A Quantitative Analysis of Power Consumption for Location-Aware Applications on Smart Phones

Arjun Anand[1], Constantine Manikopoulos[2], Quentin Jones[3], and Cristian Borcea[1]

Departments of [1]Computer Science, [2]Electrical and Computer Engineering, and [3]Information Systems
New Jersey Institute of Technology
University Heights, Newark, NJ 07102, USA

aa297@njit.edu, manikopoulos@njit.edu, qjones@njit.edu, borcea@cs.njit.edu

***Abstract*—The industry is producing new wireless mobile devices, such as smart phones, at an ever increasing pace. In terms of processors and memory, these devices are as powerful as the PCs were one decade ago. Therefore, they are perfectly suitable to become the first real-life platforms for ubiquitous computing. For instance, they can be programmed to run location-aware applications that provide people with real-time information relevant to their current places. Deploying such applications in our daily life, however, requires a good understanding of their power requirements in order to ensure that mobile devices can indeed support them. This paper presents a quantitative analysis of power consumption for location-aware applications in our SmartCampus project, which builds a large scale test-bed for mobile social computing. Based on this analysis, we conclude that carefully designed applications can run for up to six hours, while updating the user location frequently enough to support real-time location-aware communication.**

I. Introduction

Mark Weiser had envisioned an era where people would not be confined to their desks for computing, as computers would blend in our daily environments [1]. Today, a good part of Weiser's prediction has come true. Computers have become increasingly networked, mobile, and small. We see a barrage of mobile devices, such as smart phones, PDAs, and pocketPCs, with processing capacity and networking ability that was previously thought impossible. Typically, they come equipped with several wireless network interfaces, such as WiFi, cellular (including 3G), and Bluetooth, processors running at 100-400MHz, and 64-128MB of memory.

From stock tickers to city-wide social games, these devices promise to offer support for a large spectrum of ubiquitous computing applications [2]. We believe that location-aware mobile social applications that link people-to-people-to-places (P3-Systems [3]) in real-time will be the "killer applications" for ubiquitous computing environments. Until recently, the ability to locate individuals seamlessly has been very limited, but with the introductions of technologies such as WiFi, GSM, Bluetooth, and GPS, we are capable to overcome this issue in a cost-efficient manner [4, 5].

We are currently building SmartCampus, a test-bed for P3-Systems that leverages all these new technologies [6]. In the very near future, SmartCampus will consist of several hundred heterogeneous mobile devices carried by students on our campus. In this test-bed, real-time user location information is used to improve collaboration and coordination among friends, acquaintances, or people with similar interests. The applications are built over a common service-oriented middleware that captures, processes, and shares social information among mobile devices.

As expected, battery power represents the most important limitation that we face in developing P3-System applications. For instance, the basic question asked by any user who plans to run an application on her smart phone is: "how long can I use my phone if I run this application?" A software developer, on the other hand, would be interested to know how much power is consumed by each hardware component in order to optimize the applications and be able to inform the users about the power consumption of each application. Previous research has focused on topics such as energy-aware application adaptation [7], characterization of power consumption for user interfaces on handheld devices [8], and energy consumption for mobile peer-to-peer applications [9].

This paper presents a quantitative analysis of power consumption for location-aware applications running on smart phones. The main purpose of this analysis is to provide a set of experimental results and guidelines for system and application programmers. This is because the software industry must be made aware of the constraints imposed on mobile devices by their limited battery power in order to design and deploy efficient applications. Our results indicate that cellular communication and WiFi communication have similar power behavior. Furthermore, the processor can be the main source of power drainage for many applications. Overall, we conclude that carefully designed location-aware applications can run on typical current generation smart phones for up to six hours while supporting real-time communication among users.

The rest of this paper is organized as follows. Section II introduces the software architecture of the SmartCampus test-bed and a prototype location-aware application, namely

CampusMesh. Section III describes the experimental setup. In Section IV, we present the power consumption analysis for general application running on smart phones, with a special focus on the hardware components that drain the most power during execution. Section V shows experimental results for the SmartCampus location engine, which is used by all location-aware applications. We discuss the lessons learned and conclude in Section VI.

