Improving Energy Efficiency of Location Sensing on Smartphones

Zhenyun Zhuang[1]* Kyu-Han Kim[2]† Jatinder Pal Singh[2]

[1]Georgia Institute of Technology, Atlanta, GA 30332, U.S.A.
[2]Deutsche Telekom R&D Laboratories USA, Los Altos, CA 94022, U.S.A.
zhenyun@cc.gatech.edu, kyu-han.kim@telekom.com, jatinder.singh@telekom.com

ABSTRACT

Location-based applications have become increasingly popular on smartphones over the past years. The active use of these applications can however cause device battery drain owing to their power-intensive location-sensing operations. This paper presents an adaptive location-sensing framework that significantly improves the energy efficiency of smartphones running location-based applications. The underlying design principles of the proposed framework involve substitution, suppression, piggybacking, and adaptation of applications' location-sensing requests to conserve energy. We implement these design principles on Android-based smartphones as a middleware. Our evaluation results show that the design principles reduce the usage of the power-intensive GPS (Global Positioning System) by up to 98% and improve battery life by up to 75%.

Categories and Subject Descriptors

C.3.3 [**Special-Purpose and Application-Based Systems**]: Real-Time and Embedded Systems

General Terms

Design, Experimentation, Measurement, Performance, Algorithms

Keywords

Location Sensing, Energy Efficiency, Location-Based Applications, Smartphone.

1. INTRODUCTION

With the increasing pervasiveness of smartphones over the past years, several Location-Based Applications (LBAs) have been adopted by mobile users for always-on contact for social-networking, businesses needs, and entertainment. Some instances of currently popular LBAs include mobile social networking [2, 3, 9, 10], health-care [1], local traffic [7, 22, 23, 29, 36], and local restaurants [6].

*Zhenyun Zhuang was an intern at Deutsche Telekom R&D Lab USA while this research was conducted.

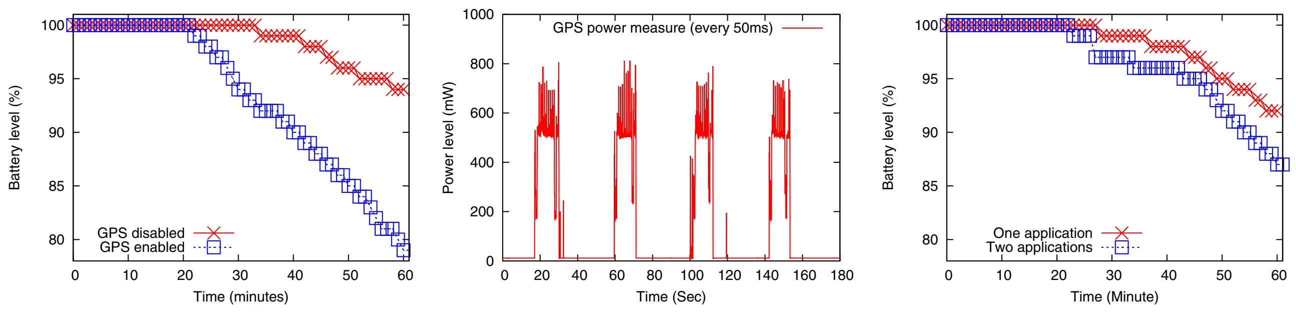
†To whom correspondence should be addressed.

In spite of the increase in processing power, feature-set, and sensing capabilities, the smartphones continue to suffer from battery life limitation, which hinders the active utilization of LBAs. Typical battery capacity of smartphones today is barely above 1000 mAh (e.g., the lithium-ion battery of HTC Dream smartphones has the capacity of 1150 mAh). Unfortunately, GPS (Global Positioning System), the core enabler of LBAs, is power-intensive, and its aggressive usage can cause complete drain of the battery within a few hours [14, 17]. While the aggressiveness of GPS usage is specific to different applications, several LBAs such as local traffic (e.g., [7]) and social networking (e.g., [9]) particularly benefit from continuous location updates. Real Time Traffic [7], for instance, requires continuous GPS location updates. Twidroid [9], a mobile version of Twitter, features a GPS accuracy booster, which provides an option to enable/disable continuous GPS sensing.

Numerous solutions have been proposed to improve the battery life of mobile devices [11, 32–34], but little rigor and attention has been devoted to the battery-efficient use of LBAs. The LBA developers are suggested to reduce the use of GPS by increasing location-update intervals (say, to more than a minute), thus allowing GPS hardware to sleep between successive location-updates. Such a simple solution can improve battery life by forcing applications to request location information less frequently, but it has fundamental limitations. For instance, although each LBA can save energy by reducing GPS invocation, the effectiveness of this approach could be compromised when multiple LBAs are running, as the asynchronous use of GPS from different LBAs unnecessarily leads to an increased number of invocations.

In this paper, we present an energy-efficient location-sensing framework that effectively conserves energy for smartphones running LBAs. In its core, the proposed framework includes four design principles: Substitution, Suppression, Piggybacking and Adaptation. Briefly, *Substitution* makes use of alternative location-sensing mechanisms (e.g., network-based location sensing) that consumes lower power than GPS. *Suppression* uses less power-intensive sensors such as an accelerometer to suppress unnecessary GPS sensing when the user is in static state. *Piggybacking* synchronizes the location sensing requests from multiple running LBAs. *Adaptation* aggressively adjusts system-wide sensing parameters such as time and distance, when battery level is low.

We implement the four design principles on a G1 Android Developer Phone (ADP) as a middleware and evaluate the implementation extensively via measurements. While the proposed design principles are general enough to be applied to any software stack, the middleware implementation allows for better application transparency in the sense that applications can be kept as-is. We choose Android-based smartphones for prototyping because of the openness of the Android platform [5]. Our evaluation results with the



(a) Real Time Traffic (b) Power spikes (c) Multiple LBAs

Figure 1: Energy Consumption of Gps

implementation show that the proposed framework significantly saves energy in location sensing. For instance, in various scenarios, our prototype reduces the number of GPS invocations by up to 98%, and thus improves the battery life by up to 75%.

To summarize, this work makes the following contributions:

- We address and explore energy efficiency of location sensing for resource-constrained smartphones that often run multiple location-based applications (LBAs).
- We study four design principles tailored for LBAs to reduce energy consumption in location-sensing on smartphones and show that the integration of the proposed design principles enables significant energy savings.
- We prototype the proposed design in Android-based smartphones, which are open to both practice and research, and demonstrate the effectiveness through real-life measurements.

The remainder of the paper is organized as follows. Section 2 motivates this work. Section 3 presents the key design principles and their integrated operation. Section 4 describes our implementation. Section 5 shows evaluation results of our prototype. Section 6 discusses related work, and Section 7 concludes this paper.

2. MOTIVATION

In this section, we motivate the present work by highlighting results from a set of experimental evaluations. We demonstrate factors impacting energy efficiency in location sensing by employing G1 ADP phones and summarize the limitations of existing smartphone usage that prevent energy-efficient location-sensing.

2.1 GPS Energy Consumption

We first assess the impact of using power-intensive GPS on smartphones. We consider a scenario where a user is driving with a traffic-monitoring LBA, called "Real Time Traffic" [7], running. The application is popularly used to determine traffic speed based on anonymous collection of users' locations, speed, and direction information. While running this LBA (version 1.0.2e(17)) on a smartphone, we measure instantaneous battery levels of the phone over an hour, using power-APIs provided by Android Software Development Kit (SDK).[1] For comparison, we also run the same LBA on the second phone with GPS disabled. For both experiments, we start with a fully charged battery after charging for the same amount of time. The screens of the phones are always kept on. The map refreshing rate and GPS invocation interval of the LBAs are set to 5 seconds.

Figure 1(a) shows the battery level of the phone during the run. As shown in the figure, when GPS is used, the battery level drops to 79% within one hour, whereas the battery level with GPS disabled drops to only 94%. Note that we ran the experiment multiple times with different setups such as charging time, and we always see the same trends in battery-level drops across all runs.

We also measure instantaneous power-spikes of GPS sensing using a digital multi-meter (Agilent 34410A) to see microscopic power usage. Figure 1(b) shows the power spikes of the phone (measured once every 50 ms) when an LBA requesting GPS runs. As shown in the figure, a typical GPS invocation consists of a locking period and a sensing/reporting period. The lengths of these two periods are about 4-5 seconds and 10-12 seconds, respectively. More importantly, the average power draw of the two periods are about 400 mW and 600 mW, respectively. For a typical battery capacity of 1000 mAh such high power consumption is very expensive as continuous GPS sensing can deplete the battery in merely 6 hours (i.e., $\frac{1000mAh \times 3.7V}{600mW}$).

2.2 Multiple Location-Based Applications

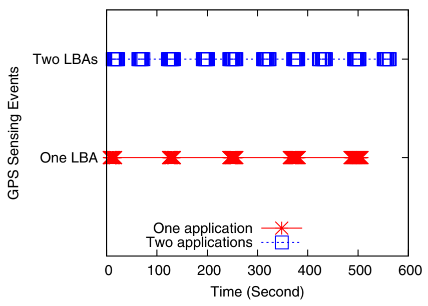
GPS power consumption becomes even more significant if multiple LBAs are running simultaneously.[2] Let us consider the following scenario. A user is initially running a social network LBA such as FaceBook on his phone and is continuously publishing his locations. After a while, he begins to drive and launches a traffic-monitoring LBA such as "Real Time Traffic". Now both LBAs run concurrently. Assuming both applications invoke GPS sensing every 2 minutes (i.e., with 2-minute invocation interval), GPS ideally needs to wake up every *two* minutes. However, if these two applications are not synchronized on GPS sensing, then GPS might need to wake up every *one* minute.

Figure 1(c) shows the impact of multiple LBAs with two scenarios. In the first scenario ('One application'), there is one LBA that runs and requests GPS sensing every 2 minutes. In the second scenario ('Two applications'), two LBAs run without synchronizing GPS sensing requests. As shown in the figure, when one LBA is running, the battery level drops to about 92% after 1 hour. However, with two LBAs running, the battery level drops to 87%.

