ACTIVE MODE POWER MANAGEMENT FOR MOBILE DEVICES

By

RICHARD J. LOY

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2001

TO MY WIFE AND SON

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Helal for all of his support, patience and most of all motivation. Additionally, I would like to thank Dr. Mafla for his inspiration and precious time in helping me with his expertise in the field of networking and device drivers. I would also like to thank all the students in the Harris Lab for the day-to-day learning experience they shared with me.

I could not have done any of this without the constant support of my loving wife, Maureen, who endured this entire ordeal while pregnant with our first child. I would like to thank her and express my sincere love and gratitude for the sacrifices she has made. Finally, I would like to thank my new son, Brendan, born February 26, 2001, who is sitting in my lap as I type this, keeping me company (and warm) while I put the finishing touches on this thesis.

TABLE OF CONTENTS

page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

ACTIVE MODE POWER MANAGEMENT
FOR MOBILE DEVICES

By

Richard J. Loy

May 2001

Chairman: Abdelsalam Helal
Major Department: Computer and Information Science and Engineering

As computer technology becomes more miniaturized and mobile, power management becomes an ever-increasing concern to users wanting freedom from conventional power supplies, yet full access to data while roaming. Mobile connectivity is fully available, but at the precious price of increased energy consumption. This increased energy consumption reduces mobility, requiring frequent, undesired recharge session. Current power management techniques revolve around the lower layers of the protocol stack. This approach treats all applications equally and leaves the burden of choosing the best power management settings with the user. But different applications require different levels of connectivity, and thus different energy requirements. Mobile devices such as the Personal Data Assistant (PDA), PocketPC, or mobile laptop, and the applications they support, cannot afford to all be treated equally. This thesis explores the idea of migrating power management to a much higher level – the application layer.

The key is to take as much information as possible from the applications that require connectivity and allow them to directly and dynamically control applicable power management techniques. Active Mode Power Management (AM/PM) can take advantage of today's modular languages through the use of Application Protocol Interfaces (APIs). This thesis looks at the plausibility of such API-based power management software and provides benchmark-testing figures to substantiate its validity. Power usage data, from a mobile device equipped with an IEEE 802.11b wireless LAN PC card, are presented. Additionally, known obstacles and pitfalls are discussed with reference to future development in this field.

CHAPTER 1
INTRODUCTION

The mobile consumer of the new millennium is hungry for seamless connectivity with his PDA. The market must satisfy this insatiable appetite for AOAC (Always On Always Connected) computing [HELA99]while obeying stringent power and size constraints. Unfortunately, these wireless necessities are not at all friendly companions, but rather fierce competitors. Current technologies involving wireless network interfaces consume far too much power to be adequately powered by current battery technology. Although many advances have been made in reducing energy consumption, there is much work for us on the horizon. Maybe sometime in the near future a new portable power supply, that is more efficient and lightweight than current batteries, will be discovered. But in the meantime, researchers will be very busy designing innovative new methods to reduce the amount of energy consumed, while maintaining connectivity with a mobile computing device.

Of course there are many different valuable angles to explore when attempting to reduce power consumption in a mobile device. Although we discuss several of those techniques in Chapter 2, the remainder of this thesis focuses on just one, Active Mode Power Management (AM/PM). This thesis proves that this new power-saving method is as viable as any others already in use.

The AM/PM increases the amount of information available through the use of power-aware APIs available to the application program developer. We show that the use

of these APIs can directly reduce the power consumption of a connected wireless interface PC card by using up to 62 times less power than no power management. The main idea is to examine closely the amount of power being used by a connected wireless PC card[1] when the connection is not really needed by the user's application. As anticipated, this time period can amount to a significant waste of energy. The AM/PM will "steal" back as much of that wasted energy as the application will allow. The true beauty of AM/PM is the added feature of user transparency. Since AM/PM is already preprogrammed into the application, the user need not do anything extra to reap the benefits of AM/PM, but rather simply enjoy a prolonged device "up-time" between battery recharges.

Chapters 2 and 3 discuss the background of power management techniques and related research. Chapter 4 expands on the details and motivation of the AM/PM approach. In Chapters 5 and 6 we fully outline the hardware used to test our theory and the specifics of our experiments used to verify and validate our methods. Finally, Chapter 7 concludes with a summary and discussion of future research in the area of Active Mode Power Management.

[1] By connected, we mean an active connection with a base station AP (Access Point).

CHAPTER 2
BACKGROUND ON POWER MANAGEMENT IN WIRELESS NETWORKS

2.1 Introduction

Most mobile devices employed today use battery power very efficiently while not connected to any type of wireless network, but what happens when a user wants to be connected to the multitude of information available via LAN, WAN[1] or the Internet? Current wireless connectivity technology allows for such availability, but at a precious cost. When connected via a wireless network interface card, the additional power requirements placed on a connected mobile device may reduce battery life by up to a factor of six for a hand-held pc, and up to a factor of two for a laptop [STEMM]. This all but negates the mobility of the user, requiring a necessary tethered re-charge much sooner than practical. This chapter introduces some of the various approaches attempting to minimize energy waste while remaining connected via a wireless network interface.

2.2 Wireless Interface Power–Saving States

First we examine the breakdown of defined operating states for wireless PC cards. Current proprietors have developed several different granularities for power consumption states in their wireless LAN cards. A PC card must have a minimum of two states, off and transmit/receive, but what if the receive state requires much less power to function adequately? Then we may separate this active transmit/receive state into two separate

[1] Local or Wide Area Network, which may not necessarily be connected to the Internet.

states, one for receive and one for transmit.[2] Typically there are a minimum of at least four basic states. Table 1 shows the four basic states available for most wireless LAN cards on the market today (2001). As you can see, the most power intensive state of any wireless card is the transmit state. The act of transmitting radio waves has a property worth mentioning for the purpose of exploiting. The power drain is asymptotically proportional to the range at which the transmitter must communicate. For example, reducing the transmission range/power by 30% of maximum reduces power consumption by up to 50% [ELL00]. This is one obvious way we can attempt to reduce energy consumption: reduce transmission range. Current research in the area of adaptive transmission power is ongoing and is expected to prove itself very useful in the near future. However, this is not always a possibility because of the architecture of the wireless network.

Table 2.1: Typical states of a wireless LAN card.

States	Description
Off	No power is being used by the card.
Doze/Sleep	Used when not expecting to receive or transmit. (10 mA power drain for Lucent WaveLAN)
Receive	Used for both receiving data and listening for incoming data. Typical default state. (180 mA power drain for Lucent WaveLAN)
Transmit	Used only for data transmission. Most power intensive. (280 mA power drain for Lucent WaveLAN)

As an example, the Lucent/Orinoco WaveLAN PC card power management suite involves only two modes, which use a combination of the states from Table 1. They are

[2] This number of states is independent of the physical hardware. We may have only one transceiver unit that is capable of both transmitting and receiving either at the same time

Active mode and Power Savings (PS) mode. Active mode uses the receive state as a default and switches to the Transmit state whenever the mobile user needs to transmit for any reason. The PS mode uses the Doze state as a default and has an additional setting to maintain connectivity; a listen interval. While in the Doze state, the WaveLAN card can neither transmit nor receive, thus it is rendered disconnected without the functionality of the listen or "sleep" interval. The semantics of the Doze State require the context of the network connection to be saved. In this manner, at the listen interval, the card can efficiently transition to receive mode for a small amount of time in order to "listen" for a beacon from the base station, notifying it that information is awaiting download. Note that for the PS mode to be functional, the base station must support this mode by buffering all outgoing information to the mobile unit and by synchronizing with the mobile unit's listen interval to notify of buffered packets. This model is strictly controlled at the device driver level through interfaces at the operating system level. The user chooses one or the other, Active mode or PS mode, regardless of which (if any) application is currently running. This could potentially add up to a terrible waste of power by the wireless network card during times when it isn't being used at all.

Industry has also attempted to standardize the power-based state model while expanding the number and types of states available. Table 2.2 outlines the Advance Configuration and Power Management Interface adopted by Bluetooth [FLEM00]. This model also takes into consideration the System State to which the wireless device is operating. As you can see the ACPI adds two additional states to the 4 previous states. This added granularity allows the driver to take advantage of a larger pool of available

---

(full duplex) or separately (half-duplex), but separate states will function differently and thus consume different amounts of power.

states in order to optimize the tradeoff of network delay vs. power consumption. In Table 2.2 we see that the sleep state has been expanded to incorporate different levels of system context. Each of these different levels requires less power until we reach a power off state.

Table 2.2: Advance Configuration and Power Management Interface (ACPI)

States	Description
S0	Working State.
S1	Low wake-up latency sleep state. No system context is lost.
S2	Low wake-up latency sleep state. CPU and cache context are not maintained by hardware.
S3	Low wake-up latency sleep state All system context is lost except for system memory.
S4	Lowest power, longest wake-up latency. Assumes all devices powered off.
S5	Soft Off state.

ACPI does a nice job at expanding upon the previous states available for any mobile device. We would like to focus on the available states for the wireless PC card. One could argue that there are really seven distinct states from which to expand. Table 2.3 exposes those seven states as defined by [SCO98]. Five of the seven states could be exploited to find energy savings through either hardware or software techniques, or a combination of the two.

2.3 The IEEE 802.11b Protocol

In 1999, the IEEE 802.11b (high rate) amendment was ratified for the IEEE 802.11-protocol standard for wireless networks [IEEE97]. It describes the Medium Access Control (MAC) and Physical (PHY) layer support as well as specifications for power management at that level. The key contribution of the 802.11b addition to the

wireless LAN standard was to standardize the physical layer support of two new speeds, 5.5 Mbps and 11 Mbps, a huge improvement over the old 1–2 Mbps speeds. It uses Direct Sequence Spread Spectrum (DSSS). To accomplish this DSSS had to be established as the sole technology[3] used, and thus will not interoperate with some older systems using the IEEE 802.11 protocol. The DSSS technique divides the 2.4 GHz band into fourteen 22-MHz channels.

Table 2.3: Seven Basic States of a Wireless Interface

State	Description
1. Off	The device is completely off.
2. Sleep	In a low power mode, but ready to switch to full power upon request.
3. Idle	Fully powered, yet neither receiving or transmitting.
4. Power-Up	Switching from an off or sleep state to a fully powered state.
5. Power-Down	Switching from a fully powered state to an off or sleep state.
6. Receive	Fully powered and receiving a packet.
7. Transmit	Fully powered and transmitting a packet.

Figure 2.1 shows how this protocol fits into the International Standards Organization (ISO) communication stack model. The typical configuration includes a wireless network access point that acts as an intermediary to the mobile units and connects directly through a wired medium to an existing network (Infrastructure Mode – Figure 2.2). The standard also provides for ad-hoc network connections (peer-to-peer communication or Ad-Hoc Mode – Figure 2.3).

Power management is supported and accomplished while operating in the infrastructure mode (Figure 2.2) through data buffering, notification and subsequent

[3] IEEE 802.11 also used Frequency Hopping Spread Sprectrum (FHSS), which would not support higher data bandwith without violating current FAA regulations.

forwarding at the access point. As stated above, the wireless PC card may be in one of several states at any given time, some requiring much more power than others.

