UserScope: A Fine-grained Framework for Collecting Energy-related Smartphone User Contexts

Wonwoo Jung, Kwanghwan Kim, and Hojung Cha
Department of Computer Science
Yonsei University
Seoul, Korea
{wwjung, kwanghwan, hjcha}@cs.yonsei.ac.kr

***Abstract*—To prolong the battery lifetime of modern mobile devices, the energy management policy should be developed in a personalized way, adequately reflecting user context or the energy behavior of the user. The first step toward this personalization is to collect the relevant information, accurately and efficiently, from the device. This paper presents a fine-grained and low-overhead framework, called UserScope, which is designed to collect energy-related user contexts in Android smartphones. We classified energy-related smartphone usage and designed an appropriate set of monitoring parameters to collect from the system. The UserScope core is then implemented as a kernel module to collect all the necessary information in an event-driven manner. This kernel-level implementation ensures monitoring accuracy and low system overhead. UserScope also provides a data-sharing mechanism with which other software components in the system can easily interface. Our experiments show that UserScope accurately extracts energy-related system information with 0.8% CPU overhead. The practicality of UserScope is also validated with real deployment and subsequent analysis of the collected data.**

Keywords-smartphones; OS kernel; system monitoring techniques; user energy behavior

I. INTRODUCTION

Both hardware and software optimization are important to reduce the amount of energy consumed by smartphones and their applications. According to recent reports [1, 2], the energy consumption of a smartphone varies with different distribution of system software, even on the same device. This provides crucial evidence of the importance of software optimization for smartphone energy management.

Regarding recent work on smartphone energy management, several issues have been addressed and studied accordingly. Much effort has been made to model and estimate the energy consumption of smartphone applications, as well as the underlying hardware [3-6]. Some studies have tried to find cases of energy waste in smartphones and identify the causes of energy leak [7]. Various energy management policies for smartphones have also been developed to deal with their energy characteristics in system level [8] and even in OS level [9-12]. In short, researchers observe the energy consumption of a system, find problems related to energy waste, and develop management policies that can be applied effectively.

In general, the usage behavior of smartphones is widely different from user to user; hence, applying the same energy management policy to every user would not work in practice [13, 14]. Such diversity of user energy behavior implies the necessity of having to develop a personalized energy management scheme for the individual user. In a personalized policy development, the energy management system should accurately collect and understand the user's energy behavior. In addition, the acquired information should be shared effectively with other software components, such as the operating system, which instigates the management policy.

To collect user behavior related to energy consumption and provide it to the system effectively, several issues should be considered. First, the overhead incurred by the collection mechanism should be minimized, and accuracy maximized. The required data, therefore, should be gathered only at the exact moment that the relevant event occurs. Second, the collected information should be shared effectively with diverse software components, from the operating system to applications, which may use it for their own energy management schemes. Third, the target information to be collected in the smartphone should be defined precisely a priori. That is, the information should be rich and complete in terms of the energy consumption patterns of the user.

A number of tools have recently been developed to collect user behavior, especially for research purposes [13, 15, 16]. Most of the tools, however, have limited usage in understanding the user's energy behavior [14, 17], as the gathering of user data is typically conducted at a high level and in coarse-grained manner. Although the user-level approach (i.e., tools as an application form) has the potential advantage of easy deployment, the scheme has limitations when applied to energy management purposes. In the user-level approach, a considerable amount of information should be collected by the system. This is commonly accomplished by means of a polling technique, which acquires data at a predefined timing point, not at the moment the data is generated. Moreover, some information cannot even be collected at a user-level on certain platforms (e.g., the information on foreground application on Android). Thus, the user-level approach hardly guarantees the accuracy and completeness of information. The method can also lead to impermissible levels of energy overhead due to active system polling. The user-level data collection scheme is also

limited in sharing information with other system components, such as the one in the kernel. For instance, most energy management policies, such as dynamic voltage and frequency scaling (DVFS) or dynamic power management (DPM), are conducted at the kernel level. The collected data should be effectively shared with all the software components, from the kernel to the applications. The user-level approach is certainly restricted in this perspective.