II. SmartCampus Software Architecture

The SmartCampus applications are developed on top of UbiCon, our service-oriented middleware for P3-System applications. UbiCon securely collects individual, community, place, and social event data from associated P3-System applications, which are continuously mined to produce an increasingly rich model of the social environment. Unlike typical distributed middleware solutions, our middleware not only shields the programmers from the distributed heterogeneous hardware, but also provides social models of the campus community and allows a large spectrum of P3-System applications to access data derived from these models in a simple and secure way through a service API

UbiCon was implemented in Java, and its core services run over the Apache Tomcat server. These services are exposed using KSOAP, a SOAP toolkit designed to work with lightweight versions of JAVA. Essentially, any P3-System application has two parts: a service that uses UbiCon to access data about users and their locations, and a thin client that runs on mobile devices such as smart phones. These clients are developed in Java using OSGi for simple software management [10].

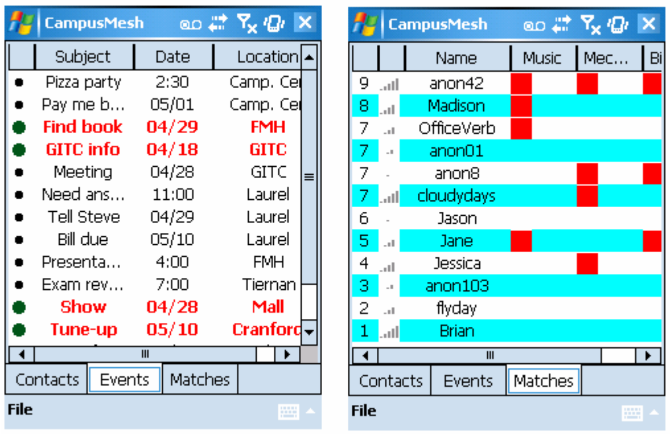
A key client component is the location engine that runs on mobile devices. This engine scans for visible WiFi access points and employs centroid and fingerprinting techniques to compute the current user location. We assume that all mobile clients have wireless Internet access either through WiFi or through cellular communication. This assumption holds as many urban places such as college campuses already have full WiFi coverage; in places that do not have full WiFi coverage, we assume the use of smart phones or cellular data cards that allow Internet access.

To illustrate a typical location-aware application in SmartCampus, we describe CampusMesh, a social matching and alert delivery application developed using UbiCon. CampusMesh runs on heterogeneous mobile devices such as smart phones, PDAs, and TabletPCs. It leverages rich user profile information, user preferences, social networks, and user mobility traces for social match generation. Additionally, it provides context-aware alerts. This application encourages the formation of new friendships, supports goal directed team formation, and geo-temporal personal relationship management. Let us consider the following scenarios to understand how CampusMesh can be used to set up a geo-temporal alert and to provide a social match.

Scenario 1: Mike and Joe regularly meet at a computing laboratory to study for their Ubiquitous Computing class. Joe keeps bringing a textbook borrowed from Mike that needs to be returned to the library, but every time they meet, other stuff comes up and he forgets to return it. Finally, he decides to set a reminding alert triggered by proximity to Mike. When they meet next time, the CampusMesh client delivers the reminder; Mike gets his book back and returns it to the library.

Scenario 2: One Monday afternoon, Ayala is relaxing at the university cafeteria when her smart phone vibrates to indicate she has a CampusMesh message. She sees that it has detected another female student with a high affinity match in the vicinity, who she might wish to meet. Though the system does not reveal the strangers identity, it does inform Ayala that they have several friends in common and a mutual interest in art movies. As Ayala is hoping to organize art movie projections on-campus, she sends a message to find out more, and after further data is exchanged through mutual user controlled progressive identity revelation, she decides to personally meet the other user.

Fig. 1 presents the CampusMesh user interface on smart phones. On the left side, we show the *event tab*, used for alerts such as the one described in the first scenario. On the right side, we show the *match tab,* used for social matching such as the one described in the second scenario.



