Sinew: A System for Improving the Efficiency of Wireless Networks

by

Hans Robertson

Submitted to the Department of Electrical Engineering and
Computer Science in Partial Fulfillment of the
Requirements for the Degree of Master of Engineering in
Electrical Engineering and Computer Science at the
Massachusetts Institute of Technology

September 2, 2003

Author_____
Department of Electrical Engineering and Computer Science
September 2, 2003

Certified by_____
Anant Agarwal
Associate Director, Lab for Computer Science
Thesis Supervisor

Accepted by_____
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Sinew: A System for Improving the Efficiency of Wireless Networks

by

Hans Robertson

Submitted to the Department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science at the Massachusetts Institute of Technology

Abstract

Wireless data networks are widespread and growing quickly. As their use increases, many wireless networks are becoming congested. In addition, as wireless data capability moves into ever-smaller devices, power becomes a significant issue.

This thesis presents a system that increases network bandwidth and decreases energy use without changing existing network hardware or protocols. We use specialized proxy servers to transparently modify the traffic sent over the mobile link such that the total energy used by the receiver is reduced and the effective bandwidth is increased. Our techniques include optimizing packet size, eliminating unnecessary traffic, and masking wireless packet losses.

We design and implement two proxies – one for access points and one for mobile devices – which when used together, achieve up to a 20% decrease in energy and 38% increase in throughput.

Thesis Supervisor: Anant Agarwal

Title: Associate Director, Lab for Computer Science

Contents

1 Overview

Wireless networks are important: excluding cell phones, there are tens of millions of devices in use today and their number is expected to continue to grow. Since wireless network bandwidth is limited, as the number of users increases congestion can become a problem. Unlike wired networks, it is difficult to upgrade wireless networks: adjacent nodes interfere with each other, standards conflict, and government regulations limit the available frequency bands. Moreover, changing hardware or networking protocols is very difficult once devices are widely deployed.

Also, since wireless network devices often operate on battery power, the energy efficiency of wireless networking is important.

This thesis describes the design and implementation of Sinew, short for *System for Increasing the Network Efficiency of Wireless*. Sinew increases the energy efficiency and improves the performance of wireless networks for common applications like file transfers. It uses a proxy server to reshape the traffic sent over the wireless link. An optional client proxy server provides additional performance enhancements, such as allowing us to use a special TCP replacement protocol over the wireless link.

Sinew proxies can be implemented in user-mode software and do not require any changes to existing network infrastructure or hardware.

Our proxy design is only one of several ways to approach the problems of wireless throughput and energy efficiency. Section 2 covers alternative approaches and previous work. Section 3 describes our system in detail, section 4 models its theoretical benefits, and section 5 presents actual results.

1.1 Constraints and Limits

Wireless networking and the problems we have outlined are broad, so it is important to state the limitations within which we are working.

- We consider only 802.11b-based networks, since they are the most widespread. However, many of our techniques will apply to other kinds of 802.11 as well as other wireless networking standards.
- We limit ourselves to approaches that minimize changes to mobile hosts and do not require changes to the 802.11b hardware or firmware or to Internet protocols.
- We do not consider ad-hoc networks since infrastructure networks are more common.
- Our target usage mode is client-initiated file transfer and Web browsing. At present few wireless hosts act as servers.
- We evaluate Sinew in an environment with little noise and few users. This is not representative of real 802.11 networks and is an area for future study.
- We focus on energy and performance improvements and do not explicitly treat jitter, fairness, congestion control, wireless interference, or handoffs.

- To realize energy savings benefits, we require that the mobile host use power saving mode[1].

2 Related Work

There have been many attempts to improve the energy efficiency and performance of Internetworking over wireless networks. We limit ourselves to an examination of approaches that, like ours, attempt to reduce energy or increase performance while requiring minimal changes to the mobile device or existing network infrastructure.

Our proxy approach is sometimes referred to as a *split-connection proxy*. Few have tried to use a split connection proxy to reduce energy, although they have been considered for performance improvements. One explanation is that, until the 802.11b PSM was widely deployed, the best way to reduce the energy consumption of a wireless networking device was not to decrease the traffic on the link but to reduce the amount of time that the wireless radio was turned on. However, with PSM, reductions in the amount of traffic yield significant power gains. Recent work on more efficient PSMs [20] makes reduction of traffic all the more important.

2.1 Hardware/Physical Layer

The power efficiency of wireless networking hardware is an area of ongoing research and development. For example, many 802.11b cards sold today use 3.3V instead of 5V. Such work complements our system.

Shih et al. [23] proposed use of a secondary low-power radio as a control channel to minimize the amount of time the power-hungry data radio is on but idle. Although discussed in the context of PDA-based telephony, their ideas and hardware could easily be adapted to improving the power-efficiency of data traffic by alerting the device when incoming packets are waiting at the base station.

The 802.15.4 working group is developing an ultra-low power radio that could also be used as a secondary channel.

Other physical layers such as 802.11a and 802.11g offer higher bandwidth. Unfortunately, these physical layers either interfere with 802.11b or they do not offer significant benefits in the presence of 802.11b. Improvements in the physical layer will likely be compatible with Sinew.

2.2 MAC Layer

MAC-layer strategies try to improve the protocol between the mobile device and the base station. [11] contains a discussion of the energy efficiency of various MAC protocols, including 802.11. MAC improvements typically require changes to the firmware, hardware, or standards.

---

[1] PSM stands for Power Saving Mode. In PSM, a device wakes up periodically to see if the access point has packets available and, if so, retrieves those packets before going back to sleep.

Stemm and Katz [25] were the first to show that putting a wireless LAN card to sleep whenever possible can dramatically reduce its power consumption. They concluded that savings obtained in this way dwarfed any that could be obtained by reducing the amount of traffic sent on the link.

In 2002 Krashinksy [20] proposed a PSM that further reduces energy consumption by 1-14% while avoiding some of the performance penalties of the original PSM. Kravets [22] proposed a similar strategy. Unfortunately, these approaches require changes to the hardware or firmware of mobile networking hardware.

Chiasserini [9] tried to improve the performance of TCP by adapting link layer parameters to changing network conditions. In particular, he dynamically changed the number of 802.11 retransmits to reflect network conditions. This approach is most likely to have an impact in noisy environments in which packets are often lost. It has the added bonus that it could be implemented by a user-level program. Wong and Leung [26] also investigated interactions between TCP and the link layer.

While there has been significant work on improving the energy efficiency of ad-hoc wireless networks [10] at the MAC layer, we are concerned only with base-station networks.

2.3 TCP Layer

In the same 1997 study referenced above, Stemm and Katz [25] also simulated the performance gains of two reliable UDP schemes. Their simulations showed that, *excluding* time when the radio was idle (neither transmitting nor receiving), reliable UDP consumed over 50% less power than TCP with delayed acks. However, the paper concluded that "...the choice of transport layer can have a significant impact on the number of packets sent and received...[but] the actual power difference is minimal...because the energy consumed simply by keeping the network interface on during the transfer contributes the most to the final energy cost."

However, in 1999 the 802.11b specification [17] defined a *power saving mode* (PSM) that drastically reduces the amount of energy used when the radio is idle.

Balakrishnan et al. [2] proposed the snoop protocol, which masks losses over wireless links from the sender by intercepting duplicate acks, which signal a packet loss, and resending the required data to the mobile host. This approach has the advantage of being transparent to sender and receiver and appears to alleviate TCP problems with noisy links. However, it does not improve energy efficiency or bandwidth over clean or simply congested links, where few duplicate acks are sent.

Balakrishnan et al. [3] proposed ack-filtering to reduce the number of ack packets sent on slow upstream links, e.g., those often used with satellite downlinks. Ack-filtering can increase downstream bandwidth when acks do not arrive quickly enough. The technique is applicable to mobile networks because they are half-duplex, and thus every ack that is sent upstream reduces the downstream bandwidth. Our SRDP protocol attempts to solve this problem by using sacks and delayed acks; ack-filtering would be another approach.