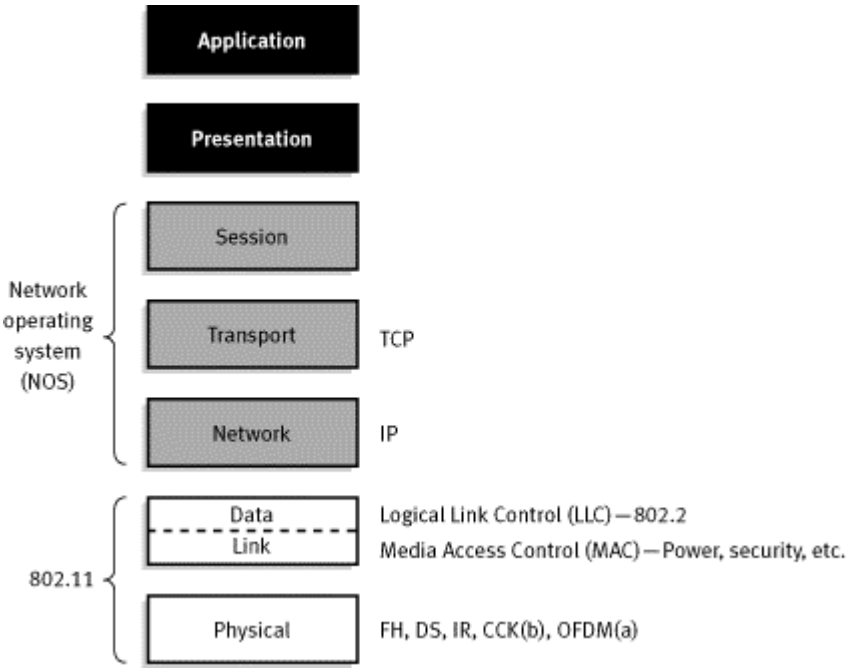

Figure 2.1: IEEE 802.11 and the ISO model

Power management as defined for IEEE 802.11b attempts to keep the PC card in the least power-hungry state for as long as possible while minimizing network delays. It is controlled by the user through the operating system level via two primary variables. First, the user must select either power management on or off. Then, if power management is turned on, an additional variable, listen interval[4], is added. With power management off, the PC card will remain in the default state of receive and will transition to transmit only when sending data. With power management on, the default state is

[4] The listen interval variable default value for the Lucent WaveLAN card used in our testing is 100ms and has a controlled input range of (0 - 65,535).

doze, and the PC card will transition to the receive state at the frequency of whatever is entered as the listen interval. This requires synchronization with the base station in that it must know that a mobile user has turned power management on and will then buffer all outgoing traffic to that particular mobile user. Once traffic is buffered, the base station will alert the mobile unit that it has traffic waiting to be received via synchronized broadcast beacon. Upon learning of queued traffic, the mobile unit may decide to switch to a receive state, alert the base station that it is ready to receive and receive the buffered traffic, or simply wait until the next time it has an outgoing message to download the queued traffic. Upon completion of the download, the mobile unit will return to the default Doze State.

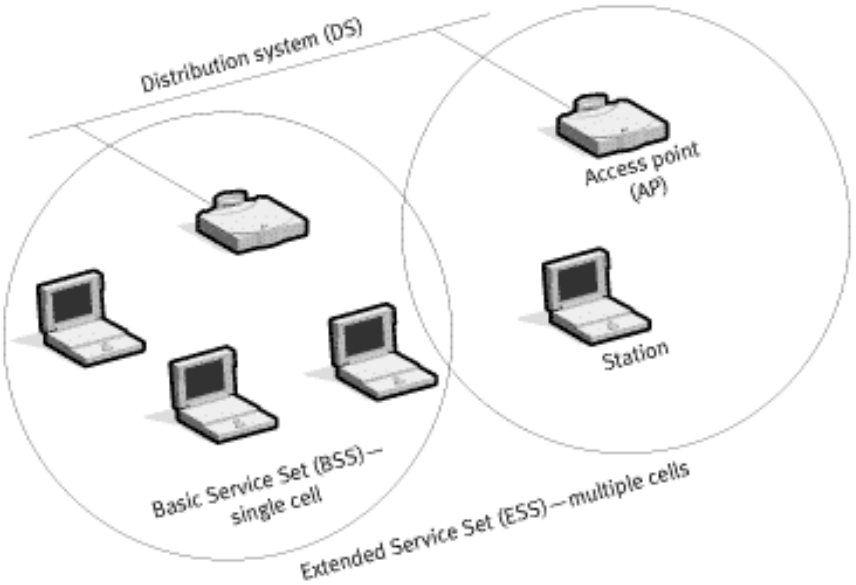

Figure 2.2: Infrastructure mode

During the PS mode the PC card continuously switches between doze and receive states until either a beacon message indicates there are queued messages or it needs to transmit

an outgoing message. This can dramatically decrease the amount of power consumed by the PC card while not actively transmitting or receiving data. In fact, our tests show that it can be up to eight times more efficient when operating in an environment where only an email application is being used and moderate message traffic exists[5].

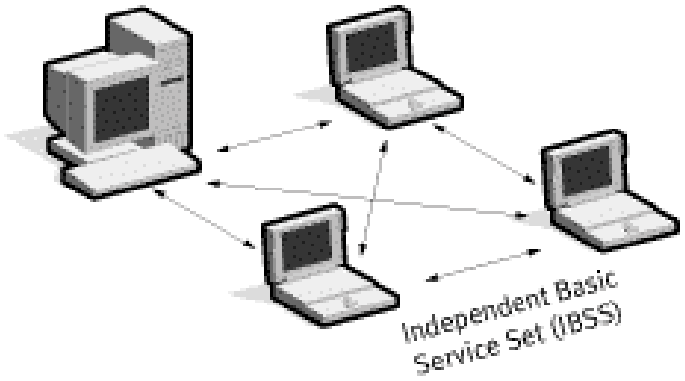

Figure 2.3: Ad Hoc mode

[5] Moderate traffic is explained fully in Chapter 6, Implementation.

CHAPTER 3
SURVEY OF RELATED RESEARCH

3.1 Power Management Approaches

There are two main approaches to conserving power in mobile devices, hardware and software. Hardware approaches attempt to improve energy efficiency by reducing the amount of power required to operate a particular device. These include improved battery technology and lower power electronics. Software approaches attempt to manage the device through outside algorithms in order to conserve more energy and include battery-saving protocols and network architectures. The two methods, however, are not mutually exclusive. Hardware advances frequently require good software techniques to take advantage of power savings. This chapter discusses a variety of software approaches for improving the efficiency of mobile devices in recent years.

3.2 Energy Aware Routing

As mentioned in Chapter 2, power management is not supported with the IEEE 802.11b protocol while in the ad-hoc mode. Yet this mode may potentially have a great need for energy savings. The conceptual differences between saving power with routing algorithms in ad-hoc vs. infrastructure architectures are small. Thus we can consider each for any particular algorithm. Research in this area includes i) reducing transmission power adaptively based on the distance between sender and receiver, ii) adaptively setting transmission power in route discovery protocols, iii) balancing hop count and latency with respect to power consumption when choosing the "best" route between two

hosts, and iv) choosing routes that fairly and evenly distribute the message traffic among the nodes in the ad-hoc network. Ellis et al. [ELL00] showed that adaptively reducing transmit power can reduce overall consumption by 32% to 56% for minimum hop count routes. This is a substantial amount of savings, which can be taken advantage of by implementing an algorithm that always chooses the minimum hop route from the sender to the receiver. Another area of research using this technique always searches for the nearest node with which to communicate. In this same manner, studies are being conducted to find out if even more energy could be saved by using directional antennas that are smart enough to know in which direction the next hop node is located and to only transmit in the correct direction using a low-power narrow beam. Directional antennas use less power to transmit a fixed distance than their omni-directional counterparts.

There are many other different wireless network architectures that use some sort of adaptive routing algorithm. These algorithms, whether dynamic or static, can take advantage of network level information to attempt to minimize the amount of power required in order to get a packet of data from point A to point B. This is a whole other area of research in and of itself.

3.3 Adaptive Network Polling

Adaptive network polling concentrates on when, and for how long mobile units communicate with an Access Point while in a power saving (standby) mode. Adaptive setting polling rates for wireless networks (trading latency for energy) was discussed by Ellis et al. The idea here is that a mobile user may be more concerned with available power than with the communication delay in the system (latency). Using adaptive polling rates, a mobile device could potentially save a great deal of power by minimizing

or eliminating unnecessary transmitting and receiving of packets. This would be equivalent to updating the IEEE 802.11b protocol or designing a whole new protocol around the issue of power management.

As we described earlier, the power saving (PS) mode of the Lucent WaveLAN PC Card has a variable listen interval time, but it is not adaptive. First of all, the PS mode must be set during Operating System (O/S) startup and cannot be changed without a minimum of a soft re-boot. Allowing either the O/S or an active application to change the value of the listen interval dynamically would prove to be a great advantage. The remainder of this thesis will make a statement for Active Mode Power Management (AM/PM) controlled through the application layer.

3.4 Power Saving Algorithms

There are many different software techniques energy conserving device resource managers could improve without making significant changes to the resource itself. For example there has been a great deal of research in the area of applying adaptive algorithms to intelligently spin-down a hard disk when it is expected to be idle for significant periods of time [HELM98, LORC98]. Past studies showed that the disk drive and the display panel on mobile computers consume the majority of power. Of course that is without considering the wireless network connection, which singularly consumes the most power in an actively connected mobile device. Nonetheless, humans rely predominantly on sight and thus are visually stimulated by robust graphical user interfaces (GUI). This, of course, increases demand for power and thus perpetuates the need for power savings algorithms in this area.

Another example applying to network architecture would be a network protocol that can reduce the average number of retransmissions needed to get a packet to its destination [FINN99]. This design is similar to that discussed in Section 3.2. Here again, minimizing the number of nodes required to handle a packet would also work to minimize the amount of energy used for the average packet transmission.

Many subsystems of computers spend a good portion of their time idle. Adaptive algorithms to predict idle periods are being heavily researched. Predicting when an idle period will appear, and accurately predicting the duration of an idle period would prove to be very beneficial to saving energy. If an upcoming idle period was predicted to be long enough, the overhead of turning a device off and back on would at some point be less than leaving a device on and idle or in a sleep state. The system could then transition to a soft off state and save even more energy. Many of the predictor algorithms are being research using artificial intelligence machine learning principles [SCO98].

3.5 Battery Technology

The reader may be inclined to ask "If power requirements are so stringent, why not just make better batteries?" This is a valid question. As it is beyond the scope of this thesis to fully address this question, it is important to mention a couple of quick notes about battery technology. First, batteries are basically enjoying advances on a linear scale. With this in mind, we shouldn't expect any huge breakthrough any time in the near future. On the other hand, technology is becoming miniaturized. This fact allows for the possibility of additional battery power included within separate devices of a mobile computer. Metrocom has used this approach with its Ricochet wireless modems, providing an on-board battery with its radio transmitter to supplement the mobile device

with its own secondary battery. Additionally, there is an environmental issue with disposal of batteries. Batteries that last longer need to be recharged less often and therefore have an extended useful lifetime. Although these battery advances are important, they alone will not satisfy the power requirements being demanded by new mobile devices. Other power saving strategies must be pursued.

3.6 Other Application Layer Power Management Research

From what we can find, there really isn't extensive research being conducted in the area covered by this thesis. If there is, it isn't well documented. The computer science department at Duke University, under Professor Carla Ellis has a MillyWatt power management project that aims to form a partnership between applications and the operating system with regard to setting power management policy [ELL99]. They advocate the following points:

- Software can have a positive impact on and should be involved in power management.
- Energy consumption will be a major concern in emerging applications of computing technology. Therefore, performance-measuring studies should regularly include appropriate energy metrics in addition to the more traditional ones (e.g. time, bandwidth).
- Tools and mechanisms for power management are currently inadequate and require more research and development.

There is no tangible evidence, beyond this project, to address the topic of application layer power management. Our research goes a step beyond theorizing about the potential benefits attainable AM/PM. We prove its worth through a series of benchmark tests using both email and www browser software packages.

CHAPTER 4
APPLICATION LEVEL POWER MANAGEMENT APPROACH

4.1 The Problem and Motivation

The currently functional, device driver-level power management installed in a Lucent WaveLan PC card works very well out of the package. In fact, the device driver, acting autonomously from any application, can save up to ten times the amount of power used with power management activated. The problem is that we can do much better. What can we improve upon? First, power management must be directly turned on and off by the user via the operating system. What if the user is unfamiliar with power management or too naïve to know the impact of increased battery life when it is used? Even when it is turned on, the receiver still wastes power while periodically turning on to listen for buffered data. What is the best listen interval? What if the application running on the mobile user's computer knows it will not be receiving any more packet communication and does not require network connectivity until the user requests such connectivity? The important issue to grasp here is that the connectivity requirements of the user are much more of an application-specific issue. Each application knows best when to take advantage of a doze mode, and, more importantly, when to receive and transmit. The application can then be tailored to needs of specific users; it can "learn" the habits and peculiarities of individual users and can tailor its power saving methods.

Each application would in essence attempt to **"steal"** back as much wasted energy[1] as possible, while still meeting the latency demands of the user. In this manner, great power savings can be accomplished. We can save an additional six times the amount of power used by the PS mode, and can use sixty times less power than with no power-savings options at all.

