A Network-Assisted System for Energy Efficiency in Mobile Devices

Joshua Hare Dheeraj Agrawal Arunesh Mishra Suman Banerjee Aditya Akella
Dept. of Computer Sciences, University of Wisconsin-Madison, 53706
{hare, dheeraj, arunesh, suman, akella}@cs.wisc.edu

***Abstract*—We present the design and implementation of Scepter, a system with explicit infrastructure support to reduce energy consumption and improve battery life of mobile devices. Scepter focuses on effective techniques that can reduce the total number of bits transmitted to communicate the same information from a mobile device to a base station. Scepter combines multiple techniques into a single unified architecture by utilizing a stateful proxy located within the infrastructure close to the wireless base station. Scepter intentionally introduces asymmetries in wireless communication tasks between a mobile device and the stateful proxy to provide greater energy advantages to the mobile device. In this work, we have implemented all capabilities of Scepter as various kernel- and user-level enhancements. Our detailed evaluation demonstrates the performance advantages of Scepter. In various experiments with different wireless conditions and traffic patterns, Scepter improves the energy consumption of these devices between 15% and 54%.**

I. Introduction

Various wireless communication technologies are being integrated into an increasing number of mobile devices, beyond just laptops and cellular phones. Examples of such integration span music players, e.g., iPod Touch, GPS-based navigation tools, e.g., Garmin Nuvi 880, digital cameras, e.g., Nikon S6, electronic readers, e.g., Kindle, and PDAs, thereby adding significantly new capabilities to them. However, as wireless communication is quite a power-hungry operation, vendors are increasingly seeking new ways to make all wireless communication technologies more energy efficient. Over more than a decade, a number of energy efficiency techniques have been developed that can be classified into hardware-based and software-based approaches. The hardware-based approaches focus on improving the energy consumption of the wireless hardware and its RF front-end, while software-based approaches often suggest either protocol-level optimizations or better configuration of device parameters.

In this paper, we propose Scepter, which is a software-only technique, for further improving the energy efficiency of mobile devices when they are communicating with the network infrastructure. Scepter stands for a Stateful proxy-based system for Control overhead Elimination and Payload reduction Through Elimination of Redundancy. In particular,

Hare, Agrawal, Mishra, and Banerjee were supported in part by US NSF awards: CNS-1040648, CNS-0916955, CNS-0855201, CNS-0747177, and CNS-0627589.

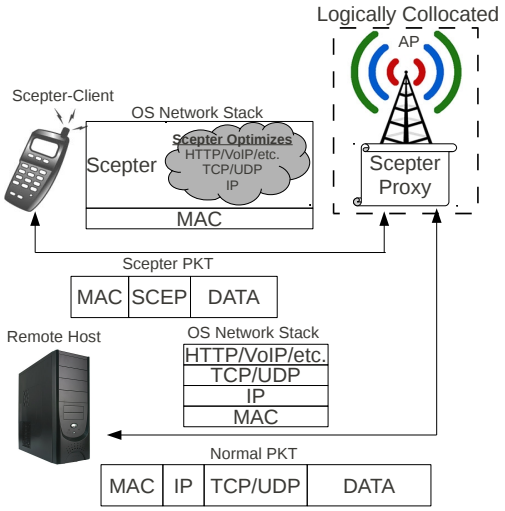

Fig. 1. **Scepter Infrastructure. This figure shows the overall Scepter infrastructure (note that the Scepter proxy does not have to be collocated on the AP). The Scepter proxy is the entity that is communicating with the remote host and can be viewed as the entity that is acting on behalf of the Scepter client. The "SCEP" in the packet is the 1 byte of overhead that Scepter uses for control information. The Scepter protocol is used in communications between the Scepter client and the Scepter proxy.**

Scepter achieves its gains by placing a stateful proxy in the network infrastructure, close to the base station, and assisting the mobile device in reducing the number of bits required to transmit given amount of data. This reduction spans both control messages of existing protocols and data payloads carried in wireless frames. We have developed a system specifically for the popular WiFi technology, which is known to be particularly energy inefficient for mobile devices. For example, in our measurements using multiple mobile phones, we found that the battery drained more than twice as fast when the device is actively transmitting using its WiFi interface, compared to when it is idle.

A. Scepter: Novelty and approach

Scepter is based on the observation that a reduction in the number of bytes transmitted over a wireless interface directly correlates to a reduction in the power consumed by the wireless interface. Scepter reduces the number of bits to communicate the same information between mobile devices and the network infrastructure. While many of the techniques used in Scepter can be beneficial to any wireless

communication scenario, we believe that its greatest value lies in mitigating energy consumption when a mobile client is communicating with the network infrastructure. This is because, in Scepter, we introduce a stateful proxy in the network infrastructure, and offload various communication functions from the mobile client into the proxy. We have created a asymmetric relationship between the mobile client and proxy such that the proxy carries a much greater communication and computation burden. The mobile client will thus offload certain communication tasks to the proxy requiring the proxy to consume more energy on behalf of the mobile client. With this asymmetric relationship we have built upon compression techniques to develop cross-layer optimizations that reduce the number of bits needed to communicate a given amount information between the mobile client and the network infrastructure. These compression techniques, which were not designed with energy efficiency in mind, have been extended and optimized to for vast energy savings. For further energy savings Scepter was designed to be complementary to other energy efficient design techniques such as "Power-Save Mode" where the radio is put in a low power state when not actively transmitting [1], [2], [3], [4].

Figure 1 illustrates the simple energy efficiency architecture of Scepter. The figure shows the logical interaction between a Scepter-capable mobile device and the way it interacts with a remote host through a stateful proxy. In general, the proxy can be placed at different points within the infrastructure. In the context of WiFi, the most convenient point is within the same broadcast LAN such as the Access Point (AP). In our implementation, we actually implement the Scepter proxy within the AP itself.

The Scepter proxy is stateful and is aware of various protocol layers at the mobile device. In particular, we design the proxy to understand the protocol semantics of the IP and the TCP/UDP layers. In addition, it can also help eliminate redundant content in the payload of various application layer protocols, e.g., HTTP, VoIP, etc., in an application-independent way. An implementation of Scepter, therefore, requires kernel level modifications in the mobile device and in the Scepter proxy. The Scepter-capable mobile device establishes state with the proxy for each of its network connections. To manage this state among other things, Scepter adds a very short (1 byte) header for all messages between the mobile device and the proxy. The TCP/IP header fields are completely subsumed in the Scepter header using an enhanced form of Robust Header Compression (RoHC).