Unfortunately, ack-filtering would likely require kernel TCP modifications on the mobile host. It might also be possible to implement it at the 802.11 layer.

Yavatkar and Bhagawat [27] showed that a split-connection proxy scheme similar to our own can increase simulated performance under various noise levels. They too used a specialized protocol over the wireless link. Since their paper was published in 1994, mobile networking technology has changed significantly, and many of the specifics of our techniques differ from theirs. For example, we use a larger MTU than is commonly available on the Internet whereas they used a much smaller one.

2.4 Higher Layer

Caching Web proxies [24] have been around for over a decade and are effective. Moreover, almost all mobile devices have support at the application level for Web proxies.

In the early to mid 90's, many researchers tried to reduce the number of bytes sent over wireless links or slow modem connections by a technique called *transcoding.* Transcoding techniques include reducing image size and quality, modifying multimedia content, changing text formats, and eliminating unimportant information. Some transcoding proxies have gone into commercial deployment [7][9][16], e.g. T-Mobile's Accelerator and AvantGo. IBM sells a commercial transcoding proxy. These products try to reduce latency and bandwidth. We propose the use of some transcoding techniques such as having the proxy resolve hostnames.

Barr [4] showed that compression of network traffic could be energy-efficient in certain situations. Our dual-proxy architecture allows this technique to be easily implemented without any modifications to the mobile device or access point software. Krashinsky [21] described a dual-proxy system similar to our own that compressed HTTP traffic.

There has been much work on the energy management of hardware devices ([12] is a representative paper). Many of these techniques operate at the OS level [14], although some work at the application level [18]. Moving control of the networking hardware power to the client proxy would enable the network hardware to be turned on whenever it is needed.

2.5 Prior Measurement Studies

Jun [19] calculated the theoretical performance limits of various 802.11 standards. These limits are applicable when a large number of devices or high-bandwidth applications use the network. In section 4 we calculate how some of our schemes will increase the theoretical performance of 802.11b, and in section 5 we measure their actual effectiveness.

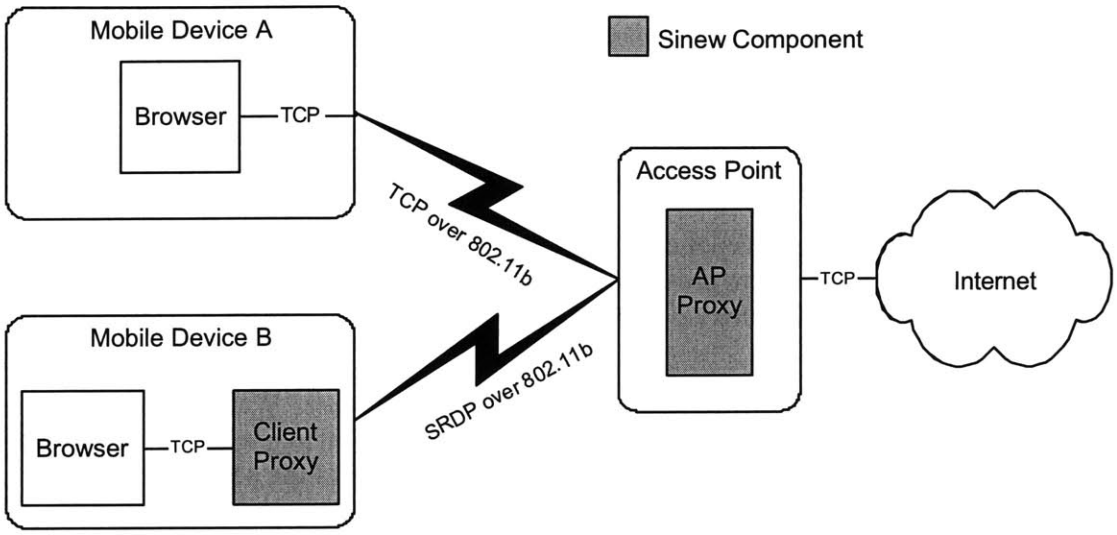
Feeny [13] published extensive power measurement data for 802.11 devices in an ad-hoc environment. Unfortunately the data cannot be used to calculate the energy usage of 802.11b devices in PSM mode.

3 Design

Sinew reduces mobile devices' network energy usage while increasing the average bandwidth. The system comprises two main components: an *access point proxy* and a *client proxy*. We implement both proxies as user-level programs so that they may be easily used on different operating systems or hardware.

Figure 1 illustrates the Sinew components.

Figure 1 – Sinew Overview



Applications and operating systems interface with our proxies through standard TCP or HTTP proxy mechanisms. This interface ensures that end users will be able to understand and use our proxies with minimal training. Also, existing applications such as Web browsers do not need to be rewritten or recompiled. Application modification would be a major barrier to adoption.

The access point proxy may be integrated with the access point (as in the figure above) or may be connected to the access point via a local network. For research purposes we used the former configuration – our access point was a PC running Linux. A low cost commercial access point could be used instead of a PC; in this case the access point proxy would run on a server elsewhere on the LAN[2].

The AP proxy can use the following energy-saving schemes, each of which is described in more detail later:

---

[2] One of our techniques uses packets larger than 1500 bytes, which is the maximum packet size for 10 and 100 Mbps Ethernet networks. To continue to use this technique, the proxy server would need to be connected to the access point via a network that supported large packets, e.g., gigabit Ethernet. Alternately, the access point would need to be modified.

- Buffering
- Packet size optimization
- DNS transcoding for HTTP traffic

The client proxy is an *optional* user-level program that runs on the mobile device. We made the client proxy optional because, while it enables additional benefits, it requires the user to install additional software. A client proxy, when used in conjunction with an access point proxy, enables the following energy-saving schemes:

- Simple Reliable Datagram Protocol, a specialized TCP replacement protocol used to communicate between the client and access point proxies
- TCP/IP parameter tuning
- Explicit power management
- Data and header compression

In Figure 1 mobile device A is not using the client proxy, and so communicates directly over TCP with the access point proxy. Mobile device B uses the client proxy and communicates with the AP Proxy over SRDP, a special-purpose reliable transport protocol.

To test our design we implement an access point and client proxy. We implement the features that we predict will have the highest energy savings, including buffering, packet size optimization, and SRDP. Section 5 contains a discussion of the performance of the system using these features.

It is important to stress that our proxy only delivers significant energy savings when used in conjunction with an efficient 802.11b power saving mode (PSM). Without PSM, the 802.11 radio is always on, and thus reductions in traffic do not save energy.[3] Thankfully, PSM is widely deployed.

3.1 Base Station Proxy

Our base station proxy is a scalable, portable, and extensible TCP and HTTP proxy. It uses a number of schemes to reduce the power and increase the performance of 802.11b networks. Our base station uses non-blocking IO to provide scalability. See Appendix B for notes on the implementation.

The following section describes those schemes in detail. We implement a subset of these features.

3.1.1 Buffering

Buffering is possible when one splits the TCP connection. When the client requests a large amount of information, e.g. a large Web page, the proxy will retrieve that file, store it in a buffer, and then resend it to the mobile client out of the buffer.

[3] Although, as we will show, we still achieve significant bandwidth increases even without PSM.

The split TCP connection shields the sender and the mobile device from losses or congestion on each others' networks. This concept is very similar to snoop [2], which also buffered packets. The energy bene fit of buffering comes from fewer duplicate transmissions and acks being sent over the wireless network.

In addition, as suggested in [1], buffering eliminates TCP timeouts due to drops on the wireless link and reduces client-proxy round trip time (RTT). The result is improved throughput and reduced latency.

Buffering is distinct from caching, which would try to save the information in case it is requested again. Caching might be a reasonable strategy depending upon the amount of memory available, although it is probably most effective when there are more users than are typically at a single access point.