4.2 Application Level Interface

The idea here is to design power-aware APIs that can be used by the application programmer when developing applications for mobile computers that require network connectivity. The APIs are able to use application-specific knowledge of the user's trends and network needs. With this information, available only at the application level, the application itself can make direct calls to the network device driver through the underlying Operating System. For example, in the Linux O/S, control commands such as **ioctl** and **cardctl**, directly change the state of the device, real time. These calls can be made at run time and would thus have the most information available about current network needs. Moreover, these calls require a good deal of knowledge about the underlying wireless hardware and therefore would be tedious for an application programmer to directly apply to an application's design. The job of the API is to create an easy-to-use and fully functional power management abstraction tool to the application programmer. This, in turn, allows the application programmer to incorporate power awareness and management into applications written with the mobile user in mind. The

[1] By wasted energy we mean power used by the receiver, transmitter or doze mode, when not specifically required by the user or the application in use.

end result is a much more power-efficient mobile device at no extra cost to the user[2]. Transparency is the key concept here. The mobile user doesn't even really have to know any of the underlying specifics, but rather just continues using the same familiar applications, only now for much longer periods of desired ubiquitous computing.

This approach does have some downsides. Similar to IEEE 802.11b, ad-hoc networks would be difficult to coordinate, due to mobile units constantly changing states to adapt to energy conservation. There would have to be some sort of distributed information concerning the current state of devices taking advantage of power management states. One idea is to make use of pathways that are weighted based on a recent-activity time stamp. When a user is active and has been sending and receiving traffic across the network then that same user is very attractive as a hop station because it is much more likely to be in an active state. On the other hand, inactive users would be assigned increasing weights over time which would make them much less likely to be chosen to forward message traffic because they have a much higher probability of being in a low-power mode. This would make sending a packet through such a user a risky choice, hence the higher weight to that path.

Although some of the other subsystems of a mobile device also need to address disruption/delay of service issues, this same disruption to a user-connected network could prove disastrous. For example what if there were a period of massive inactivity in a mobile network, and many of the nodes decided to enter a semi-connected or even disconnected state in order to save power. This could potentially trigger widespread areas of disconnection.

---

[2] A minimal cost of increased latency may be paid, but in most cases this is far outweighed by the savings in energy consumption, thereby increasing mobility.

Many wireless networks are directly supported by and are merely an extension of a well-established wire line network. This is the type of network we use as the basis of this research. This type of network relies much less on when, and for how long , a user is connected, but rather closely resembles the well-known client-server model. In this manner, mobile users have much more flexibility in their ability to connect and disconnect in order to maximize power efficiency without directly affecting the status of the network.

4.3 The Email Application

We initially focused our research with a single application in mind. This application is prolific the world over and is quickly becoming the next "can't do without" software for the public at large. The email client serves many purposes. Whether business, personal, or pleasure, email has become a daily necessity for the working lives of a multitude of computer users. These users share an ever-increasing demand to access their email anywhere, anytime. In fact, consumer demands and expectations are driving wireless technology to be the fastest growing segment of the telecommunications industry [HELA99].

With that in mind, we adapted a simple email client, GetPop, to include power management APIs that we created to prove the necessity for Active Mode Power Management. GetPop is a text-based and command-line-interfaced C program that provides the basic feature of connecting and downloading email from an input mail server. At run time, GetPop connects to the user input mail server, prompts for a user name and password and then downloads any messages on the server. Figure 4.1 shows a sample connection and download of the GetPop program in action.

It immediately becomes apparent that neither the transmitter, receiver, nor doze state are doing the user much good when not physically in the act of downloading email from a server. This causes a dilemma for the user. What power savings mode is best? Should the mobile connection be terminated and the card shut down in order to save power? Is there time and expertise available to do either of these options?

```
[root@localhost tmp]# gp earthlink.net
ioctl(): Device or resource busy
Connecting to server earthlink.net
<207.217.120.206>.
User: rmloy
Password:
User has 8 messages.
Getting mail number 1
Getting mail number 2
Getting mail number 3
Getting mail number 4
Getting mail number 5
Getting mail number 6
Getting mail number 7
Getting mail number 8
Closing connection.
[root@localhost tmp]#
```

Figure 4.1: GetPop email client snapshot.

Until this point these were the only options available to users. Either change the settings on the PC card through an O/S layer interface or accept the blatant wasting of power associated with failing to do so. In some O/Ss (e.g. Windows 95, 98, 2000, CE), the user is unable to actively change the power management settings on the card without restarting the device driver software. The user is then forced to accept this burdensome, time consuming task, necessarily diverting attention from any other task at hand.

This is where AM/PM enters the scene. As discussed earlier in Section 4.2, AM/PM makes use of power-aware APIs to optimize the power resources being used by

specific applications. Our research focused on the wireless network PC card, but it would be easy to extend this concept to any portion of the mobile computer. Coordination through the Operating System level would become more difficult, but that is not part of our research. We aim only to show that application layer power management can be implemented and at an excellent power savings.

4.4 The Web Browser Application

Upon obtaining such good results from the GetPop email client, we decided to extend AM/PM into a web browser to prove similar power savings could be achieved in this environment. We worked with several different text-based browsers, including **debris**, **dillo**, **links**, and **w3m**. Each of these programs is an open source, GPL program, freely available from the linux.org web site. The **links** browser seemed to be the best of the bunch for our purposes, so it is the one we chose to use for incorporation of our PowerSteeler APIs and benchmark testing.

The same basic principles that achieved power savings for the email application apply to the browser program. Regardless of the user, there will be some time consumed digesting the contents of a web page during which connectivity via the PC card is not required. We aim to expose this wasted period of time by taking advantage of our ability to suspend the card in order to save power. Two APIs were created specifically for this purpose. From Table 6.1, the browser_pm_start() and pm_close() functions will achieve our desired results. The first program will ensure that power is resumed to a suspended card and that it is in an appropriate power savings mode. The second will suspend power to the card immediately following successful download of a web page, sometimes even before the full contents of the page are displayed to the user. The increased latency is

minimal; well worth the wait for the trade off for power savings. We lacked a adequate tool to precisely measure the added latency, but rather took note of the user's perceived delay. We found our savings to be very favorable. From Table 6.2, the default power management used 8.33 times less power, while the AM/PM consumed 23 times less power than using no power management at all. The AM/PM saved us almost three times (279% less) the amount of power consumed by that of the default power savings mode. This savings is extraordinary and can not be overlooked as a plausible means of extending the life expectancy for a mobile device in between re-charges. Now let's get into the specifics of how we specifically implemented AM/PM.

4.5 The Device Driver and Interface Specifics

Our initial research started with two different device drivers, one for the Windows O/S and one for Linux. Both were proprietary, developed by TriplePoint Inc., specifically for the Lucent/Orinoco WaveLAN PC card, **wavelan2_cs**. We were unable to obtain the source code for the Windows based driver, and with no support there was little hope to attempt to control this driver through the Windows O/S at the application layer[3]. Additionally, the interface software for the Windows O/S does not allow the user to change the PC card settings without shutting the card down and starting it back up again. There is no "on the fly" functionality that is required for AM/PM to work

4.5.1 Linux Drivers

We then focused on the Linux driver, which had source code available, but unfortunately no support. We initially planned on developing an interface to control the

[3] We did, however, run some benchmarks using this driver, and they are available in the appendix if you wish to view them.

power management functionality of the WaveLAN card via direct manipulation of the active variables. While in the development stage for our interface, we came across two very interesting new developments in the realm of wireless devices. First, a third driver for the WaveLAN card, **wvlan_cs**. This third driver is an Open Source GPL (Gnu Public License) licensed driver developed by multiple Linux programmers and included in the Linux PCMCIA standard package. This GPL driver differs from the first two, in that it does have both support and documentation easily available. The second, and most important, discovery is called Wireless Extensions.

4.5.2 Wireless Extensions and Tools

Wireless Extensions are part of an Open Source project supported by Hewlett Packard by Jean Tourrilhes. The Wireless Extension is a generic API allowing a driver to be exposed to the user space for configuration and statistics specific to common Wireless LANs. The beauty of these extensions is that a single set of tools can support all the variations of Wireless LANs, regardless of their type (as long as the driver supports Wireless Extension)[4] [TOUR]. Another advantage is these parameters may be changed on the fly without restarting the driver (or Linux). This is exactly the functionality we set out to develop. But Wireless Extensions were not created with the distinct purpose of power management in mind. Thus we used the Wireless Extension functionality to create our own AM/PM APIs with the included Wireless Tools. The Wireless Tools are a set of tools (C functions) allowing the user to manipulate the Wireless Extensions. For now they use a textual interface only.

[4] Both the proprietary and the GPL Linux drivers support Wireless Extensions, but the latter has greater support and functionality.

The following are the three Wireless Tool functions available to create our APIs:

iwconfig – manipulate the basic wireless parameters.

iwspy – lists addresses, frequencies, bit-rates...(for statistical purposed).

iwpriv – manipulates the Wireless Extensions specific to a driver (private).

The other command line function used in our APIs to minimize the power consumption is the **cardctl** command. It supports two specific options that we are concerned with, suspend and resume. These options do exactly what one would infer; suspend the card (thereby consuming zero power), and resume from the suspended state (back into whatever the state of the card was before being suspended). With the use of the above tools, we are now ready to describe the AM/PM APIs.

4.6 The AM/PM APIs

Finally, we are into the meat and potatoes of this thesis. We will now describe in full detail the functionality of our AM/PM APIs and how they are implemented in both the **GetPop** and **links** applications. From the previous section, we know the Wireless Tools are directly available to the application programmer. But does the power aware, mobile application programmer really want to get down and dirty into the man pages for these tools? We suggest a better approach.

4.6.1 The PowerSteeler Header file

We have added another layer of abstraction to the Wireless Tools driver interface. The idea is simple. We create an extra header file, **PowerSteeler.h**, to be included at the top of any program wishing to take advantage of the PowerSteeler functions. This header file is easily adapted or updated to include new technology or protocol specific changes. Functionality for all different types of hardware and software can be added to general,

abstract functions to be used by power aware programmers. The header file and all of our code is written in C, but it would be easy to create the identical functionality for any language wishing to incorporate PowerSteeler functions. For an object-oriented approach, we would create a PowerSteeler class, which housed all of the power aware APIs for use in creating new applications. Just a couple of functions tailored for specific use in either an email client or a browser were sufficient to make a point in the name of AM/PM. We explain these functions now and use them for our benchmark testing covered in chapter six.

4.6.2 The API functions

For our research and testing purposes, we remain focused on the idea of stealing back power from an idle PC card being used by a either standard email client or a web browser application. We note that the average email client certainly does not require real time connectivity. Most people, who already enjoy a dedicated connection to their mail server, and have little worry of power management (i.e. desktop computing), still have their user preferences set up to check their email server only once every ten to thirty minutes. Now consider the mobile user who would like to enjoy that same connectivity, but who doesn't have the luxury of a dedicated line or an unlimited power supply. This user would like to have a fresh download of email, say every 15 minutes, but can't afford to stay connected (even in PS mode) because of the drain imposed by the transmitter and receiver. If only the user could automate the process of having the PC card hibernate[5], awaking only for very brief periods to quickly query the mail server for new mail, thus allowing for even more power savings. Does the user really need connectivity while

[5] A new term proposed in this thesis to mean an extended sleep period, allowing for large power savings, but still maintaining the context of a network connection.

reading email? The answer is no. What this user needs is software empowered by the PowerSteeler API utilities. This software will enable the mobile user to remain seamlessly connected, while minimizing power consumption.

The same principles apply to the web browser application. For example, consider a mobile user who wants to catch up on today's headlines while commuting to work on the subway. Does this user need to be connected while reading each individual story? What happens when the user gets distracted to discuss the news with the person sitting next to him for several minutes? What if while browsing, he could have the PC card hibernate while he absorbs the contents of a newly downloaded url? Does the user really need connectivity while reading the news from cnn.com on his mobile device? The answer again is no. PowerSteeler saves the day, by allowing the browser to suspend power to the PC card after a web page has been downloaded.[6]

With that in mind, let's look at the functionality we've included in our PowerSteeler.c source code and PowerSteeler.h header file. There are a total of nine functions available. These are defined in table 4.1 below.