B. Key contributions

Our work has the following important contributions:

- *Leverage infrastructure support:* We demonstrate that through infrastructure support we can not only reduce the number of bytes sent per packet but also are able to reduce the total number of packets sent by the elimination of redundant control traffic across multiple layers of the protocol stack.
- *Leveraging asymmetry:* This work also points out that communication tasks can intentionally be designed to be asymmetric, such that the burden falls more on the AP, which has no power constraints, and less on the mobile device. Asymmetry is also used in Scepter to optimize the system for the asymmetric nature of up-link versus down-link network traffic.
- *Complete software implementation with user- and kernel-level modules:* We have implemented a full prototype version of Scepter for Linux-based systems. This implementation involves various cross-layer optimizations ranging from the session layer all the way down to the network layer. We further quantify the energy savings of our system in our evaluation.

In the following sections we discuss the motivation and design of Scepter, and evaluate its performance through detailed experiments, comment on related work, and conclude various advantages and disadvantages of our proposal.

II. Motivation — Non-uniform bit errors and short packets

The design goal of Scepter is to reduce the energy costs of active communication by reducing the number of transmitted bits. While it is natural to expect that fewer transmitted bits lead to lower energy costs, there are specific properties of *wireless bit error distributions that significantly enhance the advantage of using smaller packets,* as shown in Section II-B.

To better present the intuition behind some of the design choices made in Scepter we first discuss the various factors that affect energy consumption in a mobile device during such active communication phases. In particular, we performed controlled power measurements of a WiFi interface during its active and idle phases to create an effective model of energy consumption to expose the bit-specific error and energy footprint.

A. Power measurement and energy profile

The measurements were collected on both a laptop and a Netgear SPH101 WiFi phone. In the case of the phone, the setup constitutes of a 0.1 Ohm sense resistor, R, connected in series to the phone's power source allowing measurement of the being current supplied to the device. For accurate measurements on the phone we removed the battery and reconnected power via external wires. In the case of the laptop, we used the Sycard 140A cardbus adapter [5], to expose the current supplied to the interface. A Data Acquisition Card, DS1M12 Stingray Oscilloscope [6], samples the current through R at a high rate, thereby giving us energy consumption information on a very fine-grained manner.

The instantaneous power consumption, P_i can be written as $P_i = V_d \times V_R/R$ where V_d is the voltage provided to the 802.11 device and V_R is the voltage drop across R at a given moment. The energy consumed, E over a given time period, T, can then be computed using the formula, $E = \sum_{i=1}^{n} P_i \times t_s$, where we take n samples of instantaneous power P_i during time T (one sample every t_s seconds).

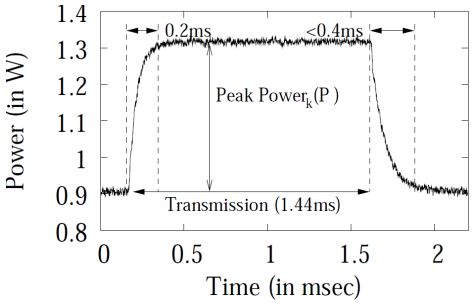

Fig. 2. **Instantaneous power consumption during the transmission of a 1064 byte packet using a Cisco wireless interface in 802.11g band.**

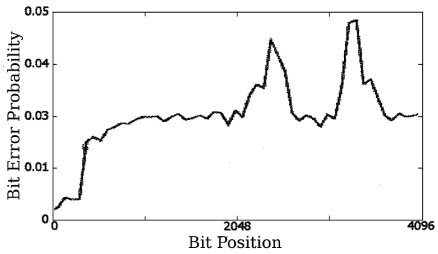

Fig. 3. **Distribution of bit-error probability within a packet computed over the set of packets received in error during the experiment. The packets were modulated at 54 Mbps.**

To better understand exactly what really goes on during packet transmission consider Figure 2, which shows the instantaneous power consumption of a packet of total size 1064 bytes (in air), transmitted at 6 Mbps in 802.11g band at a transmit power of 63 mW for a Linksys interface. The power consumption during packet transmission can be divided into three parts: *rise* in power when the card performs some preprocessing to prepare the data for transmission and then the packet transmission starts (about 200 μs), *stable* period when power consumption has reached a steady state and *fall* in the power consumption after packet transmission terminates ($<$ 400 μs).

The rounded nature of the curve can be explained as follows. The rise and fall occurs due to some capacitance in the transceiver circuit — the rise corresponds to the discharge of the capacitance to release initial current and the fall corresponds to a gradual drop in current as the transceiver goes back to idle state.

The instantaneous transmit power P_c during a packet transmission can then be represented by (verified in our analysis):

$$P_c = \begin{cases} P_t.(1 - e^{\frac{-t}{RC_1}}) & t < max(c \cdot RC_1, t_p) \\ P_t & t_r < t < t_p \\ P_t.(e^{\frac{-t}{RC_2}}) & t < c \cdot RC_2 \end{cases} .$$

Where $P_t = P_a - P_i$ is the power consumed only by the transmission circuit and does not include idle power P_i. RC_1 and RC_2 are the RC components for the rise and fall of current respectively, c is a constant and t_p is the packet transmission time.

From our empirical results, P_a serves as a good approximation to be used directly to calculate energy consumption, $E = P_a \times t_p$, for a packet of arbitrary size and data rate.

Using this information, we build an energy profile detailing the energy consumption patterns at various modulation rates for a specific WiFi interface. To compute the energy cost of transmitting a packet, the energy profile would need a lookup table indicating the power consumption, P_i for different transmit power levels i for every 802.11 mode that a given card supports. When the energy profile is queried for the energy cost for a given packet transmitted at rate r at transmit power level i it would make use of the following two steps. First we compute the packet transmission time, t, using the formula $t = t_p + l/r$, where t_p is the time taken to transmit the preamble for the particular 802.11 mode i.e. a, b or g, l/r given the time taken to transmit the payload of length l at rate r. Second we would reference the lookup table T to get P_i for the transmit power level i used and use formula, $E = P_a \times t_p$, to compute the total power consumed by the wireless transceiver and from it obtain the total energy. During regular operations, this can be further augmented to factor in observed packet error rates (that depend on packet sizes) in a manner similar to the ETX and ETT metrics [7].