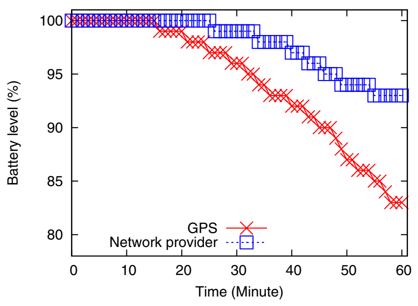
To better understand the results, we plot the GPS invocation events during the first 10 minutes for both scenarios in Figure 2(a). As shown in the figure, when the sensing events of multiple applications are not synchronized ('Two LBAs'), the GPS is indeed

[1]Though the power-APIs of Android SDK only provide coarse-grained measurement of battery levels, we use them to show macro-scale impact, which is an interesting factor in this work.

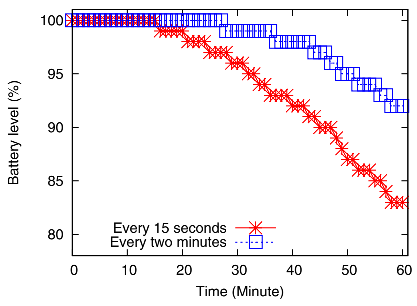
[2]Smartphones such as those based on Android or Symbian support multi-tasking. The background LBAs still triggers location sensing.



(a) Sensing events



(b) Gps vs. Net



(c) Sensing intervals

Figure 2: Energy consumption of Gps and Net

invoked for a total of 10 times rather than the desired 5 times ('One LBA'), thus causing more energy consumption than when multiple LBAs are synchronized.

2.3 Multiple Sensing Mechanisms

Today's smartphones support multiple location-sensing mechanisms (or location providers). Android, for example, supports two mechanisms: GPS and Network-based triangulation. Network-based mechanism collects information about reachable cell towers (or WiFi access points) from a mobile device and determines its location by retrieving a location database. For simplicity, in the following presentation, we use 'Gps' and 'Net' to refer to these two location-sensing mechanisms, respectively. We use the capitalized notation GPS to refer to the physical device of Global Positioning System.

These two mechanisms have different accuracy and power consumption levels. In Figure 2(b), we show the power consumption of each mechanism, as one LBA is running with a location sensing interval of 15 seconds. The Net mechanism uses GSM cell towers to determine locations as both WiFi and 3G are turned off for both experiments. Net only causes the battery level to drop to about 93% and consumes much less power than Gps does (i.e., 83%).

We also perform experiments to show the two mechanisms' accuracy. As shown in several prior studies, Gps can achieve an accuracy of as high as 10m in outdoor areas, while Net's accuracy varies depending on environments. To further understand such characteristics, we also perform experiments to calculate Net's accuracy as follows. Due to the lack of a more accurate measure, we use Gps as ground truth to measure the accuracy of the Net. We perform experiments in an urban area around Silicon Valley, California. Net accuracy is measured as the average distance between the Gps-reported location and Net-reported locations. We observe that Net achieves an accuracy of about 30 meters to 100 meters during most of the time. Although Net still provides much coarser accuracy than Gps does, such accuracy might be sufficient for many LBAs (e.g., weather information).

2.4 Sensing Intervals

For many mobile platforms including Android, applications are allowed to explicitly specify the location sensing granularity in terms of updating time interval and distance interval. Intuitively, larger time and distance intervals can help save energy. In some scenarios, particularly when the battery level is low, LBAs can cooperate by explicitly increasing location-sensing intervals of time and distance (e.g. updating every 1 minute or 20 meters rather than every 30 seconds or 10 meters). To study the impact of adapting sensing intervals, we consider two LBAs with GPS invocation intervals of 15 seconds and 2 minutes, respectively. Figure 2(c) shows the battery level. As shown in the figure, by simply enlarging the update interval from 15 seconds to 2 minutes, the application can help conserve 9% of the battery in an hour.

2.5 Problem Characterization

Figure 3 summarizes the problems identified above and additional intuitions in the energy efficiency of location sensing.

- *Static use of location sensing mechanisms:* In many cases, mobile platforms lack the dynamic selection of location sensing mechanisms. Many smartphones today support two major types of location-sensing mechanisms—Gps and Net. These sensing mechanisms have performance tradeoffs in terms of accuracy, power consumption, and availability. However, mobile platforms tend to statically use their sensing mechanisms, and this can lead to energy inefficiency in many scenarios.
- *Absence of use of power-efficient sensors to optimize location-sensing:* Depending on specific environments (e.g., inside buildings) or contexts (e.g., phones being static), certain location-sensing operations may be impossible or unnecessary to perform, and blindly requesting location sensing can lead to battery power wastage. The environment and context information, interestingly, can be obtained by using other types of sensors that are more power-efficient. Many smartphones are typically equipped with multiple sensors such as accelerometer and orientation sensors, which consume much less power than those used for location sensing. Therefore, leveraging these sensors can optimize location sensing and conserve energy.
- *Lack of cooperation among multiple LBAs:* When multiple LBAs run and request location sensing independently, they are not aware of each other, and their location-sensing operations are not coordinated. This results in *redundant* location sensing invocations and causes unnecessary energy consumption.

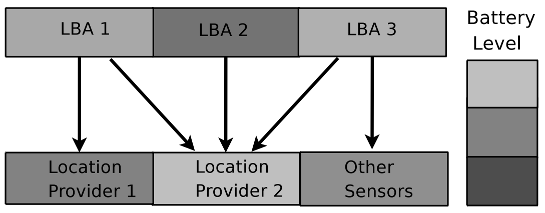
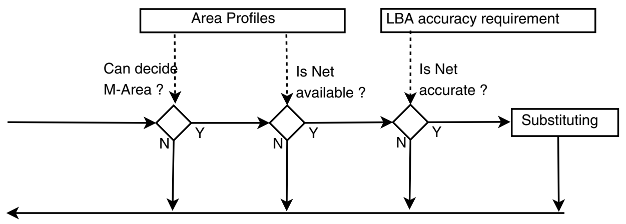
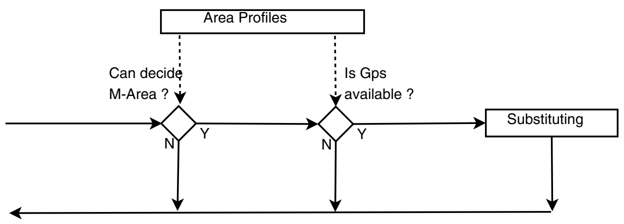


Figure 3: Problem characterization



(a) LBA requesting Gps



(b) LBA requesting Net

Figure 4: Sensing Substitution

- *Unawareness of battery level:* When the battery power level is low, users are usually willing to tolerate degradation of location-accuracy or, at least, seek such an option in favor of longer operation time. Current mobile platforms, including Android, typically lack advanced battery-aware location management to strike a balance between location sensing accuracy and operation life-time.

3. DESIGN PRINCIPLES

To overcome the limitations characterized in the previous section, we present four design principles and their integrated operations in a smartphone. Furthermore, we discuss performance tradeoffs in employing these design principles.

3.1 Sensing Substitution (SS)

Current smartphones lack the capability of selecting the most appropriate location sensing mechanism on-the-fly to strike the performance balance amongst energy consumption, availability and accuracy. LBAs are allowed to choose location-sensing mechanisms at the moment when they register their location-sensing requests to underlying systems. For instance, current Android SDK 1.5 allows an application to specify criteria indicating the applications' requirement about accuracy, power consumption, bearing (e.g. direction) and speed. Based on such criterion, the underlying framework chooses the most appropriate mechanism (e.g. Gps or Net). Thereafter, the chosen mechanism will always be invoked, irrespective the changing environment or context.

Lack of dynamic selection of location sensing mechanisms leads to energy inefficiency as well as failure in satisfying LBA requirements. For example, in certain indoor environments and dense urban areas, Gps may not be able to provide accurate location information. Similarly, the performance of Net is heavily affected by the environment. For instance, in certain urban areas, studies have shown that Net can achieve as much accuracy as Gps does. On the other hand, in rural areas with only a few cell towers available, Net shows low accuracy. Thus, with static selection of location sensing mechanisms, applications may not be able to effectively function, especially when the user moves around with LBAs running. For example, when Gps is used, applications expect to receive accurate location information all the time. However, if the environment prevents Gps from working, continuously invoking GPS apparently is wasteful in terms of battery energy. The same is true for using Net.

Our solution to these problems is a dynamic selection approach which we refer to as "Sensing Substitution (SS)". SS can choose the most appropriate location sensing mechanism on-the-fly. Specifically, SS is context-aware and can learn the characteristics of the location providers along the routes where phones move. It then performs location sensing in a more energy efficient manner by choosing the best sensing mechanism, given the context. Because typical mobile users routinely follow certain routes (e.g., commuting between offices and home) and visit familiar locations (e.g., restaurants, malls), and because these places exhibit consistent location-sensing related environment characteristics, such as GPS, signal strength and the number of APs, utilizing the environmental information can assist in choosing the most appropriate location provider.

To achieve dynamic selection of location providers, SS relies on learning environmental characteristics such as the availability and accuracy of location providers (e.g., Gps and Net). For this reason, the design of SS includes a location-sensing characteristic profiler. The profiler monitors and stores relevant information, including current locations, visit frequency, and sensing characteristics (e.g., availability, positioning accuracy) of location providers. The profiled data consists of a list of entries, and each entry corresponds to a profiled area which we refer to as *M-Area*. M-Areas represent physical areas with geographical boundaries. In particular, the locations in the same area exhibit similar location-sensing characteristics. We will detail the rationale and data structure of M-Area in Section 4.7.

Based on the profiled areas, SS dynamically decides an optimal location-sensing mechanism as follows. For ease of illustration, we consider Android platform and show the high-level operations of SS as shown in Figure 4. Specifically, let's assume that the currently registered location-sensing mechanism is Gps. When the user moves into an area where Net is available and its accuracy can fulfill the LBA's requirement, then the LBA uses Net to replace Gps. As shown in Figure 4(a), SS first attempts to decide the most appropriate M-Area. Then, it checks Net's availability and accuracy. If Net's accuracy can satisfy the requirement of the LBA, SS performs substitution. Similarly, as shown in Figure 4(b), when the current location-sensing mechanism is Net and the phone moves into areas where Net is not working, SS invokes Gps, instead of Net. Since GPS consumes more power, SS uses less frequent GPS sensing to maintain the same level of energy consumption.