These are very simple functions, making use of the system() function in the C language. It is important to see here, that multiple actions can occur through the use of one API call, regardless of the what type of connection interface is being maintained (hence the interface variable). It should also be noted that these APIs are functional for a IEEE 802.11 wireless PC card that supports Wireless Extensions. They could easily be expanded to incorporate many different types of network devices, but we did not have the resources available to test other options.

---

[6] In some cases, the PC card is suspended even before the entire contents of the display is filled.

Table 4.1: PowerSteeler AM/PM APIs

API Name	Definition
int browser_pm_start(int interface)	Resumes PC card into PS Mode. The listen interval is set at 4 seconds.
int browser_pm_nap(int interface, int nap_period)	Suspends PC card for a specified period of time. Zero power while napping.
int email_pm_start(int interface)	Resumes PC card into PS Mode. The listen interval is set at 60 seconds.
int pm_close(int interface)	Suspends PC card. Zero power will be consumed by the card unit resumed.
int pm_off(int interface)	Turns off device power management
int pm_on1(int interface)	Turns on device power management, and sets the sleep period to the default (100ms)
int pm_on2(int interface, char *sleep_period)	Turns on device power management, and sets the listen interval to the specified argument (0−65 seconds)
Int resume(int interface)	Resumes power to the PC card.
Int suspend(int interface)	Suspends Power to the PC card.

This concept of application layer power management is only in its infancy stage and would require an industry paradigm shift to fully embrace this way of saving power. The main purpose of this thesis was not to create a perfect "real world" functionality that could immediately be used by industry, but rather to stimulate thought and creativity via a new perspective. The desired goal would be to have more interest in this area grow into a standard PowerSteeler package that supported all mobile devices and environments. As technology changes, PowerSteeler will adapt, deprecating some APIs while updating others. An application programmer would only need to confirm the latest version of the PowerSteeler package and make use applicable functions for the application at hand.

The key features of the PowerSteeler AM/PM APIs are transparency, utility and functionality. We aim to provide a higher level of abstraction that is easy for the application programmer to use without becoming too bogged down with the details of power management. These APIs should also provide a higher return on investment and thus save the mobile user substantial amounts of energy for a minimized tradeoff in user

latency. Finally, the user should not be involved in this entire scheme to steal back wasted idle power. Rather the user just continues to operate with familiar applications while receiving the added benefit of extended wireless connected computing.

Now that we have identified what an AM/PM looks like and how it works, we'll cover the specific hardware used to run our benchmark tests in chapter five and the tests themselves in Chapter 6.

CHAPTER 5
IMPLEMENTATION

5.1 Introduction

For our testing we used an IEEE 802.11b protocol compliant wireless network set up in the Harris Lab on the fourth floor of the CISE building located on the University of Florida campus. The hardware consisted of the following:

1. A wireless access point (Lucent WavePOINT II Access Point)
2. A wireless PC card (Lucent WaveLAN Silver – 11Mbs)
3. A PCMCIA card extender (Accurite PC ExtenderCard)
4. A multimeter (Hewlett Packard 34401A with HP/IB interface)
5. A laptop (Dell Latitude, PII 233 MHz, 64 MB RAM, Linux/Windows98)
6. A desktop (Dell, PIII 600 MHz, 128 MB RAM Windows2K, HP BenchLink)

Figure 5.4 shows the configuration of all of these items connected during a simulation in the Harris Lab.

5.2 Wireless PC card

We use a Lucent subsidiary wireless PC card, the Orinoco WaveLAN Silver. The card is fully IEEE 802.11b interoperable and compliant with WECA (Wireless Ethernet Compatibility Alliance) Wi-Fi 'wireless fidelity' standard. The card is capable of up to 11Mbs bandwidth and thus works very nicely as a wireless extension to an Ethernet wired framework. The nominal values for current drawn while in doze, receive, and transmit states are 10 mA, 180 mA, and 280 mA respectively. Figure 5.1 shows the unconnected PC card.

Figure 5.1: Orinoco IEEE 802.11b silver wireless PC card.

5.3 The PC Card Extender

5.3.1 The PC ExtenderCard Features

The ExtenderCard features four layer PCB with extensive grounding. All of the mnemonic test points are clearly labeled and are provided for each of the 68 bus pins. There are 16 DIP switches that can break the connection of key signals to aid problem isolation. Built-in push button switches simulate removal and reinsertion of a PC Card (i.e., hotswapping), while prolonging the life of the computer's PC Card slot.

5.3.2 The PC ExtenderCard Usage

Although the PC ExtenderCard has the functionality of allowing access to all 68 bus pins from the PCMCIA slot of the laptop, only two pins were needed for the testing incorporated in this thesis. The two pins used were the VCC main power pins that are connected via a standard jumper connector when not being used. The jumper connector was removed and the two j-hook style connectors (positive and negative) from the multimeter where attached to read DC current (Amps) across the PC card's VCC main power supply line.

Figure 5.2: The Accurite PC Extendercard

The card worked very well, but it is rather delicate. On one occasion, the two j-hook connectors must have gotten too close and the corresponding protection fuse blew. The fuse was replaced within a day and the testing continued without further difficulty from the PC ExtenderCard.

5.4 Hewlett Packard 34401A Multimeter

The HP 34401A was used to measure DC current (Amps) directly from the exposed main power supply bus pins on the PC ExtenderCard. A front view is pictured in Figure 5.3 and the specifications follow.

5.4.1 The HP/Agilent 34401A Multimeter Specifications:

- Measure up to 1000 volts with 6½ digits resolution .
- 0.0015% basic dcV accuracy (24 hour)
- 0.06% basic acV accuracy (1 year)
- 3 Hz to 300 kHz ac bandwidth
- 1000 readings/sec direct to GPIB
- Shock and vibration: meets MIL–T–28800D, Type III, Class 5
- Power: 100/120/220/240 V, 45–65 Hz, 360–440 Hz
- Net weight: 3 kg (6.5 lbs)
- Size: 88.5 mm H x 212.6 mm W x 348.3mm D (4 x 8.5 x 14 in)

Figure 5.3: HP/Agilent 34401A multimeter.

This multimeter was relatively easy to use. The interface commands and Benchlink software took some time to become familiar with operating. Some pitfalls were encountered when dealing with maximum sampling frequency. When we first started testing with the multimeter we were using the RS–232 (serial) interface via a laptop. But we realized that this would not provide a large enough sampling frequency[1], so we installed the PCI card in a local desktop running Windows 2000 in hopes of gaining a much higher sampling rate. The multimeter is rated to provide 1000 readings/sec direct to the GPIB interface.

Unfortunately, with the configuration of our test, it will not provide this for a long period of time. Since the software needs to communicate with the multimeter in order to access data and continue to trigger readings, a constant window of samples can not be taken at this rate. Rather, a very high sampling frequency[2] may be obtained, but it is always followed by a short period (gap) of missed readings while the software communicates with the multimeter. The reading must be stored in the memory of the multimeter before they are transmitted across the GPIB interface. The memory system for the 34401A is only 512 readings total. This allows for only about one half second of

[1] The best sampling frequency for the RS-232 is 55 reading/sec.

readings at the sampling frequency we desire. We attempted to have the reading sent directly over the GPIB interface, but the 34401A only supports this at 80 readings/sec, and therefore isn't much better than the RS−232. With this in mind, we opted for the fastest readings we could get for a short period of time, and accepted the small gaps in sampling periods (which average out over time anyway). We made sure to tailor our testing with this fact in mind. See the appendix for a brief overview of a better device for the measurement needed in this line of testing.

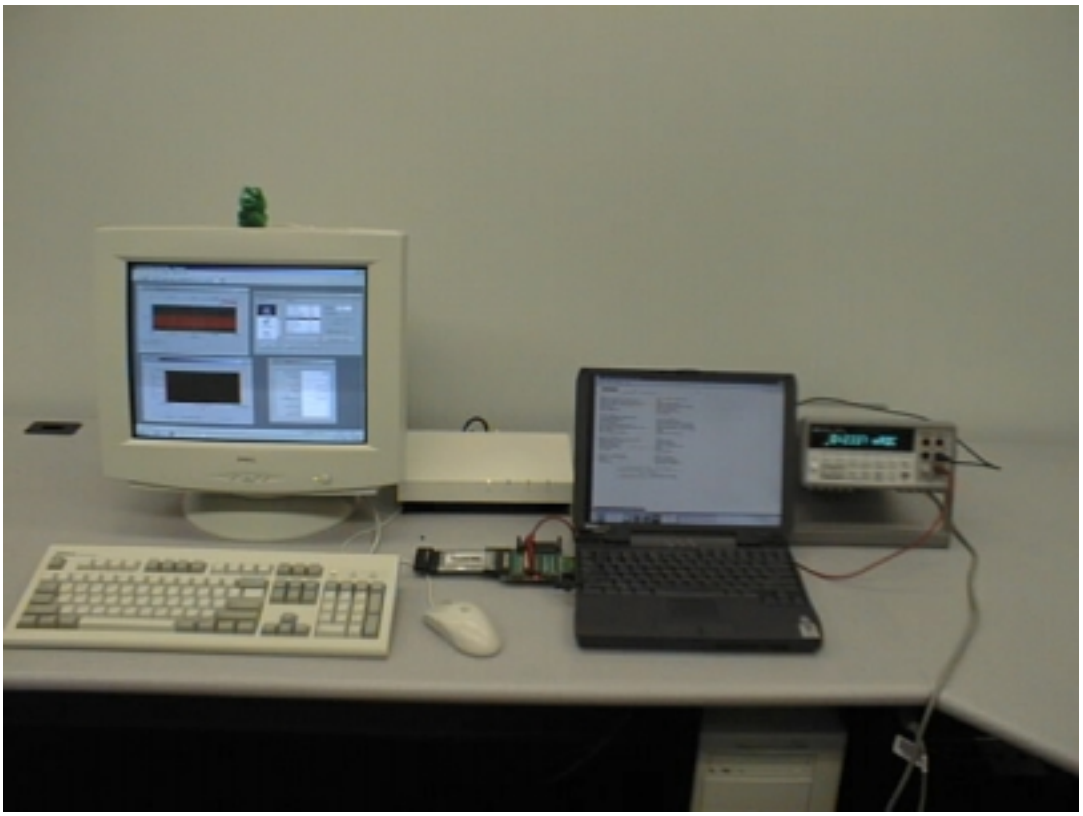

Figure 5.4: The hardware package. (**links** browser running on the laptop).

[2] The very high sampling frequency is 1000 readings/sec.

CHAPTER 6
EXPERIMENTAL VALIDATION

6.1 Introduction

The goal of these experiments was to obtain three specific benchmark test figures for each of the **GetPop** email client and the **links** www browser. The goal is to compare the average power consumed from each of the three tests. With this information we can better analyze the benefits of AM/PM. The three tests use the PC card with the native PS mode disabled, enabled, and finally enabled with our AM/PM incorporated with PowerSteeler APIs. We then compare the three data sets to get an accurate idea of just how much energy savings can be expected using the PS mode alone and then with AM/PM incorporated. The findings are quite interesting.

6.2 Power Saving Mode Disabled – Email Client

For this benchmark test we first connected all the hardware as described in chapter five. A connection to the local access point was made with the wireless PC card. The experiment commences with the multimeter set to trigger for a sample every 100ms and take 512 (the maximum allowed) on every trigger. The test was run for four minutes, yielding 90,000 samples for an average of 375 samples/sec.[1] The following sequence was used for each of the three tests:

1. Initiate GetPop from the command line prompt.
2. Enter valid user name when prompted.

[1] As discussed in chapter five the multimeter actually reads 512 readings at 1000 Hz and then misses approximately 800 readings while transferring data from memory to disk.

3. Enter valid password when prompted.
4. Six resident email text messages are downloaded to the local machine, simulating an average email user.
5. Repeat above four times in the four-minute period.

Figure 6.1 shows a screen shot of the Benchlink software running on the desktop after the completion of the test. As you can see from the statistics in the bottom right hand corner of Figure 6.1, the maximum and minimum DC current readings during the four-minute test were 268.5 mA and 160.2 mA respectively. More important is the average current value of 161.2 mA. We will focus our attention on this average mA reading to compare differences in savings. Notice the Strip chart and the histogram show almost all readings in the 160 mA range, with only a few periodic spikes in the 265 mA range where the PC card is transmitting. This is because the receive state is the default state while not operating in PS Mode. It is noteworthy to mention that this value (160 mA) is actually much better than the nominal rating of 180 mA in the PC card specifications. Even so, this is too large an amount of energy being wasted by a mobile unit who is only checking email, even if it is rather frequently. Let's see how the native PS mode stacks up in the same test environment.

