PADA: Power-aware Development Assistant for Mobile Sensing Applications

Chulhong Min[1], Seungchul Lee[1], Changhun Lee[1], Youngki Lee[2], Seungwoo Kang[3], Seungpyo Choi[1], Wonjung Kim[1], Junehwa Song[1]
[1]School of Computing, KAIST, [2]School of Information Systems, Singapore Management University, [3]School of Computer Science and Engineering, KOREATECH
[1]{chulhong, seungchul, changhun, spchoi, wjkim, junesong}@nclab.kaist.ac.kr, [2]youngkilee@smu.edu.sg, [3]swkang@koreatech.ac.kr

ABSTRACT
We propose PADA, a new power evaluation tool to measure and optimize power use of mobile sensing applications. Our motivational study with 53 professional developers shows they face huge challenges in meeting power requirements. The key challenges are from the significant time and effort for repetitive power measurements since the power use of sensing applications needs to be evaluated under various real-world usage scenarios and sensing parameters. PADA enables developers to obtain *enriched* power information under diverse *usage scenarios* in *development environments* without deploying and testing applications on real phones in real-life situations. We conducted two user studies with 19 developers to evaluate the usability of PADA. We show that developers benefit from using PADA in the implementation and power tuning of mobile sensing applications.

Author Keywords
Power-aware development; Mobile sensing applications

ACM Classification Keywords
H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous

INTRODUCTION
Mobile sensing applications (MSAs) bring developers unprecedented challenges for power-aware development. Due to their power-intensive nature, it is inevitable for developers to optimize the power use of MSAs through multiple iterations of power evaluation: measuring power, identifying power-intensive code blocks, and changing logic or tuning relevant parameters. Such iterations are extremely burdensome because MSAs need to be tested repetitively under diverse real-life situations as a result of their highly variable power use in different user contexts [21,22,23]. Moreover, even a single evaluation of an MSA's power use is highly cumbersome and time consuming; developers should be in a real usage scenario for realistic power testing, e.g., being in a bus for a transportation mode detector or being near people for an interaction monitor.

Our exploratory study shows that, even in the industry, mobile developers do not seriously take power requirements into accounts, or deal with them in an ad-hoc manner, due to a lack of supporting tools. We surveyed 46 professional developers (78% of whom have more than two years of development experiences). 73% of them noted that energy efficiency is an important aspect, but 35% did not put explicit efforts into measuring and optimizing energy use during the development process. 35% of them evaluate the power use of applications only after they are fully developed, which often causes huge changes in their logic. We also interviewed 7 MSA developers and showed that the current practices for power evaluation are laborious, inconvenient, and inaccurate.

We present PADA, a novel tool to assist developers with power-aware development of MSAs. Its key idea is to equip developers with power emulation environments upon which they can instantaneously replay their codes. It has three unique features: (1) *accurate* power estimation of an MSA in a *development environment*, without executing it on a phone nor measuring its power, (2) power estimation under *diverse input workloads* reflecting real-world usage scenarios, and (3) provision of *enriched* power information, e.g., per hardware components and over a timeline, to help developers obtain a holistic understanding of the power behavior of MSAs.

It is important to note that easy and repetitive power evaluation is not yet feasible with the tools that are currently available. In practice, developers use either battery profilers provided by mobile OSes [3,8] or an external power monitoring device like Monsoon. Battery profilers are too coarse-grained to identify and narrow down causes of energy issues, and require executing applications on actual devices. Also, power monitoring devices significantly limit real-world testing because they are not very portable and need to be attached to a phone to be tested; note that real-world testing is critical because power consumption for MSAs varies significantly based on users' physical contexts [21,23].

Our user studies with the PADA prototype show that PADA is useful for power-aware implementation and tuning of MSAs. Thirteen participants out of 14 (strongly) agreed that PADA was useful after they used it to develop an example

UbiComp '16, September 12-16, 2016, Heidelberg, Germany
 ISBN 978-1-4503-4461-6/16/09…$15.00
DOI: http://dx.doi.org/10.1145/2971648.2971676

MSA given power requirements. They highlighted three key benefits of PADA as follows: (1) lowering the burden of repetitive power measurements in real usage scenarios, (2) providing rich, in-depth power use breakdown over a timeline, and (3) guaranteeing the repeatability of the power evaluation on the same scenarios. Moreover, the other 5 participants used PADA to tuned the power use of another MSA and stated that it greatly accelerates the tuning process compared to current practices. They tested 10 potential settings (2 scenarios × 5 duty cycles) with PADA to quickly determine the right duty cycle, which would otherwise have likely taken several days with current practices.

The contribution of this paper can be summarized as follows.

- We show that, even in the industry, power use of an MSA is hardly considered despite its importance based on an online survey of 46 mobile developers and interviews with 7 MSA developers. Even if it was the case, the power was often considered at the last stage of the application development cycle, not throughout the whole development process.
- We propose PADA, a novel developer support tool to enable quick provision of detailed power information in a development environment, which is essential for power-aware implementation and optimization of MSAs.
- We demonstrate the usefulness and benefits of PADA in the power-aware implementation and power tuning of MSAs by conducting two user studies with 19 developers.

RELATED WORK

Much effort has been given to estimate the power use of mobile applications and to identify power bugs without a separate hardware. Mobile OSes provide tools such as iOS Instruments [8] and BatteryStats [3] to monitor hardware use and power consumption of applications. Also, many research prototypes have been proposed to estimate the power use of applications based on pre-built power models [14,19,26,32]. Some works provide more detailed power use information, e.g., at the system-service level [13], method level [26], and source line level [19]. These tools help developers measure power and avoid complications from using separate power monitors. However, they commonly require execution of applications on real phones, incurring significant time and efforts from developers. Unlike such tools, PADA provides the MSA's power use in development environments without executing it on a real phone nor using a separate monitor, thereby greatly reducing the burden of the developers.

Recently, a few works have proposed the emulation of the power behavior of mobile applications [21,22,24]. Mittal et al. presented WattsOn, an energy emulation tool that allows developers to explore power behaviors of applications under multiple operating conditions in development environments [24]. It mainly targets interactive foreground applications and thus focuses on power modeling and resource scaling of the display, network, and CPU. Thus, it hardly supports the power emulation of MSAs due to a lack of power emulation of the sensor components. In this work, we propose a novel tool enabling developers to examine the power behaviors of MSAs under diverse emulated usage environments. Also, we evaluate the effectiveness of the proposed tool through user studies, which has not been explored in existing works.

In our previous work, we presented PowerForecaster, a system that provides users with the personalized power impact of MSAs at the pre-installation time [21,22]. Its core component is the power emulator that emulates the power behavior of MSAs. PADA is developed based on the power emulator of PowerForecaster; however, the contributions of the two works are greatly different because the goals, target users, and usages are different. More specifically, PADA is designed to assist MSA developers with power-aware development, whereas PowerForecaster targets end users. In this work, we newly discovered that repetitive power evaluation is key for the power-aware development of MSAs and explored the challenges of laborious power evaluation. Based on the findings, PADA extends the power emulator of PowerForecaster to help developers (1) facilitate such costly power evaluation and (2) understand the detailed power behavior of MSAs with enriched power information. We further conducted user studies to evaluate its effectiveness.