	Subject	Date	Location
●	Pizza party	2:30	Camp. Ce
●	Pay me b...	05/01	Camp. Ce
●	**Find book**	**04/29**	**FMH**
●	**GITC info**	**04/18**	**GITC**
●	Meeting	04/28	GITC
●	Need ans...	11:00	Laurel
●	Tell Steve	04/29	Laurel
●	Bill due	05/10	Laurel
●	Presenta...	4:00	FMH
●	Exam rev...	7:00	Tiernan
●	**Show**	**04/28**	**Mall**
●	**Tune-up**	**05/10**	**Cranford**

	Name	Music	Mec...	Bi
9	anon42	🟥	🟥	🟥
8	Madison	🟥		
7	OfficeVerb	🟥		
7	anon01			
7	anon8		🟥	🟥
7	cloudydays		🟥	
6	Jason			
5	Jane	🟥		🟥
4	Jessica		🟥	
3	anon103			
2	flyday			
1	Brian			

Fig. 1. CampusMesh User Interface on Smart Phones

III. Experimental Setup

For our quantitative analysis, we ran both micro-benchmarks to evaluate the individual power consumption of the main hardware components (e.g., processor, WiFi interface, backlight) and experiments that measured the power consumed by the SmartCampus location engine. The experiments were performed on iMate KJam smart phones, each with the relevant specifications as listed in Table I.

TABLE I
SMART PHONE CHARACTERISTICS

Battery Type	Lithium Polymer 1250mAh
Memory	128 MB
OS	Windows Mobile 5.0
Processor	195 MHz OMAP processor

The following software was used to measure the battery life and the CPU usage of the devices: ACB Power Meter[11] and Pocket Hack Master[12]. Previously, the unanimously accepted best way to measure battery usage by smart phones was by connecting an Ampere counter between the battery and the smart phone [13]. Although it is potentially more accurate, this solution was very difficult and virtually impossible without soldering wires onto the smart phone. The ACB Power Meter simplified significantly the task of monitoring battery usage. The Pocket Hack Master was used to analyze the CPU usage of various processes performed on the device.

IV. Smart Phone Power Consumption

Numerous factors affect the battery lifetime of a smart phone. Despite the fact that each application that runs on the phone contributes differently to its battery drain, there are five fundamental hardware components that consume most of the power: backlight, Bluetooth, CPU, WiFi, and the cell radio. By understanding their relative contribution to the overall power consumption, application programmers can estimate the power requirements for each individual application based on its specific logic. Furthermore, they can design the applications to make better use of the limited power resources. In the following, we present experimental results for these five components and conclude the section with the component distribution of power consumption for a typical use of the smart phone.

A. *Backlight*

Smart phones have larger screens compared to regular mobile phones. Hence, they require larger backlights that consume more power. Most of these phones come with settings for backlight levels that may be adjusted by users. Our smart phone has five levels of backlight, with 0 being no backlight and 4 being the brightest. We conducted experiments for each level, starting with no backlight and moving up to the 4th level. We observed that the 1st level of backlight consumes 45-65mAh. Subsequent levels of backlight burn approximately 30-40mAh more power than the previous level. At the 4th level, the battery consumed is almost 200mAh. Thus, keeping the backlight continuously on at the 4th level would completely drain the battery in slightly over 6 hours.

B. *Bluetooth*

A commonality among recent smart phones is the presence of a Bluetooth interface, which provides the convenience of sharing files or using Bluetooth headsets and keyboards. We measured the power consumed by various activities using the Bluetooth adapter, and the results are summarized in Table II.

TABLE II
BLUETOOTH ADAPTER CHARACTERISTICS

Bluetooth On	20-35mAh
Scanning for Devices	10mAh
Data Transfer	30-40mAh

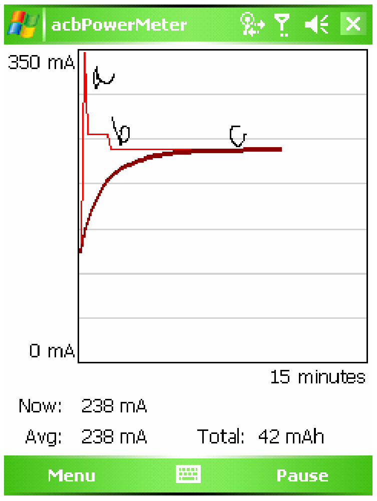
C. *CPU*

The CPU of a handheld device is the component that drains the most battery power, assuming that WiFi is not used very often. The faster a processor, the more power it consumes. Even in standby mode, the CPU is used to keep the Smart phone "awake," and consumes approximately 5-10mAh. In the following, we list a few important characteristics of the CPU power consumption on our smart phones:

- On average, the CPU remains in the idle state for 95% of the time. The idle time consumes much less power than when the CPU is being actively used.
- Turning on/off devices such as the WiFi network card and Bluetooth adapter causes a spike in the usage of the CPU, but maintaining the state of the respective devices does not take much CPU time. For example, turning on the WiFi network card of a smart phone shows the usage of 90-120mAh.
- The CPU does not participate in scanning for WiFi access points or Bluetooth devices.
- We observed that most common games on the phones are very CPU intensive, and hence, they drain a lot of battery (a significant factor that influences these results is the software emulation of floating point operations by the processor).

D. *WiFi*

Most new smart phones come equipped with a WiFi network interface. Although this interface can be very useful for ubiquitous computing applications, it could lead to a quick exhaustion of the battery if its use is not controlled properly. Fig. 2 depicts the power consumption for the WiFi interface. The top line represents the instantaneous power consumption, while the bottom line represents the average power consumption over time. The markers on the figure indicate the points when the WiFi is switched on (a), when the device is being authenticated into the network (b), and when the connection with the network has been established (c).



Now: 238 mA
Avg: 238 mA Total: 42 mAh

Fig. 2. Battery Usage while Connecting to WiFi Network

Just having the WiFi card on, without any data transfers, drains about 130-150mAh. Note that this value includes the power consumed by the smart phone in steady state. This means that when connected to a network, the battery would last 8-9 hours without any computation or communication. The network card can drain more battery than the CPU if it remains on for longer periods than the CPU utilization periods. Also, the network card drains more power while scanning for available networks than when maintaining an existing network connection. Continuous data transfers over the network also drain the battery of the device by an additional 150-200mAh in transmit/receive mode. We observed that the power consumption is similar in transmit and receive mode.

E. Cell Radio

The cell radio burns about 2-4mAh on top of the 5-10mAh standby usage of the smart phone. A phone call made over the GSM cellular service costs the battery about 250-300mAh. This value is just slightly less than transferring data over the WiFi interface. Therefore, considering that WiFi is typically free, while the cellular network incurs a certain cost, WiFi should be used as the network interface of choice for data transfers each time it is possible.

F. Power Consumption for Typical Use of a Smart Phone

To obtain a rough estimate the percentages of time when a typical user makes phone calls, browses the Internet, uses her Bluetooth headset, plays games, or plays media files, we interviewed 18 students on our campus. Based on this input, Fig. 3 illustrates the hardware component distribution of power consumption. We observe that the processor consumes the most power, followed closely by the WiFi and the GSM radio. The Bluetooth is still significant because the typical user keeps the headset on most of the time. The backlight is not very important because the users turn it off or maintain it at a minimum level most of the time. The other parameters that affect the power consumption include the memory and the speaker.

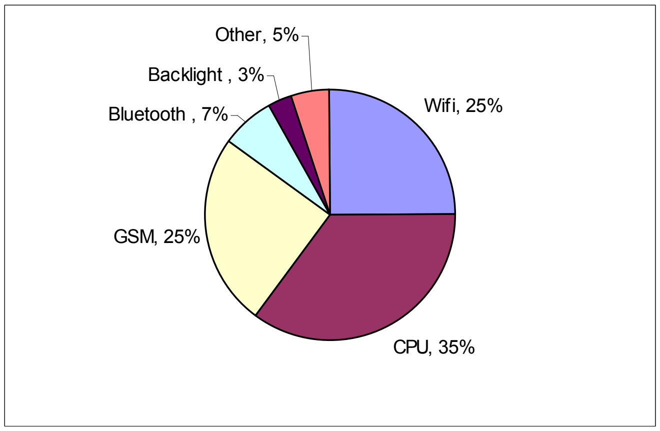


Fig. 3. Power Consumption for Typical Smart Phone Usage

V. <u>Location Engine Power Consumption</u>

All SmartCampus location-aware applications use a common location engine that runs on the smart phone. This section presents experimental results that quantify the power consumption of this engine. The engine uses a database of known access points and their associated locations (latitude and longitude). This database is loaded as a hash table into the smart phone memory at runtime for fast access. Each time the location is computed, the WiFi network card is made to scan for visible access points, and subsequently the location of these access points is retrieved from the database. In the centroid method that we used for experiments, the position of the user is computed as the average of these locations. Once the computation is done, the location is sent over the network to a server as a SOAP (Simple Object Access Protocol) message. This procedure is repeated at regular scan intervals to update the location of the user. From this description, it is clear that the CPU and WiFi are the biggest power drain components for the location engine.