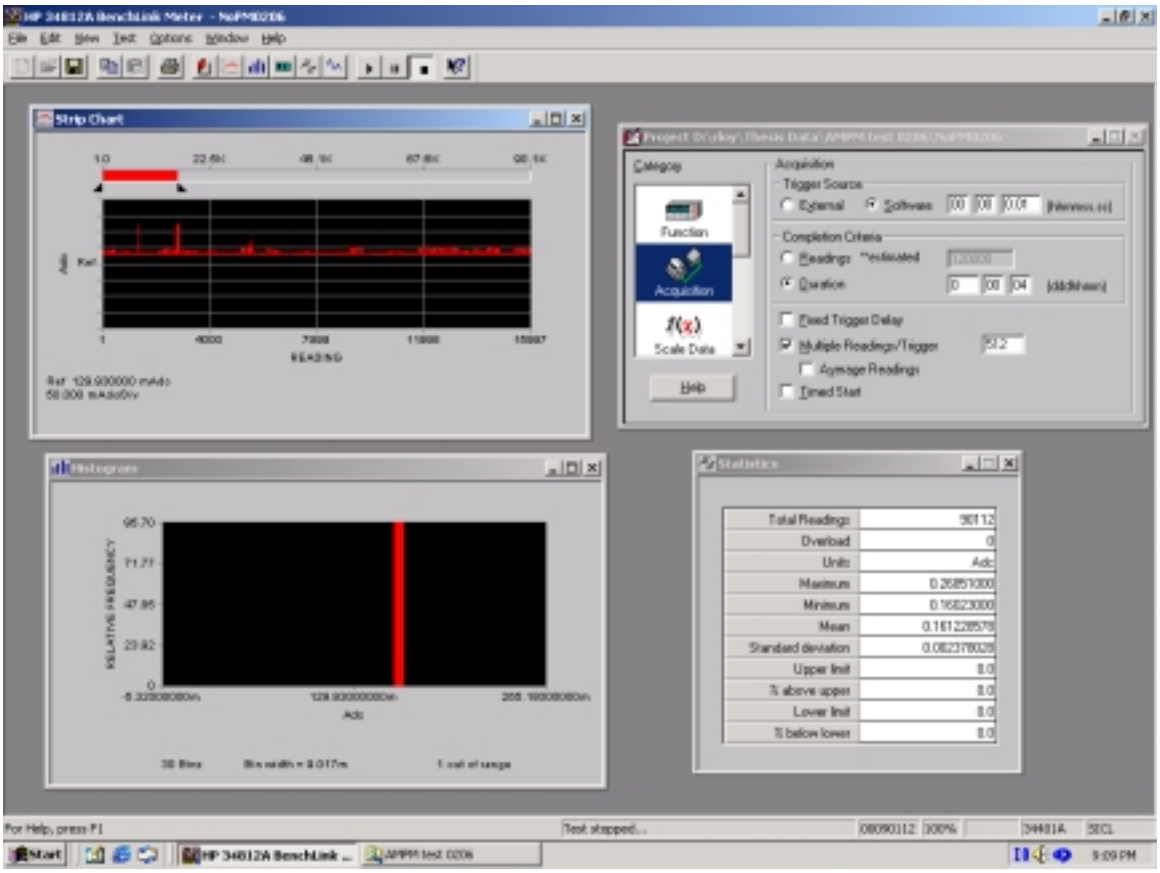

Figure 6.1: **GetPop** email client PS mode disabled benchmark test results.

6.3 Power Saving Mode Enabled – Email Client

The identical benchmark test environment as above was recreated, only this time PS mode was enabled and the listen interval was set at the default level of 100 ms. Now the PC card will default to the doze state and check for a beacon indicating buffered traffic at the access point ten times per second (every 100 ms). Figure 6.2 shows a screen shot of the Benchlink software upon completion of this test.

For this test, the maximum and minimum DC current readings were 266.9 mA and 11.7 mA respectively. The average for this test was only 16.8 mA. This amounts to a whopping ten times improvement in energy efficiency! The user perceived delay in receiving the email messages was negligible. Now let's see AM/PM is worth its weight

in power savings. Take a look at the histogram in the lower left-hand portion of Figure 6.2 and notice that nearly all of the samples collected are in the 10 mA range, while only a few (barely noticeable red trace) are in the 160 mA area.

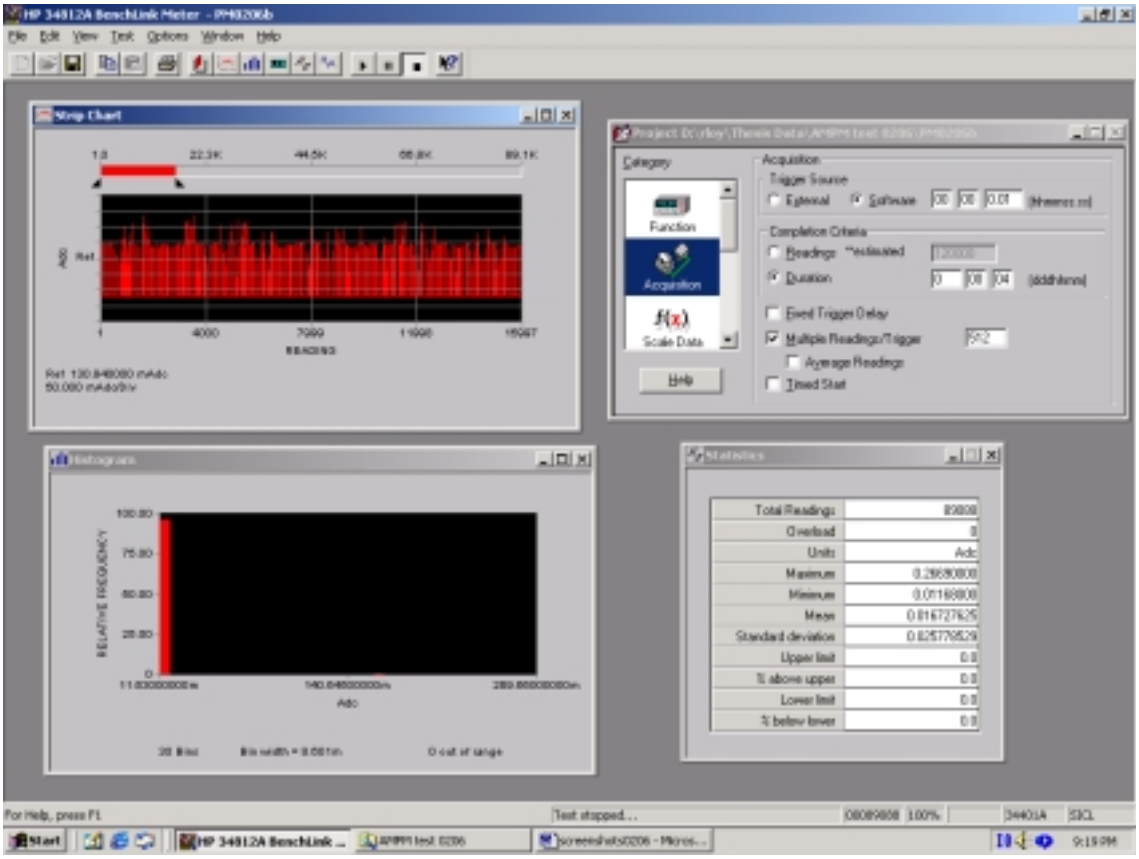

Figure 6.2: **GetPop** email client PS mode disabled benchmark test results.

6.4 Active Mode Power Management Delivers – Email Client

Although huge power savings were found with the PS mode enabled in this scenario, there is still current being wasted by the PC card. Figure 6.3 shows the Benchlink software after this test. .The AM/PM yields a similar maximum but a much less minimum DC current reading; 266.5 mA and .6mA respectively. The reason for

such a small minimum reading is that PC card is actually being suspended in between **GetPop** invocations. The average for this test was only 2.6 mA.

This is an additional 6.5 times improvement in energy efficiency over the PS mode alone and a gigantic 62 times improvement over using the card with no PS saving mode enabled! Again, the user perceived delay in receiving the email messages was negligible. It is interesting to note the upper left-hand portion of the Figure 6.3 and notice the current being drawn up until the point of the first invocation of **GetPop**. At that point **GetPop** downloads the new email messages and immediately suspends the PC card, causing the readings to drop to near zero.

Table 6.1 summarizes the values obtained in all three benchmark tests. Figures 6.8–6.10 are included to get a closer look at the strip charts of each of the three benchmarks. Only 16,000 data points may be displayed at one time in the strip chart and the points displayed in each figure are the first 16,000 data points of the test run. Each figure has a reference value of approximately 150 mA nominally and highlights the points made above. Figure 6.6 shows the PC card starting the test in the PS mode and then immediately getting suspended after the first download of email.

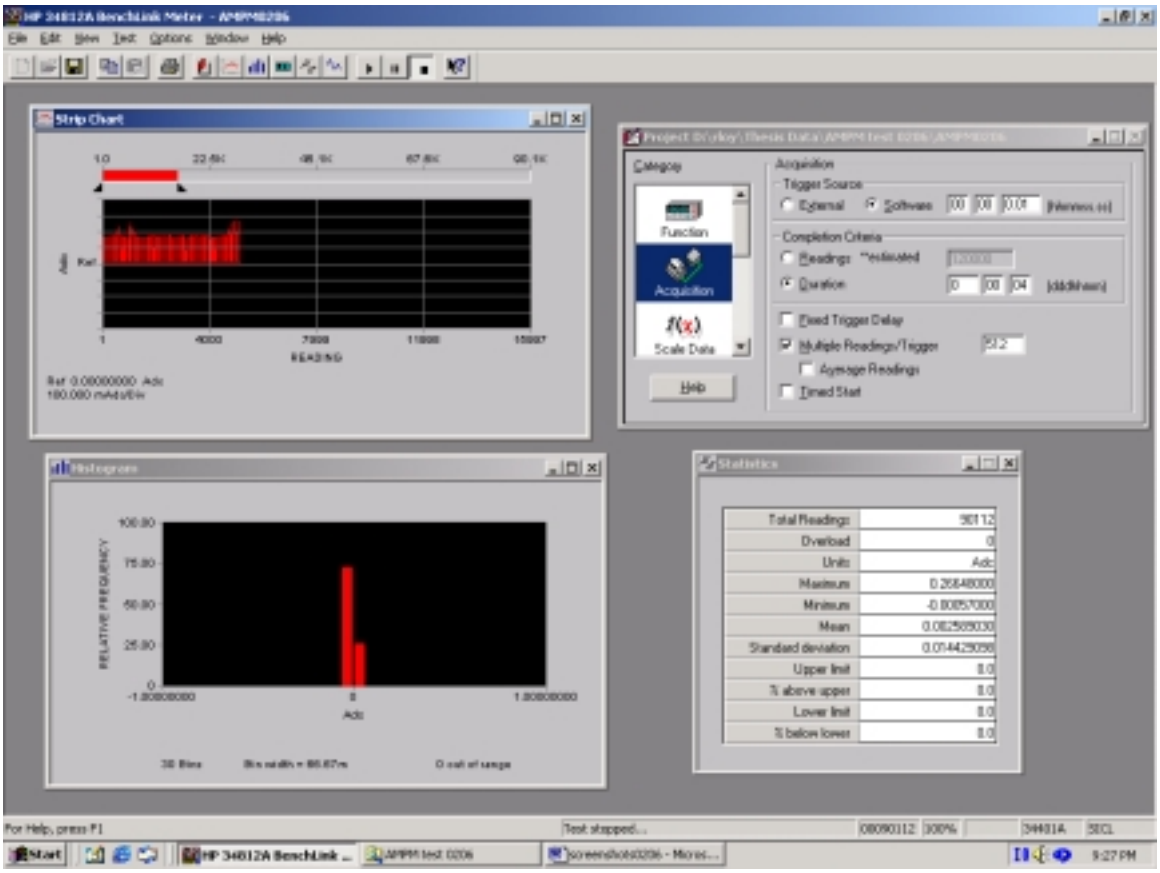

Figure 6.3: **GetPop** email client AM/PM benchmark test results.