In this paper, we propose UserScope, which is a fine-grained and efficient framework to collect user energy behavior at the kernel level. With the kernel-level approach, UserScope extracts information that cannot be collected at the user level. All information in our scheme is collected in an event-driven method, in which UserScope accurately collects a complete set of information with minimum energy overhead. Furthermore, the information that is obtained at the kernel level can be shared efficiently with any software component in the system. Thus, UserScope is a pragmatic framework for smartphones that provides detailed user information for energy management.

The contributions of our work are as follows:

- We propose a kernel-level framework to extract energy-related user information accurately and with minimum overhead.
- We provide an efficient mechanism to share information among diverse software components in the system.
- We categorize user behavior that is related to energy consumption and clarify its meaning in terms of smartphone energy consumption.

II. Related Work

Smartphones are highly personalized mobile computing devices. These features (i.e., personalization and mobility) have led to active study on the analysis of user behavior and the development of diverse monitoring tools. MyExperience [15] is a smartphone monitoring tool that collects various types of user behavior for Windows Mobile phones. The tool extracts the user's personal usage information, as well as system data. LiveLab [16] provides user behavior information, such as device usage information, for the iPhone. In particular, the tool allows the reconfiguration of the monitoring system during its deployment. SystemSens [13] collects information on user behavior for Android phones. For a large-scale deployment, the system uses a passive method, which does not require active participation by users, and limits target information types to run the monitoring application on Android phones. Based on such tools, research efforts have led to understanding the usage characteristics of smartphone users, especially in the context of analyzing user behavior related to energy consumption. Falaki et al. [14] studied the behavior of Android and Windows Mobile phone users. They found that it is difficult to classify users into small groups based on behavior characteristics; thus, they claimed the need for personalized management policies, such as the one based on a statistical usage model, which improve the user experience and reduce energy consumption. Olive and Keshav [17] collected battery usage statistics for BlackBerry users and analyzed energy-related behavior. Based on their results, they proposed a technique to predict battery residual.

From an implementation viewpoint, various user monitoring tools have been developed as user applications [13, 15, 16]. The user-level scheme might be suitable for developing loggers for high-level understanding of mobile users' behavior. However, the approach is not adequate to be used for other purpose such as energy management which should reflect the collected information. To be used more widely, the information should be shared with other components in the system, including kernel-level objects, but sharing the information is not easy with user-level approach. Moreover, the coarse-grained information and high monitoring overhead limit the user-level scheme to be used for only high-level log analysis. Certainly, the kernel-level scheme is more suitable to handle this kind of information [10]. Kernel-level monitoring techniques have been developed to collect detailed information based on an event-driven scheme, thus guaranteeing low overhead and high accuracy. For example, AppScope [3], a runtime energy estimation tool for Android applications, guarantees a high level of accuracy using a kernel-level scheme to monitor the usage of hardware components. However, its objective is to understand the behavior of applications in accessing the hardware resources, whereas our goal is to collect user behavior related to energy consumption, and to provide an efficient mechanism for energy management policy development.

TABLE I. Categorization of the Energy Parameters

Category	User Pattern of Smartphone Use		Representative Parameter
Implicit Energy Behavior	Environmental factors	Network Environment	*3G_Signal_Strength*
		User Moving Behavior	*3G_Cell_Info*
	Energy habit	Interaction	*Screen_State*
		Battery Charging Behavior	*Battery_Charging_State*
		System Reboot	*Power_State*
		Display Setting	*Display_Brightness*
Explicit Energy Behavior	Service preference	Network Service	*Network_Traffic*
		Calling Service	*Call_State*
		Location Service	*GPS_State*
		Camera Service	*Camera_State*
		Storage Media Service	*SDCard_State*
		Graphic Service	*GPU_State*
		Voice Service	*MIC_State*
	Application preference		*App_Foreground*

III. Extracting Energy Parameter

The objective of UserScope is to understand the energy behavior of smartphone users as accurately as possible and to share the information efficiently with other software components in the system. To achieve this goal, we designed a set of parameters to be collected from the system to cover the energy behavior of the user. Table I summarizes the categories of user behavior classified in our work. We classified energy-related behavior into two categories, according to whether the user's intention is involved.