A few works have addressed energy bug detection through static analysis at compile time, e.g., the misuse of wakelock [27]. However, it is well known that such static analysis-based methods are limited in capturing dynamic application behavior at execution time, which is very important for MSAs because their power use varies significantly depending on the physical behavior of users [21,22,23].

Several works developed automatic testing frameworks for mobile applications using mobile emulators [6,20,29] recently. They execute an application on the emulator with a *monkey* tool, feed streams of user interaction such as touch events, and analyze runtime properties such as application exceptions. PADA is distinguished from those in three aspects. First, PADA targets MSAs and thus replays the stream of sensor data in the emulation environment. Second, it focuses on analyzing the power-related behaviors of MSAs. Last, PADA provides broad support to facilitate repetitive, burdensome power evaluation of MSAs, thereby allowing convenient power tuning, energy bug detection, and so on.

A variety of MSAs have been actively proposed in the literature [7,10,12,17,25]. In common, they provide situational services based on continuous monitoring of user contexts. To manage their power use at runtime, there has been extensive effort on mobile platforms for MSAs [4,9,15,16,18]. PADA complements such runtime systems by helping developers develop energy-efficient MSAs.

EXPLORATORY STUDY WITH DEVELOPERS

Survey of Mobile Developers

We conducted an online survey with 46 participants who currently work at mobile software companies. Table 1 shows their demographics; we did not collect company names due to privacy concerns. We recruited the participants through

Age	20s (19), 30s (22), 40s (5)
Development experiences (years)	0-2 (12), 2-4 (6), 4-6 (14), 6-8 (7), >8 (7)
MSA development experience	yes (16), no (30)
Occupation	developers (42), QA/tester (1), manager (2), other (1)
Self-reported development skills	expert (13), intermediate (23), beginners (10)

Table 1. Participant demographics of the online survey; the parenthesized numbers are the number of the participants

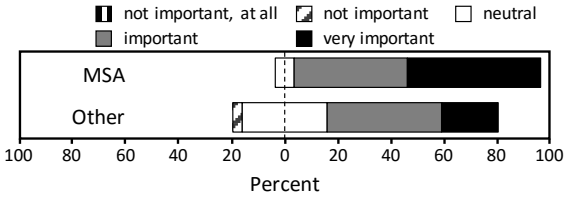

Figure 1. Awareness of energy efficiency

online community sites for mobile developers. Each participant was provided a gift card equivalent to USD 1.4.

Awareness of energy efficiency and current practices

Q1: *Do you think energy efficiency is important in developing mobile applications?* Figure 1 shows the results; each distribution is centered at "neutral" and the width of the segment is the frequency of each response. Twenty participants (43%) reported very important and 14 important (30%). We further examined whether developers who have experience with building MSAs considered energy more important. We split the participants into two groups: one with MSA development experience and the other without such experience. The answers show that the former (M=4.4, SD=0.6) considers energy efficiency more important than the latter (M=3.8, SD=0.9): the difference was statistically significant (p-value=0.005). This is mainly due to the unique characteristics of MSAs which continuously drain the battery in the background. Thus, developers need to be more careful about energy efficiency not to degrade user experience.

Q2: *From which stage do you (desire to) start considering energy efficiency?* We asked if the participants actually consider energy efficiency when developing their applications, and 30 participants (65%) reported that they did so. This was lower than expected, considering their concern about energy efficiency. Then, we asked from which stage they actually start considering energy efficiency. Figure 2 shows that 16 (53%) of 30 participants start evaluating power use only after their applications are fully implemented. We further asked from which stage they want to start if possible. We classified these 30 participants into two groups: one who starts considering energy efficiency before the full implementation (14 participants) and the other after the full implementation (16 participants). Interestingly, only 2 in the former group wants to start power evaluation in an earlier stage, whereas 8 in the latter group wants to do it before full implementation. This shows that power evaluation from an early stage in the development is preferable. The reasons

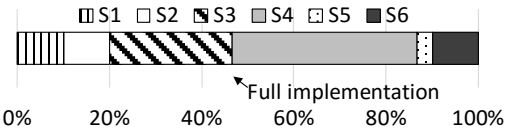

Figure 2. Stage of development when energy efficiency is considered: S1: project planning, S2: early stage of development, S3: middle stage of development, S4: alpha/beta testing, S5: after release, S6: upon user feedback *(Question: From which stage do you start considering energy efficiency?)*

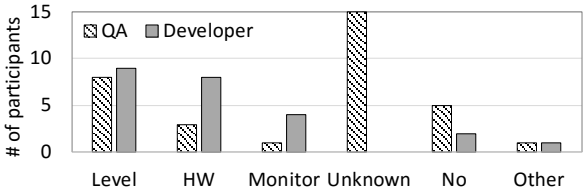

Figure 3. How to measure the power; Level: long-term observation of battery level decrease, HW: observation of h/w usage, Monitor: power measurement equipment, Unknown: I don't know, No: do not measure power
(Direction: please select how you (or QA team) measure the power of mobile applications. You may choose more than one)

why they were not able to start earlier were that it requires too much time and effort (7 participants) and their companies have specific policies about this matter (3 participants). Interestingly, 3 of the former group reported that they prefer the later stage, whereas none of the latter group did so.

Q3: *Do you or anyone in your team measure power use of an application during the development process?* We first asked whether the participant's company has quality assurance (QA) teams and how the QA teams measure the power use. Twenty-eight participants (61%) answered that they have QA teams and the rest of them (18 participants, 39%) answered that they do not. Interestingly, despite their high interest in energy efficiency, 15 (54%) out of the 28 are not aware of how their QA teams evaluate power use of applications (See Figure 3). Additionally, we noticed that most QA teams measure the power in a coarse-grained way, i.e., battery level monitoring and hardware usage observation using BatteryStats or custom logs. Only one team uses power monitoring equipment for fine-granule power measurements. For the participants who do not have the QA teams, we asked if developers themselves measure the power and if so, how they do it. The most common way of measuring the power by developers is also based on battery levels and hardware usage (14 participants out of 18, 78%). Interestingly, the ratio of using power monitoring equipment by developers (22%) is higher than that by the QA teams (4%).

Knowledge on battery use of smartphones

We further investigated MSA developers about their level of understanding on the battery use of smartphones. For this purpose, we asked the MSA developers to rank the average power consumption of frequently-used power-hungry hardware components, i.e., GPS, accelerometer, storage, and screen. The expected ranking is (1) screen, (2) GPS, (3) storage, and (4) accelerometer. Surprisingly, 12 (75%) of the

16 MSA developers did not give the correct answer; 11 developers ranked correctly only for a subset of the options and the other one marked "*I don't know*" without ranking. This is against our expectation that MSA developers in the industry are aware of battery use of smartphones. Moreover, this may imply that even commercial MSAs could be developed without careful consideration of the power use of various hardware components triggered by the applications.