This model suggests that the energy consumed to transmit a packet is proportional to its size. Hence, for a given packet, reducing its size by half would reduce the energy consumption in half as well. However, there is a critical factor that significantly enhances the advantages of shorter packets: non-uniformity in bit errors across the packet, as shown in Section V.

B. Leveraging non-uniform bit errors

If a packet is S bits long, and if p is the uniform bit error rate of all bits, then the probability of a packet loss is $1-(1-p)^S$. Once a packet is lost, in typical implementation the entire packet would need to be re-transmitted. This mechanism is implemented in the 802.11 MAC. Hence, if $E(S)$ is the energy cost of transmitting the packet once, then the total energy costs of successfully transmitting the packet, including loss recovery (under independent packet loss assumptions) is $E(S)/(1-p)^S$, which grows super-linearly with the size of the packet.

However, bit errors are not distributed uniformly within the frame. This observation implies the packet size has an even greater impact on energy costs.

We show this through our collected measurements in Figure 3. The figure shows the likelihood of errors in different bit positions of a 4096-bit packet, among all packets received in error. The key observation in this plot is that the first 704 bits (88 bytes) of a packet have a lower likelihood of error than the remaining bits. This is because the PLCP header of the physical layer includes a training sequence to synchronize the transmitter and receiver interfaces, prior to sending the remaining physical layer payload. The first few bytes of the

packet are, therefore, decoded when the communicating ends are better synchronized and have lesser errors. In contrast, the synchronicity between the two ends worsens with time and the accuracy of the receiver in decoding the later bytes diminish.

This result suggests that shorter packets gain even further energy efficiency. Bits in shorter packets stay synchronized at the receiver as they follow the PLCP header. It implies that even *the average bit error rate of shorter packets is lower than longer packets* which further separates the error rates of packets of different sizes. To take an example, if the bit error rate of all packets, independent of packet size, was uniformly 1×10^{-4}, then a 128-byte packet will have a packet error rate of 0.09, while a 256-byte packet will have a packet error rate of 0.17. However, in practice the average bit error rate of the longer packet is lower, say 2×10^{-4}, then the 256-byte has a packet error rate of 0.31, i.e., about 2 times more, incurring about 35% more re-transmission overheads. Overall, it suggests that use of smaller packets on the wireless link is particularly critical in reducing re-transmission overheads and in turn saving energy.

III. Scepter Design

In Scepter, we based our design on the observation that every bit transmitted or received by a mobile device has an energy cost associated with it. While the infrastructure also pays an equivalent cost, its access to unlimited power supply makes this cost irrelevant in this scenario. Hence, in Scepter we increase the communication and computation burden of the infrastructure-based entity (the proxy) to reduce the burden on the mobile device. We call this proxy the *Scepter proxy* and we call a client that uses a Scepter proxy a *Scepter client*. By offloading redundant transmissions and processing to the Scepter proxy, the Scepter client is able to send fewer bits and thus conserve energy. In this work, we have assumed that the Proxy's functionality is within the same broadcast domain as the WiFi AP.

The interaction between the Scepter client and the Scepter proxy is specifically designed to remove any redundant traffic that might be seen, whether within the control information or in the data payload. Eliminating redundant control information is possible, because the link between the Scepter client and the Scepter proxy is predictable. Such knowledge is leveraged at the proxy to remove signaling messages and other information that does not need to be explicitly transmitted by the Scepter client, but can be correctly predicted at the Scepter proxy.

In addition, all user generated content (carried as data payloads) tend to have various forms of redundancies as well. To eliminate all such forms of redundancies, we use a toolbox available from a range of prior efforts, including Robust Header Compression (RoHC) [8], Huffman encoding [9], Split TCP [10], and strategies to eliminate redundancies across a sequence of packets [11], [12], none of which were design for energy efficiency.

In this section, we explain how we use each of these prior techniques in the context of energy efficiency in Scepter. We first describe our approach to reducing redundant payloads.

A. Redundant Payloads

Reducing the redundancies found the in payloads of packets may provide significant savings when the payload dominates the size of the packet, such is the case with HTTP, which was designed to be simple and human-readable having many redundancies.

Elimination of redundant payload content within a single packet: Since HTTP is a text-based protocol, one simple approach would be to take each packet and compress it with a zip tool [13] that uses a combination of Huffman encoding [9] and the LZ77 algorithm [13]. This would effectively remove patterns found in the textual payload of the given packet. We call this approach of payload redundancy elimination, *per-packet redundancy elimination* and is used in most web browsers. However, web browsers use compression to improve bandwidth efficiencies and not for energy reasons. We build upon this approach to quantify the energy savings from such techniques. Further discussion of using compression techniques and their computational overhead are in Section VI.

Recent work [14], has shown that many such compression algorithms have an asymmetric nature where decompression is often much faster and less resource intensive as compared to compression. During compression the algorithm must search for redundancies and as the algorithm removes these redundancies it must also create a mapping between the original content and the compressed content. During decompression the algorithm simply uses the mapping that was created by the compression step to return the content back to its original state. This asymmetry is additionally beneficial to Scepter since the network traffic of mobile devices is heavily weighted in the down-link direction.

Elimination of redundant payload content across multiple packets: In addition, we also consider the case where redundancy exists across a sequence of packets exchanged between the mobile client and the proxy. For example, consider two different IP packets (say, corresponding to content of different websites) carrying the following text-based substrings "energy efficient 802.11" and "energy efficient laptop." Between these two packets, the substring "energy efficient" is redundant, while the rest of the packet's content are not. It is well known that various forms of such redundancies exist in the Internet traffic [11]. In recent work, Anand et. al. [12] showed how to implement some redundancy elimination primitives, using Rabin fingerprints [15] within core Internet routers to reduce the total traffic carried by an individual ISP. The basic idea of this technique is to cache various fragments of a packet locally and remove redundancies when these fragments are observed in future packets. In Scepter, we tailor this technique for use within the mobile device and the Scepter proxy.