The profiler can be designed to automatically obtain profiling results, including physical location and availability/accuracy of location providers. To ensure higher degree of accuracy and energy efficiency, the profiler design also includes the following mechanisms. (i) The profiler may also involve users to explicitly control the profiling process. For example, users may specify the area boundaries of the profiler. (ii) The profiler calibrates either periodically or conditionally, depending on the changes in the profile characteristics. Essentially, whenever there is need to run profiling, the process will be invoked on demand. For instance, when the user moves to a new city to join a new job, the profiler will detect that change and proactively initialize the profiling process to accommodate the environmental change. In particular, when the profiler is first initialized, it performs profiling. After that, the profiling process keeps monitoring the necessity of performing profiling again. The necessity is measured by an opportunistic verification process. Specifically, it is periodically invoked to measure the location-sensing characteristics and compare them with the information stored in the profiler database. If the comparison results in a large discrepancy value, it indicates that another profiling is needed. Also, because the periodic verifications are enabled when other location-sensing requests exist and because the verifications are piggybacked on the location-sensing results of the existing requests, they do not incur additional sensing overheads.

3.2 Sensing suppRession (SR)

Smartphone users may use the phones in various scenarios, and continuous location sensing may often not be needed. For instance, when the smartphone is in static state such as being put on a table in an office, continuous location sensing is unnecessary. It is desirable to "suppress" the sensing from energy efficiency standpoint. The design principle of Sensing suppRession (SR) is to detect phones' mobility state by using less-power-intensive sensors and to suppress unnecessary invocation of location sensing. The basic mobility-state information is whether the phone is static or moving, but it can contain more sophisticated information such as moving speed and direction.

The fundamental requirement of this design principle is to learn the mobility state (e.g., static or moving) of a phone with energy-efficient sensors. There are many existing research efforts ([35], [27], [24]) that attempt to profile users' mobility pattern. For example, SoundSense [27] uses the microphone to determine the user's logical location. In this work, we are primarily interested to use *low-power* sensors to suppress *high-power* location sensing. Specifically, we attempt to use sensors such as accelerometer and orientation sensors to profile smartphones' states. Other sensors such as camera and microphone used by the mentioned works, typically consume much more power than the low-power sensors, and thus are not considered in this work.

A challenge that arises is to ensure the correctness of mobility state detection. False positives (i.e., falsely detecting that the phone is moving while it is not) will lead to the unnecessary location sensing, while false negatives will bear more serious consequences on LBA performance for changing locations. We propose various methodologies to reducing these errors, particularly the false negatives. First, a configuration option is exposed to a user, allowing the user to manually enable/disable a suppression option. Second, the aggressiveness of suppression is automatically adjusted based on information such as the confidence levels of the learned mobility context. The confidence levels reflect the familiarity with the current mobility contexts such as commuting routes. Third, suppression is adjusted based on the application requirements. For example, if the application requires very coarse-grained location information, suppression will be invoked. Fourth, a verification mechanism is employed to verify the correctness of the detection. Briefly, location sensing is periodically invoked for verification purpose even in a suppression mode.

3.3 Sensing Piggybacking (SP)

Sensing Piggybacking (SP) is designed to improve the energy efficiency of location sensing when multiple LBAs are concurrently running. It can re-use the existing sensing registrations by piggybacking new sensing requests on existing ones, thus eliminating some location-sensing invocations. For example, let us assume that an existing LBA registers GPS location-sensing every 2 minutes. When a new LBA starts and requests Gps with the same time interval, it can simply piggyback on the existing registration requests, thus avoiding separate sensing. Reducing the number of separate sensings can help save the energy associated with sensing as the sensing hardware can go to sleep between consecutive invocations.

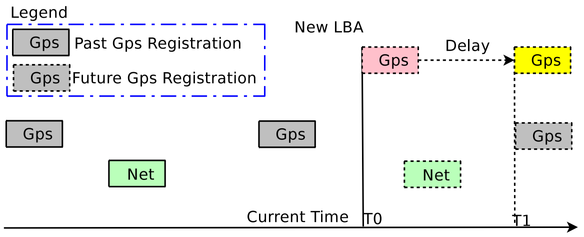


Figure 5: Sensing Piggybacking

Applications may request and register location sensing in various ways, as supported by the underlying framework or system. Android platform, for example, allows application designers to perform two types of sensing registrations. In the first type, the application statically registers a location listener to the underlying framework, and the framework periodically notifies the listener of location updates based on the specified parameters such as time interval and distance interval. This method is simple, but it relies on the underlying framework for GPS to sleep between two sensing invocations. For example, if a Gps request takes 30 seconds to perform one invocation of sensing and if the specified time interval is more than 30 seconds, then the framework can turn off the GPS and put it into sleep to conserve energy.

The other type of registration is to explicitly register/unregister GPS requests to enable hardware sleeping. For instance, if the preferred location update interval is 1 minute, the application can register/unregister the request every one minute. Assuming unregistering Gps will turn GPS off, this method does not rely on the underlying framework to support energy conservation through GPS sleeping. The downside of this method is the increased complexity of application design. It needs more involvement from the applications by requiring the application to control when to start and stop location sensing. But such involvement also gives the user/application more control over when and how to perform location sensing. For instance, the user may require different degrees of accuracy and frequency when performing locations sensing in different scenarios. Such requirements are hard to satisfy with single-time registration and not supported by current APIs and systems. We refer to the first type of registration as *One-time* Registration, while the second type as *Multi-time* Registration. For One-time Registration, depending on the mobile systems, optimizations might be applied to save energy. Whether and how to apply the techniques depends on the GPS location management of multiple registrations. Specifically, when there are multiple sensing registrations, the underlying location manager needs to accommodate multiple registrations with different sensing requirements. For example, if there are two registrations with 2-minute and 1-minute update interval, respectively, the location manager may combine these two registrations by simply considering the finer one, i.e., every 1 minute.

In this work, we focus on Multi-time Registration, as mobile platforms such as Android have already employed mechanisms to synchronize the location sensing actions for One-time Registration scenarios. For Multi-time Registration, we propose to piggyback the otherwise wasteful sensing on other sensing invocations. Specifically, we present Sensing Piggybacking (SP) with respect to the following two scenarios which involve the joining of a new LBA. We assume the joining LBA has location sensing requirement of (G_1, T_1, D_1), where G_1 is the granularity of sensing (e.g., fine (or Gps) and coarse (or Net)), T_1 is the minimum time interval and D_1 is the minimum distance interval for location updating. We also consider the cases where other applications are running when the LBA joins. We use (G_f, T_2, D_2) to denote the finest existing

Gps registration, where T_2 and D_2 are the finest sensing intervals. Similarly, we use (G_c, T_3, D_3) to denote the finest Net registration.

- *The joining LBA has Gps request:* When a new Gps registration with (T_1, D_1) comes, the currently registered requests (G_f, T_2, D_2) are retrieved. (i) If Gps requests have been registered so far with (T_2, D_2) and if $(T_1, D_1) > (T_2, D_2)$, SP does not invoke sensing in response to the new request, but wait for the next sensing of (T_2, D_2) request. Statistically, the new registration request waits, on average, for $\frac{T_2}{2}$ time. If $(T_1, D_1) < (T_2, D_2)$, the new request is registered immediately. (ii) If only Net requests are registered, then SP immediately registers the new Gps request. Figure 5 illustrates one piggybacking scenario where both Gps and Net registrations have been maintained. The joining LBA requests Gps, and the new registration is delayed to piggyback on other Gps registrations.
- *The joining LBA has Net request:* When a Net registration with (T_1, D_1) comes, the current registered requests are checked. (i) If there are Net requests registered so far and $(T_1, D_1) > (T_3, D_3)$, SP waits for the firing of next sensing. On average, the new request waits for $\frac{T_3}{2}$ time. (ii) If only Gps requests are registered, then SP checks to see whether Gps registrations satisfy its requirement. If so, SP uses the current one; otherwise, it registers a Net request.

3.4 Sensing Adaptation (SA)

There are different ways to save phone battery power and each of these focus on adapting a specific phone attribute. Such measures may include adjusting the screen light, sleep-time, or even the volume of ringtones. In the present work, we focus on energy-saving methodologies in the context of location sensing.

The key rationale behind the Sensing Adaptation (SA) principle is to adapt the location sensing frequency based on the current battery level, driven by user's general preference of longer phone-operating time over higher location accuracy. Except for running several accuracy-critical applications, users are most likely willing to trade accuracy for longer battery life. For instance, when the battery level is low and a user is running Twitter on his mobile phone and using the Gps for the location sensing, the user would generally be more willing to run the LBA with less-accuracy in return for longer phone use time.

SA is designed to respect the preference for longer operation time. When the battery level is low, SA is invoked to adapt the location sensing parameters to save energy. SA can be implemented in three ways: (i) changing the sensing frequency or interval, (ii) changing the sensing distance interval, and (iii) adjusting the aggressiveness of other design principles. The first two ways adapt the sensing intervals of location requests and registrations. For newly joining LBAs, this can be done by hooking into the registration process and directly changing the registration requests. For already-running LBAs, SA needs to remove existing registrations and add new registrations with adjusted parameter values. Specifically, when the battery level is low and the user wants to conserve energy, the sensing time and distance intervals will be increased correspondingly based on two adaptation functions f_{time} and f_{dist}, respectively. Denoting the requested time update interval, distance interval, and current battery level by T_i, D_i, and L_b, respectively, T_i and D_i can be obtained by $(T_i, D_i) = (f_{time}(L_b), f_{dist}(L_b))$. Furthermore, users may be given the opportunity to manually input the desired adaptation degrees rather than using pre-defined ones. For this, the user can be greeted by a GUI interface that solicits user input for controlling the adaptation degree.

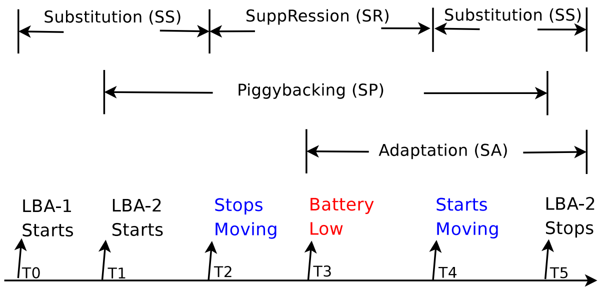


Figure 6: Integrated Operations

3.5 Integrated Operation

So far we have separately described four design principles to improve energy efficiency. The four design principles can work together for better energy saving in various scenarios. We show the integrated operation for an exemplary scenario in Figure 6. In the scenario, the user is initially in motion and the battery level is high. After the user starts LBA-1 at time T_0, SS begins to work. After the second LBA starts at T_1, SP becomes operational. When the user becomes static, SR kicks in. When the battery level becomes low, SA comes into play. As the user starts moving again, SR stops, and SS is invoked if possible.