3.1.2 Packet Size Optimization

IP packets received from the Internet at large typically range in size from 50-1,450 bytes. Smaller packets are acks, small files (e.g. buttons on a Web page) or, more rarely, fragments. The largest packet that can be transmitted from one point to another without fragmentation is known as the maximum transmission unit (MTU). The default MTU on the Internet today is ~1,500 bytes, which is the maximum packet size allowed by Ethernet.

We can save energy and increase performance by changing the size of the 802.11b packets. Our proxy accomplishes this by buffering incoming data and then sending it out in optimally sized packets.

802.11 offers 2,296 bytes[4] (called the frame body or data field) of payload to higher-level protocols. In the absence of losses, large packets are more efficient. This is true because 802.11b, like Ethernet, requires a contention period between each packet sent. Amortizing this contention period over a greater number of bytes reduces the total amount of delay per byte, and hence increases the effective bandwidth.

With losses, determining the optimal packet size becomes much more complicated, and is out of the scope of this paper. Techniques such as those used in [9] or online learning could be used to set the optimal packet size

In section 4 we show the theoretical improvement due to a higher MTU, and in section 5 we show that in our test setup a larger MTU offers a significant performance improvement.

---

[4] Actually, implementations must support frame bodies of 2,312 bytes to accommodate WEP overhead. 8 bytes are then used for WEP and 8 for Link Layer headers. Some implementations, such as the Linux hostap and orinoco-cs drivers, set the maximum payload to 2,290.

3.1.3 Jumbo IP Packets with Fragment Burst

Figure 2 – 802.11b Packet Timing without Fragments

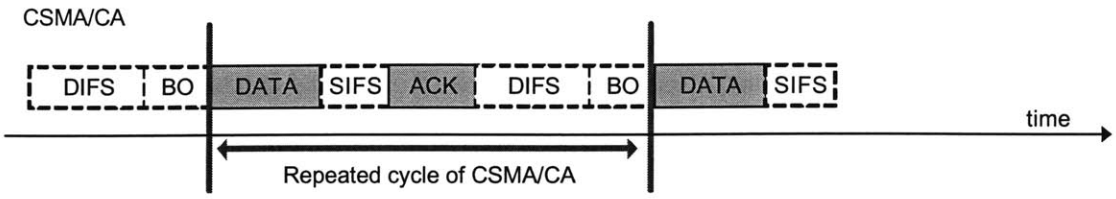
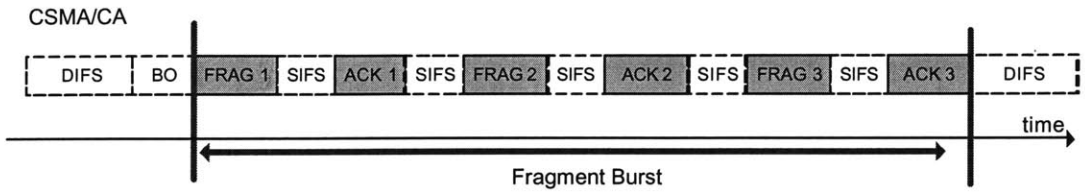


Figure 3 – 802.11b Packet Timing with Fragments (Fragment Burst)



There may be a way to send very large IP packets over 802.11b without fragmenting them at the IP level, thus providing for efficient transport.

Non-fragment packets are sent after DIFS[5] amount of time plus a backoff period (BO), as in Figure 2. However, as Figure 3 shows, 802.11b fragments are sent after SIFS amount of time. Since SIFS is much smaller than DIFS, and no contention or backoff period is required, more bytes can be sent per unit time with fragments than without.

The maximum number of fragments is 16, so that the maximum number of bytes that we might send in this way is 36,736. The maximum IP packet size is 2^{16} or 65,536 bytes.

It is unclear whether existing 802.11b implementations would tolerate a packet with a payload greater than 2,296 bytes, but it does not appear to be unreasonable. The benefits of this scheme are modeled in section 4. We did not implement this scheme.

3.1.4 Transcoding

Classical Web transcoding techniques change the resolution of images, the frame rate of video, or cut out advertisements. These changes are often visible to the user and are typically used in situations where bandwidth is severely limited, such as with cell phones. We are not interested in these lossy approaches here.

However, there are certain cases where transcoding can still be useful. For example, one traffic trace done with a Compaq iPaq showed that approximately 20% of all packets were DNS lookups.[6] This can occur when browsing Web pages that use fully qualified domain names instead of relative names, or that have many links to external domains. A

---

[5] DIFS is the Distributed Inter-Frame Space, SIFS is the Short Inter-Frame Space. In 802.11b DFIS is 50 μs and SIFS is 10 μs.

[6] Some browsers will cache DNS lookups.

browser may do a DNS lookup for each one of these URLs, thus increasing latency and wasting bandwidth. With an HTTP proxy already running, it is an easy step to replace all hostnames in the HTML with IP addresses.

3.2 Client Proxy

The client proxy is an optional component of Sinew. It is optional because some mobile hosts might not have the computational power or space to run the proxy, or some users may not wish to use it.

Use of our client proxy in conjunction with the access point proxy allows the following performance-enhancing schemes to be used.

3.2.1 Simple Reliable Datagram Protocol (SRDP)

We create a new protocol, SRDP, to replace TCP over 802.11b links. SRDP is a reliable UDP-based protocol similar in spirit to SRP [27]. Like TCP, it provides in-order, reliable delivery of bytes. Unlike TCP, it adjusts its ack rate and does not need to worry about congestion control (we rely on 802.11b for that). SRDP is designed to appear identical to TCP to applications.[7] Table 1 summarizes the key feature differences between TCP and SRDP. SRDP is described fully in Appendix A.

Table 1 – SRDP Feature Comparison

Feature	TCP	UDP	SRDP
Reliable	X		X
In-Order	X		X
Congestion Control	X		
SACK[8]			X
Max Packets between acks	2		no limit

SRDP should be faster than TCP because it does not have to ramp up its congestion window or back off when packets are lost. Also, since it uses sacks or delayed acks, SRDP does not have to ack every packet. In a half-duplex environment, acks reduce the amount of data that can be sent. It would also be possible to use 802.11b acks to generate TCP acks, but this would likely require kernel modifications.

As its name implies, SRDP is designed to be simple and, hence, easy to implement. It is suitable for implementation in user mode.

While SRDP has several advantages, the introduction of a new protocol must be considered carefully. SRDP may have adverse interactions with lower and higher-layer protocols, routers, [9] or applications that expect TCP or TCP-like performance. Since

[7] Our SRDP socket implementation has the same methods as a normal TCP socket.

[8] TCP SACK has been widely discussed and different approaches have been proposed; however, it is not widely deployed. RFC 2018 and 2883 and others discuss TCP SACK.

[9] Since SRDP operates over a single hop or LAN it should not be going through routers.

SRDP sends at the maximum rate allowed by lower layers, it has the potential to crowd out protocols that back off when faced with congestion. It is also possible that by not backing off, SRDP could aggravate congested wireless networks. Further studies and simulations are required to address these issues.

3.2.2 Compression

A dual-proxy platform is also an excellent way to implement a protocol-level compression scheme. While we have not done so ourselves, an approach similar to that taken by Barr [4] or Krashinsky [21] would be appropriate.

3.2.3 Application Power Management

Web browsing, email, and file transfer do not require a mobile device to be able to accept incoming connections. [10] Thus it should be possible to turn on the radio whenever the client requests a new page or file. Our client proxy provides a convenient platform on which to build this functionality.

3.2.4 IP Parameter Tuning

Several papers have suggested adjusting or dynamically tuning TCP/IP parameters based on network conditions. Our dual-proxy platform enhances such efforts by providing an easy way to coordinate adjustments between the sender (the access point) and receiver (the mobile host). We modify some parameters, such as send and receive buffer size, on the mobile host.

4 Theoretical Calculations

We modeled the increase in the maximum theoretical throughput to 802.11b from our various performance-enhancing schemes. We drew heavily on the data in [19].