Interview of MSA Developers

We conducted in-depth interviews of 7 MSA developers (denoted D_1 to D_7) to deeply understand the challenges in power evaluation in developing MSAs. We performed a half-hour semi-structured interview with each of them. They belong to different teams in different companies. Each one was provided with a gift card equivalent to USD 9. Our key questions consisted of "*What process do you go through to evaluate power use of an MSA*?", "*Which tools do you use to measure the power of an MSA?*", "*Do you find any difficulties in using those tools?*", "*How do you optimize power use of your MSA after the power measurement?*"

Complying with our survey results, they all agree that energy efficiency is very important to consider. They are even under pressure from their manager and companies to reduce the power consumption of their MSAs. D_1 stated, "*My company has been focusing on energy efficiency for several years.*" D_2 mentioned, "*It depends on the type of application. We put much effort into applications running in the background.*"

However, they find it challenging to optimize the power consumption of their MSAs. One major issue was that repetitive power measurements were required. D_5 who has been developing beacon-based advertisement applications talked about his efforts in the power evaluation process. He stated, "*(Even for a lab-setting test) We installed ten beacons at 50 m intervals indoor, had to move 500 m for a single energy measurement, and repeated this many times. […] For on-site testing, we installed beacons inside Gangnam station, tested for two days with our prototype. […] It was still not easy. It takes too much time to do all the preparation and actual measurements.*" D_7 described their power evaluation method, "*We install our MSA on several smartphones, monitor their battery levels for a day, find issues, and repeat this for several more days."* Their experiences show that it takes from several hours to several days even for a single measurement, and this needs to be repeated, making the power evaluation very time consuming and laborious. These sometimes lead developers to make undesirable choices. D_3 said, "*Our team sometimes fails to address power issues before a deadline. We had to leave the issues unresolved.*"

They pointed out the necessity of anticipating various situations that their MSAs might encounter. D_6 stated, "*Our team uses a service to profile the general performance of applications in various usage scenarios. It plays a crucial role in our production process. But sadly, I haven't heard about a similar tool for power use.*" D_5 also stated "*To test whether the application consumes the energy as intended in various situations, we have to put ourselves into the situation. For example, we manually repeat the following actions, 'stay near the beacon for 10 seconds and walk away', 'stay near the beacon for 20 seconds away' and so on.*"

Due to a lack of supporting tools, developers commonly have difficulties in considering the energy efficiency during development. D_1 stated, "*I first implement the functional requirements without considering power. When our QA team gets back with power issues, there isn't much I can do. I just try simple tuning such as increasing the duty cycle.*" D_5 stated, "*To determine whether sensors operate as intended without causing energy issues, I manually log and analyze sensor activities and related information. It is very tiresome."* D_6 said, "*My team members commonly make mistakes due to lack of (sensor usage) information. […] For example, they request the GPS sensor more frequently compared to what's necessary. The effect of frequent GPS use on the battery is not noticeable from short observations.*"

Key Takeaways

- Developers perceive that the energy efficiency of their mobile applications is an important concern.
- Developers often fail to consider power due to limited time and efforts for repetitive and costly power evaluations.
- The current practices for power evaluations are laborious, inconvenient, and inaccurate, and developers thought supporting tools are lacking.
- For the power evaluation of MSAs, it is desirable but difficult to consider various real-world usage scenarios.

PADA: POWER-AWARE DEVELOPMENT ASSISTANT

Design Goals

PADA is a system tool to assist developers with the power evaluation of MSAs by addressing the above-described difficulties in power evaluations. For its design, we elicited three key requirements from the survey and interviews, along with our intuitions and experiences with MSA development.

Facilitating repetitive power evaluation: PADA enables developers to quickly and repetitively perform power evaluations of MSAs in the development environment, thereby significantly improving development productivity.

Supporting various real-life situations: MSAs often operate differently depending on the user behavior and thus their power use varies largely depending on user contexts [21,23]. PADA supports developers in evaluating the power use of MSAs under various real-life scenarios of the target users.

Providing enriched power information: A single power value (mW) is the most representative, commonly-used metric for power analysis. However, it is insufficient for developers to understand the detailed power behavior of MSAs to optimize their power use and detect causes of power-related problems. PADA provides enriched power information.

PADA Overview

We take a *context-driven record-and-replay* as our key design approach to meet the design requirements. The idea is

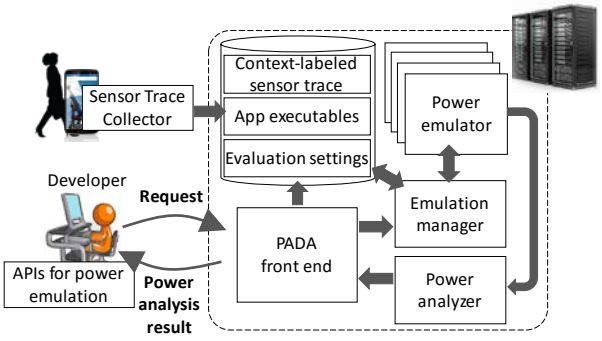

Figure 4. PADA overview

to equip developers with power emulation environments upon which they can instantaneously replay their codes. The emulation is performed with pre-recorded sensor data taken from the real contexts of users. The environment replays a target MSA over the sensor data and emulates its power behavior, allowing developers to analyze the power use without field trials and repetitive measurements. Compared to the current practice, i.e., to actually run the codes on a phone, our emulation-based approach has the following benefits. First, it lowers the burden of developer by removing the hassles of executing and observing the application for a long period of time under real-world usage scenarios. Second, it enables the repeatability of the power evaluation, i.e., comparing the power behavior of different application logics in the same situations. Note that even seemingly similar trials may generate different sensor data, which in turn change the power use of an MSA. Third, it enables fast and scalable power evaluation, leveraging cloud resources.

Figure 4 shows an overview of PADA. It consists of the following components. First, the *PADA front end* provides a web-based interface with which developers request a power evaluation and view the analysis report. Second, the *emulation manager* governs the power emulation process which involves accepting power evaluation requests from the front-end and distributing the requests to multiple power emulator instances. Each *power emulator* emulates the power behavior of the target MSA using selected sensor traces and then passes the emulation results to the power analyzer. The *power analyzer* turns the raw emulation results into informative power analysis reports. PADA also provides *auxiliary APIs* that enable developers to take full advantage of the features of PADA such as configuration changes and custom logging. Note that, prior to the power evaluation request, the sensor traces are collected on the mobile-side *sensor trace collector* once and reused for various requests.

PADA User Interface

PADA currently provides a web-based interface that consists of two major sections: evaluation request and power analysis.

Evaluation request: Figure 5 shows the interface to issue a new power evaluation request. Its primary purpose is to facilitate power evaluation requests of developers for various execution scenarios with minimal effort. The evaluation request process is as follows. First, the developer uploads an

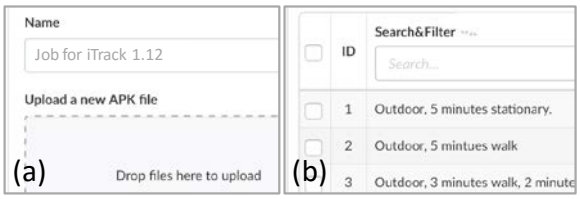

Figure 5. Power emulation request on PADA;
(a) apk upload and (b) trace selection

application executable to be evaluated with drag-and-drop (See Figure 5 (a)). Second, the developer chooses a set of sensor traces on where the application would run (See Figure 5 (b)). Last, the developer uploads the setting file if she wants to test her MSA under diverse application configurations. If specified, PADA generates multiple application executables automatically and runs them with different configurations specified. PADA also enables the submission of multiple requests in parallel to ease the evaluation.