3.6 Inherent Tradeoffs

So far we have presented four design principles and their integrated operation to save energy associated with location sensing. These design principles essentially trade accuracy and timeliness of location sensing for energy saving. Along these lines, we do note that some applications might be sensitive to the location accuracy and sensing timeliness, regardless of the battery level and power consumption. Examples of such applications include health-care and military LBAs. For these applications, all adaptation techniques have to respect application requirements. Thus, one way to safely perform the adaptation without violating the application requirement is to be application-aware and application-specific. In other words, the four design principles can be selectively adopted by application designers, when LBAs are developed. For instance, an LBA can be designed to detect the phone's mobility state and perform SR when possible. In this way, the decision about whether to apply a specific design principle and how to apply is made by the designer, and the application requirement regarding location sensing accuracy is not violated.

However, the aforementioned application-layer adoption has an associated implementation cost and is not scalable, particularly because of a plethora of existing and future applications. Realizing this, we propose another adoption model—a middleware approach—which maintains transparency of application requirements. We will elaborate on this functionality in Section 4. With the middleware approach, smartphone users are explicitly asked to decide whether to apply a design principle or not to maintain application requirement about accuracy. Practically, users may be greeted with an user-interface asking the preferred action. Users can even be given finer controls such as deciding the adaptation parameters.

The primary reason for users' involvement is to equip them with final decision-making authority. For any LBA, different users may require different levels of location accuracy. For example, given a health-care LBA, a healthy teenager may think that high location-accuracy is unnecessary, while an elder patient may think otherwise. Furthermore, even for the same application and the same user, the importance of location accuracy may also vary. For in-

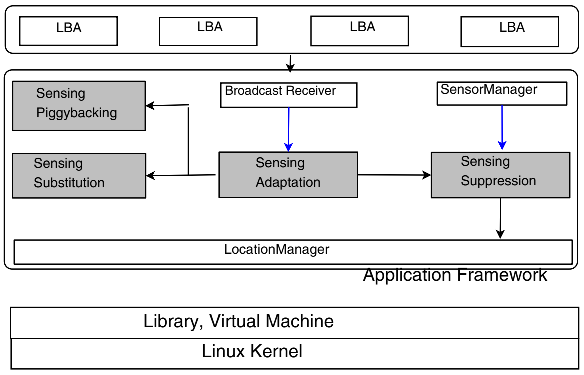


Figure 7: Software Architecture

stance, when a person is sick, the health-care LBA becomes more important. A more intelligent design is to remember or even predict the users' selection, thus reducing the users' overhead in such decision making. We see this enhancement as part of future work.

4. SOFTWARE ARCHITECTURE AND SYSTEM IMPLEMENTATION

We now present the software architecture of a system that incorporates the design principles discussed in the previous section. We explain its detailed system implementations on Android Development Phones (ADPs).

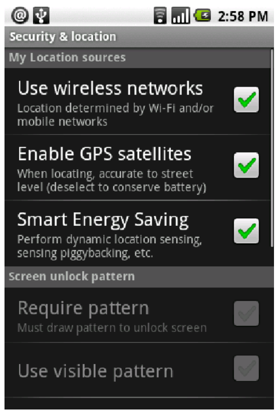
4.1 Architecture and Deployment Model

Even though our solution can potentially be applied to any mobile platform that deploys location-based services, we specifically present the architecture on Android OS for the ease of presentation and the concreteness. Such a selection is also justified by Android's open nature and increasing popularity. Note that the architecture and design principles can also be implemented on other platforms such as Symbian, Windows Mobile. As illustrated in Figure 7, the system is realized as a middleware solution, residing between applications and underlying Linux kernels. Specifically, Android platform includes Application Framework that packages many useful classes in Java. The solution is implemented inside the Android Application Framework by modifying existing classes as well as creating new classes. As illustrated in Figure 7, SA supplies the other three design principles with adaptation information, and all principles work closely with several existing components such as LocationManager and SensorManager in Android Framework.

With this deployment model, the adoption of the proposed solution on Android phones is through a new system image, which includes both new Application Framework and embedded applications. Users may choose to re-compile the source code to obtain the new system image or simply download the system image from Internet, and then update the phones with $fastboot$ utility provided in Android SDK to flash the phones.

4.2 Implementation Overview

We prototype the proposed solution on G1 Android Developer Phone (ADP1) with OS version 1.5 Cupcake. All the four design principles are implemented in Java inside Android Framework. The prototype contains Graphic User Interface (GUI) which allows a user to enable, disable and finely configure the prototype. Figure 8(a) shows the interface for enabling/disabling the adaptive location-sensing framework. The interface is implemented inside



Security & location

My Location sources

Use wireless networks ✓
Location determined by Wi-Fi and/or mobile networks

Enable GPS satellites ✓
When locating, accurate to street level (deselect to conserve battery)

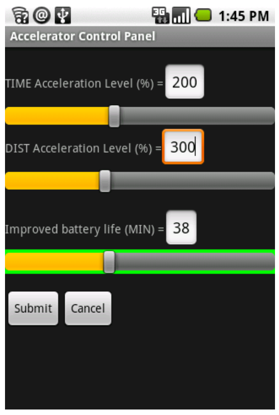
Smart Energy Saving ✓
Perform dynamic location sensing, sensing piggybacking, etc.

Screen unlock pattern

Require pattern
Must draw pattern to unlock screen

Use visible pattern

(a) Enable/Disable interface



Accelerator Control Panel

TIME Acceleration Level (%) = 200

DIST Acceleration Level (%) = 300

Improved battery life (MIN) = 38

(b) Configuration

Figure 8: Two prototype interfaces

the default "Security & location" setting menu of G1 phones. The new menu item, called "Smart Energy Saving", has been added. Figure 8(b) shows the configuration interface for the desired SA degree in time (TIME) and distance (DIST). The interface also shows the expected battery saving time with the current LBA requests and SA degrees. Briefly, the prototype first calculates the expected number of saved GPS invocations with SA. Then, assuming a typical operation of making a phone call and its associated power level, the prototype estimates the improved battery life from the saved energy.

With current Android APIs, GPS is invoked through a major function call, requestLocationUpdates(), which takes at least four input parameters: LocationProvider (i.e., Gps or Net), reporting frequencies in term of time and distance, and an PendingIntent or LocationListener. Our prototype mainly captures this function call and embeds intelligence inside the function as well as other relevant functions. Specifically, SS may substitute another LocationProvider for the requested one, SR may freeze the further execution of the function when necessary, SP may piggyback the current call on existing registrations and freeze further execution of the function call, and SA may adjust reporting frequency based on battery level or user preference.

We illustrate the high-level operations of the four design principles, as well as the major data structures, information flows and the function calls in Figure 9. SP is hooked into the location-sensing registration function, requestLocationUpdate(). Whenever the framework detects a new location sensing registration, SP records the registrations into Registration State and obtains the piggybacking time by checking this state. SA and SR are implemented in separate threads, and their invocations are triggered by battery level changes and timers. SA registers for battery change updates with Broadcast Receiver. SR periodically checks the user's mobility state for the purpose of registering or unregistering sensor readings. SS reads the state of Area Profiles, periodically determines the current M-Area and selects the most appropriate location provider.

4.3 Sensing Substitution (SS)

SS aims to determine the most appropriate location provider on-the-fly. Specifically, when Net is available and currently Gps is being used, SS may decide to use Net to replace Gps for location sensing. The decision of whether to perform SS is controlled by the user with a pop-up dialog informing the Net accuracy and asking for actions. Similarly, when Net is being used and becomes un-

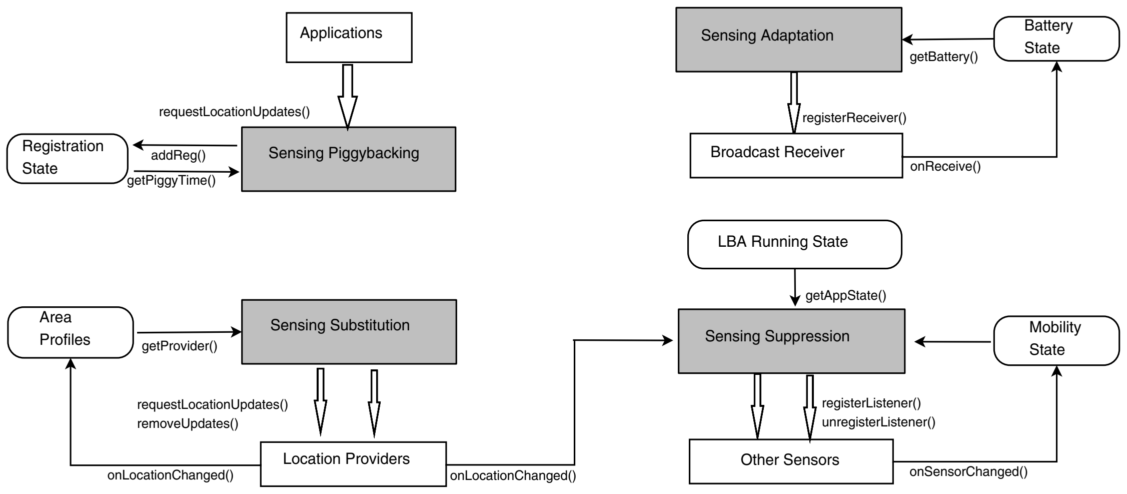


Figure 9: Prototype on G1 Android Phone

available, SS may turn to Gps. Since Gps consumes more power, Gps is requested with reduced location update frequency to maintain the same level of power consumption as Net.

In order to perform dynamic selection of location providers and accommodate the mobility of the phone, SS needs to be invoked periodically. The Handler class in Android SDK is used to implement a separate thread inside the LocationManager Class for this purpose. As shown in Figure 10(a) line 1-2, whenever the task is invoked, SS attempts to determine the most appropriate M-Area where the phone resides. After finding such an M-Area, SS then determines the available location provider with getProvider() call. Specifically, the prototype captures the registration of the provider, and records the registered provider, the listener, the registered time update interval and the distance interval. This information is used for new registrations (with the same interval values and the same listener). If the available provider is Net, the requested provider is Gps, and Net can satisfy LBA's requirement (Lines 3-4), then SS unregisters the current provider and registers the available one (Lines 5-6). If the available provider is Gps and the requested provider is Net, then SS unregisters Net and registers Gps appropriately (Lines 7-10).