TABLE II. PARAMETERS MONITORED BY USERSCOPE

Component	Parameter	Kernel Events	Accessibility by monitoring method			Event Type
			Polling (User Level)	Event Driven (User Level)	Event Driven (Kernel Level)	
CPU	*CPU_Usage*	CPU Frequency Switch	O	X	O	3
	CPU_Frequency	CPU Frequency Switch	O	X	O	2
Network	*Network_Traffic*	Send/Receive	O	X	O	1
	3G_Network_Type	Network Type Switch	O	X	O	2
	3G_Signal_Strength	Report for Signal Strength	O	O	O	2
	3G_Cell_Info	Cell Information	O	O	O	2
	WiFi_Scanning	Wi-Fi Scanning Request	O	X	O	1
	WiFi_Signal_Strength	Report for Signal Strength	O	X	O	2
Display	*Display_Brightness*	Brightness Level Change	O	X	O	2
	Framebuffer	Framebuffer Content Change	X	X	O	3
	Screen_State	Screen On/Off	O	O	O	1
	Screen_Lock_State	Screen Lock and Unlock	X	X	O	1
GPU	*GPU_State*	Access to GPU	X	X	O	1
GPS	*GPS_State*	GPS On/Off	O	O	O	1
Call	*Call_State*	Incall/Outcall Event	O	O	O	2
Memory	*Memory_Usage*	Process Creation and Destruction	O	X	O	3
Battery	*Battery_State*	Battery State Information	O	O	O	2
	Battery_Charing_State	Battery Charging State Change	O	O	O	1
SD Card	*SDCard_State*	Read/Write to SD Card	X	X	O	1
Camera	*Camera_State*	Camera On/Off	X	X	O	1
MIC	*MIC_State*	MIC On/Off	X	X	O	1
Power	*Power_State*	Device On/Off	O	O	O	1
App	*App_Foreground*	Foreground Application Information	O	X	O	2

A. Implicit Energy Behavior

This category of energy behavior includes the situations where the user's intentions are not practically involved in terms of energy usage. This category is further classified into either user's environment dependency or user's energy habit.

The energy-related behavior of the user is often influenced by the factors related to surrounding environments, such as network conditions, which change with the user's whereabouts. In other words, it is information about the user's environment or status that affects energy consumption. For example, the unnecessary attempts of frequent Wi-Fi scanning in a place where no Wi-Fi access points are found consume a significant amount of smartphone energy, and this can be monitored with UserScope.

Energy habits are user behaviors inherent in the user's device usage, such as the habit of repetitively switching on and off the screen. This type of information is necessary in order to develop a personalized energy management policy for the individual user. For instance, a user who does not charge his battery often would have less freedom in terms of energy use. Thus, we might apply a more aggressive energy management policy (e.g., prohibiting energy-intensive services) to a user who charges his battery less frequently. Although the low power mode of operation is available with current smartphones, we can improve energy management policies with UserScope in a more intelligent and personalized way.

B. Explicit Energy Behavior

This category includes the situation where the user's intention is deeply involved in using the device and applications. We classify this type into two classes: user's service preference and application preference.

Service preference refers to the usage statistics on the services that the device can provide. With this information, we can know the types of services the user mainly acquires with the smartphone. Table I illustrates seven classes we have categorized for the services provided by smartphones. According to our classification, each service is related to the energy consumption of a particular set of hardware components. That is, the service preference provides information about how the user utilizes the hardware components, as well as information on the purpose of using the device.