Power analysis reports: PADA features two types of power analysis reports: *in-depth analysis* and *comparative analysis*. The in-depth analysis report provides detailed power information for a single request shown in Figure 6. It consists of the following four sections. First, PADA shows the power use overview (Figure 6 (a)) with the average power consumption (mW), the total wakelock time, the total alarm count, and the total activation time of individual hardware components. Second, it provides an area chart to visualize the estimated power consumption over time (Figure 6 (b)) along with the hardware use pattern on a timeline, indicating activations of hardware components including CPU, GPS, sensors, microphone, and Bluetooth with colored bar segments (Figure 6 (c)). The information helps developers get a holistic understanding of the power characteristics of their MSA and gives them a fine-grained view of power behavior from the perspective of hardware and system.

PADA additionally displays the labeled context information and custom log messages above shown in Figure 6 (d). Such information helps developers determine under which user contexts the application consumes high power and which part of the application logic contributes more to the overall energy use. Note that the power consumption of an MSA varies significantly depending on user contexts [21,22,23].

The second is the comparative analysis. PADA provides an interface to compare important metrics, e.g., average power consumption and hardware usages, across multiple power emulation requests. Figure 7 shows an example of the comparative analysis; 'avg. power', 'wakelock time' and 'GPS time' are compared. Developers can customize the reports by selecting the metrics of interest. The comparative analysis is a unique advantage of PADA because it can repeat power evaluations over exactly the same sensor traces. If a developer was to manually carry out such comparative power evaluations, it would be extremely challenging. Not only would it take much more time and efforts, the developer would need to replicate the exact same situations, which might be almost physically infeasible in many cases.

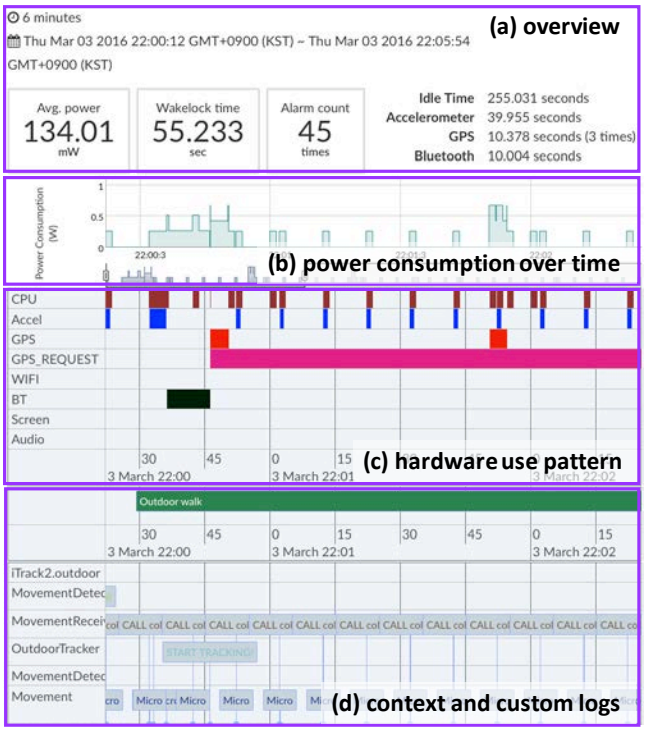

Figure 6. In-depth analysis of the power behavior on PADA

Comparative analysis is useful in several cases. To optimize the power consumption even under worst-case scenarios, a developer needs to find a particular user situation with higher power use. Moreover, after the addition of new features or changes in the parameters, the developer might examine the power impact of the revision. PADA allows the developer to make more informed decisions about important changes. While the current prototype provides an aggregate summary for the comparison, we will add an in-depth comparison feature to compare two requests in a detailed timeline view of the power and hardware use pattern.

Auxiliary APIs

Adjusting MSA configurations: Developers often need to frequently compare the power use of an MSA while varying application configurations. For example, assume a developer tests an MSA for 10 different cases, e.g., different sensor sampling rates and monitoring intervals to find the most desirable parameters. However, it is extremely burdensome to manually build the application executables for every time and compare their power behaviors. PADA provides APIs to facilitate such a process. Developers first specify a set of parameter types and candidate values in a configuration file. Upon a request, PADA automatically generates the executables for all combinations of parameter values and runs them. Figure 8 shows an example code snippet. In the '*parameter.xml*', three values are specified for the parameter type, '*location_interval*'. Then, the executables are made and getParameterValue() returns one of the values for each executable at runtime.

Tagging runtime information: To help developers match the hardware usage of an MSA to relevant source codes,

Figure 7. Comparative analysis of power behavior on PADA

```
<parameter.xml>
  <parameter type="location_interval">
      <value>10</value>
      <value>20</value>
      <value>30</value>
  </parameter>

<application code>
  interval = PADA.getParameterValue(
       "location_interval", "parameter.xml");
  requestLocation(GPS_PROVIDER, interval, 0, this);
```

Figure 8. Code snippet for the application configuration

PADA provides two types of logging APIs, *Log.p*(*tag*, *msg*) and *Log.p*(*tag*, *msg*, *isOn*). The former is to display a power-related log message at a specific point and the latter is to display a message as a range. *tag* is to identify the source of a log message, *msg* is the message that developers would like logged, and *isOn* indicates the start and stop of the range. The log messages are collected during the power emulation of the MSA and later displayed along with hardware usage and power use information shown in Figure 6 (d). Assume a developer finds a location in the source code where an acquired wakelock is not released properly. The developer can put a log with the custom message whenever his/her application acquires a wakelock. This later can tell the developer where in the code the problematic wakelock is acquired and what application logic is responsible.

Sensor Trace Collection

Developers collect the sensor traces under real-world usage scenarios before they use PADA. The trace consists of a time series of sensor values and events. The context information can be labeled additionally if necessary. The scenarios under which the traces are collected are carefully crafted by reflecting the goal of the power testing for the MSAs. For example, assume a location tracking MSA that activates GPS only when a user is moving. Developers might want to test their MSA under scenarios such as 'standing for a while' and 'stopping after walking for 5 minutes'. For power tuning of their MSA, they also can consider one day-long scenarios of target users, e.g., 20's undergraduate student's weekend date and 30's office worker's weekday. Then, they can observe the power use of the MSA by varying the sensing parameters.

We develop a mobile-side sensor trace collector with which developers can collect the sensor traces of interest. Developers specify the types and sampling rates of sensors

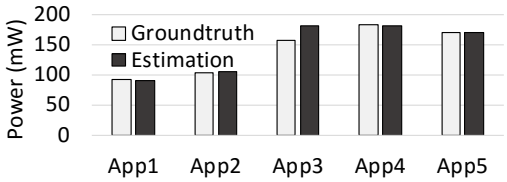

Figure 9. Power estimation; App1: Accupedo, App2: Pedometer 2.0, App3: SleepBot, App4: Android-pedometer, App5: iTrack

required for the execution of their MSA. The collected traces are reusable throughout the development process. Moreover, the developers can generate traces that can be commonly used for different MSAs by collecting sensor data from all available sensors on the phone at the highest sampling rate. Note that PADA is already equipped with a pool of sensor traces for various common user scenarios which enables basic power evaluation even without data collection.