The amount of battery drained by the WiFi is function of the scanning frequency. Obviously, as the scanning period increases, the battery life increases. Hence, for a longer battery life, the scanning interval should be as large as possible. However, with a very large scanning period, the location updates to the server are also much less frequent. Thus, it defeats the purpose of having real-time updates of user's position. Ideally, the location needs to be updated every few seconds for good accuracy, but because of a limited battery power, the scanning interval becomes an important design decision. This decision is made by taking into consideration the trade-off between update frequency and battery drainage.

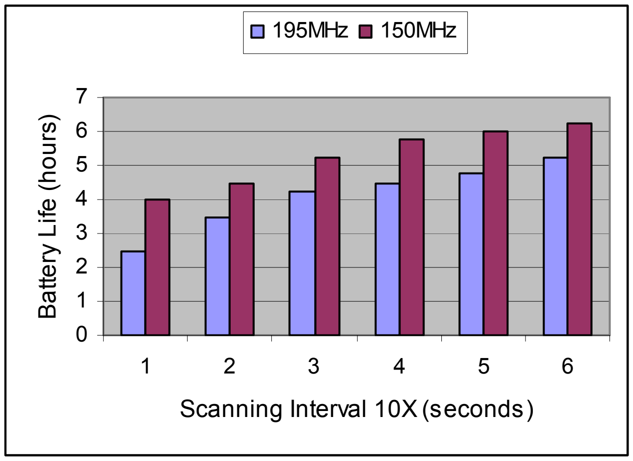
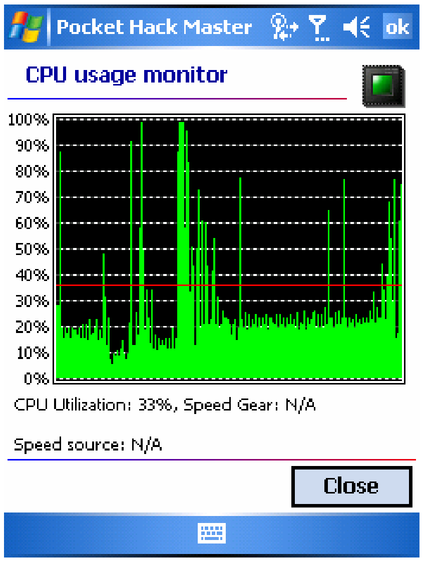


Fig. 4. Battery Lifetime as Function of Location Scan Interval and Processor Speed

Fig. 4 shows the battery lifetime as function of WiFi scanning frequency. We also plot the battery behavior when under-clocking the CPU at 155 MHz instead of its regular 195 MHz. In this experiment, we consider the entire process of scanning for access points, computing the location on the phone, and sending this location encapsulated in a SOAP message to a server. If we assume that a student will be on-campus at least 3 hours (i.e., attending one 3-hour course), the results at 195MHz demonstrate that the location update frequency should be at most three times per minute. We also discovered that the CPU speed affects the battery consumption significantly (at least 20% in our experiment).

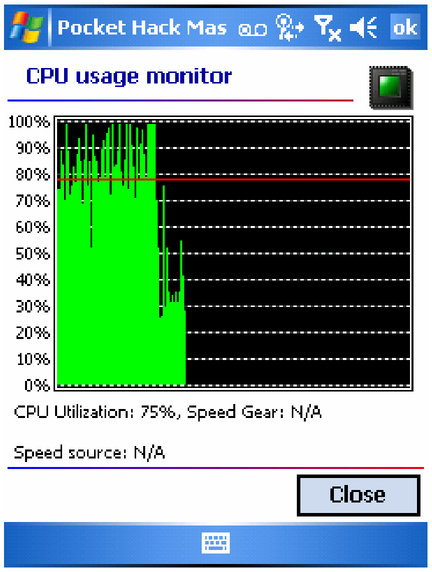


CPU Utilization: 33%, Speed Gear: N/A

Speed source: N/A

Fig. 5. CPU Usage for Location Scan Interval of 20 Seconds

Computing the location uses up a fair amount of the CPU time. Fig. 5 shows the CPU usage at 195MHz for a scanning interval of 20 seconds (as determined above). The results demonstrate that the CPU is utilized 33% only by the location engine. Therefore, the applications will be capable to use it just for the rest of the time.