We made the following simplifying assumptions:

- We use CSMA/CA, not RTS/CTS.[11]
- We use the 11Mbps data rate.[12]
- TCP always sends at the maximum rate allowed by the hardware (ignore slow start and back-off).
- There are no dropped packets and no collisions.

---

[10] Inability to receive incoming connections does cause problems with some protocols, such as FTP. However, firewalls have prevented incoming connections for many years so there are well-established work arounds.

[11] CSMA/CA and RTS/CTS are different techniques a host may use to determine if a wireless channel is available for sending. RTS/CTS is typically used in noisier or more crowded environments.

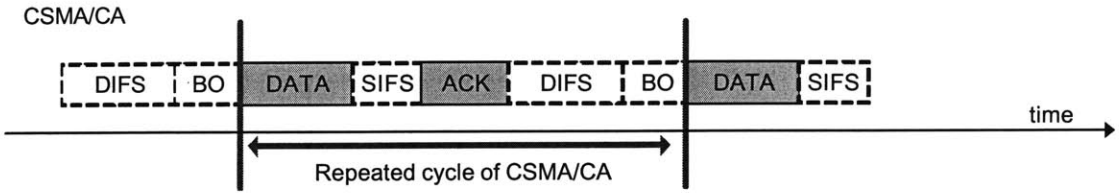
[12] The model does take into account the fact that some management frames are sent at lower speeds.

Let D be the delay per packet, X be the number of data bytes sent per packet, B be the bit-rate at which the packet is sent, and TMT be the theoretical maximum throughput of 802.11b (without any TCP or IP headers). Then

$$TMT = \frac{X}{\frac{X}{B} + D}$$

One can think of X/B as the variable time per packet and D as the fixed time per packet. As Figure 4 illustrates, D is the sum of multiple delays.

Figure 4 – 802.11b Packet Timing



T_{DIFS} is the wait time before the contention period can begin, T_{BO} is the average contention time, T_{SIFS} is the wait time before the 802.11b ack can be sent, T_{ACK} is the time it takes to send the 802.11b ack, and T_{DATA} is the time required to send the payload.

$D = T_{DIFS} + T_{SIFS} + T_{BO} + T_{ACK} + T_{DATA}$.

TCP requires that we send an ack for every two data packets received.[13] The delay for a single 40 byte TCP packet is 920 µs, so the effective delay, T_{TCPACK}, is half that, or 460 µs. We can treat a TCP ack as a delay because the 802.11 channel is half-duplex, so that we must contend for each TCP packet. A full duplex channel would allow acks to be sent without affecting the data flow.

$D_{TCP} = T_{DIFS} + T_{SIFS} + T_{BO} + T_{ACK} + T_{DATA} + T_{TCPACK}$

We also need to take into account the overhead of the TCP and IP headers, which amount to 40 bytes total. A 1500 byte Ethernet packet has at most 1460 bytes of non-header data.

Plugging in these figures yields $TMT = 4.8$ Mbps. This is perhaps surprising given that the bit rate of 802.11b is 11Mbps. Table 2 shows the variable values used and the TMT for each of our performance-enhancing schemes.

SRDP is our Simple Reliable Datagram Protocol with an ack sent for every 10 packets received. We found this setting to work in practice. We did not use Sack.

BIGSDU is modeled using a 2290 byte 802.11b packet.

[13] This is a conservative assumption; without delayed ack, TMT decreases to about 4.0 Mbps

BIGIP refers to the fragmentation burst scheme explained in section 3.1.3. The effectiveness of this scheme comes from the fact that we can amortize the DIFS and BO period over each of the fragments. As noted above, however, it is unclear whether existing hardware would support this scheme.

DCOMP is a simple data compression scheme. The 35% compression ratio is based on [21] and serves to illustrate how data compression could increase effective throughput. The number is likely to change significantly depending on the kind of traffic – images are already compressed while text compresses well.

We point out that the SRDP+BIGSDU scheme, which we implement, is predicted to increase the total maximum throughput to 6.9Mbps, a 31% increase. In section 5.2.2 we

Table 2

Parameter	Units	Scheme						
		TCP	SRDP	BIGSDU	BIGIP	DCOMP	SRDP+ BIGSDU	SRDP+ BIGIP+ BIGSDU
802.11b								
Effective DIFS	ms	50	50	50	10	50	50	10
SIFS	ms	10	10	10	10	10	10	10
Average Backoff	ms	310	310	310	62	310	310	62
Time Ack	ms	304	304	304	304	304	304	304
Time Overhead	ms	192	192	192	192	192	192	192
Overhead Bytes	bytes	34	34	34	34	34	34	34
IP								
Number of IP Frags		0	0	0	5	0	0	5
Amortized IP Overhead	bytes	20	20	20	4	20	20	4
UDP								
UDP Overhead	bytes	0	8	0	0	0	8	1.6
TCP / SRDP								
Amortized Overhead	bytes	20	13	20	4	20	13	2.6
Time Ack	ms	460	92	460	184	460	92	18.4
Packets per Ack		1	10	1	1	1	10	10
Compression								
Data		0%	0%	0%	0%	35%	0%	0%
Bitrate	mbit/s	11	11	11	11	11	11	11
Time per Packet	ms	2442	2074	3023	1878	2442	2655	2294
Throughput	mbits/s	4.8	5.6	6.0	6.4	7.4	6.8	8.0
Bitrate Increase over TCP	mbits/s	0%	18%	25%	33%	55%	42%	67%

show a measured increase to approximately 6.3Mbps.

The combination of SRDP, BIGSDU, and BIGIP would result in 8.0MBps throughput.

5 Measurements

The following section presents performance and energy efficiency measurements. We first describe the test setup and then present and analyze the data.

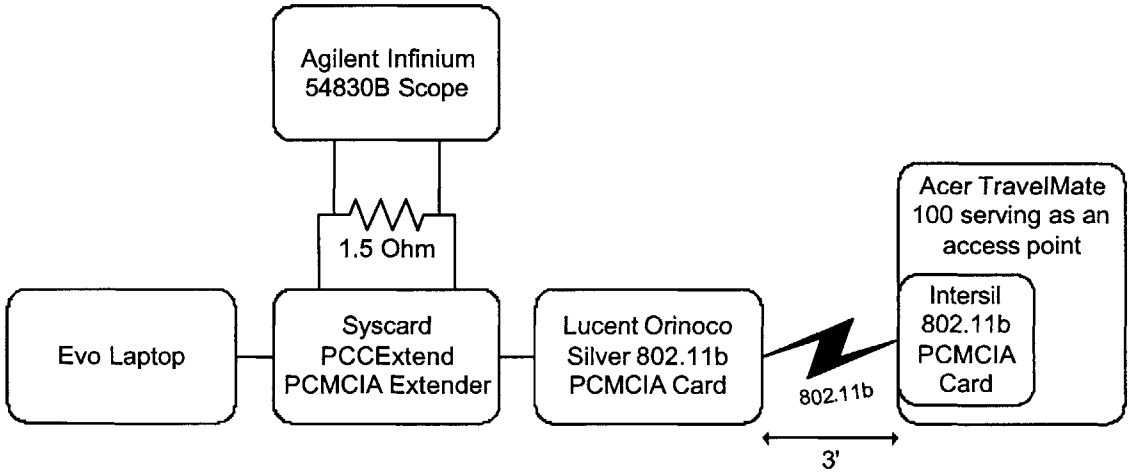
5.1 Test Setup

We desired an environment that would give us accurate and precise results. To that end we used precise equipment and operated on a low noise channel with no other users. Tests were conducted between 10PM and 2AM to minimize other possible traffic, such as random nodes trying to use our access point. Our mobile host was approximately 3 feet from our base station.

To measure power we created a test circuit to measure the power use of 802.11b card. We placed a small, high-accuracy, resistor R in series with the power supply to the card, V_{cc}. We then used two voltage probes[14] to measure the voltage drop V_r over the resistor. The current flowing to the card I_r is thus V_r/R. We then calculated the instantaneous power use P of the card by $P = V_{cc} \cdot I_r$.[15] To calculate the energy use we summed the voltage samples.