Power Emulation

For the power emulation of MSAs, we implemented the PADA emulation manager with Java Spring framework on a server machine. As an input, the power emulator takes a sensor trace and an application executable. It then executes the application while replaying the trace and monitors the hardware usage. It outputs a time series of system call events by the application and estimated power consumption.

Existing mobile emulators are not feasible to emulate the power behavior of MSAs due to a lack of physical sensor devices and power monitoring functionality. For power emulation, we adopt and modify the power emulator proposed in [21,22]; the emulator is built on the Android framework. To enable the target MSA to run on the emulator, the power emulator hooks system calls made by the SensorManager in the Android framework and feeds sensor values from the pre-collected sensor trace as target application requests. While the target MSA is running, the power emulator records system calls related to the power behavior of the MSA at the same time, e.g., *gps_on()* and *gps_off()* for GPS. After the emulation is finished, the power analyzer computes the power consumption of the application with a system call-based power estimation [26].

We evaluated the power estimation accuracy of PADA under diverse types of MSAs. We used three commercial MSAs (Accupedo, Pedometer2.0, SleepBot), one open source MSA (Android-pedometer), and one research MSA we developed (iTrack which is described in the evaluation section). The ground truth is obtained by using a Nexus 5 (Android 4.4) phone with a Monsoon power monitor under a one-hour real-world usage scenario. As shown in Figure 9, PADA achieves 95.9% accuracy on average.

EVALUATION

We evaluate PADA with Android developers: (a) power-aware implementation and (b) power tuning of MSAs.

Power-aware Implementation of MSAs with PADA

To evaluate the usability of PADA in the implementation of MSAs, we conducted a user study with 14 programmers: 10

#		Traces [minutes, context]	Power requirements
1	Outdoor	[5, staying]	$Duty_{CPU} \leq 20\%$, # of BT/GPS calls = 0, …
2		[5, walking]	# of BT calls = 1, # of GPS calls = 5, …
3		[3, walking], [2, staying]	$Duty_{ACC} \leq 15\%$, # of alarm calls ≤ 50, …
4	Indoor	[5, staying]	$Duty_{CPU} \leq 20\%$, # of BT/GPS calls = 0, …
5		[5, walking]	$Time_{BT} \leq 120s$, # of GPS calls = 0, …
6		[3, walking], [2, staying]	$Duty_{CPU} \leq 40\%$, # of BT calls = 3, …

Table 2. Traces and corresponding power requirements

undergraduate students (P_1 to P_{10}), 2 graduate students (P_{11}, P_{12}), and 2 research engineers (P_{13}, P_{14}). We asked them to implement an MSA and to meet power requirements. During the implementation, they were allowed to use PADA freely. Then, we investigated the usability of PADA in an actual implementation process. In particular, we focused on how PADA helped developers fulfill the power requirements. All participants majored in computer science and had 0.5 to 5 years (M=2.0,SD=1.6) of Android programming experiences. None of them had any prior knowledge of PADA. They were rewarded with a gift card equivalent to USD 40. The study was conducted across five sessions, with two to four participants in each. We provided a Nexus 5 phone for testing and debugging purposes. We pre-collected six five-minute-long sensor traces shown in Table 2 to be used in PADA.

Development requirement: We asked the participants to develop *iTrack*, an Android MSA for energy-efficient location tracking. Its main functionality is to continuously record the location of a user. To save energy, it monitors the location only when a user is moving. It detects the movement of the user through periodic accelerometer sensing. The location is monitored with Bluetooth scan (indoor) or GPS (outdoor); we assume that a user is indoor when a designated beacon is detected. In addition to the functional requirements, the participants were asked to fulfill the power requirements in Table 2. To put them succinctly, the hardware components should be activated only when the relevant operation needs to be executed. For example, after movement detection, wakelock should be released until the next detection. Similarly, GPS should not be activated when a user is indoor.

According to our preliminary study, even experienced developers took quite a long time to develop iTrack from scratch. To focus on the impact of PADA on the power-related implementation activities, we provided the participants with the skeleton code of iTrack. It contains basic boilerplates and implementations of modules such as the movement detection algorithm. The participants were asked to implement power-related application logics, e.g., registering alarms, activating/deactivating sensors, and using wakelocks on top of the given skeleton code.

Procedures: The study consisted of three parts. First, we gave the participants a one-hour tutorial about the specification and requirements of iTrack and how to use PADA. Second, three hours were given for developing iTrack with PADA at their disposal. At the end, we conducted a one-hour long semi-structured interview about

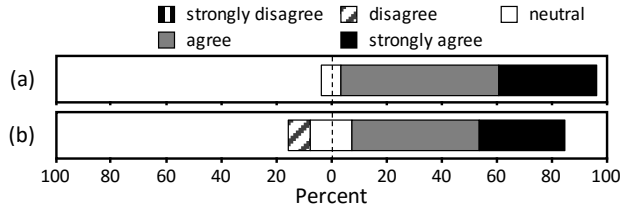

Figure 10. Participants' responses distributions;
(a) *"Do you think PADA is useful?"*
(b) *"Do you think PADA is easy to use?"*

their use of PADA during the development. Two researchers analyzed the transcripts individually and reached consensus for high-level themes by discussing together [5].

Usefulness of PADA

Overall, all the participants were satisfied with the usefulness of PADA. As shown in Figure 10 (a), eight participants reported PADA was useful for the power-aware implementation and 5 very useful.

Lowering the burden of power evaluation: According to the interviews, 11 participants were surprised about the ease of the power evaluation. The common reason was that they measured the power consumption within the development environment without actually moving around. P_7 stated, comparing with his past experiences, "*I implemented a GPS-based mobile application before. To test the application in the office, I had to put my hand out of window and keep shaking. When using PADA, I didn't need to go out for testing and I really liked it.*" P_1 described a similar experience, "*When I was testing a fall detector application, I had to fall down tens of times to collect data and measure the performance. I might have had to go through a similar situation today, without PADA.*" Some participants highlighted the ease of the power evaluation in PADA compared with existing tools such as Android BatteryStats and iOS energy instrument. P_{10} said, "*Without any effort, I was able to examine multiple scenarios (with PADA) in parallel.*" P_5 added, "*BatteryStats just provides continuous power logs after I clear the log using ADB. To minimize the number of ADB connection, I had to log several scenarios sequentially. But analysis was not easy.*"

Five participants complained that they had to wait for a while to receive the result. The main reason was that the progress of the evaluation was hidden to the developers. P_7 mentioned, "*I just kept gazing at the screen and refreshed the job list several times until the job was finished.*" We realize that the future version of PADA needs improvements in the emulation speed and real time feedback about the progress.