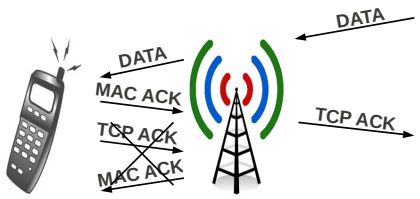

Fig. 4. **MAC to TCP acknowledgments. With our infrastructure the MAC layer acknowledgment provides the same information as a traditional TCP acknowledgment making the TCP ACK completely redundant.**

Since we apply this technique over multiple traffic flows we call this technique *multi-flow redundancy elimination*.

B. Reducing Redundant Control Messages

We next examine how the infrastructure-based proxy can use cross-layer optimizations to reduce certain control messages through its knowledge of redundancies between multiple layers of the protocol stack. We do this through the following example.

Elimination of redundant control messages. When a TCP source sends a data segment the receiver will reply with a TCP ACK. If this ACK is not received before a set period of time TCP will retransmit the packet. Similarly, when an 802.11 source sends a packet and the receiver responds with a MAC layer ACK. If this ACK is not received before some set period of time the MAC layer will retransmit the packet. Within a single hop, wireless link between the Scepter client and the Scepter proxy, a MAC layer ACK provides the same information that a TCP ACK provides. As shown in Figure 4, both the MAC layer ACK and TCP ACK are transmitted from the mobile client. When using Scepter, the proxy can translate the MAC layer ACK into a TCP ACK eliminating the need for the client to transmit the TCP ACK and also eliminate the need to receive the subsequent MAC layer ACK.

In practice Scepter must provide cross-layer guarantees over the data so that the end-to-end semantics are not violated. We claim this optimization is an acceptable due to three reasons. First, it is very unlikely that the wireless NIC is able to correctly decode the original data packet and send a MAC layer ACK, but is unable to pass the packet up to the higher network layers. Second, the Scepter proxy is sending the TCP acknowledgment only after it receives the MAC layer ACK from the mobile device and hence, is certain the mobile device has received the original data packet. Finally, if the MAC layer fails to transmit the packet, our kernel implementation will receive notification of such a failure and Scepter can simply retransmit the packet, as detailed in Section IV.

C. Reducing Other Redundancies

The first approach to reducing control overheads is to reduce the number of header bits within a packet. Header reduction is critical to energy efficiency for small packets — ones in which headers form a significant overhead. For example, with the usual voice aggregation delay of 20-30 milliseconds, popular voice codecs such as G.711, G.723, etc., create packets sized 20 to 160 bytes. Hence the 28-byte UDP/IP header can be a significant fraction of a VoIP packet. Based on this motivation from our measurements, we describe techniques used in Scepter to reduce and eliminate avoidable header fields. We start with techniques seen in RoHC for removing static header fields [8]. Once RoHC compression is in place we implement further compression techniques to build a system that extends far beyond RoHC.

Elimination of dynamic header fields, moving beyond RoHC. We also remove some of the dynamic header fields due to redundancies across multiple layers. For example, the UDP checksum field can be removed in Scepter, since the 802.11 MAC frame also contains a CRC field that protects the entire MAC payload. Similarly, the UDP length field can also be eliminated as the same information can be gleaned from the 802.11 MAC headers. However, in order for this packet to be correctly received at the remote destination, the proxy is now made responsible for creating and placing both the UDP checksum and the UDP length, before sending the packet onwards. This adds additional computation and processing burden on the proxy, but relieves these burdens from the client.

In case of TCP, we also introduce some header optimizations. An example of this is the 32-bit sequence and ACK number fields of TCP. The advantage of a large sequence number field is that multiple unacknowledged segments may be left in transit increasing throughput achieved on the end-to-end connection. However, the 802.11 MAC operates in a stop-and-go manner, i.e., it sends a packet and obtains its ACK before sending the next packet. Therefore, we consider a split-like design to leverage the design differences between the 802.11 protocol and higher layer protocols.

We split some of the header construction functions between the mobile device and the proxy. The first part of this split is the link between the mobile device and the proxy, which uses a smaller sequence number space. In our current implementation we need only to reliably disambiguate between two consecutive packets, which only requires the use of a single bit sequence number. Thus, the link operates like a stop-and-go protocol, matching the 802.11 MAC's underlying behavior. The rest of the path between the proxy and the remote destination uses the full 32-bit TCP sequence number space. Specifically, when the proxy receives a TCP segment from the mobile device it expands the sequence number into a regular TCP sequence number by inferring the correct value in the 32-bit space. Thus, we further reduce the number of bits sent over-the-air by shifting the computation burden from the client to the proxy.

By using a stateful proxy, we are able to replace all fields of the IP, UDP, and TCP headers in regular packets with a total of 3 bytes of Scepter header. Two of these bytes are embedded within the 802.11 MAC header, allowing us to reduce regular TCP/IP and UDP/IP headers to a single byte.

Call Type	Number of Calls	Total Percentage
Recv()	23228	93.1%
Send()	811	3.3%
Connect()	534	2.1%
Getsockopt()	215	0.9%
Other	164	0.6%
Bind()	1	<0.1%

TABLE I

This table contains the number of socket type system calls made during a 10 minute period of time browsing the Internet with the firefox web-browser. General browsing, web-email, and streaming video were all performed during this period of time. Counters were incremented from within the kernel each time a socket type system call was made by an application.

IV. In Kernel Implementation

In this section we present implementation details and design considerations for Scepter. We implemented Scepter as a kernel module, which provides us the vantage point needed for many of our cross-layer optimizations. The kernel also provides us the opportunity to conduct a side-by-side comparison with the traditional Linux network stack.

The protocol was implemented using Remote Procedure Call (RPC) style messages. The choice of RPC style messaging provides a layer of abstraction that other implementation methods could not provide. This layer of abstraction produces a clean implementation allowing the current socket API to go unchanged and provides an accurate comparison to the traditional Linux network stack. The goal of an RPC messaging system is to send a message to a remote host requesting that a particular action be execute on the remote host. In our system the Scepter client sends messages to the Scepter proxy requesting a system call be executed on behalf of the Scepter client, e.g., `connect()` or `send()`.