Figure 5 illustrates the test setup.

Figure 5 – Experimental Test Setup



Unless otherwise noted the 802.11b card was in power saving mode with a beacon period of 0.1s and was operating at 11Mbps. 802.11b fragmentation was disabled and the card was in CSMA/CA mode.

---

[14] A more accurate measurement circuit would use a single differential probe instead of two passive probes.

[15] This formula is approximately correct for small values of R; to get an exact reading one would use the voltage actually going into the card.

We used two techniques to give the mobile host access to the local network and the Internet. If the experiment used the access point proxy, then we setup the access point routing tables to route packets through the mobile host over the wireless interface and all other packets over the wired interface. If the experiment did not use the access point proxy then we used the `brctl` utility to bridge the mobile host to the local network. We could not use `brctl` in the former case because it did not allow packets larger than 1500 bytes.

5.1.1 Equipment Details

The mobile host was a Compaq Evo laptop running Debian Linux 2.4.21. It used a Lucent Orinoco Silver PCMCIA 802.11b card.

The base station was an Acer TravelMate C100 laptop running Debian Linux and a 2.4.21 kernel. We used an Intersil PCMCIA Ethernet card. The hostap-0.0.4 driver provided the access point functionality.

Power measurements were conducted using an Agilent Infinium 54830B scope. We collected data at 25-50 KSamples/s. [16] We used a Syscard PCMCIA extender to get access to the power pins on the 802.11b card. We used two passive probes to measure the voltage drop across a 1.5 Ohm (±1%) resistor in series with the VCC terminal of the PCMCIA card extender.

5.1.2 Data Analysis Procedure

Most of our measurements lasted ~40s and captured ~1 million data points, the maximum allowed by our scope. To compute the total energy used we exported the waveform and manipulated the data points with an analysis program.[17]

5.2 Measurements

Our two performance metrics were *energy* (in Joules) and *time to completion*. Energy is important to users because it determines how long their battery will last. Time to completion is important because users do not like waiting for Web pages to load or files to transfer. In some cases we report bandwidth, which is inversely proportional to time to completion.

Energy and time to completion are closely related – the longer a transmission takes, the longer the radio is transmitting and receiving, and so the more energy it uses.

---

[16] We believe that the shortest amount of time that an 802.11b radio turns on is about 0.2ms – the time it takes to send a CTS or RTS packet. If we want to make sure that we capture every time the radio is not idle, we need to make sure we sample that event multiple times. We arbitrarily decided that we wanted 5 samples/event. This results in a minimum sampling rate of 25,000 Samples/s. We could not determine the sampling rate used in other studies.

[17] The Agilent 54830B did not accurately compute averages over large waveforms, which would have given us enough information to calculate the energy used, so we were forced to do these calculations ourselves.

The most interesting measurements quantify what users are actually doing. We constructed 4 data sets that correspond to typical user activities. The following table describes the data sets and how they relate to user activity.

Table 3 – Description of Data Sets

Data Set	Typical User Activity
Baseline	The user is doing nothing
Bulk	The user is downloading a large file (10MB or more) from the LAN; 802.11 is the bandwidth bottleneck
Single File	The user is downloading a medium-sized (500K) file from the Internet; the bottleneck is likely on the Internet
Web Suite	The user is browsing Internet Web pages.

It is important to note that when we refer to energy we are referring to the energy used by the network card, *not* the total system. Our client proxy is certain to incur some incremental CPU and memory energy cost, although rough profiling using `top` indicated that our client proxy rarely used over 10% of the CPU. An efficient implementation would likely reduce this even further. This is an area of future work.

5.2.1 Baseline

Baseline power use is an important figure. If it were the case that baseline power use were similar to transmit or receive power, as was true in the early 90's [25], then changing the number or size of packets sent over the link would have little impact on energy.

However, as the following table illustrates, Idle is about 10 times more expensive than PSM. This factor of 10 is the key reason why we are able to save energy by increasing the effective bandwidth of the 802.11b link.

Table 4 – 802.11b Card Characteristics

Lucent Orinoco Silver	Manufacturer Spec	Measured
Sleep	45mW	73mW
Idle	-	890mW
Receive	925mW	-
Transmit	1425mW	-
Average in PS Mode	-	94mW
Power Supply	5V	4.74-5.07V

We are not able to explain the discrepancy between the manufacturer's and our sleep mode power measurements.[18]

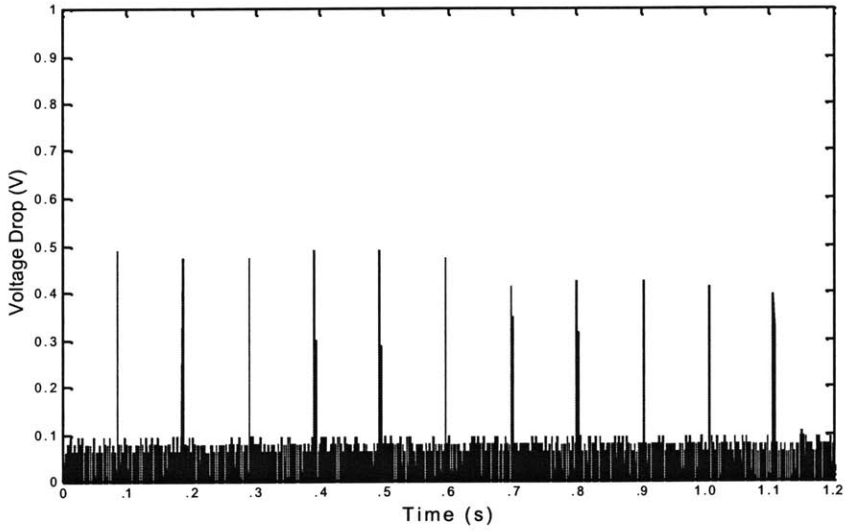
To obtain the above data we placed the 802.11b card in PSM mode and measured the voltage drop over the resistor. Our sample period was about 40s.

Table 5 – 802.11b PSM Measurements

Measurement	Average Value (n=2)
Average voltage	28.9mV
Average voltage between wakeup periods	22.0mV
Average voltage during wakeup time	259mV
Wakeup period	102ms
Duration of wakeup	1.80ms

Figure 6 is the actual output of the scope and helps visualize the data above. Notice that the card is waking up every .1s as anticipated.

Figure 6 – Power Saving Mode Scope Output



5.2.2 Bulk Data

The bulk data set measures the maximum throughput of the 802.11b link with and without our proxies. Maximum throughput is the figure of interest when a user is transferring a large file over a LAN, in which case the slow 802.11b link is likely to be the bottleneck.

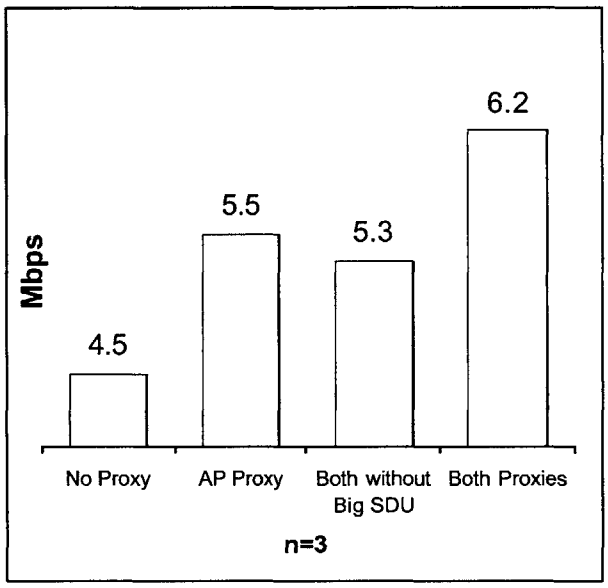
---

[18] We note that [13] also observed a significant (25mW) difference between measured and spec. There may be some variance based on what mode the card is in.