Area Profiles are initialized with training data and updated by monitoring the sensed environmental characteristics when running LBAs. A separate profiler process keeps running when the user carries the phone and moves around. The process records GPS locations, network-based locations, and the time. The profiled data are stored in files, and then further extracted into M-Areas. Area Profiles are read into the memory whenever the instance of LocationManager is created. Profiled locations are organized as a list of M-Areas, each of which has the same characteristics of the two location providers. In other words, locations inside the same area has the same physical characteristics of Gps and Net (i.e., availability, accuracy, precision). The structures and operations of the M-Areas are presented in Section 4.7.

To reduce false negatives of area determination, the prototype uses both current location and mobility properties to decide the current M-Area. The mobility properties include current moving speed and direction. For each invocation of SS, if the current location is inside the same M-Area and if the moving direction and speed suggest that the user will be in this area for a while, then the M-Area is determined to be a candidate M-Area. If multiple candidate M-Areas exist, the most appropriate one is chosen based on a set of criteria including visiting frequency, most recent visit time and area size.

4.4 Sensing suppRession (SR)

SR monitors user's context with less-energy-intensive accelerometer and orientation sensors. When the user is in a static state, the prototype saves energy by suppressing the new location sensing. When LBAs are running and the location services are registered, a thread is created to monitor and identify whether the phone is in static or moving state. If the current state is static, then the current location sensing registration is removed; if the state is non-static, SR re-registers the previous sensing request, as shown in Figure 10(b) (Lines 8-12). The thread is invoked periodically (e.g., every 1 minute) and the reading for each invocation lasts for several seconds. The reason for doing so rather than continuous monitoring is that otherwise the continuous sensor reading and computation become expensive in terms of energy consumption. However, the disadvantage of periodic reading as compared to persistent reading is that short-term static states might not be detected. Thus, periodic invocations work better for long-term static states.

Inside the thread, the prototype reads accelerometer and orientation sensors to detect mobility (Lines 1-7). The basic rationale is that whenever there is change of the state, these sensors will see a large variation in readings. As the motion sensors may report updates quite frequently (e.g., 20 times per second), the user state is detected to be static only when both microscopic state and macroscopic state are static. Microscopic state is determined by finer successive sensor readings, while macroscopic state is determined by coarser reading changes (e.g., 2 second). We notice that both microscopic and macroscopic detections are necessary since there are scenarios where slow changes (i.e., macroscopic) happen, but such changes cannot be detected by microscopic checking. For instance, the most infrequent sensor reading rate (i.e., by supplying SENSOR_DELAY_NORMAL in the registerListener() call) on Android platform is about 10-20 times per second, as observed in our experiments. When the state change is slow, simply comparing two continuous readings does not allow detection of the state change. Furthermore, to reduce the false negative (i.e., mobility being detected as being static) probability, our prototype takes one step further. If no mobility is detected, then the user state is considered to be *transiently* static, and this state has to sustain for certain period before inferring that the state is static.

(a) Sensing Substitution (SS)

```
Variables
    $provider$: Requested location provider
    $Set_{Area}$: Profiled M-Areas
    $Area_{prev}$: Previous M-Area
    $Area_{cur}$: Current M-Area
1   Obtain most recently sensed location
2   Determine $Area_{cur}$ based on $Set_{Area}$
3   If provider == Gps
4       If $Area_{cur}$'s Net can satisfy LBA
5           Unregister the corresponding Gps
6           Register a new Net
7   Else // provider == Net
8       If Gps is not available AND Net is available
9           Unregister the corresponding Net
10          Register a new Gps
11      End
12  End
```

(b) Sensing suppRession (SR)

```
Variables
    $State_{cur}$: Current motion state (static or moving)
    $State_{prev}$: Current motion state (static or moving)
    $State_{micro}$: Micro transient motion state
    $State_{macro}$: Macro transient motion state
    $State_{Gps,Reg}$: Currently requested Gps state
1   Obtain motion sensor readings
2   Determine $State_{micro}$ and $State_{macro}$
3   If $State_{micro}$ and $State_{macro}$ == static
4       $State_{cur}$ = static
5   Else
6       $State_{cur}$ = moving
7   End
8   If $State_{prev}$ and $State_{cur}$ == static
9       Unregister the corresponding Gps
10  Else
11      Register a new Gps based on $State_{Gps,Reg}$
12  End
13  $State_{prev}$ = $State_{cur}$
```

(c) Sensing Piggybacking (SP)

```
Variables
    $State_{Gps}$: Gps registration state
    $State_{Net}$: Net registration state
    $time$: Requested location sensing frequency
    $dist$: Requested location sensing distance
1   Received requestLocationUpdate(provider, time, dist,...)
2   Store information about provider, time, distance
3   Check validity of $State_{Gps}$ and $State_{Net}$
4   If provider == Gps
5       Compare $State_{Gps}$ to $time$ and $dist$
6       If $State_{Gps}$ allows piggybacking
7           Delays the registration to enable piggybacking
8       End
9   Else // provider == Net
10      Compare $State_{Net}$ to $time$ and $dist$
11      If $State_{Net}$ allows piggybacking
12          Delays the registration to enable piggybacking
13      Else
14          Compare $State_{Gps}$ to $time$ and $dist$
15          If $State_{Gps}$ allows piggybacking
16              Delays the registration to enable piggybacking
17          End
18      End
19  End
```

(d) Sensing Adaptation (SA)

```
Variables
    $Bat_{cur}$: Current battery level
    $Bat_{thr}$: Battery level threshold to trigger SA
    $f_{time}$: Function to adjust time parameter
    $f_{dist}$: Function to adjust distance parameter
1   If provider == Gps AND $Bat_{cur} < Bat_{thr}$
2       $time = time * f_{time}$
3       $dist = dist * f_{dist}$
4       Obtain user preference
5       If SA is allowed
6           Unregister the current Gps
7           Register a new Gps with $time$ and $dist$
8       End
9   End
```

Figure 10: Pseudo-code : (a) Sensing Substitution, (b) Sensing suppRession, (c) Sensing Piggybacking, and (d) Sensing Adaption

4.5 Sensing Piggybacking (SP)

LBAs request location sensing through a registration function call of requestLocationUpdates(), which takes several parameters including the location provider, time interval and distance interval. The essential idea of SP is to force the incoming registration request to synchronize with existing location-sensing registrations. SP predicts the next sensing registration request from currently running LBAs and asks the incoming LBA to delay the registration. SP learns and maintains the location-sensing registration history, stored in two array lists—one for Gps and the other for Net. Each element of the lists contains three values: registration time, time interval and distance interval.

SP first needs to determine the validity of the maintained states. Since the prediction of future registrations is based on historically maintained states, the states can be outdated because the requesting LBAs might stop running or change the registration. A state is valid only when the most recent registration time recorded is no more than certain time earlier than the current time. The default threshold value for determining the validity is 200% of the time interval. In other words, if the predicted registration which is supposed to occur after T time does not come in $2T$ time, then the state is invalid, indicating either the application changed the registration pattern or the application has stopped running.

As shown in Figure 10(c), SP is hooked into the registerLocationUpdate() function in the LocationManager Class of Android Framework. When receiving the above function call, SP checks the validity of the maintained registration state (Lines 1-2). If the state is invalid, the request is passed through and is added to the registration history by the addReg() function. If the state is valid, then SP determines the piggybacking time (i.e., the delay) with getPiggyTime() function (Lines 4-16). The current prototype determines the piggybacking time in six different usage scenarios, based on the currently maintained registration-state types as well as the incoming new registration type. In the following, we will discuss each of the six scenarios below. For simplicity, we use the notation of {(Maintained states), Incoming state} to denote each scenario. We use (t, T_0, D_0) to denote the incoming request, where t is the time, T_0 is the requested update time interval, and D_0 is the requested distance interval. For the maintained states, we use (Gps, T_1, D_1) to denote the Gps state with the finest time interval being T_1 and finest distance interval being D_1. We use (Net, T_2, D_2) to denote the Net state with the finest time interval being T_2 and the finest distance interval being D_2.

- {(Gps), Gps}: The prototype checks whether the (Gps, T_1, D_1) state is valid. If so, then it compares (T_1, D_1) to (T_0, D_0). If $T_1 < T_0$ and $D_1 < D_0$, then piggybacking is enabled, and the piggybacking time is calculated.
- {(Gps), Net}: As Net typically has coarser location information than Gps, the operations are similar to the ({Gps},Gps) scenario, but the comparison is between (T_2, D_2) and (T_0, D_0).
- {(Net), Net}: Similar to {(Gps), Gps} case by replacing Gps with Net.
- {(Net), Gps}: Since Gps is typically finer than Net, the request cannot piggyback on existing Net registrations. The new registration is passed through immediately.
- {(Gps, Net), Gps}: Similar to {(Gps), Gps}.
- {(Gps,Net), Net}: The prototype firstly checks the Net state, which is similar to that of {(Net), Net}. If not possible to piggyback, then it checks the Gps state, which is similar to {(Gps), Net} scenario.

4.6 Sensing Adaptation (SA)

The operations of SA are shown in Figure 10(d). SA is invoked when Gps is used and the phone's battery level is low. When the battery level is below a user-specified threshold (e.g. 20%), SA determines the preferred adaptation degree for both time and distance intervals of Gps registrations (Lines 1-3). SA also asks a user's intention on whether to perform SA or not. If adaptation is enabled, the user can choose the preferred adaptation degrees. The prototype then functions based on the decision and values provided by the user (Lines 4-7).

SA learns the current battery level information with Android power-APIs. It registers a BroadcastReceiver to handle the Intent of ACTION_BATTERY_CHANGED. The function used to register is registerReceiver(), which is a method of the Context class in Android SDK. Because of this, the prototype piggybacks the registration on an existing application in Android platform: SecuritySettings, which is extended from Context. Specifically, in the onCreate() method, SecuritySettings registers the BroadcastReceiver and an IntentFilter. Whenever the battery level changes, the receiver is notified and appropriate information is recorded.