The protocol begins when the Scepter client program makes a `socket()` function call. Instead of going to the traditional network stack of the Linux kernel, the call goes to the Scepter stack. In the routine, the Scepter client sends a packet to the Scepter proxy to execute the call on the Scepter proxy. The Scepter proxy then allocates a Scepter identifier which is returned to the Scepter client and used for all subsequent communications. It is important to note that our implementation was designed to allow side-by-side comparisons with the traditional network stack. However, by implementing a completely separate network stack we would require applications to choose between the traditional stack and the Scepter stack. For our research purposes this was the desired functionality, however, in practice integration of the Scepter network stack into the traditional network stack would allow the kernel to use the Scepter functionality only if support from within the network existed. This would not require any changes to application code. A discussion on bootstrapping methods will be deferred to Section VI.

A naive approach would send messages to the Scepter proxy for every socket call. However, socket calls such as `getsockopt()` change socket options that are applicable to the Scepter client and not the Scepter proxy.

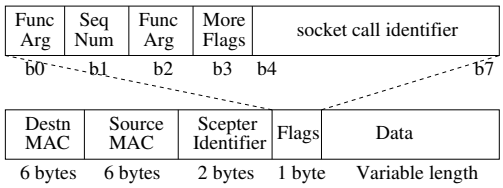

Fig. 5. **Scepter client to proxy Ethernet frame used to communicate with the Scepter proxy. The type field in Ethernet frame is replaced by the Scepter identifier**

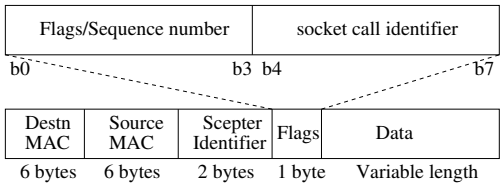

Fig. 6. **Scepter proxy to client Ethernet frame used to communicate with the Scepter client. The type field in Ethernet frame is replaced by the Scepter identifier.**

Thus, we execute only the socket calls which modify the connection state on the Scepter proxy to optimize the control overhead of our system. Furthermore, we have focused on optimizing `send()` and `recv()` calls since other socket calls such as `socket()` or `getsockopt()` constitute <5% of total socket calls in a typical network application, as shown in Table I. Table I further demonstrates that there is a highly asymmetric nature between the number of `recv()` calls and all other socket calls. Scepter leverages this asymmetry to optimize the `recv()` call. When the Scepter client issues a `recv()` call a RPC message is not sent to the Scepter proxy, unlike the behavior of all the other socket calls. Instead the Scepter proxy will proactively forward packets destined for the Scepter client. On reception of these packets, the Scepter client enqueues them to the appropriate socket queue. Later, when the application on the Scepter client requests a packet, it is fetched from the local socket queue.

A. Scepter Packet Structure

Figures 5 and 6 show the structure of packets used for communication between the Scepter client and the Scepter proxy. All packets are Ethernet frames. To minimize packet header length, we have reused the 2-byte Ethernet type field of the 802.11 MAC header as the Scepter identifier. However, we make sure that the Scepter identifier that we use does not conflict with any of the types (e.g., IP, ARP, RARP) used on the given network. This 2-byte type field can simply be passed to the Ethernet driver requiring no changes to the driver code at all.

The following byte (byte 15) is an extensible flags field present both in Scepter proxy and Scepter client packets which behave differently based on whether it is a Scepter client packet or Scepter proxy packet. The last 4 bits (b4 to b7) of the flags field are common to

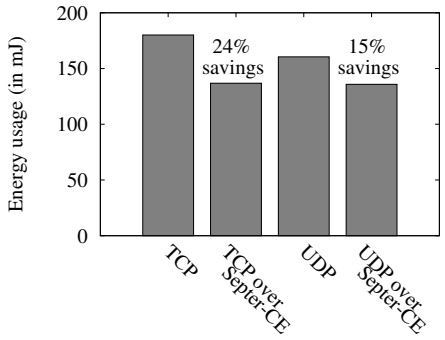

Fig. 7. **VoIP or interactive messaging traffic (consisting of small packets) over a good quality link. Scepter-CE is well suited for such traffic, since it focuses on header elimination.**

both client and proxy and represent the socket call (e.g. `bind()`, `sendmsg()`, `recvmsg()`) to be executed at the Scepter proxy.

Scepter client flags field. The first 4 bits (b0 to b3) are used for socket call specific options. For example when `sendmsg()` is called with MSG_DONTWAIT flag set, it is set in bit position b2. Bit b3 (MORE_FLAGS_INDICATOR) if set implies that the following byte can be used to for setting more flags. Bit b1 is used as a sequence number.

Scepter proxy flags field. The first 4 bits, b0 to b3, of the flags field are used for `sendmsg()` calls and for forwarding packets to Scepter client. For `sendmsg()` calls, the bits are used to send control information to Scepter clients. For example when the Scepter client sends a UDP packet, the first packet contains the receiver's IP and port numbers, the Scepter proxy creates an entry for the receiver and sends a new Scepter identifier back to the Scepter client which represents the given IP and port and sets a bit in the flags field to reflect that. For subsequent packets, the Scepter client can use the identifier instead of IP and port information.

When forwarding TCP packets, the Scepter proxy uses bits b0 to b3, for sequence numbers. Our stop-and-go protocol only requires the use of a 1 bit sequence number, which is sufficient to differentiate subsequent packets, but we implemented a 4 bit sequence in packets from the proxy to the client to allow for block aggregation. Block aggregation is used in protocols such as 802.11n and allow the sender to combine multiple packets that would otherwise be transmitted separately into a single transmission. Our 4 bit sequence number provides support for such optimizations.

V. Evaluation

In this section we present a detailed evaluation of Scepter obtained through a wide range of experiments under different scenarios.