We used a custom program to generate and send random bytes from the base station and the `netcat` utility to receive those bytes on the mobile host. The bash `time` command was used to measure the transfer time on the mobile host. Our byte generation program could send data at at least 100Mbps, so it was never the bottleneck.

In PSM mode our proxies increased the throughput of the 802.11b link by 38%, from 4.5Mbps to 6.2Mbps. With only the access point proxy – meaning that we used TCP over the link instead of SRDP – we achieved a 22% increase. Figure 7 illustrates the gain.

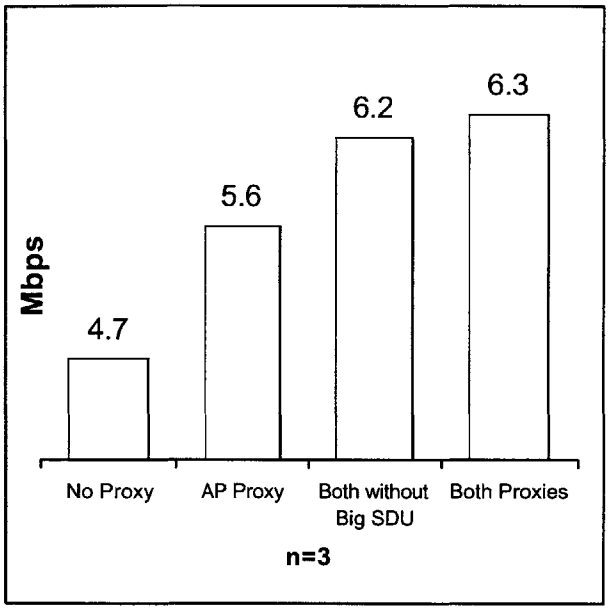
Figure 7 – Effect of Proxies on Maximum Throughput with PSM



We also measured the effect of the proxies on maximum throughput without PSM mode. This measurement would be interesting in a real-world deployment since there would likely be a combination of devices with and without batteries. Without PSM we achieved a 35% increase in maximum throughput, from 4.7 to 6.3Mbs. The numbers are similar because the host is not sleeping during the file transfer.

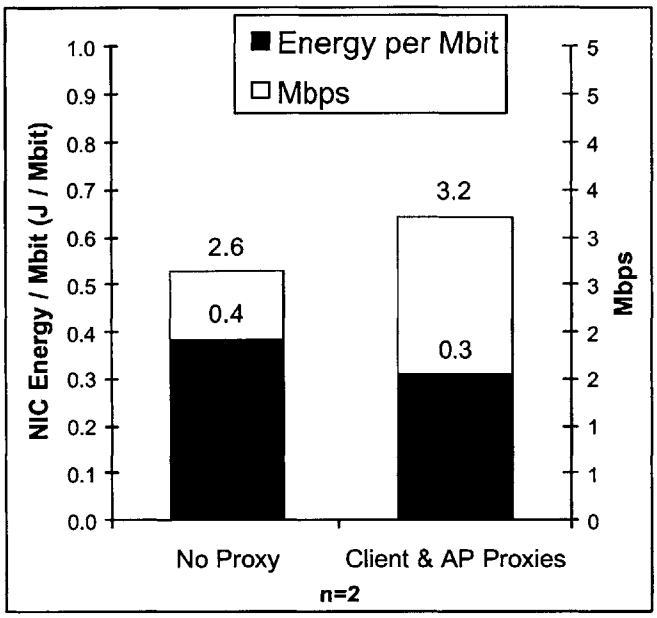
It is interesting to note that even with a 1500 byte SDU, we were able to achieve a 32% increase in bandwidth. SRDP is likely the major contributor.

Figure 8 – Effect of Proxies on Maximum Throughput without PSM



We also measured the energy consumption of the network card during a bulk transfer. Using both our proxies we reduced energy consumption per Mbit by 20%. As Figure 9 illustrates, energy is roughly proportional to speed.

Figure 9 – Energy per Mbit with and without Proxies



5.2.3 Single File

The single file test gives us an idea of how our system performs when the 802.11b link is not the bottleneck.

To perform the single file transfer test we downloaded a 325K file from the Internet using `wget`[19]. For each energy measurement we downloaded the same file 10 times, with a 1 second delay between each download. A new TCP connection was established for each retrieval. We chose the file arbitrarily but tried to ensured it was on route with few dropped packets and no routing abnormalities. We subtracted the baseline energy cost from the total energy of the capture after the 10 files were retrieved.

For a single file we obtained a 24% decrease in the energy required when using the access point proxy and a 23% decrease when using both proxies. It is possible that the extra latency introduced by the client proxy outweighed the benefits of reduced ack traffic from SRDP. In addition to saving energy, our proxies reduce the latency experienced by the user by up to 17%.

Figure 10 displays this data graphically and shows the correlation between energy savings and time to completion. Figure 11 shows the throughput with and without the proxies.

---

[19] wget is a unix utility to fetch web pages. The actual command line used in the transfer tests was `wget -C off -r -p -H -U "Internet Explorer" -e robots=off --no-http-keep-alive --delete-after [URLs]`

Figure 10 –Single File Energy Savings

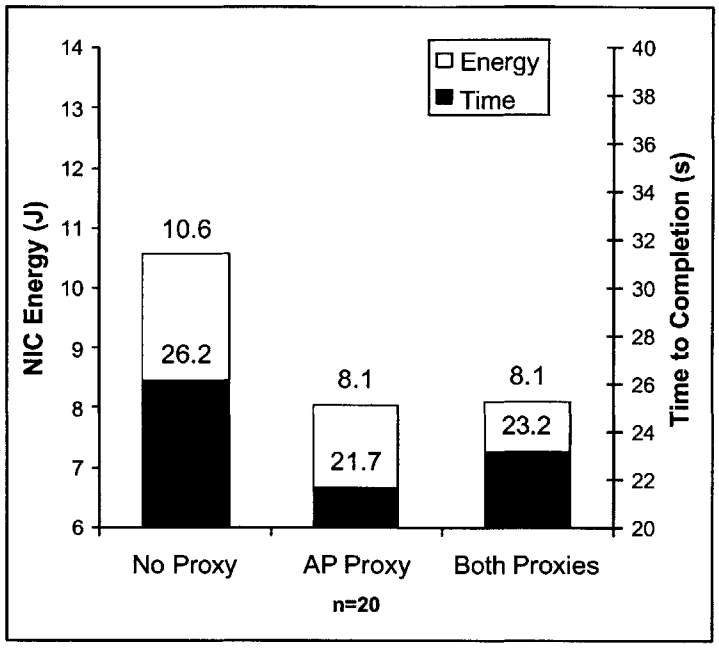
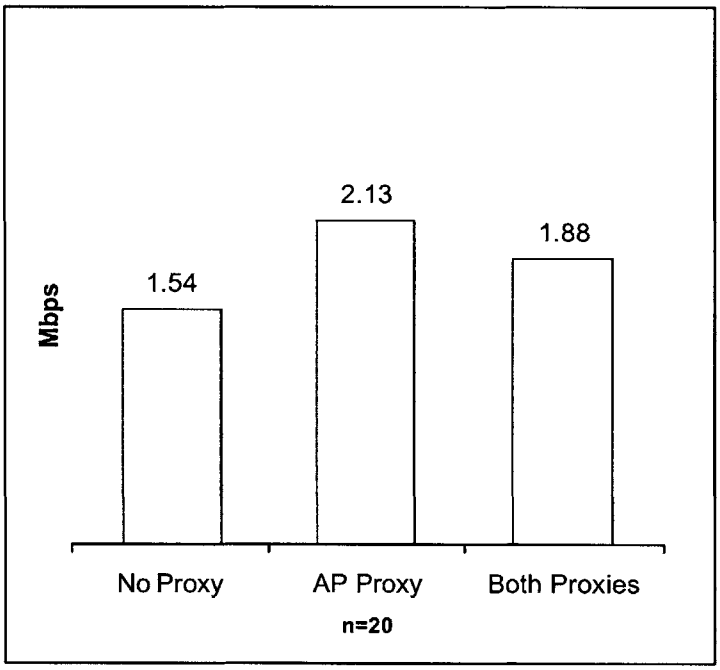


Figure 11 – Single File Throughput



5.2.4 Web Suite

We downloaded 5 popular sites from the Internet and measured the performance of the AP proxy. The client proxy was not used because it introduced additional overhead. Table 6 lists the contents of the suite and Figure 12 shows our performance.