Usefulness of the power information: Most participants were satisfied with the detailed, enriched information on power behavior provided by PADA. P_5 stated, "*I thought that obtaining some system information, such as the GPS activation time, is really hard on the application side. I was happy to see a graph provided by PADA. The graph seemed to be telling me that I was doing well.*" P_{11} and P_{12}, who experienced a Monsoon power monitor to develop an MSA, mentioned that PADA provides more detailed information regarding the power behavior of the application than Monsoon. P_{12} said, "*Monsoon shows the overall power, but I can't get the amount of power consumed by an application or a hardware component. (With PADA), I can figure out how my application uses sensor devices. (In the experiment,) I could easily catch the misbehavior of my application.*"

Repeatability of the power evaluation: Five participants found that repeating the evaluation for the exact same situation is useful because they easily caught how changes in the source code affect the power use of iTrack. P_9 recalled, "*When I saw the power report of my modified application, I realized that my modification was really bad. Without PADA, it could be harder to notice this mistake.*" They also reported that the repetitive evaluation has an advantage in testing MSAs in certain scenarios where developers need to spend a significant amount of time and effort. P_5 stated, *"I pictured myself developing a transportation mode detector. Then, my application should be able to recognize 'taking a train'. You know, it would be quite expensive and time consuming to test out. If I don't have PADA, whenever I update the code, I might have to take a train several times to test. If the app crashes during the test… Oh, no… I don't want to imagine.*"

Logical bug detection: We observed that the participants utilized PADA to find logical bugs as well as testing power-related requirements. They examined activation intervals and duration of hardware usages to find logical flaws in their applications. P_6 stated, *"I intended the accelerometer activation interval to be 10 seconds, but was 4 seconds. It also remained activated while a user is walking. I thought I made a mistake in managing movement detection."*

Here is one of the interesting use cases that we found during the experiment. P_{12} used two alarm services for periodic sensing of the accelerometer and Bluetooth. He mistakenly managed these services with the same identifier. It led to a malfunction of turning them on and off together. On the first use of PADA, he easily spotted the bug. This bug might be overlooked without PADA, because no functional failure or crash would be caused in most execution scenarios. Defects in background services such as this bug might not have an immediate impact on the UI and require long observations to find, so even detecting their existence can be challenging.

Usability of PADA

The participants were very positive about their user experience with PADA (see Figure 10 (b)). All participants liked that they need only a few clicks to perform power evaluations of their applications. P_{14} emphasized, "*(It was) really convenient! All I had to do was to drag and drop the apk file (for upload) and wait some time. I'm willing to use PADA if it is released.*" Some participants complimented the intuitive interface of PADA. P_6 said, "*It was very easy to use PADA. The interface mostly operates as expected.*"

Many participants reported that they could see a variety of power information at a glance. P_8 stated with awe, "*It's really*

Elapsed time (min)	0-30	30-60	60-90	90-120	120-150	150-180
# of uses	1	1	5	8	13	13

Table 3. PADA usage counts over time

beautiful! I think anyone can understand how the application consumes the energy." P_{11} stated, "*I saw my application reduces the energy more, it was really exciting. I had pleasure seeing it.*" However, P_3 and P_{12} were concerned about the difficulty in understanding the in-depth analysis page. P_{12} stated, "*The result page was less intuitive than other pages. (…) But, ironically, I like that page the most.*"

Use patterns of PADA in the development

During the development, the participants requested power evaluations 2.9 times on average (min: 0, max: 6. SD: 1.8), which means that they performed power emulations 17.4 times on average (six scenarios per evaluation request). Table 3 shows the usage frequency of PADA over time. The results show that 66% of PADA uses occurred after 90 minutes. This is because, in the beginning, the participants spent time on filling in the skeleton code. Then, after they made the code to be executable, they tried to test and debug the power use of iTrack using PADA. P_4 said, "*I used a real phone to find any runtime error. Then I tried to measure the power with PADA.*" The average interval between successive PADA uses was 21.4 minutes. In the interview, the participants commonly stated that they would perform the power evaluation less frequently if they could not use PADA. P_2 stated, "*If PADA is not provided, I might skip several outdoor scenarios because going outside is burdensome.*"

Suggestions

The participants suggested additional features to improve PADA. P_1 and P_9 suggested to display the interim results in real time. They stated that real time results could enable faster bug detection if a bug occurs at the earlier part of the emulation. P_5 wanted to compare the hardware usage traces of two application versions. He stated, "*I opened PADA in two Chrome browser tabs side by side to identify whether I correctly modify the power bug in the previous version. It would be great if PADA displays two traces at once and emphasizes on the different parts like code diff.*"

Power Tuning of MSAs with PADA

We further conducted a case study with developers to examine the usefulness of PADA in the power tuning of MSAs. For this study, we prepared *SocioHotspot*, an MSA that tracks a user's social hotspots where the user meets her friends frequently by periodic Bluetooth scans for nearby friends' phones. It has two configurable parameters that affect the power and accuracy: the Bluetooth scanning interval while a user is either (1) moving or (2) stationary.

We recruited five participants: four graduate students (P_A, P_B, P_C, and P_D) and one undergraduate student (P_E). The study consisted of two phases: balancing the accuracy and the power consumption of SocioHotspot (1) with the current practices of power turning and (2) with PADA. At the first phase, two Nexus 5 phones were provided to each participant, one for running SocioHotspot and the other for ground truth collection. The ground truth was logged by performing Bluetooth scan every 30 seconds. They were asked to carry two phones at least nine hours a day in daily life. For five days, they were asked to select and change the parameter values every day. We provided them with the accuracy and the power consumption for the selected values each day.

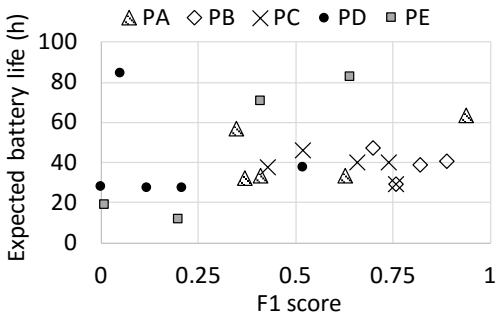

Figure 11. Power tuning with current practices

After five days, the second phase started. The participants visited our laboratory and tried power tuning with PADA. They selected five more sets of parameter values that they wanted to further test. Then, we provided the power results obtained from PADA with two pre-collected sensor traces. At the end of each phase, we conducted a half-hour semi-structured interview with each of them and obtained their power tuning practices. Each one was compensated with a gift card equivalent to USD 30. To promote the participation, we provided a gift card equivalent to USD 10 more to a winner of each session, i.e., who achieved the best balance.

Drawbacks of current practices

As shown in the exploratory study, a common practice for power tuning of an MSA in real-life situations is to observe the battery levels while running the MSA for a long time. However, our study revealed that it has several drawbacks.

Inconsistency of daily behavior: All participants reported that the most difficult part was the inconsistency of their daily behavior. This is surprising because we did not divulge the unique power characteristics of MSAs, i.e., user behavior dependent power use, to the participants. Four participants applied the same parameter values for two different days to confirm the impact of their daily behavior on the power consumption of SocioHotspot. P_B stated, "*I tried to move around similar to yesterday, but it was practically infeasible.*" P_A stated, "*Developers should not do this if the application is for commercial release. It will be biased to their life.*"

Nontrivial burden: The participants reported that power tuning with current practices took all day to test one set of parameter values. In this regard, all participants pointed out that even five days are not enough to achieve satisfactory balancing even though SocioHotspot has just two parameters to be configured. P_D mentioned, "*I may need ten more days to find satisfactory values.*" P_C stated, "*If I am the startup company president, I will test SocioHotspot for one month.*"

P_E stated his experience, "*On Sunday, I was told that the accuracy was zero, and realized I was at home all day. So I'm supposed to do this for one more day.*" They all agreed that, it would be almost infeasible to perform power tuning with a number of configurable parameters in this manner.