Table 6.1: Summary of **GetPop** email test results (all values in mA DC current)

	No PM	**PM (100 ms)**	**AM/PM**
MAX reading	268.5	266.9	266.5
MIN reading	160.2	11.7	.6
MEAN	161.2	16.8	2.6
Power savings factor		10x	62x

6.5 Power Saving Mode Disabled – www Browser

We now turn our attention to the results obtained using the **links** www browser. We aimed to simulate a user who is connecting for the purpose of checking the headline news. Each benchmark test uses a four-minute period, just as the email client tests above.

For this reason, identical sampling frequencies were obtained and the figures should look very familiar to the reader. The following sequence was used for each of the three tests:

1. Initiate links from the command line prompt.
2. Go to www.yahoo.com. Realize that cnn.com is preferred for news.
3. Go to www.cnn.com. Find World news link.
4. Take internal link to World news.
5. Take internal link to read contents of the full story of the leading article.
6. Take internal link to U.S. News.
7. Take internal link to read contents of the full story of the leading article.
8. Take internal link to Science and Technology.
9. Take internal link to read contents of the full story of the leading article.

Just as with the above benchmark tests using the **GetPop** email client, Figure 6.4 shows a screen shot of the Benchlink software after the completion of the test using the **links** www browser. The maximum and minimum DC current readings during the four-minute test were 265.6 mA and 159.9 mA respectively. The average value is 160.9 mA. As we would suspect, this is almost identical to the readings obtained from the GetPop email client. This is because each test is run for 4 minutes, and with the PC card not in PS mode, the receive state is the default. The figures obtained from with PS mode enabled are nearly identical as well. This has to do with the fact that even though we are working with separate software applications, the network requirements are actually quite similar. In general, AM/PM pays the most dividends for the loosely connected user. As connectivity requirements increase, AM/PM savings decrease. We shall see this trend in Section 6.7.

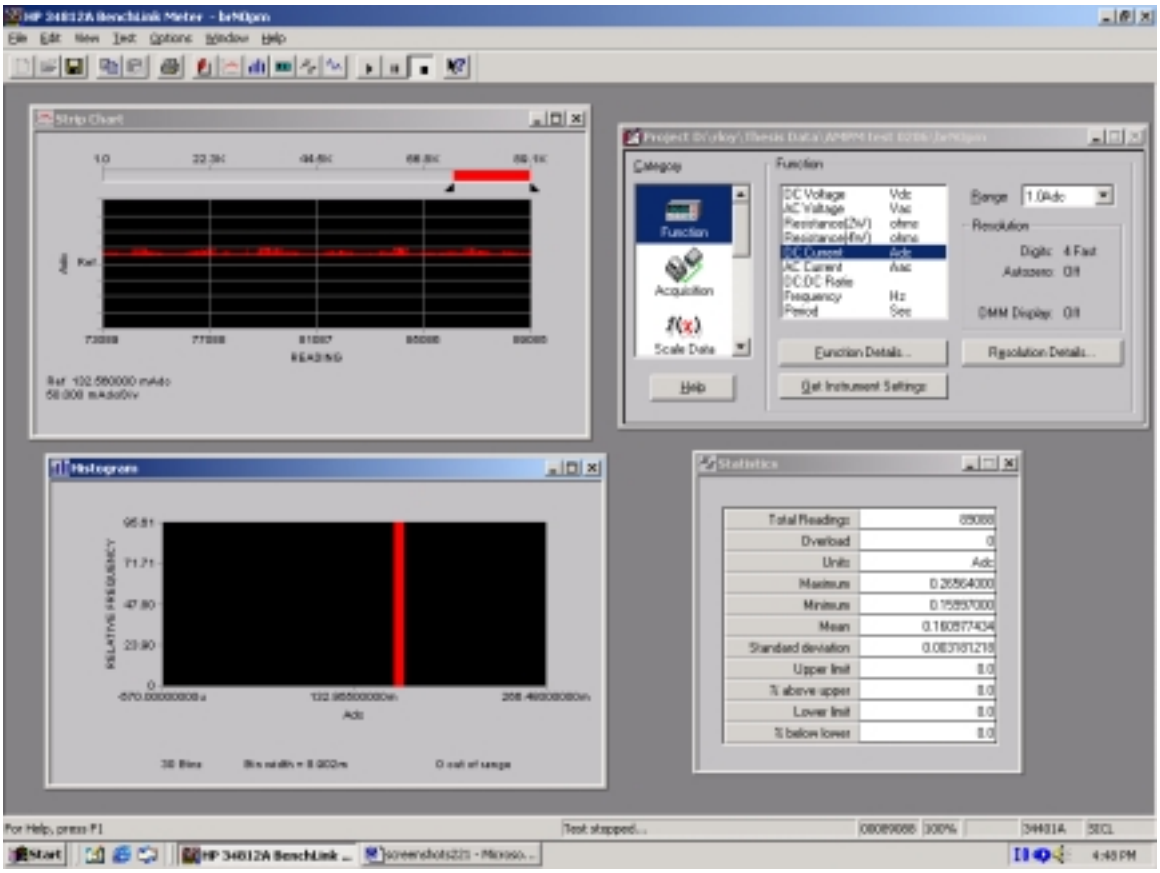

Figure 6.4: **links** browser PS mode disabled benchmark test results.

6.6 Power Saving Mode Enabled – www Browser

The identical benchmark test environment as above was recreated, only this time PS mode was enabled and the listen interval was set at the default level of 100 ms, just as before with the email client tests. Figure 6.5 shows a screen shot of the Benchlink software upon completion of this test.

For this test, the maximum and minimum DC current readings were 299.3 mA and 11.6 mA respectively. The average for this test was only 19.3 mA. This amounts to a significant 8.33 times improvement in energy efficiency! This is only slightly less than the savings with the email client. The user perceived delay in receiving the email

messages was negligible. Now let's see if AM/PM delivers the same power savings with an active www browser.

6.7 Active Mode Power Management Delivers – www Browser

Just as with the email client, there is still current being wasted by the PC card, even in PS mode. Figure 6.6 shows the Benchlink software after this test. AM/PM yields a similar maximum but a much less minimum DC current reading; 281 mA and .52 mA respectively. The reason for such a small minimum reading is that PC card is actually being suspended whenever an active connection is not sending or receiving information. We have effectively made the soft off (using no power), the default state! The average for this test was only 6.9 mA. This is an additional 2.8 times improvement in energy efficiency over the PS mode alone and a gigantic 23.3 times improvement over using the card with no PS saving mode enabled! Again, the user perceived delay in receiving a fresh downloaded web page was negligible.

Table 6.2 summarizes the values obtained from all three benchmark tests. As you can clearly see, AM/PM provides distinct and significant power savings. It does come at a cost. There will be periods when the user is not connected. For most users acting as a client, this is no problem. Consider a mobile server. This server will not be able to take advantage of AM/PM unless it can afford to "serve" clients on a limited basis. Figure 6.7 shows the connection between AM/PM and the native PS mode when the number of network requests per unit of time vary. The results are to be expected. The more active the network, the least amount of savings gained from AM/PM, and vice-versa. The two are inversely proportional. Part of the reason is because the more connected a user is, the more the transmitter will be used and thus the more power will be consumed.

Nevertheless AM/PM is undoubtedly an extremely viable option for increased power saving in the future of mobile devices!

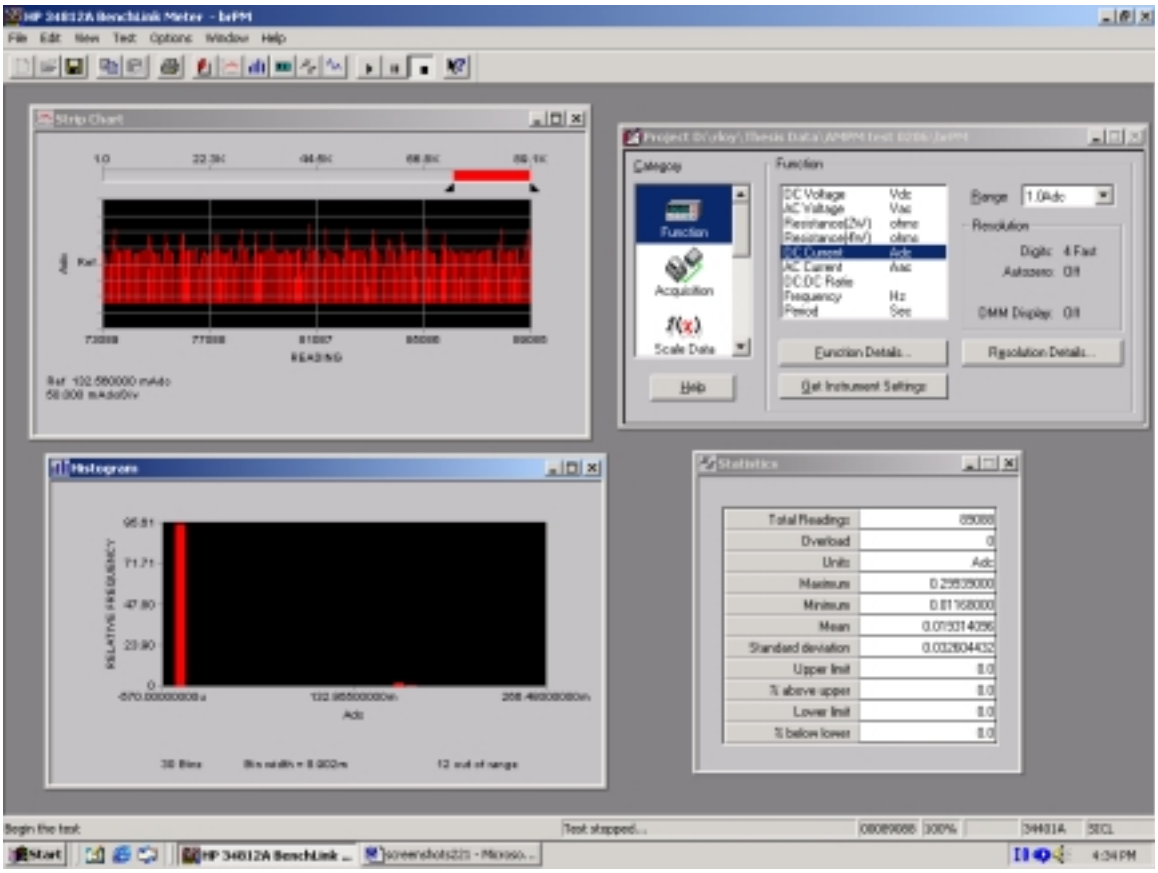

Figure 6.5: **links** browser PS mode disabled benchmark test results.