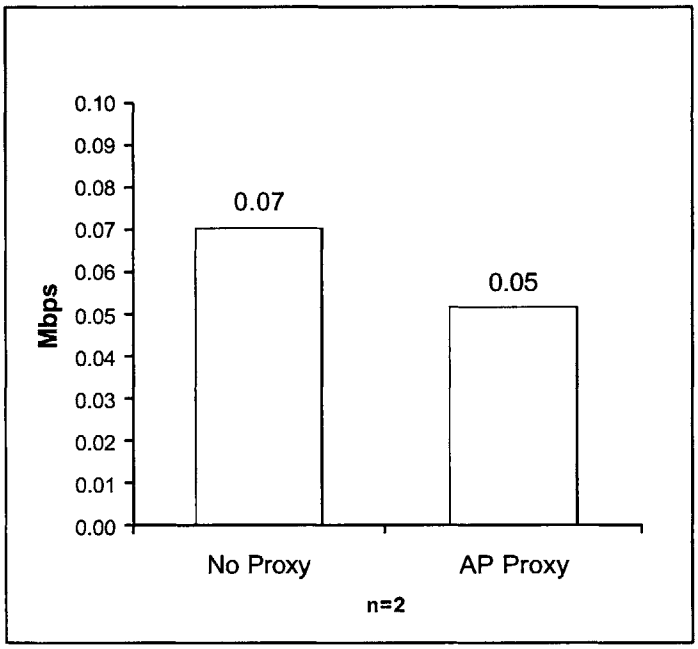
The effective transfer rate is very low due to the large number of small files (184 in total, with an average wise of about 3K). Because of the large number of small files, the overhead incurred by going through the proxy becomes significant.

One way to fix this problem is to have the proxy *prefetch* each file once it received the original request. That is, when the initial request is received, the proxy guesses (based on the links in the HTML) which files the browser will request next and caches them.

Table 6 – Web Suite

Site	Size (bytes)	Number of Files	Average Size (bytes)
www.cnn.com	204,732	67	3,056
www.netscape.com	91,618	31	2,955
www.yahoo.com	52,054	15	3,470
www.slashdot.org	63,814	5	12,763
www.news.com	145,565	66	2,206
Total	**557,783**	**184**	**3,031**

Figure 12 – Web Suite Throughput



6 Future Work

6.1 Measure Total System Energy Change

While we do not believe our client proxy is resource intensive, it is certainly using some amount of energy. It would be very valuable to know what the total system energy savings is.

6.2 SRDP

Our research implementation of SRDP was not optimized; the proxy had high latency and resource usage. While SRDP should remain simple, it might benefit from some additional features like window. Also, we have not studied SRDP in a congested environment. Modifications might be necessary.

6.3 Implement Additional Performance Enhancing Schemes

As [4] and [21] showed, compression can significantly increase energy efficiency and performance. Our models show that compression should have a incremental benefit. With the existing architecture it would be relatively easy to compress the traffic between the client and access point proxies. More challenging, perhaps, would be to determine under what situations compression is appropriate.

While we do not implement the application power management, jumbo IP packet scheme, or transcoding we believe they hold significant promise. Other schemes may also be possible.

6.4 Other Wireless Networks

We believe many of the approaches we have taken are applicable to other wireless networks, including 802.11a and 802.11g. Indeed, the code would run unmodified on these networks and should scale easily.

6.5 Hostile Network Environments

Our system has not been tested in noisy or congested networks, in part because such testing is difficult to do. Such testing is clearly a prerequisite for deployment. How our system will perform at slower or higher network speeds is uncertain, although [19] has some data on the theoretical effect of SDU.

6.6 Other Architectures

Our access point and access point proxy ran on the same physical computer. It should be possible to have one access point proxy serve multiple commercial access points, although to get the benefits of large SDU packets the connecting network would have to support jumbo frames.

7 Conclusion

7.1 Summary

We design, build, and test a system to decrease the energy consumption and increase the performance of wireless networks. Our system is minimally invasive, requiring only user-level software programs running on commodity hardware. With our Java-based research implementation, we achieve up to a 20% energy savings and a 38% increase in network throughput.

7.2 Acknowledgements

I would like to thank Jason Waterman for his extensive help with systems administration and answering my many Linux questions. Eugene Weinstein and Ken Steele provided valuable feedback and support.

8 References

[1] H. Balakrishnan, S. Seshan, and R.H. Katz. Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks. *ACM MOBICOM*, 1995.

[2] H. Balakrishnan, V. Padmanabhan, S. Srinivasan, and R. Katz. A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. *IEEE/ACM Transactions on Networking*, December 1997.

[3] H. Balakrishnan, V. Padmanabhan, and R. Katz. The Effects of Asymmetry on TCP Performance. *MONET*, 1999.

[4] K. Barr and K. Asanovic. Energy Aware Lossless Data Compression. *First International Conference on Mobile Systems, Applications, and Services*, May 2003.

[5] A. Barke and R. Badrinath. I-TCP: Indirect TCP for Mobile Hosts. *Proc. 15th International Conf. On Distributed Computing Systems (ICDCS)*, May 1995.

[6] A. Fox and E. Brewer. Reducing WWW Latency and Bandwidth Requirements by Real-Time Distillation. *5th Annual WWW Conference*, 1996.

[7] C. Brooks, M. Mazer, S Meeks, and J. Miller. Application-Specific Proxy Servers as HTTP Stream Transducers. *Fourth International World Wide Web Conference*, December 1995.

[8] H. Bharadvaj, A. Joshi, and S. Auephanwiriyakul. An Active Transcoding Proxy to Support Mobile Web Access. *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*, 1998.

[9] C.F. Chiasserini and M. Meo. Improving TCP over Wireless through Adaptive Link Layer Setting. *IEEE GLOBECOM*, November 2001.

[10] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: an Energy-Efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks. *ACM Wireless Networks*, 8(5), September 2002.

[11] J-C. Chen, K. M. Sivalingam, and P. Agrawal. Performance Comparison of Battery Power Consumption in Wireless Multiple Access Protocols. *ACM Wireless Networks*, 5(6):445-460, 1999.

[12] F. Douglas, P. Krishnan, and B. Marsh. Thwarting the Power Hungry Disk. In *Proceedings of the 1994 Winter USENIX Conference*, January 1994.

[13] L. Feeney and M. Nilsson. Investigating the Energy Consumption of a Wireless Network Interface in an Ad Hoc Networking Environment. *Proc. IEEE INFOCOM*, April 2001.

[14] J. Flinn. Extending Mobile Computer Battery Life through Energy-Aware Adaptation. Ph.D. dissertation, TR# CMU-CS-01-171, Carnegie Mellon University, December 2001.

[15] A. Fox and E.A. Brewer. Reducing WWW Latency and Bandwidth Requirements by Real-time Distillations. *Fifth International World Wide Web Conference*, May 1996.

[16] IBM Transcoding Proxy Website. http://www.research.ibm.com/networked_data_systems/transcoding/index.html

[17] IEEE Computer Society LAN MAN Standards Committee. IEEE Std 802.11: Wireless LAN Medium Access Control and Physical Layer Specifications, August 1999.

[18] Intel Corporation. Application Power Management for Mobility. White Paper, March 20, 2002.