Application preference is the information about the usage statistics of the applications installed in the device. We only consider the applications running in the foreground; thus, we obtain the user's intention exactly [13]. Application preference includes detailed information about the application itself; hence, we uncover the user's application tendency more concretely.

IV. USERSCOPE

UserScope is a kernel-level software framework that monitors the user's energy behavior while guaranteeing high accuracy and low overhead. UserScope captures the energy behavior in an event-driven manner and provides information to software components in the system. Fig. 1 shows an overview of the UserScope framework. To collect user information, UserScope uses kprobes [18], which provide a mechanism to hook kernel routines dynamically. Upon the occurrence of an energy-related event, UserScope detects the event via the hooking mechanism and extracts the related parameter values. The collected parameters are stored

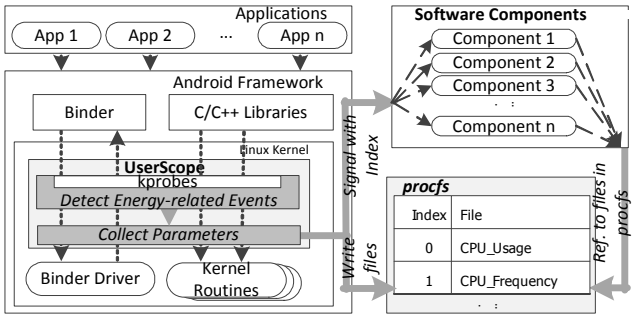


Figure 2. Overview of the UserScope Framework

in */proc* file system (*procfs*), and used by other software components that would require them.

A. Kernel-level Event Detection

Because UserScope is a kernel module that works as part of the operating system, it should work accurately and efficiently, with low overhead. UserScope collects information inside the kernel in an event-based manner. The event-based method accesses information only at the moment that related events occur. The method provides high accuracy with low overhead, compared to the polling method, which collects information at predefined intervals. All the events detected by UserScope are the kernel function calls related to the user's energy behavior. That is, the user's energy behavior triggers a chain of function calls in the kernel.

In the Android framework, UserScope can utilize three types of kernel functions as events associated with user behavior. The first type is the *binder_transaction()* function, which is a core of the Android's binder framework. Android uses the binder as an RPC/IPC mechanism to exchange various types of messages between processes. The message types are determined with the parameters of *binder_transaction()*; therefore, the input parameters should be analyzed. The second type of kernel function is internal functions of the device drivers. The driver functions generally depend on the hardware components that are equipped in the device. Hence, event detection with these functions may lead to dependency on the hardware platform. The third type is general functions of the standard Linux kernel. They are relatively independent of the Android framework or hardware platform. For portability issues, UserScope mainly utilizes the third type of function as the events to detect energy behavior. However, the current version of UserScope also uses the first or second type in some cases, for a fast prototyping of the proposed system.

B. Extracting Kernel Parameters

Table II lists the events that are related to the classified set of parameters in Table I. For each event, UserScope collects the related parameters in a predefined manner for each event type. Fig. 2 illustrates the overall operation associated with different types of events. Every event is classified into three types. We now define the event types and explain the parameter extraction method for each type.