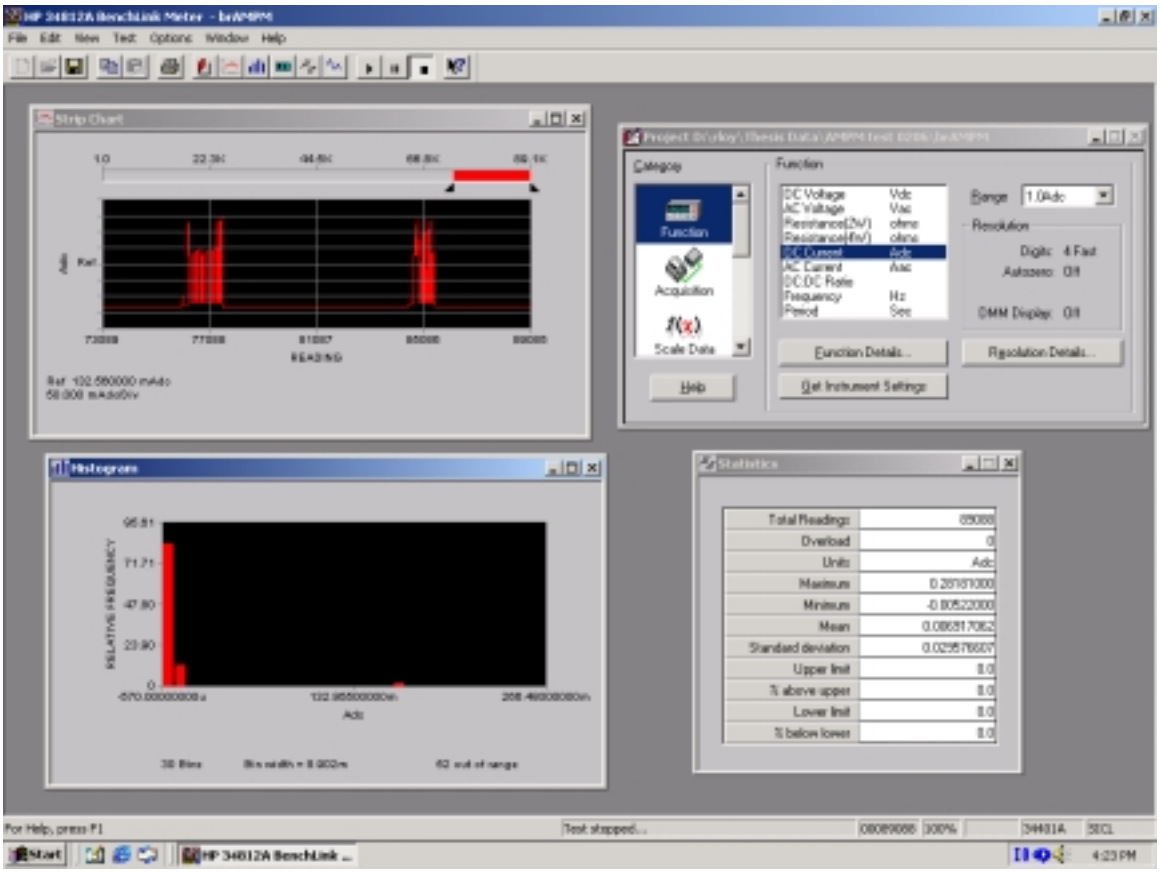

Figure 6.6: **links** browser AM/PM benchmark test results.

Table 6.2: Summary of **links** browser test results (all values in mA DC current)

	No PM	**PM (100ms)**	**AM/PM**
MAX reading	265.6	299.3	281.0
MIN reading	159.9	11.6	.52
MEAN	160.9	19.3	6.9
Power savings factor		8.3x	23.3x

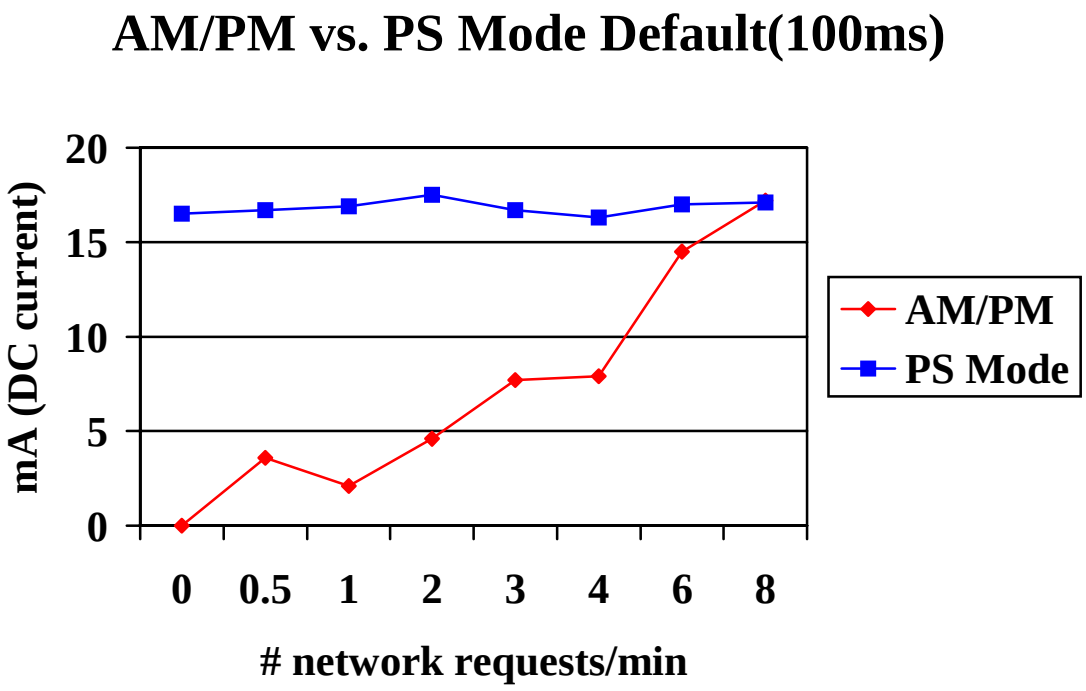

Figure 6.7: Native PS mode vs AM/PM.

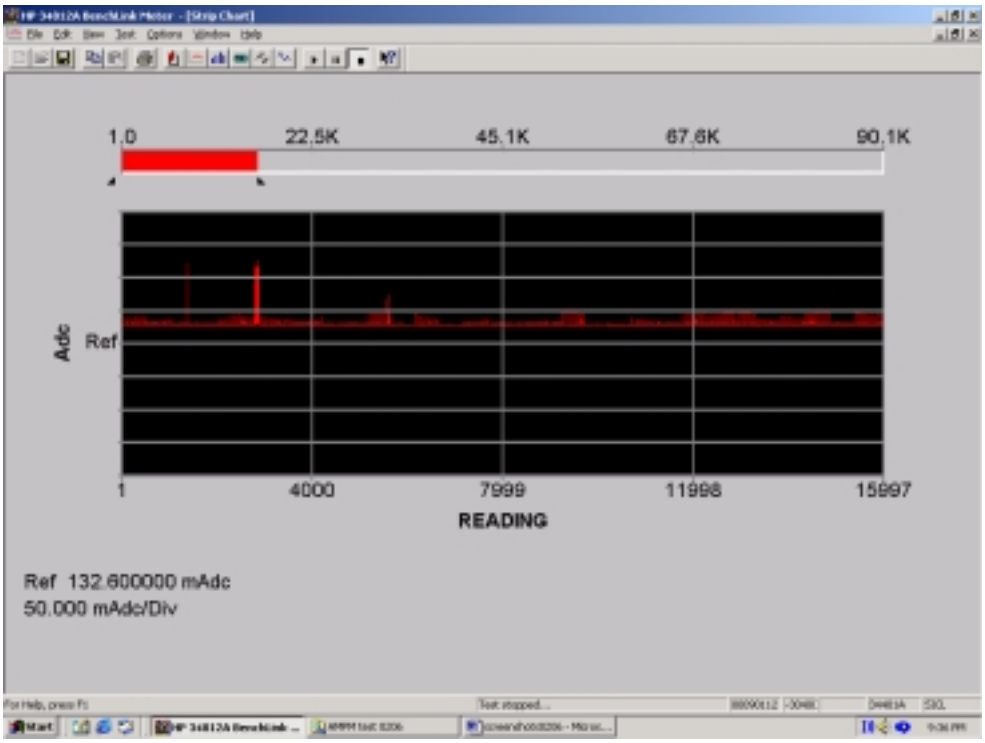

Figure 6.8: Expanded strip chart from Figure 6.1.

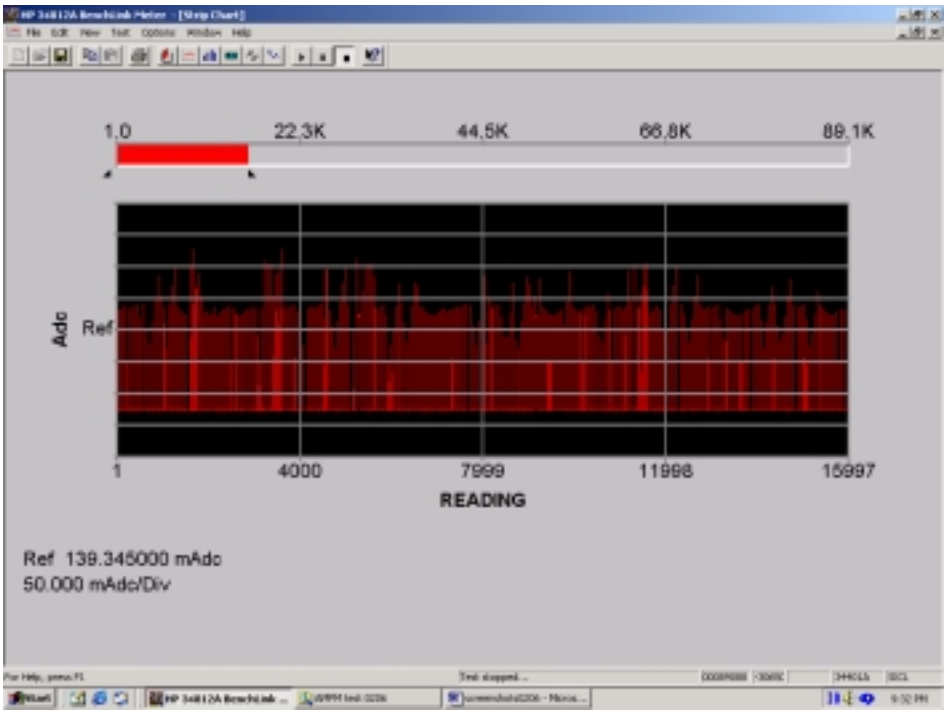

Figure 6.9: Expanded strip chart from Figure 6.2.

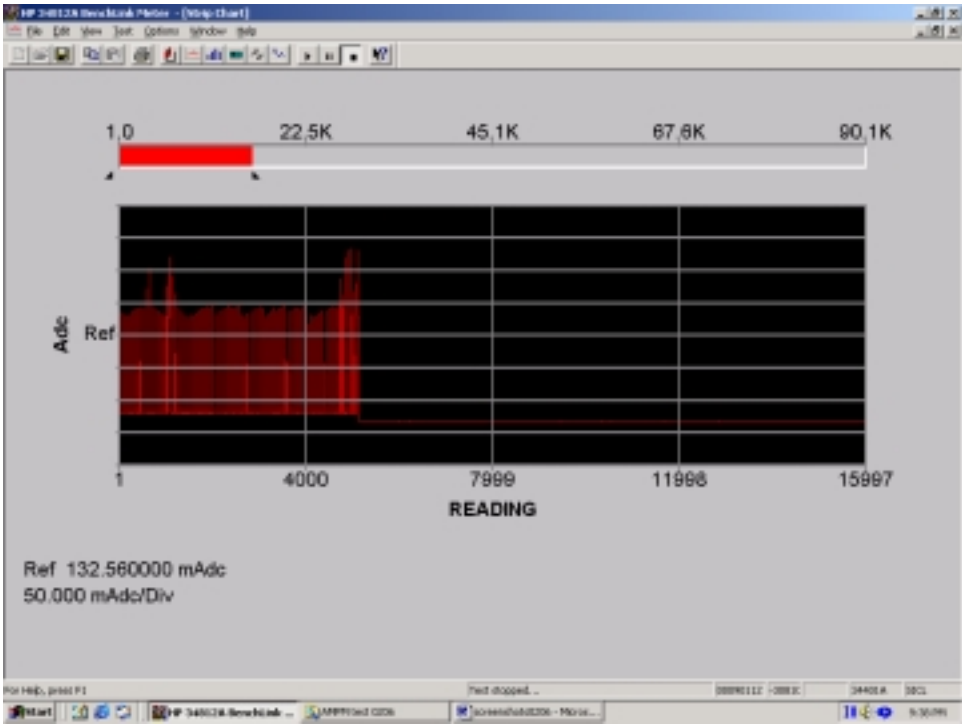

Figure 6.10: Expanded strip chart from Figure 6.3.

CHAPTER 7
FUTURE WORK AND CONCLUSION

Future work in this area has only just begun. Much more investigation into coordination between the application layer and the MAC/PHY layers that currently control the PC card will be needed. The integration of AM/PM poses many new questions to the rapidly expanding field of mobile device power management.