For each packet there is a preamble, MAC header, TCP/UDP/IP or Scepter header, and payload. For all experiments at least 25,000 packets were sent for the given traffic type. We used two different types of traffic load to demonstrate the benefits of Scepter. We emulate both VoIP traffic (operates on UDP) and live interactive messaging services (operates on TCP) which have small packets (order of 20 bytes) and can be encrypted for data privacy. We generate similar sized packets and account for encryption by randomly generating payloads. The other type of traffic is web traffic which is dominated by packets 1KB or greater.

	Battery Life (min)
Idle	121
w/o Scepter	52
w/ Scepter	72

TABLE II
Battery drain test. VoIP style traffic was continuously transmitted from the laptop until the battery of the device was exhausted. Scepter provides a 38% improvement in the battery lifetime.

We have implemented Scepter within a Linux kernel version 2.6.19.6 running on a 1.66 GHz laptop. We compare the energy savings of Scepter to the default TCP/IP implementation running on the laptop. We consider different variations of Scepter: (i) *Scepter-CE:* includes elimination of redundant header fields and control messages (Sections III-C and III-B), (ii) *Scepter-RE (per-packet):* includes elimination of redundant payload as applied to an individual packet through a combination of Huffman encoding and LZ77 algorithms for data compression within a packet (Section III-A), and (iii) *Scepter-RE (multi-flow):* includes elimination of redundant payload as applied across a sequence of packets between a single mobile device and the proxy (also Section III-A). The term Scepter-CE-RE (multi-flow) implies that both Scepter-CE and Scepter-RE (multi-flow) have been applied together. Similarly, Scepter-CE-RE (per-packet) implies that both Scepter-CE and Scepter-RE (per-packet) are being utilized. Scepter-RE (per-packet) and Scepter-RE (multi-flow) are never combined since the CPU and memory consumed during the second phase of redundancy elimination outweigh the minimal additional reductions in packets sizes.

For our experiments, we conducted measurements for the WiFi interface of a laptop. Thus most of our numbers represent the energy savings across the WiFi interface alone. To transmit a fix amount of data across such a WiFi link we define "energy savings" as follows:

$$energy\ savings = Energy_{Legacy} - Energy_{Scepter} (in\ mJ)$$

where

$Energy_{Legacy}$ = energy to transmit the data without Scepter
$Energy_{Scepter}$ = energy to transmit the data with Scepter

We then compute the percent difference between $Energy_{Legacy}$ and $Energy_{Scepter}$ and present this number throughout this section.

Overall energy gains: The relative impact of the battery lifetime of a mobile device would also depend on the regular power drawn by the rest of the mobile device. A comparison of this battery lifetime metric on the laptop will certainly be biased by the power drawn by the large screen. For this reason

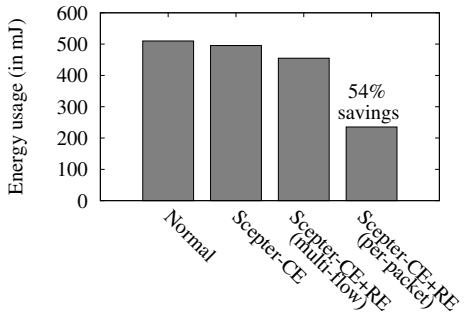

Fig. 8. **Web style traffic over a good quality link. Scepter-RE is well suited for web style traffic as can be seen. The energy savings for Scepter-RE(per-packet) stems from high compressibility of web traffic.**

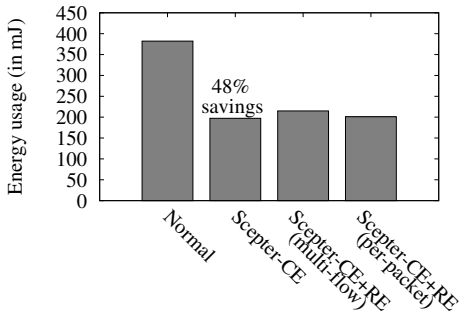

Fig. 9. **VoIP style traffic over poor quality link. High gains of Scepter-CE due to drop in re-transmission requirements in this case, when compared to other schemes.**

the measurements were conducted with the screen turned off. As seen in Table II, Scepter improves the battery lifetime by 38%. The battery was continuously probed and the remaining battery charge was logged while sending VoIP style traffic over the link.

In the following experiments, we describe the performance of different schemes in terms of both energy consumed and throughputs achieved. Note that the comparison in energy is based on the same amount of data successfully transmitted, i.e., mechanisms with lower throughputs operate longer to carry all the data.

Good quality link, small packets: Figure 7 compares the performance of different schemes when operating on a link of good signal quality and carrying VoIP style traffic. This is the scenario where Scepter-CE provides significant advantages by eliminating the TCP/IP or UDP/IP header. This is also shown in the figure where Scepter-CE provides 24% savings over TCP and 15% over UDP. The gains of Scepter-CE is higher in TCP due to the larger size of TCP headers that can be eliminated.

Figure 8 shows the comparative performance of the schemes with good link quality for large packets. The packets were collected from web traffic consisting mostly of full 1500 byte payloads and had observable textual redundancies. This is the case where Scepter-RE provides more gains than Scepter-CE, as can be seen in the additional gains of Scepter-CE-RE over Scepter-CE. In addition, the per-packet Scepter-RE scheme outperforms the multi-flow scheme since the former uses a more effective compression algorithm that is computationally expensive and can thus achieve greater payload reduction. Even though the multi-flow approach has greater opportunities of redundancy elimination, it was designed to be computationally efficient. It only probabilistically identifies redundant content, missing various opportunities.

Impact on Throughput: Most components of Scepter have no perceptible impact on end-to-end throughput, barring one in its current implementation — Scepter-RE (per-packet), which performs computation-intensive tasks of data compression. In our experiments the data compression techniques within a single packet includes a combination of Huffman encoding and the LZ77 algorithm. The ensuing delays impacted achievable throughput (TCP or UDP) by up to 25%, e.g., in some moderate channel conditions we achieved a throughput of 1.5 Mbps with Scepter-RE (per-packet) and a throughput of 2 Mbps without it. Note however, that this loss in throughput always comes with energy savings when comparing the transfer of the same number of data bits as already reported in this section. We believe that some of the inefficiency lies in our user level implementation of this component, while others are inherent to the compression algorithm itself. In either case we feel this is a reasonable trade-off a mobile user applying Scepter-RE (per-packet) should make, i.e., they can choose to be more energy conserving and at the expense of some throughput, or they can consume more energy and to maintain full throughput.