CPU Utilization: 75%, Speed Gear: N/A

Speed source: N/A

Fig. 6. CPU Usage for Location Scan Interval of 1 Second

Furthermore, Fig. 6 demonstrates that a small scanning interval such as 1 second not only would consume a lot of energy due to WiFi scanning, but also would use up to 75% of the processing power. Thus, it would leave very few processor cycles for applications.

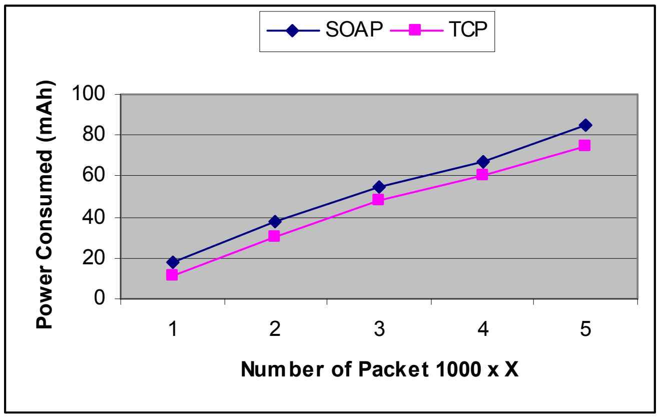


Fig. 7. SOAP vs. TCP Power Consumption

Once the location is computed, the smart phones will send it to a location server via SOAP. We decided to expose the SmartCampus services via SOAP because this protocol offers language independence and SOAP clients are available for many popular languages. Additionally, it provides a clean transport mechanism, with the client services communicating over HTTP. We use KSOAP J2ME library that has a memory footprint less than 50KB. This makes it extremely suitable for resource constraint devices such as smart phones. The KXML parser provides good performance comparable to XML-RPC. We compared the power consumption when using SOAP vs. TCP. The results from Fig. 7 demonstrate the SOAP overhead in terms of power consumption increases just slightly with the number of packets. This is due to the efficient processing of the KXML parser.

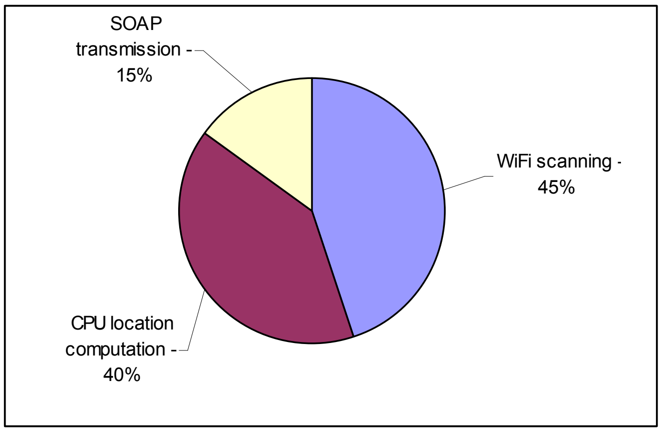


Fig. 8. Power Consumption Distribution for the SmartCampus Location Engine

Fig. 8 summarizes our findings for the power consumption of the location engine. The experiments indicate that 45% of the power is spent for WiFi scanning to discover nearby access points, 40% is consumed by the processor to compute the user location, and 15% is consumed by both the processor and the WiFi interface to encapsulate the location data in a SOAP message and send it to the location server.

VI. Conclusions and Lessons Learned

Based on our experiments, we derived a number of guidelines for software designers in order to maximize the battery lifetime of smart phones running location-aware applications such as CampusMesh. Since the CPU consumes a significant amount of power, the programmers should try to offload the computation to a server each time it is possible. In making this decision, they must trade-off CPU power consumption against WiFi power consumption (i.e., the WiFi will be used to exchange parameters and results with the server).