Limited results: Figure 11 shows the expected battery life and F1-score of the hotspot detection, which the participants achieved during the first session. The results show that the performance, i.e., balancing the power and accuracy, largely varies depending on the individual. While P_A achieved 64 hours of expected battery life and 94% of F1-score, P_D only achieved 37 hours and 52%. This is mainly due to the developers' own intuition and a limited number of trials.

Power tuning using PADA

All participants agreed that PADA brings unprecedented productivity compared with current practices. They were commonly surprised that they were able to produce comparable results with PADA in a day, whereas it took five days without PADA; actually, they achieved better results due to their prior knowledge on the impact of the parameter values obtained from the first session. They said that, for the same amount of time, they could test their MSA much more frequently with PADA, compared to that without PADA. Then, they reported that it is obvious that they could achieve better accuracy and energy efficiency with more trials using PADA. Especially, they liked seeing the power results without the laborious burden of manual measurement. Some participants found PADA very useful for power tuning because it can compare the results under the same scenarios. We omitted a detailed analysis in this respect because the responses were similar to those in the previous study.

New tuning practices with PADA: All participants agreed that, with the results from PADA (Figure 7), they obtained better understanding on the impact of the parameter values. P_C reported that he easily determined the power behaviors of SocioHotspot with the hardware use pattern (Figure 6). The participants also felt positive about the automated repetition of power tuning without laborious measurements. Interestingly, after using PADA, P_A and P_D wanted to try it more to find satisfactory performance. P_D stated, "*(Without PADA,) I increased the interval by minutes. Now I want to change the interval by 15 seconds.*" P_A said, "*I want to find optimal values by changing one parameter at a time.*"

The participants also expressed interests in performance dependency on a user's behavior. P_C stated, "*It would be essential to consider, at least two representative types of user behaviors.*" P_B suggested, "*After comparing power results from two scenarios, I was convinced that the power impact differs even depending on the individual. […] Isn't it possible to use different parameter values for individual users?*"

Suggestions for systematic power tuning: The participants had the following suggestions for systematic power tuning with PADA. P_D who majors in machine learning suggested, "*I think it seems possible to model users' behavior and then we might make an automated power tuning system based on PADA.*" P_A stated, "*It will be perfect if PADA finds optimal parameter values if I specify their range."*

DISCUSSION

Extending application coverage: PADA currently targets MSAs that perform sensing and processing periodically. However, some MSAs may involve user interactions or cloud offloading [28] that PADA does not currently account the power use for. We will extend PADA to cover the power cost caused by such operations. For instance, PADA can adopt monkey tools [1] to record and replay user interactions, thereby accounting the power consumed by user interactions. Also, network power cost can be addressed by capturing the network usage such as the packet size and signal strength [24] and replaying them on our power emulator.

Supporting modern hardware architecture: Recent mobile phones adopt dedicated processors, designed to optimize the power use of sensor-related operations. For example, Android sensor hub [2] collects and processes sensor data without involving the CPU. We plan to extend our power emulator to support such recent hardware architectures. A possible way would be to track and replay usages of the sensor hub and build its power model as in [30].

Supporting energy debugging: We noticed that developers often want to localize the code block that contributes the most to the power consumption when they see unexpectedly high power numbers. PADA currently supports logging APIs to help developers map the hardware use (and corresponding power) with a specific code block. We plan to extend this feature by adopting bug localization techniques from runtime energy diagnosis tools like Adel [31] and WakeScope [11].

Longitudinal, quantitative evaluation: We acknowledge that our study is based on the short-term use of PADA by a small number of developers, and the findings are mainly derived from qualitative interviews. We will improve PADA based on our findings (e.g., providing feedback on progress, pinpointing problematic code lines, etc.) and conduct a longitudinal evaluation with a larger pool of developers.

CONCLUSION

MSAs induce persistent battery drain due to continuous sensing and processing. Nonetheless, developers have difficulties considering energy efficiency in the development process. We developed PADA, a tool to assist power-aware development of MSAs. We show the usefulness of PADA by conducting two user studies: power-aware implementation with 14 developers and power tuning with 5 developers.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their invaluable comments. This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIP) (No. 2011-0018120) and Civil Military Technology Cooperation Center (12-DU-EB-01) in Agency for Defense Development of Republic of Korea. The corresponding author is Seungwoo Kang.

REFERENCES

1. Android monkey tool. Retrieved June 15, 2016 from https://developer.android.com/studio/test/monkey.html
2. Android sensor hub. Retrieved June 15, 2016 from https://source.android.com/devices/sensors/sensor-stack.html#sensor_hub
3. BatteryStats. Retrieved June 15, 2016 from https://developer.android.com/studio/profile/battery-historian.html
4. David Chu, Nicholas D. Lane, Ted Tsung-Te Lai, Cong Pang, Xiangying Meng, Qing Guo, Fan Li, and Feng Zhao. 2011. Balancing energy, latency and accuracy for mobile sensor data classification. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems* (SenSys '11). 54-67. http://dx.doi.org/10.1145/2070942.2070949
5. Juliet Corbin and Anselm Strauss. 2007. *Basics of Qualitative Research Techniques and Procedures for Developing Grounded Theory.* Sage Publications.
6. Shuhai Hao, Bin Liu, Suman Nath, William G.J. Halford, and Ramesh Govindan. 2014. PUMA: Programmable UI-Automation for Large Scale Dynamic Analysis of Mobile Apps. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*. (MobiSys '14), 204-217. http://doi.acm.org/10.1145/2594368.2594390
7. Inseok Hwang, Chungkuk Yoo, Chanyou Hwang, Dongsun Yim, Youngki Lee, Chulhong Min, John Kim, and Junehwa Song. 2014. TalkBetter: family-driven mobile intervention care for children with language delay. In *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing* (CSCW '14). 1283-1296. http://dx.doi.org/10.1145/2531602.2531668
8. Energy instrument. Retrieved June 15, 2016 from https://developer.apple.com/library/prerelease/content/documentation/Performance/Conceptual/EnergyGuide-iOS/MonitorEnergyWithInstruments.html
9. Younghyun Ju, Youngki Lee, Jihyun Yu, Chulhong Min, Insik Shin, and Junehwa Song. 2012. SymPhoney: a coordinated sensing flow execution engine for concurrent mobile sensing applications. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems* (SenSys '12). 211-224. http://dx.doi.org/10.1145/2426656.2426678
10. Seungwoo Kang, Sungjun Kwon, Chungkuk Yoo, Sangwon Seo, Kwangsuk Park, Junehwa Song, Youngki Lee. 2014. Sinabro: Opportunistic and Unobtrusive Mobile Electrocardiogram Monitoring System. In *Proceedings of the 15th ACM Workshop on Mobile Computing Systems and Applications* (HotMobile '14), Article No. 11. http://dx.doi.org/10.1145/2565585.2565605
11. Kwanghwan Kim and Hojung Cha. 2013. WakeScope: runtime WakeLock anomaly management scheme for Android platform. In *Proceedings of the Eleventh ACM International Conference on Embedded Software* (EMSOFT '13). Article 27, 10 pages.
12. Nicholas D. Lane, Petko Georgiev, and Lorena Qendro. 2015. DeepEar: robust smartphone audio sensing in unconstrained acoustic environments using deep learning. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (UbiComp '15). 283-294. http://dx.doi.org/10.1145/2750858.2804262
13. Seokjun Lee, Wonwoo Jung, Yohan Chon, and Hojung Cha. 2015. EnTrack: a system facility for analyzing energy consumption of Android system services. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (UbiComp '15), 191-202. http://dx.doi.org/10.1145/2750858.2807531
14. Seokjun Lee, Chanmin Yoon, and Hojung Cha. 2014. User interaction-based profiling system for Android application tuning. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (UbiComp '14). ACM, New York, NY, USA, 289-299. http://dx.doi.org/10.1145/2632048.2636091
15. Youngki Lee, S. S. Iyengar, Chulhong Min, Younghyun Ju, Seungwoo Kang, Taiwoo Park, Jinwon Lee, Yunseok Rhee, and Junehwa Song. 2012. MobiCon: a mobile context-monitoring platform. *Commun. ACM* 55, 3 (March 2012), 54-65. http://dx.doi.org/10.1145/2093548.2093567
16. Youngki Lee, Chulhong Min, Younghyun Ju, Seungwoo Kang, Yunseok Rhee, Junehwa Song. 2014. An Active Resource Orchestration Framework for PAN-scale Sensor-rich Environments. *IEEE Transactions on Mobile Computing (TMC)*, Vol. 13, No. 3, 596-610. http://dx.doi.org/10.1109/TMC.2013.68
17. Youngki Lee, Chulhong Min, Chanyou Hwang, Jaeung Lee, Inseok Hwang, Younghyun Ju, Chungkuk Yoo, Miri Moon, Uichin Lee, and Junehwa Song. 2013. SocioPhone: everyday face-to-face interaction monitoring platform using multi-phone sensor fusion. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services* (MobiSys '13). 375-388. http://dx.doi.org/10.1145/2462456.2465426
18. Youngki Lee, Younghyun Ju, Chulhong Min, Seungwoo Kang, Inseok Hwang, and Junehwa Song. 2012. CoMon: cooperative ambience monitoring platform with continuity and benefit awareness. In *Proceedings of the 10th international conference on Mobile systems, applications, and services* (MobiSys