Other Scepter variants which include the implementation of Scepter-RE (multi-flow) do not have a significant computation overhead. This observation is also validated by the absence of any throughput degradation in our experiments.

Bad quality link, small packets: We repeated all the experiments in a location which exhibits consistently poor wireless signal between the mobile device and the AP. As seen in Figure 9, Scepter-CE reduces energy consumption by 48% as is expected with small packet sizes, while Scepter-RE provides almost no additional gains. Again, this is due to the lack of redundancy in randomly generated content, emulating data compression and encryption of many VoIP codecs.

Bad quality link, large packets: Our final set of energy measurements consisted of large packets (web traffic) over the same poor quality link. As seen in Figure 10, Scepter-RE reduces energy consumption by 39%. Again we notice that Scepter-RE performs better than Scepter-CE because of the larger packet sizes.

A. Understanding Scepter-RE

The performance of Scepter-RE to some extent is dependent on the amount of redundancy present in the traffic. To understand what kind of redundancy may be present in regular user traffic we collected large traces of data from our on-campus wireless network. The data was collected on a busy

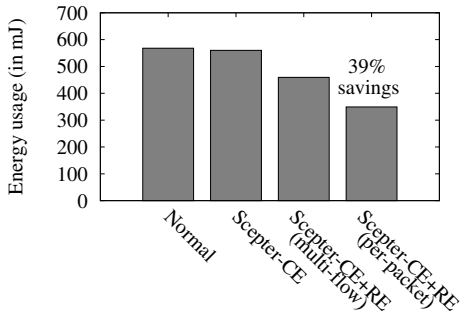

Fig. 10. **Web style traffic over poor quality link. Scepter-RE provides the best gains.**

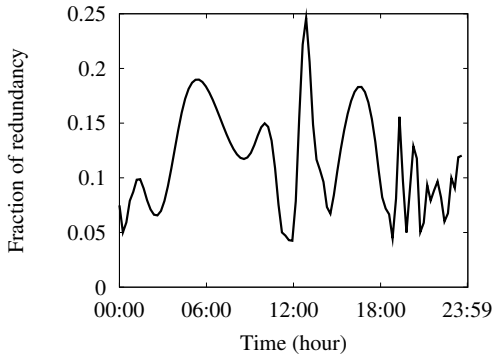

Fig. 11. **Scepter-RE (multi-flow) redundancies seen over a 24 hour period from actual traces collected from a busy university library. Each point corresponds to the percentage of redundant bytes computed using the Scepter-RE (multi-flow) algorithms over all user traffic in grouped into 10 minute bin.**

day in our university library across an entire period of 24 hours. The total amount of traffic collected in this experiment amounted to about 110 GB.

We calculated the degree of exploitable redundancy present in the traffic trace by running the multi-flow Scepter-RE algorithm over the traces, binning the entire trace into 10 minute durations. This emulates the effect of identifying redundancies available in 10 minute bins throughput the day. We chose a small bin size, e.g., 10 minutes, so that would not retain very old traffic as it would require the proxy or the mobile device to store a large volume of old data. Figure 11 plots the available redundancy as it varied throughout the day. We observed the maximum amount of redundancy was during the time segment between 1:00pm and 1:10pm where there was on the order of 25% redundancy. The spike in the graph likely corresponds to a large number of students visiting the library during the lunch hour. On average there was 8.8% redundancies on this particular day.

An interesting aspect of the above result is the fact that it is computed over traffic transmitted by multiple users. Since this entire traffic is observable from a set of close-by vantage points, many of these users are actually within communication range of each other, i.e., they can hear each others packets to some extent. This can allow for an interesting design point where payload can be eliminated between packets of different mobile devices, by requiring each to retain observed content of others. We have avoided the complexity of such a design in our current work, and will explore this concept as part of our future direction.

VI. Discussion

In this section we will discuss additional enhancements to the Scepter system that will improve both usability and integration into current mobile devices and network infrastructure.

Web Browser Compression. Most modern web browsers support the compression of the HTTP payload. Two common compression algorithms used by browsers are the gzip and deflate algorithms. It is important to note that even if both the server and web browser support payload compression, this does not guarantee that compression will be used. Further the compression is only performed by the server and if compression is performed, the server only compresses the HTTP payload and not the highly compressible HTTP header itself. However, if the HTTP payload is already compressed it would be a waste of time and energy to try to compress it again with Scepter. Therefore Scepter would need to track the state of HTTP traffic to avoid compressing data that has already been compressed.

Scepter could track the compression ratios achieved on a per-flow bias to detect if the ratios are consistently low, which would be the case if the payload was already compressed. If the ratios are consistently low for a particular flow, Scepter would then decide to turn off Scepter-RE for that particular flow.

Computation overheads. As mentioned most web browsers support the gzip and deflate compression algorithms. We have based our measurements on the deflate algorithm since the CPU and memory requirements are significantly smaller for the deflate algorithm [14]. Barr and Asanovic [14] further show that when the compression algorithm is properly tuned that the CPU and memory consumption is minimal.

The computational overheads also add additional latency to network traffic. Only one variant of Scepter, specifically Scepter-RE (per-packet), has such computation overheads that leads to loss in throughput performance, when this scheme is applied. We believe that this overhead provides device users with a trade-off between greater energy efficiency and better throughput, a choice that can depend on the battery level of the device. Further optimizing the computation overheads will continue to be part of our future work.

VII. Related Work

As discussed in Section I, there has been multiple complementary approaches to energy efficiency. Hardware-based approaches attempt to optimize the circuitry to improve power efficiency of these devices [16], [17]. Similarly, a large number of software-based approaches have also been proposed. They include: (i) those that attempt to put the wireless NIC

in a "power-save" mode [18], [19], [20]. (ii) those that power-down the primary wireless NIC but keep an additional low-power interface awake to quickly revive communication [3], [4], and (iii) those that use transmit power control [21]. Scepter uses a fourth alternative, in that it reduces the number of control and data bits to achieve the same communication, thereby reducing the costs of active transmissions when they have.

