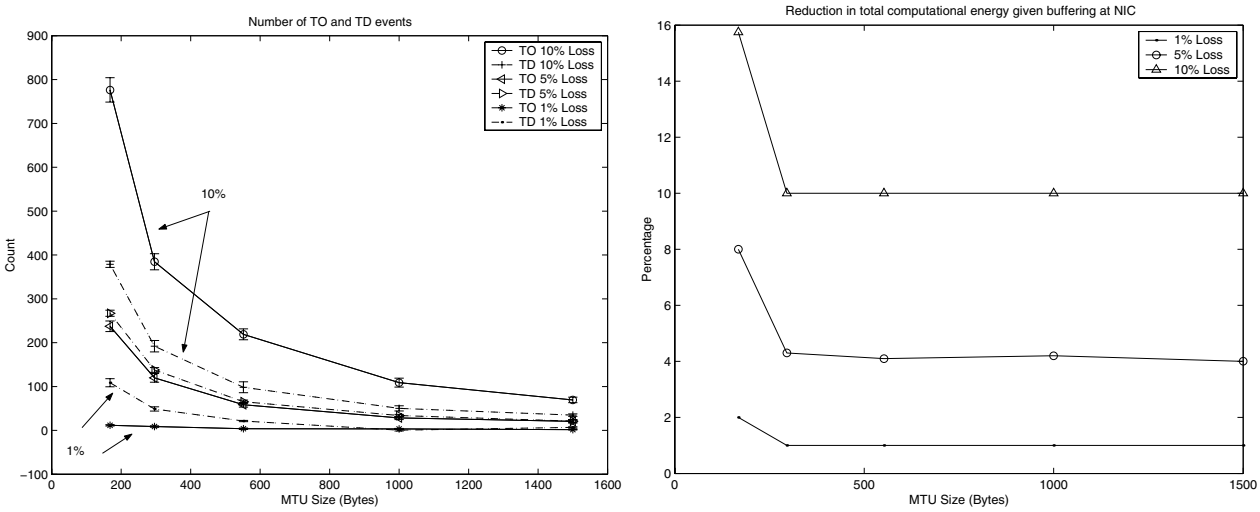

Fig. 8. Reduction in total computational energy with NIC-based window.

[12] S. G. A. J. G. K. G. Y. D. C. Anderson, J. S. Chase and M. J. Feeley, "Cheating the i/o bottleneck: Network storage with trapeze/myrinet," in *Proceedings of the USENIX 1998 Annual Technical Conference*, Berkeley, USA, June 15 – 19 1998, pp. 143 – 154,.

[13] B. Z. K. Kleinpaste, P. Steenkiste, "Software support for outboard buffering and checksumming," in *Proceedings of ACM SIGCOMM'95*, August 1995.

[14] J. C. Hsiao-Keng, "Zero-copy TCP in solaris," in *USENIX Annual Technical Conference*, 1996, pp. 253–264. [Online]. Available: citeseer.nj.nec.com/chu96zerocopy.html

[15] A. M. B. P. M. A. Druschel, P. and L. L. Peterson, "Network subsystem design," *IEEE Network*, vol. 7, no. 4, pp. 8 – 17, July 1993.

[16] G. P. Zubin Dittia and J. Cox, "The apic approach to high performance network interface design: Protected dma and other techniques," in *Proceedings of INFOCOM'97*, April 7 – 11 1997.

[17] D. D. Clark and D. Tennenhouse, "Architectural considerations for a new generation of protocols," in *Proceedings of ACM SIGCOMM'90*, September 1990.

[18] G. R. Wright and W. R. Stevens, *TCP/IP Illustrated, Volume 1 (The Protocols)*. Addison Wesley, 1994.

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

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

[21] L. Rizzo, "Dummynet: a simple approach to the evaluation of network protocols," *ACM Computer Communication Review*, vol. 27, no. 1, January 1997. [Online]. Available: citeseer.nj.nec.com/rizzo97dummynet.html

[22] 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.