[19] J. Jun, P. Peddabachagari, and M. Sichitiu. Theoretical Maximum Throughput of IEEE 802.11 and its Applications. *Second IEEE International Symposium on Network Computing and Applications*, April 2003.

[20] R. Krashinsky and H. Balakrishnan. Minimizing Energy for Wireless Web Access with Bounded Slowdown. *MOBICOM*, September 2002.

[21] R. Krashinsky. Efficient Web Browsing for Mobile Clients using HTTP Compression. Course term paper [Unpublished], December 2000.

[22] R. Kravets and P. Krishnan. Application-driven Power Management for Mobile Communication. *ACM Wireless Networks*, 6(4):263-277, 2000.

[23] E. Shih, P. Bahl, and M. Sinclair. Wake on Wireless: An Event Driven Energy Saving Strategy for Battery Operated Devices. *MOBICOM*, 2002.

[24] Squid Web Proxy. http://www.squid-cache.org/.

[25] M. Stemm and R.H. Katz. Measuring and Reducing Energy Consumption of Network Interfaces in Hand-held Devices. *IEICE Transaction on Fundamentals of Electronics, Communications, and Computer Science*, August 1997.

[26] J. Wong and V. Leung. Improving End-to-End Performance of TCP Using Link-Layer Retransmissions over Mobile Internetworks. In *Proc. of ICC*, 1999

[27] R. Yavatkar and N. Bhagawat. Improving End-to-End Performance of TCP over Mobile Internetworks. *Mobile 94 Workshop on Mobile Computing Systems and Applications*, December 1994.

Appendix A: SRDP (Simple Reliable Datagram Protocol)

SRDP is a simple, easy to implement, reliable transport protocol. It is the goal of this document to provide a description of SRDP, rather than a full specification. A specification that includes states and state transitions exists but is not included to save space.

SRDP is designed, though not required, to operate on top of UDP. SRDP provides in-order delivery of bytes. It does not provide reliability, congestion control, multicast, or any other feature not explicitly defined below. We rely upon UDP to provide reliability and 802.11b to provide congestion control.

Like TCP, SRDP relies upon *sequence numbers* to provide reliable delivery. Before bytes may be sent a *connection* must be established, which establishes the sequence numbers both sides will use. Connections should also be shutdown, although that is not required.

SRDP is not a general protocol. It is designed to operate over 802.11b networks of one hop, although it may have other uses. Because of this specialization, it is missing many features commonly found in reliable transport protocols like TCP. But it is also because of this specialized use that we are able to create a simple protocol.

The following sections describe SRDP in more detail.

Packet Framing:

All numbers below are 8-bit bytes unless otherwise noted.

Every SRDP packet contains a 13-byte Header and may optionally contain a variable-length Body.

13	*0-65,460*
Header	Body

Header:

The header is a fixed-size 13 bytes, broken down as follows.

1	*4*	*4*	*4*
Type	Field0	Field1	Field2

Type	Binary	Packet Type
1	00000001	Data
2	00000010	Ack
3	00000001	Sack
4	00000011	Syn
5	00000100	Syn-Ack
6	00000101	Close
7	00000110	Close-Ack
8	00000111	Reset

The uses of field0, field1, and field2 are specific to the type, but every type should contain a fixed-length fields. This greatly eases implementation and parsing of packets. Any future extensions should adhere to this principle and put additional information in the body, if necessary.

Body:

0-65,460

Body

The body is raw bytes. The length of the body is not specified in the header.

Packet Types

Definitions:

Sender:	The sender of the packet described.
Recipient:	The recipient of the packet described.

Type 1: Data

4	*4*	*4*	*0-65,460*
Seq	Unused	Unused	Data

This packet contains data.

Seq is the sequence number of the first byte in the packet.

Type 2: Ack

4	*4*	*4*
Seq	Unused	Unused

This packet acknowledges data received.

Seq is the next sequence number *expected* by the sender, not received.

Type 3: Sack

4	*4*	*4*	*0-65,460*
Seq	Number of Sack Blocks	Unused	More Sack Blocks (8 bytes each)

This packet acknowledges data received when that data is not contiguous with existing data received.

Seq is the last contiguous sequence number *received* by the sender.

The number of sack blocks indicates the number of sack blocks, including the one in the header, which this packet contains. This is not strictly necessary since we can just look at the packet length but may help in parsing.

A sack block is a pair of sequence numbers (x,y) where x is the sequence number of the first byte received and y is the sequence number of the last byte received.

Type 4: Syn

4	*4*	*4*

Seq	Unused	Unused

This packet is used to begin an SRDP connection.

Seq is the sequence number of the next packet that the initiator will send. It is a number from $[0, 2^{32}-1]$.

Type 5: Syn-Ack

4	*4*	*4*
Seq	Synack	Port

Seq is the sequence number of the next packet that the sender of the Syn-Ack packet will send. Note that this is not the same as the Seq received in a Syn packet; that Seq is placed in Synack.

Synack is the sequence number that the receiver of the Syn packet will send. Seq is the sequence number of the next packet that the initiator will send. It is a number from $0-2^{32}-1$.

Port is the port on the receiver host to which the sender should send packets for this connection from now on. We place no restrictions on valid port numbers.

Type 6: Close

4	*4*	*4*
Last Sent	Last Received	Unused

Last Sent is the sequence number that the sender last sent.

Last Received is the sequence number that the sender last received.

Type 7: Close-Ack

4	*4*	*4*
Last Sent	Last Received	Unused

Last Sent is the sequence number that the sender last sent.

Last Received is the sequence number that the sender last received.

Type 8: Reset

4	*4*	*4*
Unused	Unused	Unused

The sender has closed the connection.

Design Notes:

- A one-hop connection is typically low-latency connection, so it is unlikely a host will be sending a receiving data at the same time. Thus, to keep things simple, we opted to have separate ack and data packets. An alternative is to have an ack flag and field, as in TCP.
- For simplicity, there is no two-sided shutdown. If one side closes the socket, it's closed.

- We used 4-byte fields and a fixed-length header to ease implementation, although it does waste some space.
- We have left much room for expansion and modification of the protocol, e.g., extra bits in the type field.

Appendix B: Implementation Notes

The client and proxy servers were written entirely in Java 1.4 to allow for portability.

In Java 1.4 Sun introduced non-blocking IO (dubbed NIO) to allow development of high-performance server applications. Our access point and client proxies were developed using the NIO model.

SRDP

SRDP is designed to mimic a TCP socket. To accomplish this we created a class, `SRDPSocketChannel`, which has identical methods and functionality to `SocketChannel`, the class that implements non-blocking TCP. We did not subclass `SocketChannel` because it would have been complex and difficult. Instead, `SRDPSocketChannel` uses a non-blocking `DatagramChannel`, which implements non-blocking UDP, for all of its communications. To use SRDP, a programmer simply replaces all references to SocketChannel with SRDPSocketChannel.[20] `SRDPSocketChannel` could thus be used in other applications where a reliable UDP is desired.

Our SRDP implementation is built on top of UDP. We wanted to keep our implementation simple and did not want to modify the `Selector` or `SelectionKey` classes, which implement `select()`. We thus `select` on the underlying `UDPChannel`. This has the effect that when the `SRDPSocketChannel` is selected for read, there may or may not be actual data available (for example, the packet awaiting delivery could be an ack). This is not a problem since we do the packet processing (e.g., processing the ack) whenever `SRDPSocketChannel.read` is called and then return data only if there were data packets available. We use a similar mechanism for `write`.

Real-World Deployment

In an actual deployment, we envision that the first-time user of a power-saving base station would be redirected to a Web page offering the client proxy software for download. Use of Java in this situation is highly desirable since the chances of installation and compatibility problems are minimized, assuming the device can run Java. Depending on the target device, it may be more feasible to provide a pre-compiled native binary or perhaps a pre-compiled Java run time environment.

[20] There are a couple of other subtle modifications but these are explained in the code documentation.