Applications running on Android platforms are essentially independent in the sense that each application has a private directory and each application runs in a separate Java virtual machine. For communication between activities within a single application and between different applications, Android SDK provides several mechanisms including shared preferences, content providers and database. Unfortunately, none of these mechanisms works neatly for the communication between application layer and framework layer. Our prototype uses files (under /proc) as the intermediate media for these two layers to communicate. Specifically, applications and frameworks both access the same files under the data directory of the system, which can be obtained by getDataDirectory() call. There are various types of data that need to be shared. For simplicity, we use a separate file for each type of data.

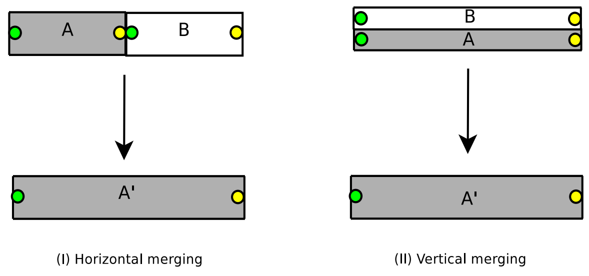


Figure 11: Merging operations

4.7 Mobility Profiling

Both SR and SS use the M-Area structure to organize the locations. Each M-Area contains three types of properties. The first type is boundary property. Each M-Area is a rectangle area bounded by a starting point, an ending point, and a width value. The points are specified with latitude and longitude coordinates. The second type is usage property. M-Areas also contain the number of visits and the last visit time (i.e., LastTime). The third type is provider property. M-Areas also maintain the sensing characteristics, such as availability and accuracy, of Gps and Net.

The construction of M-Area consists of the two steps. Initially, each M-Area is constructed as a rectangle, based on the two neighboring location readings from the mobility traces. Later, M-Area can be merged and replaced. Two M-Areas can merge into one when they have compatible boundary-related properties and same provider-related properties. There are two types of merging scenarios: Horizontal and Vertical. Horizontal merging occurs when the starting point of one M-Area is adjacent to the ending point of the other M-Area or the starting point is inside of the other M-Area. Vertical merging occurs when the two neighboring areas have adjacent starting-points and ending points. When conditions are met, merging is performed and the properties of the new M-Area are updated. The two merging operations are illustrated in Figure 11. Specifically, the starting/ending points and the width are updated to represent the new M-Area. The LastTime is updated to the more recent LastTime of the previous two M-Areas, and the Frequency is set to be the average of the two Frequency values.

One important design issue is the size of the profiled M-Areas. Since the size impacts the efficiency of processing speed and suppression effectiveness, there is a performance tradeoff with regard to the number of M-Areas maintained. Specifically, increasing the size results in higher suppression probability. However, it also occupies more storage space and inflates the processing time. We propose to adjust this size based on the hardware capability of the smartphones. If smartphones can afford to provide more space and process the operations sufficiently fast, maintaining in general more M-Areas benefits Sensing Substitution. In addition, replacement mechanism that only maintain higher-utility M-Areas can be easily applied to alleviate the storage concern and maintain scalability. The prioritization is enforced in the following order: Frequency, LastTime, and Area size.

5. PERFORMANCE EVALUATION

We evaluated the effectiveness of our prototype. We first model the energy saving when each of the four design principles is applied. Then, we show the effectiveness of each design principle by considering a typical scenario where each design principle works. Finally, we evaluate the integrated operations of the prototype and show its aggregated saving.

5.1 Analysis

We analyze energy-saving benefits coming from reduced GPS invocations. For simplicity, we assume that LBAs request r number of GPS invocations per hour by default and that the energy cost of per-GPS invocation is E_g. Similarly, we use E_n to denote the energy cost of per-Net invocation, and use E_o to denote the energy cost of running each design principle in an hour. The energy-saving benefits are expressed in the reduced number of GPS invocations and, for a more concrete understanding, they are translated into the extended battery life when other tasks are performed. Specifically, we choose the representative task of making phone calls, and the power consumption level of the task is denoted by P_c. We use N_g to denote the number of GPS invocations reduced by each design principle in an hour, and use T_c to denote the extended operation time, when making calls.

- *Sensing Substitution* Assuming p_u percentage of GPS invocations are replaced by Net invocations:

$$N_g = rp_u \text{ , and } T_c = \frac{rp_u(E_g - E_n) - E_o}{P_c}$$

- *Sensing Suppression* Assuming p_s percentage of GPS invocations are suppressed, we have,

$$N_g = rp_s \text{ , and } T_c = \frac{rp_s E_g - E_o}{P_c}$$

- *Sensing Piggybacking* Assuming p_g percentage of otherwise-independent GPS invocations can piggyback on other invocations. We have,

$$N_g = rp_g \text{ , and } T_c = \frac{rp_g E_g - E_o}{P_c}$$

- *Sensing Adaptation* Assuming the time-interval adaptation degree is $d_t(\%)$, we have,

$$N_g = r(1 - \frac{100}{d_t}) \text{ , and} \qquad (1)$$

$$T_c = \frac{r(d_t - 100)E_g - E_o}{d_t P_c} \qquad (2)$$

We now show exemplary values based on experiments and assumptions mentioned above. Our measurements show that each GPS invocation costs about 9 Joules (i.e., 150 mA×3.7V×15 seconds). The average energy overhead of running the design principles on our smartphone prototype is negligible (i.e., a few mW), compared to GPS sensing power, so for simplicity in the following presentation we ignore this cost. Next, though the power level of making phone calls varies on different phones and conversation scenarios, we choose an averaged value of 600 mW, measured in an ADP. For an LBA requesting Gps every half minute, we have $r = 120$. Thus, with SA and $d_t = 300$, the energy saved per hour is $E_s = 120 \times \frac{2}{3} \times 9 = 720$ (J) with Equation 1. We have $T_c = \frac{720}{0.6} = 1200$ (seconds) with Equation 2. In other words, with SA, for every hour of running an LBA, about 20 minutes of phone-call time can be saved.

5.2 Sensing Substitution (SS)

We evaluate SS by asking a person, who carries a smartphone, to walk along a route. The route is manually split into four areas with pre-defined different characteristics of Gps and Net. For ease of evaluation, we pre-set the characteristics of the four areas as follows. In Area 1, both Gps and Net are available, with Net being much less accurate than Gps. In Area 2, both Gps and Net are available, with Net having accuracy similar to Gps. In Area 3, only Gps is available. In Area 4, only Net is available. We run an LBA requesting Gps updates every 5 seconds. The substitution checking thread uses an interval of 15 seconds. We then record the events of SS and location updates in Figure 12. As shown in the figure, in Area 1, Gps is used to perform location updating. As the user moves into Area 2, Gps is replaced by Net, since Net has accuracy similar to Gps's. Then, as the user moves into Area 3, the component substitutes Gps for Net, since only Gps is available. Finally, when in Area 4, Net again replaces Gps to perform location sensing.

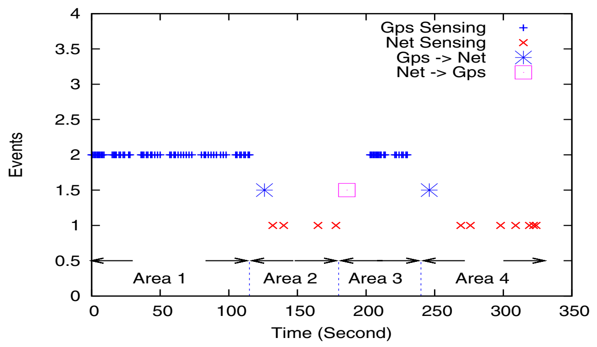
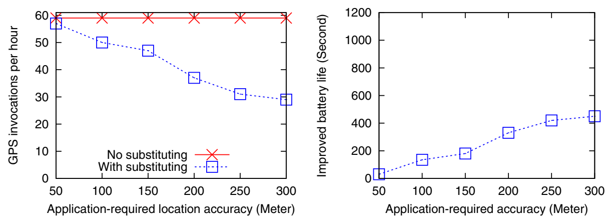


Figure 12: Sensing Substitution (Events)



(a) GPS invocation times (b) Improved call time

Figure 13: Sensing Substitution

Figure 13 shows the recorded GPS invocation times and improved battery life in our experiments. Because SS replaces Gps with Net only when Net provides the desired location sensing accuracy, we vary the location accuracy required by LBAs from 50 meters to 300 meters. We set the Net accuracy according to the traces collected from a particular user who commutes along a walking route. The user lives and works in Bay Area of California, USA. As shown in the figures, with coarser requirements, the number of GPS invocations decrease. While 50-meter accuracy requirement does not see much improvement, 300-meter requirement effectively reduces the number of invocations by about 50%. Correspondingly, improved call-making time increases as accuracy requirements become coarser.

5.3 Sensing suppRession (SR)

SR is invoked only when the phone is in static state. We consider a scenario where an LBA is running and user's mobility states vary between being static and moving. The State-Checking thread is invoked every 1 minute. Figure 14 shows the various events such as thread invocations, starting and stopping of the application, and the user's mobility. As shown in the figure, the phone is initially static. After LBA starts, accelerometer is invoked. Since the phone is not moving, the State-Checking thread puts the phone into a suppression mode, after a while. Once the phone starts moving, the

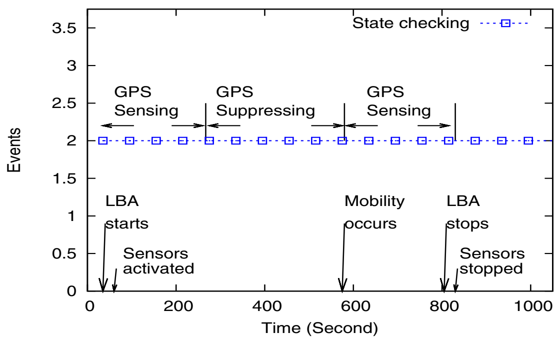
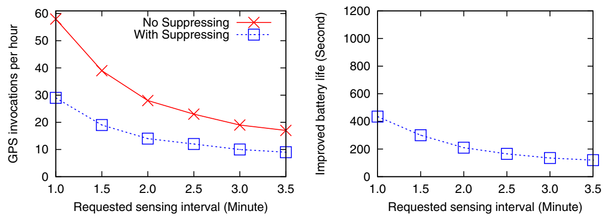
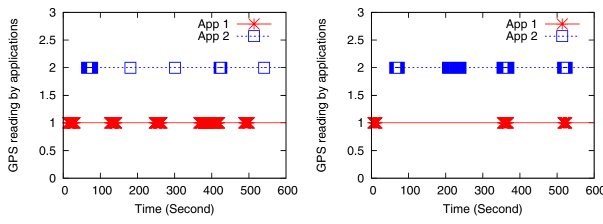


Figure 14: Sensing Suppression (Events)



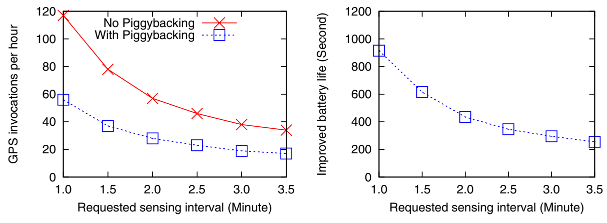
(a) GPS invocation times (b) Improved call time

Figure 15: Sensing Suppression



(a) No piggybacking (b) With piggybacking

Figure 16: Sensing Piggybacking (Events)



(a) GPS invocation times (b) Improved call time

Figure 17: Sensing Piggybacking

thread detects the mobility and takes the phone out of the suppression mode. Finally, after the application stops, the accelerometer is unregistered.