Various design components of Scepter are built upon prior work. For example, there has been significant prior work on bandwidth efficiency which overlaps with some of the principles on which Scepter is based. Robust Header Compression (RoHC)[8] originally proposed by Van Jacobson attempts to reduce the header size of packets between a sender and a receiver connected by a point-to-point link by having extra state at the receiver. RoHC is a good method to improve throughput efficiency. In contrast, Scepter is optimized for energy efficiency and goes beyond the compression methods used in RoHC.

Part of the Scepter design builds upon ideas from the split connection protocols such as iTCP[22] and split-TCP[10]. Such prior split connection, split-TCP in particular, suggests the use of a separate TCP connection between the mobile host and a proxy for better differentiating the causes of losses on the end-to-end Internet path. In our split design, we completely eliminate the TCP header in the wireless link, and use a stop-and-go protocol for flow control across this link.

Our work in removing redundancies in the payload leverages similar approaches applied in the Internet to reduce the traffic load on core routers or other such wide-area entities [11], [12]. We focus on using such an approach for energy efficiencies. An in-depth study of compression algorithms and their CPU and memory consumption can be found in prior work [14]. Using this work, we selected an algorithm with low CPU and memory need such that our focus could remain on the energy consumed by the wireless hardware itself.

Finally, there has also been some work that aims to leverage the infrastructure in a way that would allow the entire client machine to go into a sleep state [23], [24]. The network would queue up traffic until a significant amount of traffic was waiting to be processed by the client machine. The network would then wake the machine up and forward the traffic for processing.

VIII. Conclusion

We have designed and fully implemented Scepter to achieve large energy gains. These gains are realized through the use of an asymmetric relationship between the Scepter client and the Scepter proxy. Further gains are achieved from our cross-layer optimization. Our kernel implementation not only provided an ideal vantage point to implement these cross-layer optimization but also provided an ideal comparison to the traditional Linux network stack.

The energy savings observed by Scepter are very encouraging. Coupled with a design that is complementary to prior approaches makes Scepter a very attractive method for reducing energy consumption of mobile devices. As mobile device continue to require faster processing, larger screens, and faster connectivity we feel that Scepter will prove to be very valuable for users who enjoy long battery lifetimes.

References

[1] R. Zheng and R. Kravets, "On-demand power management for ad hoc networks," vol. 1, 2003, pp. 481–491 vol.1.

[2] "Ieee 802.11 standard," http://standards.ieee.org/getieee802/802.11.html.

[3] E. Shih, P. Bahl, and M. Sinclair, "Wake on wireless: An event driven energy saving strategy for battery operated devices," in *ACM Mobicom*, 2002.

[4] T. Pering, Y. Agarwal, R. Gupta, and R. Want, "Coolspots: Reducing the power consumption of wireless mobile devices with multiple radio interfaces," in *ACM MobiSys*, 2006.

[5] "Sycard cardbus adapter," http://www.sycard.com/ext140.html.

[6] "Stingray digital oscilloscope," http://www.usb-instruments.com/data_ds1m12.html.

[7] R. Draves, J. Padhye, and B. Zill, "Routing in multi-radio, multi-hop wireless mesh networks," in *MobiCom '04*. New York, NY, USA: ACM, 2004, pp. 114–128.

[8] IETF RoHC Charter, "Robust Header Compression (RoHC) IETF Charter ," http://www.ietf.org/html.charters/rohc-charter.html, 2001.

[9] D. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098 –1101, sept. 1952.

[10] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 756–769, 1997. [Online]. Available: citeseer.ist.psu.edu/article/hari96comparison.html

[11] N. T. Spring and D. Wetherall, "A protocol-independent technique for eliminating redundant network traffic," in *SIGCOMM '00*. New York, NY, USA: ACM, 2000, pp. 87–95.

[12] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker, "Packet caches on routers: the implications of universal redundant traffic elimination," in *SIGCOMM '08*. New York, NY, USA: ACM, 2008, pp. 219–230.

[13] "Deflate compressed data format specification," http://www.ietf.org/rfc/rfc1951.txt.

[14] K. C. Barr and K. Asanović, "Energy-aware lossless data compression," *ACM Trans. Comput. Syst.*, vol. 24, no. 3, pp. 250–291, 2006.

[15] M. Rabin, "Fingerprinting by random polynomials," in *Technical Report*. TR-15-81, 1981.

[16] "Ultra-low-power wifi chip targets windows mobile devices," http://www.windowsfordevices.com/news/NS3141723896.html.

[17] "Broadcom introduces ultra-low power wi-fi chips optimized for mobile devices," http://www.broadcom.com/press/release.php?id=919562.

[18] M. Anand, E. Nightingale, and J. Flinn, "Self-tuning wireless network power management," in *ACM Mobicom*, 2003.

[19] M. C. Rosu, C. M. Olsen, C. Narayanaswami, and L. Luo, "Pawp: A power aware web proxy for wireless lan clients," in *WMCSA '04*, 2004.

[20] F. R. Dogar, P. Steenkiste, and K. Papagiannaki, "Catnap: exploiting high bandwidth wireless interfaces to save energy for mobile devices," in *MobiSys '10*, 2010.

[21] S. Banerjee and A. Mishra, "Minimum energy paths for reliable communication in multi-hop wireless networks," in *MobiHoc*, 2002.

[22] A. Bakre and B. Badrinath, "I-tcp: Indirect tcp for mobile hosts," in *Technical Report DCS-TR-314, Rutgers University*, 1995.

[23] S. Nedevschi, J. Chandrashekar, J. Liu, B. Nordman, S. Ratnasamy, and N. Taft, "Skilled in the art of being idle: reducing energy waste in networked systems," in *NSDI'09: Proceedings of the 6th USENIX symposium on Networked systems design and implementation*. Berkeley, CA, USA: USENIX Association, 2009, pp. 381–394.

[24] Y. Agarwal, S. Hodges, R. Chandra, J. Scott, P. Bahl, and R. Gupta, "Somniloquy: augmenting network interfaces to reduce pc energy usage," in *NSDI'09: Proceedings of the 6th USENIX symposium on Networked systems design and implementation*. Berkeley, CA, USA: USENIX Association, 2009, pp. 365–380.