'12). 43-56. http://dx.doi.org/10.1145/2307636.2307641

19. Ding Li, Shuai Hao, William G. J. Halfond, and Ramesh Govindan. 2013. Calculating source line level energy information for Android applications. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis* (ISSTA 2013), 78-89. http://dx.doi.org/10.1145/2483760.2483780
20. Chieh-Jan Mike Liang, Nicholas D. Lane, Niels Brouwers, Li Zhang, Börje F. Karlsson, Hao Liu, Yan Liu, Jun Tang, Xiang Shan, Ranveer Chandra, and Feng Zhao. 2014. Caiipa: automated large-scale mobile app testing through contextual fuzzing. In *Proceedings of the 20th annual international conference on Mobile computing and networking* (MobiCom '14), 519-530. http://dx.doi.org/10.1145/2639108.2639131
21. Chulhong Min, Youngki Lee, Chungkuk Yoo, Seungwoo Kang, Sangwon Choi, Pillsoon Park, Inseok Hwang, Younghyun Ju, Seungpyo Choi, and Junehwa Song. 2015. PowerForecaster: predicting smartphone power impact of continuous sensing applications at pre-installation time. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems* (SenSys '15), 31-44. http://dx.doi.org/10.1145/2809695.2809728
22. Chulhong Min, Youngki Lee, Chungkuk Yoo, Seungwoo Kang, Inseok Hwang, Junehwa Song. 2016. PowerForecaster: Predicting Power Impact of Mobile Sensing Applications at Pre-Installation Time. GetMobile: Mobile Comp. and Comm. 20, 1 (July 2016), 30-33. http://dx.doi.org/10.1145/2972413.2972424
23. Chulhong Min, Chungkuk Yoo, Inseok Hwang, Seungwoo Kang, Youngki Lee, Seungchul Lee, Pillsoon Park, Changhun Lee, Seungpyo Choi, and Junehwa Song. 2015. Sandra helps you learn: the more you walk, the more battery your phone drains. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (UbiComp '15). 421-432. http://dx.doi.org/10.1145/2750858.2807553
24. Radhika Mittal, Aman Kansal, and Ranveer Chandra. 2012. Empowering developers to estimate app energy consumption. In *Proceedings of the 18th annual international conference on Mobile computing and networking* (Mobicom '12). 317-328. http://dx.doi.org/10.1145/2348543.2348583
25. Jeongyeup Paek, Joongheon Kim, and Ramesh Govindan. 2010. Energy-efficient rate-adaptive GPS-based positioning for smartphones. *In Proceedings of the 8th international conference on Mobile systems, applications, and services* (MobiSys '10). 299-314. http://dx.doi.org/10.1145/1814433.1814463
26. Abhinav Pathak, Y. Charlie Hu, Ming Zhang, Paramvir Bahl, and Yi-Min Wang. 2011. Fine-grained power modeling for smartphones using system call tracing. In *Proceedings of the sixth conference on Computer systems* (EuroSys '11), 153-168. http://dx.doi.org/10.1145/1966445.1966460.
27. Abhinav Pathak, Abhilash Jindal, Y. Charlie Hu, and Samuel P. Midkiff. 2012. What is keeping my phone awake?: characterizing and detecting no-sleep energy bugs in smartphone apps. In *Proceedings of the 10th international conference on Mobile systems, applications, and services* (MobiSys '12), 267-280. http://dx.doi.org/10.1145/2307636.2307661
28. Kiran K. Rachuri, Cecilia Mascolo, Mirco Musolesi, and Peter J. Rentfrow. 2011. SociableSense: exploring the trade-offs of adaptive sampling and computation offloading for social sensing. In *Proceedings of the 17th annual international conference on Mobile computing and networking* (MobiCom '11), 73-84. http://dx.doi.org/10.1145/2030613.2030623
29. Lenin Ravindranath, Suman Nath, Jitendra Padhye, and Hari Balakrishnan. 2014. Automatic and scalable fault detection for mobile applications. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services* (MobiSys '14), 190-203. http://doi.acm.org/10.1145/2594368.2594377
30. Haichen Shen, Aruna Balasubramanian, Anthony LaMarca, and David Wetherall. 2015. Enhancing mobile apps to use sensor hubs without programmer effort. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (UbiComp '15), 227-238. http://dx.doi.org/10.1145/2750858.2804260
31. Lide Zhang, Mark S. Gordon, Robert P. Dick, Z. Morley Mao, Peter Dinda, and Lei Yang. 2012. ADEL: an automatic detector of energy leaks for smartphone applications. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis* (CODES+ISSS '12), 363-372. http://dx.doi.org/10.1145/2380445.2380503
32. Lide Zhang, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, Robert P. Dick, Zhuoqing Morley Mao, and Lei Yang. 2010. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis* (CODES/ISSS '10), 105-114. http://dx.doi.org/10.1145/1878961.1878982