A faster processor is not necessarily better for mobile devices running ubiquitous computing applications. This is due to the high power consumption of the processor. A lower processor speed (e.g., 155MHz instead of 195MHz) does not influence the functionality of certain applications, while it saves a good amount of battery power. As shown by the results from Fig. 4, the battery life was extended by over an hour when using under-clocking.

Since keeping the WiFi card off is not an option with applications that need real-time communication, one way to increase the battery lifetime is to reduce the frequency of WiFi scans used by the location engine. Another potential solution is to have a mechanism that switches the WiFi network card on and off at certain intervals. However, this can end up in a worse drain of power than keeping the interface on all the time. A possible solution is to track the location of the smart phone, and based on its relocation frequency, to determine when it makes sense to switch the WiFi. For instance, if a student is in a classroom, she would not be moving. Sensing this stagnancy from a few scans, the application may turn off the WiFi device and may turn it on every 10 minutes to see if the location data has changed or if any new messages have been buffered at the server for this user. If the location data has changed, the scanning can resume at the normal interval specified. If the location data has not changed, the WiFi device may be turned off again.

The decision on the value of scan interval depends on the expected overall system performance. Shorter scan intervals consume more power, but the location of the smart phone is updated faster. This is appropriate for applications that need accurate location or real-time location-based communication. Longer scan intervals improve the battery life at the expense of having slower updates, and consequently the applications decrease their location-aware responsiveness. From our experiments, we believe that a 30 second to 1 minute scanning interval might be a good trade-off. In such a situation, the battery lifetime can be as much as 6 hours when only the location engine is used. Applications that use the processor and the WiFi moderately can, therefore, run for a significant period of time.

Acknowledgments

This work was supported in part by the NSF grants CNS-0454081 and IIS-0534520.

References

1. M. Weiser, *The Computer for the 21st Century*. Scientific American 265(3):94-104, September 1991.
2. L. Iftode, C. Borcea, N. Ravi, P. Kang, and P. Zhou. *Smart Phone: An Embedded System for Universal Interactions.* In Proceedings of the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS'04).
3. Jones Q., and Grandhi S.A., *Supporting Proximate Communities with P3-Systems: Technology for Connecting People-To-People-To-Geographical-Places*. The Interaction Society: Practice, Theories, & Supportive Technologies, 2004.
4. B. Schilit, A. LaMarca, G. Borriello,W. Griswold, D. McDonald, E. Lazowska, A. Balachandran, J. Hong, and V. Iverson. *Challenge: Ubiquitous Location-Aware Computing and the Place Lab Initiative*. In Proceedings of the 1st ACM International Workshop on Wireless Mobile Applications and Services on WLAN (WMASH 2003), San Diego, CA, Sep 2003.
5. Placelab Project: http://placelab.org
6. SmartCampus Project: http://smartcampus.njit.edu
7. J. Flinn and M. Satyanarayanan, *Managing Battery Lifetime with Energy-Aware Adaptation*, ACM Transactions on Computer Systems, Vol. 22, No. 2, May 2004, Pages 137-179.
8. L. Zhong and N. Jha, *Energy Efficiency of Handheld Computer Interfaces: Limits, Characterization, and Practice*, In Proceedings of 3rd International Conference on Mobile Systems, Applications, and Services (Mobisys 2005), Seattle, Washington, Jun 2005, Pages 247-260.
9. S. Gurun, P. Nagpurkar, and B. Y. Zhao, *Energy Consumption and Conservation in Mobile Peer-to-Peer Systems*, In Proceedings of the 1st ACM International Workshop on Decentralized Resource Sharing in Mobile Computing and Networking (MobiShare 2006).
10. Knoplerfish: Open source OSGi implementation. http://www.knopflerfish.org
11. ACBPowerMeter:http://www.pocketpcfreewares.com /en/index.php?soft=1564
12. Pocket Hack Master: http://www.pocketgear.com/ software_detail.asp?id=7258
13. P. Divevey, N. Lorenzon, and C. Tambary, *Measuring wireless energy consumption on SmartPhones and laptops.* Technical Report, DISI, Politecnico Di Milano.