For example, AM/PM would not be available to the user who chooses to operate Windows O/S. The concept of transparency is not fully implemented. Presently, each user wishing to take advantage of Powerman APIs in their software suite will require some privileges that are normally reserved only for the superuser. Normal users are unable to use system commands such as **cardctl** and **ioctl**, both required by the AM/PM APIs described in this thesis. Connectivity issues including AM/PM and ad hoc networks, and highly mobile users requiring network handoffs would need to be added to the Powerman functionality. Right now, there is no power management available when operating in the ad hoc environment. How would coordination between multiple users who define the network itself be able to agree on who can take advantage of power savings modes, when they can do such and for how long this can occur? Research in this area could investigate the possibility of network layer power management to help answer some of these questions. Perhaps the same could help with highly mobile users, moving from access point to access point, yet wanting to hibernate while moving. What kind of

hand-off occurs when a user wakes up from hibernation only to find a new Access Point? All of these issues question the validity of an AM/PM approach to this problem.

The numbers are just too good to ignore. Active Mode Power Management performs much better than that of techniques built into the device driver layer. And this makes perfect sense. There is so much more information available about how much power is needed and when it will be needed next at the application layer. AM/PM is the future of enabling users the freedom they desire and connectivity they require.

We have only scratched the tip of the energy iceberg with this approach to saving power. Of course there are other concerns involving the smooth integration of AM/PM into mainstream industry. These include incorporation by standards such as IEEE 802.11, and incorporation of wireless extensions into each wireless device that would like to reap the power savings of AM/PM. Extensive future research in this area is required before we may truly feel the effects of the paradigm for Active Mode Power Management.

APPENDIX

WINDOWS DRIVER BENCHMARK TESTS

The goal of this experiment was to obtain three specific benchmark test figures for the purpose of comparing the three tests to which consumed the least amount of average power. The tests use the PC card with PS mode disabled and then enabled.

Power Saving Mode Disabled

For this benchmark test we first connected all the hardware as described in chapter five. A connection to the local access point was made with the wireless PC card. An email application (Outlook Express) was opened and set to check email every one minute. The multimeter was set to trigger for a sample every 100 ms and take 512 (the maximum allowed) on every trigger. The test was run for 30 consecutive minutes, yielding 463,360 readings at 1000 Hz for an average of 257 readings/sec. During this time one large email[1] was transmitted every five minutes. Figure A1 shows a screen shot of the Benchlink software running on the desktop after the completion of the test.

As you can see from the statistics in the bottom right hand corner of Figure A1, the maximum and minimum DC current readings during the 30-minute test were 288 mA and 159 mA respectively. More important is the average current value of 161 mA. This figure is relatively close the reading of the PC card while in the receive state, which is a near constant 160 mA. This is because the receive state is the default state when not in

PS Mode. It is noteworthy to mention that this figure is actually much better than the rating in the PC card specifications of 180 mA.

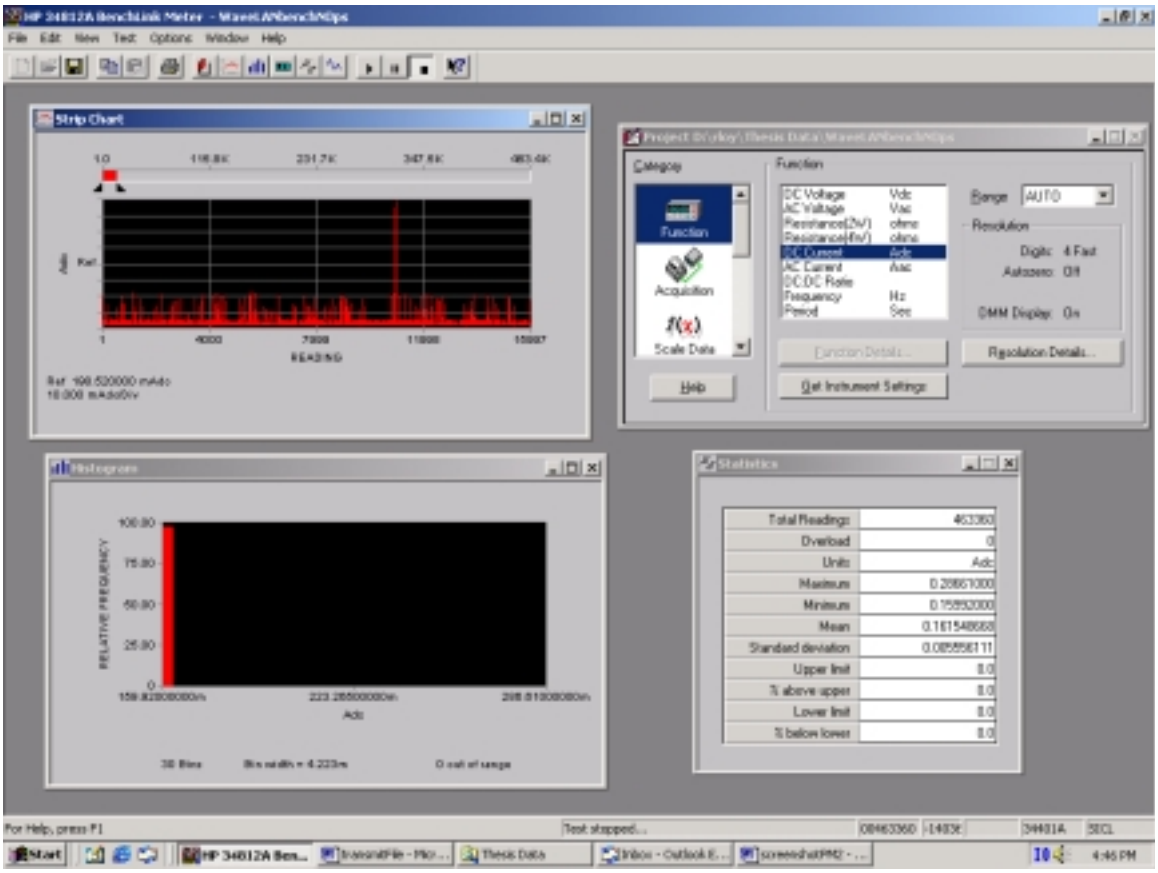

Figure A1: Power Savings mode disabled benchmark test results.

Even so, this is a large amount of energy being wasted by a mobile unit who is checking email quite frequently. Let's see how the PS mode stacks up in the same test environment.

Power Saving Mode Enabled

The identical benchmark test environment as above was recreated, only this time PS mode was enabled and the listen interval was set at the default level of 100 ms. So

[1] The large email was an attachment of size 1446 KB.

now the PC card will default to the doze state and check for a beacon indicating buffered traffic at the access point ten times per second. Figure A2 shows a screen shot of the Benchlink software upon completion of this test.

This time 483,840 samples were collected. The maximum and minimum DC current readings were 291 mA and 11 mA respectively. The average for this test is only 19 mA. This amounts to a whopping 800% improvement in energy efficiency! The delay in receiving the email messages was negligible.

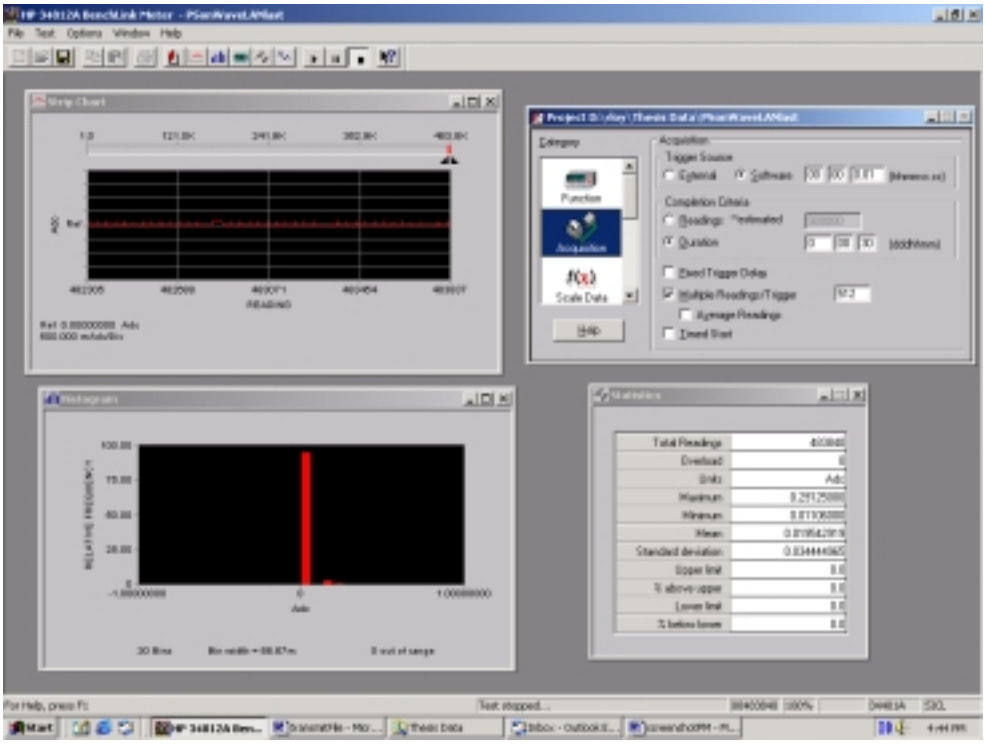

Figure A2: Power Savings mode enabled benchmark test results.

REFERENCES

[ELL99] Ellis, C., The Case for Higher Level Power Management. Duke University, March 1999.

[ELL00] Ellis, C., Leveck, A., and Vahdat, A., Every Joule is Precious: The Case for Revisiting Operating System Design for Energy Efficiency. Duke University, September 2000.

[FINN99] Finn, J., Satyanarayanan, M., Energy-aware Adaptation for Mobile Applications. Kiawah Island, SC, December 1999.

[FLEM00] Fleming, K., Hunter, R., Inouye, J., Schiffer, J., Enabling Always On, Always Connected (AOAC) Computing with Bluetooth Technology, 2000.

[HELA99] Helal, S., Any Time, Anywhere Computing. Kluwer Academic Publishers, Norwell, MA, 1999.

[HELM98] Helmbold, D., Long, D., Sconyers, T, and Sherrod, B., Adaptive Disk Spin-Down for Mobile Computers. University of California, Santa Cruz, October 1998.

[IEEE97] IEEE 802.11 Standard. Wireless LAN Medium Access Control (MAC) and Physical Layer Specification. Technical Report, IEEE, 1997.

[LORC98] Lorch, J., Smith, A., Software Strategies for Portable Computer Energy Management. University of California, Berkeley, February 1998.

[LOUG97] Lough, D., Blankenship, T., Krizman, K., A Short Tutorial on Wireless LANs and IEEE 802.11. *http://www.computer.org/students/looking/summer97/ieee802.htm*, 1997.

[SCO98] Sconyers, T., An Examination of Power Consumption in Wireless Modems, University of California, Santa Cruz, June 1998.

[STEMM] Stemm, M., and Katz, R., Measuring and Reducing Energy Consumption of Network Interfaces in Hand-Held Devices, University of California, Berkeley.

[TIWA96] Tiwari, V., Malik, S., Wolfe, A., Lee, M., Instruction Level Power Analysis and Optimization of Software. *Journal of VLSI Signal Processing*, vol. 13, pp. 1−18, 1996.

[TOUR] Tourrilhes, J., Wireless Tools for Linux, http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html

BIOGRAPHICAL SKETCH

Richard J. Loy was born on July 14, 1970, in Bristol, Pennsylvania. He received a bachelor's degree in mathematics from Carnegie Mellon University in May 1992 and was commissioned an Ensign, United States Navy at that time. He was designated a Naval Flight Officer January 20, 1995, and a Radar Intercept Officer in October 1995. He has completed two successful six-month deployments to the Arabian Gulf while attached to Fighter Squadron Two One Three aboard the carriers USS Kitty Hawk and USS Carl Vinson. During his career he has accrued 1000 flight hours in the F-14 Tomcat.

He joined the University of Florida in August 1999 to pursue a master's degree in computer science and has worked with Dr. Helal in the Harris Lab since August 2000. He remains on active duty with the United States Navy, and is attached to the University of Florida Navy ROTC Unit. His duties include teaching two undergraduate courses, NSC 1110 – Introduction to Naval Science, and NSC 1140 – Seapower and Maritime Affairs, advising the freshman NROTC class, sailing instructor, and recruiting officer. He also partially teaches NSC 4230 – Leadership and Management and NSC 4233 – Junior Naval Officer.

His future research interests are in advanced weapons/aviation systems technologies.