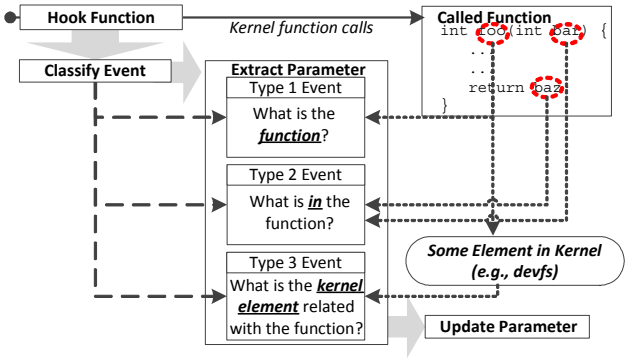


Figure 1. Event Type and Parameter Extraction

1) Type 1 event: This type of event informs UserScope of the parameter values once the related parameters have been updated. Each event (i.e., a function call) represents a specific value of an associated parameter. The on/off state of the hardware component is a typical example of this type of parameter. UserScope only considers the function that is called, and it updates the parameter with a value predefined for the function call. For example, the kernel accesses the SD card with *do_sync_read()* and *do_sync_write()* functions. UserScope detects the calls and updates the *SDCard_Usage* parameter, which represents the usage state of the SD Card.

2) Type 2 event: The second type of event informs UserScope of the fact that parameters are updated, after which the updated values are defined in the internal information of the event (i.e., called function). That is, the parameter values should be taken from the input parameters or return values of the invoked functions. UserScope updates the parameters by extracting the values in the function. For example, CPU frequency is defined in the input parameter *cpufreq_cpu_put()* (i.e., the cpufreq_policy structure); thus, UserScope updates the *CPU_Frequency* parameter with the member variable of *cpufreq_policy*.

3) Type 3 event: This type only notifies that the associated parameters have been updated. In this case, UserScope collects the actual values of some elements in the kernel, such as the */dev* file system (*devfs*) and the internal data structures of the kernel.

We used this type of event for the parameters that continuously change, such as memory usage. In our implementation, three types of parameters are associated with type 3 events: *CPU_Usage*, *Framebuffer*, and *Memory_Usage*. UserScope collects the *CPU_Usage* value only at the moment *cpufreq_cpu_put()* is invoked. That is, it extracts the CPU usage information when the CPU frequency actually changes. At that point, UserScope accesses the kernel's internal data structure, which maintains the information related to the CPU, via the *kstat_cpu()* macro, and then updates the value of *CPU_Usage*. Regarding framebuffer information, changes in framebuffer contents are detected by analyzing the input parameters of the *binder_transaction()* function. UserScope uses a similar

technique to extract the framebuffer information [8, 19] and updates the *Framebuffer* parameter in *devfs*. For *Memory_Usage*, the parameter is updated in UserScope only when processes are created and destroyed, by calling the related kernel functions, such as *do_fork()* and *do_exit()*. UserScope accesses the kernel data structure via the *si_meminfo()* function, and then updates the value of *Memory_Usage*.

C. Information-sharing Mechanism

To share information with other software components, UserScope utilizes a POSIX signal and *procfs*. UserScope writes the extracted parameters into predefined files in *procfs* and transmits signals to software components, which are preregistered to acquire the information. Every signal is delivered to the registered components with an index indicating the event type (i.e., updated file in *procfs*). The components then acquire the information from the updated file. With our sharing mechanism, the software components access the collected information at the exact moment it is updated.