Figure 15 shows the recorded GPS usage with varying GPS intervals requested by LBAs. Note that we put the phone into static state for half of the entire period (i.e., 30 minutes in a hour). We plot the improved battery life when making calls in Figure 15(b). SR effectively suppresses about half of the GPS sensing, which improves the battery life when making calls by up to 400 seconds.

5.4 Sensing Piggybacking (SP)

SP can help reduce the number of GPS invocations by piggybacking GPS sensing requests from multiple LBAs. We run two LBAs concurrently but with different starting time. Both applications request GPS sensing every 2 minutes. Figures 16(a) and (b) show the sensing updates received by the two applications. We see that when SP is not working, GPS is invoked for a total of 10 times in 10 minutes, while when SP is used, GPS is only invoked 6 times. Note that in Figure 16(b) the last two GPS invocations notify both applications about the new location updates.

Figures 17(a) and (b) show the GPS invocation times and improved battery life time during experiments. We vary the GPS requesting frequencies of LBAs from every 1 minute to every 3.5 minutes. With SP, the number of GPS invocations is reduced by half, and correspondingly, call-making time is improved by up to 910 seconds.

5.5 Sensing Adaptation (SA)

We evaluate SA by considering two scenarios with different battery levels. We set the adaptation degree for time interval to $d_t = 200$. The Gps location updates received by the applications are plotted in Figure 18. As shown in the figure, the LBA requests the location sensing updates every 1 minute, and with this component, the update interval is increased to every 2 minutes.

Figure 19(a) shows the GPS invocation times at low battery level. We vary the adaptation degree from 100% (i.e., without SA) to 350%. We observe that no-adapting results in about 60 times of GPS sensing, as requested by the applications. The higher adaptation degree results in the less number of GPS invocations, and specifically, with $d_t = 350$, GPS is only invoked 15 times. As shown in Figure 19(b), SA helps improve call-making time by up to 650 seconds per hour.

5.6 Integrated Results

We also evaluate the effectiveness of integration operations in energy saving. We run two LBAs concurrently at low battery level to enable corresponding components of SA and SP. The adaptation degree is set to be 200%. The two LBAs request GPS sensing with same frequency of every 30 seconds, but start with 15-second difference. We use the traces collected from a particular user who commutes along a route in the Silicon Valley, California. We also vary the user states to invoke the SR. Specifically, we vary the time length of the user being static. The GPS usages are plotted in Figures 20. We see that by default GPS is invoked about 240 times per hour. By invoking all the four components, GPS invocations can be reduced to about one-fifth even when the phone is constantly moving (i.e., SR is not invoked). Even more significant reduction on the number of GPS invocations can be achieved when the phone is put in longer static state (i.e., up to 98%). Also, with our prototype, improved call-making time is more than 2,700 seconds for all considered scenarios.

Even though the above evaluation results show the savings in terms of GPS invocation times and predicted operation time, it is also necessary to show the improved battery life since operating the design components (e.g. computation) also consumes energy. We show the improved battery life with our prototype with a scenario where two LBAs are running, each requesting GPS every 1 minute. The two LBAs start with 30-second difference. To show the effect of SA, we invoke the component for all battery levels, i.e., the battery level threshold is set to 100%. A user carries the phone and walks along the commuting route with different moving/static

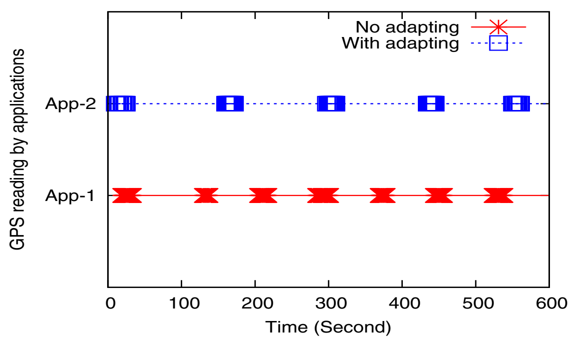
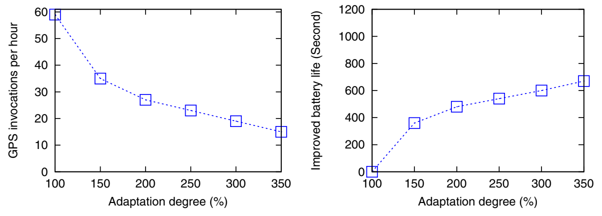


Figure 18: Sensing Adaptation (Events)



(a) GPS invocation times (b) Improved call time

Figure 19: Sensing Adaptation

time. As shown in Figure 21(a), our prototype can improve the battery life from 81% to 92% after an hour.

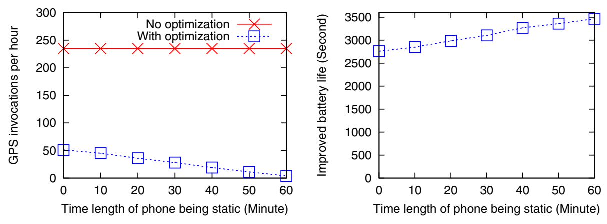
We also use the LBA of Real Time Traffic to measure the effectiveness of our prototype. Using the same configurations as described in Section 2. The user carrying the phone follows the commuting route and spends half time walking and half time being static. The instantaneous battery level results are shown in Figure 21(b). We observe that our prototype can improve the battery life from 79% to 88% after an hour—up to 75% improvement.

5.7 Profiling Results

To evaluate the location-sensing characteristic profiler in SS, we ask three users to carry phones with our prototype installed, and we continuously obtain their location information on a daily basis for 3 weeks. The users live and work in the Bay Area of California, U.S.A. We show part of a M-Area map (defined in Section 3.1) both before and after the merging operations in Figure 22. We see that there are totally 5 M-Areas before merging, and these areas result in 3 new M-Areas after merging.

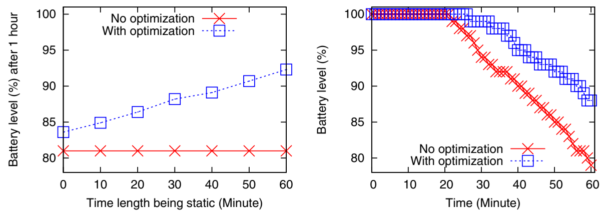
The profiling process has several pre-defined parameters for extracting and merging M-Areas. One of the parameters is the initial width of extracted M-Areas. The setting of this value particularly affects the merging operations since only adjacent M-Areas can be merged. A larger width value encourages merging and leads to smaller M-Area sets, while the accuracy of the M-Area extraction might be compromised since all the locations inside the same M-Area are supposed to have the same characteristics. As shown in Figure 23(a), setting the width to 10 meters rather than 30 meters increases the resulting M-Area set by more than 70%.

We also measure the Net accuracy with profiled data of 3 users, and we show CDF in Figure 23(b). We see that for the locations visited, more than 70% of locations have a Net-accuracy finer than 100 meters. This suggests that for an LBA requiring location accuracy coarser than 100 meters, SS can be invoked most of time.



(a) GPS invocation times (b) Improved call time

Figure 20: Integrated Results



(a) Custom-built LBA (b) Real Time Traffic

Figure 21: Battery Level of Integrated Results

6. RELATED WORK

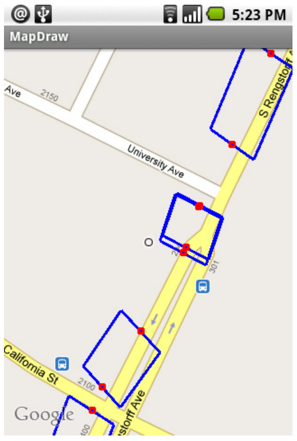
Recently, the use of smartphones (e.g., iPhones, Windows Mobile, Symbian and Android) becomes pervasive. These smartphones are equipped with location sensing capability to enable LBAs. As mentioned before, to the best of our knowledge, existing mobile platforms including Android do not employ techniques similar to our designs to improve energy efficiency of LBAs, although application developers partially adopt similar concepts (e.g., increasing sensing interval).

As users are increasingly adopting a wide variety of LBAs on smartphones [2, 4, 10], several research efforts have been made to the design and use of LBAs. For example, work in [22, 23, 29, 36] presents traffic monitoring designs. BikeNet [18] describes an extensible mobile sensing system for cyclist experience mapping. StarTrack [12] extracts users' sequences of locations in the form of tracks so that other applications can take advantage of the information. Other works aim to improve the performance of positioning mechanisms such as GPS. For instance, Skyhook [8] improves the response time of positioning by combining the unique benefits of GPS, cell-tower triangulation and WiFi positioning.

Since typical smartphones are equipped with multiple types of sensors, applications that take advantage of these sensors are booming, and many existing works attempt to detect and extract users' states and context based on the readings from these sensors [13, 15, 21, 25]. Many approaches have been proposed to combine the information obtained from sensors including Bluetooth, accelerometer, audio, camera and GPS [16, 20, 24, 28, 35].

Realizing the battery shortage problem of mobile systems, various solutions have been proposed to save energy [11, 32]. The challenges and general approaches for energy management on handheld devices are described in [34]. Turducken [33] presents a hierarchical power management architecture for mobile systems.

