Active Mode Power Management for Mobile Devices[†]

Richard J. Loy and Abdelsalam (Sumi) Helal
Computer and Information Science and Engineering Department
University of Florida, Gainesville, FL 32611, USA
http://www.harris.cise.ufl.edu/projects/powermgmt.htm

ABSTRACT

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 sessions. 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 be treated equally. This paper 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 paper 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.

KEYWORDS: Mobile Computing, Active Mode Power Management.

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 [2] 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.

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 paper focuses on just one, Active Mode Power Management (AM/PM). This paper proves that this new power-saving method is as viable as any other 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 saving up to 62 times the amount of power used as compared to using no power management. The main idea is to examine closely the amount of power being used by a connected wireless PC card 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 covers related research. Chapter 3 expands on the details and motivation of the AM/PM approach. In Chapters 4 and 5 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 6 concludes with a summary and discussion of future research in the area of Active Mode Power Management.

[†] This research was partially supported by Microsoft Research, und grant number: 4514203-12.

2. SUMMARY OF RELATED RESEARCH

The only other documented research we could find in this area is being done at the computer science department at Duke University. Under Professor Carla Ellis their MillyWatt power management project aims to form a partnership between applications and the operating system with regard to setting power management policy [1]. 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.

No tangible evidence exists, 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.

3. APPLICATION LEVEL POWER MANAGEMENT APPROACH

3.1. 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 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.

3.2. 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 end result is a much more power-efficient mobile device at no appreciable extra cost to the user. 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.

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.

3.3. 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. Email 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 [2].

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. Which 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?

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. We aim to show that application layer power management can be implemented and at an excellent power savings.

3.4. 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 chose to use the **links** browser. It is an open source, GPL program, freely available from the linux.org web site.

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. 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 cannot 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.

3.5. 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. 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

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 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.

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) [3]. 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.

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.

3.6. The AM/PM APIs

Finally, we are into the meat and potatoes of this paper. 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.

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.

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[1], 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 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

[1] A new term meaning an extended sleep period, allowing for large power savings, but still maintaining the context of a network connection.

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.

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.

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 paper 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.

4. EXPERIMENTAL VALIDATION

4.1. Implementation

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 Lucent WavePOINT II Access Point
2. A Lucent WaveLAN Silver pc card– 11Mbs
3. An Accurite PC ExtenderCard
4. A Hewlett Packard 34401A multimeter
5. A Dell Latitude laptop, PII 233 MHz, 64 MB RAM
6. A Dell desktop, PIII 600 MHz, 128 MB RAM

4.2. Benchmark Tests – Email Client

The following sequence was used for each of the following three tests:

1. Initiate GetPop from the command line prompt.
2. Enter valid user name when prompted.
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 four times in the four-minute period.

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.

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). 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.

The AM/PM yields a similar maximum but a much less minimum DC current reading; 266.5 mA and .6 mA 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.

4.3. Benchmark Tests – WWW Browser

For this test 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. Decide on CNN instead.
3. Go to www.cnn.com. Find World news link.
4. Take internal link to World news.
5. Take link to read the leading article.
6. Take link to U.S. News.
7. Take l link to read the leading article.
8. Take link to Science and Technology.
9. Take link to read the leading article.

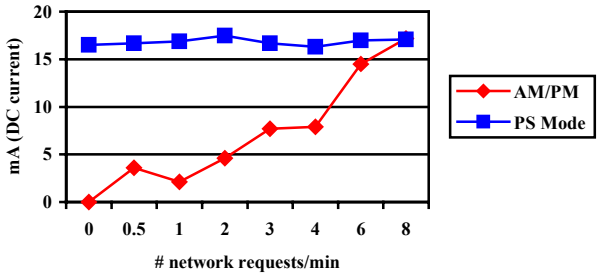
The maximum and minimum DC current readings during the four-minute PS mode disabled test were 265.6 mA and 159.9 mA respectively. The average value was 160.9 mA.

For the PS mode enabled 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.

AM/PM yields a similar maximum but a much less minimum DC current reading; 281 mA and .52 mA respectively. Again, the 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.

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 6.1 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 6.1 Native PS mode vs. AM/PM.



5. 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.

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.

6. REFERENCES

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

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

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