To address the power consumption problem of GPS sensing, some works attempt to trade accuracy of GPS for energy [14, 17, 19, 26, 30]. Work [14] proposes to use accelerometers to sense movements for saving energy, and the mechanism bears similarity with Sensing suppRession. ENloc [17] addresses the optimal location sensing


(a) Before merging

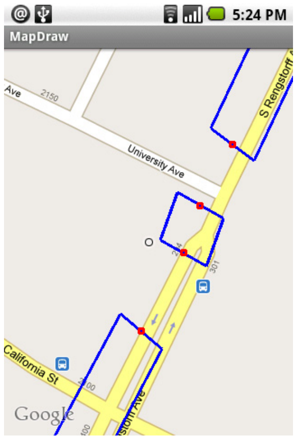
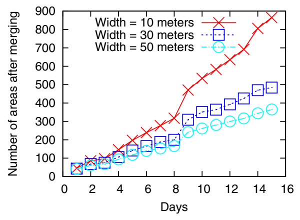

(b) After merging

Figure 22: Merging Operations


(a) Number of areas

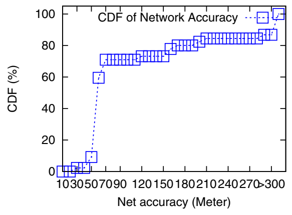

(b) Net Accuracy

Figure 23: Profiling Results

problem given an energy budget. Micro-Blog [19] proposes to balance the competing goals of accurate location coordinates and long battery life by infrequently using more accurate, but power-hungry localization services such as WiFi to offset the error introduced by less accurate, but more power-efficient localization services (e.g., GSM localization). These two works share certain features with Sensing Adaptation. In addition, work [31] selects between two data services driven by history, which bears the idea of substitution. Parallel to our work, a-Loc [26] presents a method to dynamically trade-off location accuracy and energy, using probabilistic models of user location and sensor errors. A-Loc chooses the most energy efficient location sensor to meet application accuracy requirements. The accuracy requirements may be specified explicitly by the applications, or automatically determined by a-Loc for important classes of applications such as mobile search and social networking. Work [30] proposes to trades-off location accuracy for reduced energy use by using a combination of spatio-temporal location history, user activity, and celltower-RSS blacklisting to selectively activate GPS only when necessary to reduce position uncertainty. The work also proposes sharing position readings among nearby devices using Bluetooth in order to further reduce GPS activation. Though sharing certain degree of similarity with the above approaches, our work differs from them in the exact usage scenarios and detailed designs. In particular, compared to these approaches, our work provides a comprehensive energy-saving solution tailored for smartphones running multiple LBAs, and it has been implemented as a middleware on an Android smartphone.

7. CONCLUSION

In this paper, we consider the problem of energy efficient location-sensing on smartphones. We first identify four critical factors that affect energy efficiency of location-sensing with GPS through extensive experiments. These factors are static use of location sensing mechanisms, absence of use of power-efficient sensors to optimize location-sensing, lack of sensing cooperation among multiple LBAs, and unawareness of battery level. We then present an adaptive location-sensing framework that includes the design principles of Sensing suppRession, Sensing Substitution, Sensing Piggybacking, and Sensing Adaptation to reduce the usage of GPS in various scenarios. We implement these design principles as a middleware on Android-based smartphones by modifying the Application Framework. Our evaluation results on the implementation show that our prototype can significantly reduce the GPS usage by up to 98% and improve battery life by up to 75%.

Acknowledgement

We would like to thank Raghupathy Sivakumar at the Georgia Institute of Technology for his valuable comments on this work. We also would like to thank our shepherd Doug Terry for his suggestions and kind guidance. Finally, we would like to thank the anonymous reviewers and PC members for their encouraging and insightful feedback.

8. REFERENCES

[1] Android market. http://www.android.com/market.
[2] Facebook. http://www.facebook.com/.
[3] Foursquare. http://www.foursquare.com/.
[4] Myspace. http://www.myspace.com/.
[5] Open handset alliance. http://www.openhandsetalliance.com/.
[6] Opentable. http://www.opentable.com/.
[7] Real time traffic. http://monthorin.net/tiki-index.php.
[8] Skyhook. http://www.skyhook.com/.
[9] Twidroid. http://www.twidroid.com/.
[10] Twitter. http://www.twitter.com/.
[11] M. Anand, E. B. Nightingale, and J. Flinn. Self-tuning wireless network power management. In *Proceedings of ACM MobiCom '03*, San Diego, CA, USA, 2003.
[12] G. Ananthanarayanan, M. Haridasan, I. Mohomed, D. Terry, and C. A. Thekkath. Startrack: a framework for enabling track-based applications. In *Proceedings of ACM MobiSys '09*, Kraków, Poland.
[13] M. Azizyan and R. R. Choudhury. Surroundsense: mobile phone localization using ambient sound and light. *SIGMOBILE Mob. Comput. Commun. Rev.*, 13(1):69–72, 2009.
[14] F. Ben Abdesslem, A. Phillips, and T. Henderson. Less is more: energy-efficient mobile sensing with senseless. In *Proceedings of ACM MobiHeld '09*, Barcelona, Spain, 2009.
[15] T. Brezmes, J.-L. Gorricho, and J. Cotrina. Activity recognition from accelerometer data on a mobile phone. In *Proceedings of IWANN '09*, Salamanca, Spain, 2009.
[16] A. T. Campbell, S. B. Eisenman, K. Fodor, N. D. Lane, H. Lu, E. Miluzzo, M. Musolesi, R. A. Peterson, and X. Zheng. Transforming the social networking experience with sensing presence from mobile phones. In *Proceedings of ACM SenSys '08*, Raleigh, NC, USA, 2008.
[17] I. Constandache, S. Gaonkar, M. Sayler, R. R. Choudhury, and L. Cox. Enloc: Energy-efficient localization for mobile phones. In *Proceedings of IEEE INFOCOM Mini Conference '09*, Rio de Janeiro, Brazil, 2009.
[18] S. B. Eisenman, E. Miluzzo, N. D. Lane, R. A. Peterson, G.-S. Ahn, and A. T. Campbell. The bikenet mobile sensing system for cyclist experience mapping. In *Proceedings of ACM SenSys '07*, Sydney, Australia, 2007.
[19] S. Gaonkar, J. Li, R. R. Choudhury, L. Cox, and A. Schmidt. Micro-blog: sharing and querying content through mobile phones

and social participation. In *Proceedings of ACM MobiSys '08*, Breckenridge, CO, USA, 2008.

[20] H. W. Gellersen, A. Schmidt, and M. Beigl. Multi-sensor context-awareness in mobile devices and smart artifacts. *Mob. Netw. Appl.*, 7(5):341–351, 2002.

[21] N. Györbíró, A. Fábián, and G. Hományi. An activity recognition system for mobile phones. *Mob. Netw. Appl.*, 14(1):82–91, 2009.

[22] B. Hoh, M. Gruteser, R. Herring, J. Ban, D. Work, J.-C. Herrera, A. M. Bayen, M. Annavaram, and Q. Jacobson. Virtual trip lines for distributed privacy-preserving traffic monitoring. In *Proceedings of ACM MobiSys '08*, Breckenridge, CO, USA, 2008.

[23] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan, and S. Madden. Cartel: a distributed mobile sensor computing system. In *Proceedings of ACM SenSys '06*, Boulder, Colorado, USA, 2006.

[24] S. Kang, J. Lee, H. Jang, H. Lee, Y. Lee, S. Park, T. Park, and J. Song. Seemon: scalable and energy-efficient context monitoring framework for sensor-rich mobile environments. In *Proceedings of ACM MobiSys '08*, Breckenridge, CO, USA, 2008.

[25] J. Lester, T. Choudhury, G. Borriello, S. Consolvo, J. Landay, K. Everitt, and I. Smith. Sensing and modeling activities to support physical fitness. In *Proceedings of UbiComp '05*, Tokyo, Japan.

[26] K. Lin, A. Kansal, D. Lymberopoulos, and F. Zhao. Energy-accuracy aware localization for mobile devices. In *Proceedings of ACM MobiSys '10*, San Francisco, California, USA.

[27] H. Lu, W. Pan, N. D. Lane, T. Choudhury, and A. T. Campbell. Soundsense: scalable sound sensing for people-centric applications on mobile phones. In *Proceedings of ACM MobiSys '09*, Kraków, Poland, 2009.

[28] E. Miluzzo, N. D. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S. B. Eisenman, X. Zheng, and A. T. Campbell. Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application. In *Proceedings of ACM SenSys '08*, Raleigh, NC, USA, 2008.

[29] P. Mohan, V. N. Padmanabhan, and R. Ramjee. Nericell: rich monitoring of road and traffic conditions using mobile smartphones. In *Proceedings of ACM SenSys '08*, Raleigh, NC, USA, 2008.

[30] J. Paek, J. Kim, and R. Govindan. Energy-efficient rate-adaptive gps-based positioning for smartphones. In *Proceedings of ACM MobiSys '10*, San Francisco, California, USA.

[31] A. Rahmati and L. Zhong. Context-for-wireless: context-sensitive energy-efficient wireless data transfer. In *MobiSys '07: Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 165–178, New York, NY, USA, 2007. ACM.

[32] E. Shih, P. Bahl, and M. J. Sinclair. Wake on wireless: an event driven energy saving strategy for battery operated devices. In *Proceedings of ACM MobiCom '02*, Atlanta, Georgia, USA, 2002.

[33] J. Sorber, N. Banerjee, M. D. Corner, and S. Rollins. Turducken: hierarchical power management for mobile devices. In *Proceedings of ACM MobiSys '05*, Seattle, Washington, 2005.

[34] M. A. Viredaz, L. S. Brakmo, and W. R. Hamburgen. Energy management on handheld devices. *ACM Queue*, 1(7):44–52, 2003.

[35] Y. Wang, J. Lin, M. Annavaram, Q. A. Jacobson, J. Hong, B. Krishnamachari, and N. Sadeh. A framework of energy efficient mobile sensing for automatic user state recognition. In *Proceedings of ACM MobiSys '09*, Kraków, Poland, 2009.

[36] J. Yoon, B. Noble, and M. Liu. Surface street traffic estimation. In *Proceedings of ACM MobiSys '07*, San Juan, Puerto Rico, 2007.