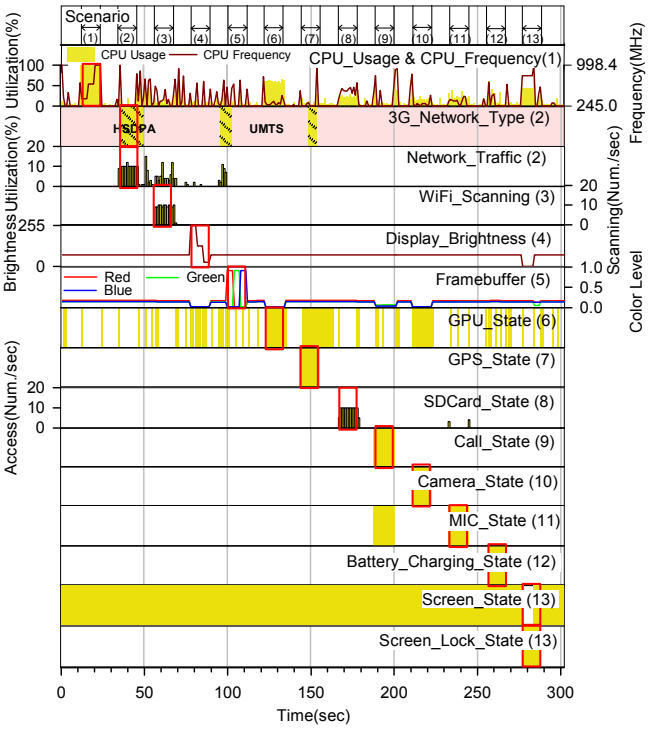
V. EVALUATION

UserScope is implemented as a kernel module on the Android platform. We evaluated UserScope, focusing particularly on its accuracy and overhead. The overall functionality of UserScope is validated, including its event-detection mechanism, parameter-value collection, and information-sharing attributes. All experiments were conducted on a Google Nexus One running Android 2.3.

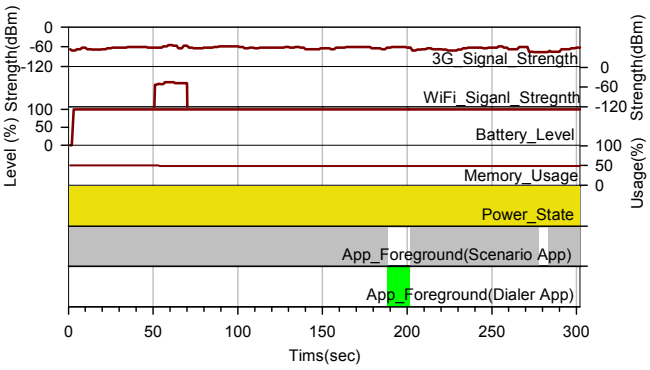
A. Experimental Methodology

To validate the functionality and overhead of UserScope, we first conducted experiments based on a predefined sequence of event-generation scenario. That is, specific events were generated to change parameter values with predefined sequences at specific times. With this scheme, we determined the exact moments and types of events that occurred. The information was then used as the ground truth for the accuracy evaluation. This scenario-based scheme is also suitable for overhead evaluation, as we can repeatedly generate the same workload with the same scenario.

Fig. 3 shows the predefined event-generation scenario, as well as the trace data obtained with UserScope while running the scenario for 300 seconds. The application scenario involves various events with different parameter characteristics. The parameters collected by UserScope are classified into three types. The first is the parameter that is automatically generated by the application. The parameters in events 1–11 in Fig. 3(a) belong to this type. The second type of parameter is from the associated events that are manually generated by the experimenters. Events 12 and 13 in Fig. 3(a) belong to this type. The third type is the parameter that does not have associated events. Fig. 3(b) represents this type of parameter. For example, the network environment and battery level are hard to control in a laboratory environment. We excluded such events from the explicit scenarios, but instead, observed the values with UserScope.



(a) Event generation scenario



(b) Parameters automatically collected by UserScope

Figure 3. Validation of UserScope

We implemented an application that automatically generated the events based on this scenario. We also wrote an application to log the collected data.

B. Monitoring Accuracy

We first checked whether UserScope collected the parameter values at the exact moment the associated events occurred and shared them with a registered software component (i.e., a logging application in this case). Fig. 3(a) shows that UserScope accurately tracked the parameter values that changed with the events in the scenario (red-boxed areas in the figure). For instance, with event 1 in the figure, we changed the CPU frequency three times (i.e., 384.0 MHz, 652.0 MHz, and 998.4 MHz) while maintaining 100% utilization. The results show that UserScope detected the changes of the frequency at the exact moment and updated the *CPU_Frequency* parameter with the collected

values. In addition, UserScope correctly analyzed CPU utilization in previous periods and updated the *CPU_Utilization* parameter accordingly.

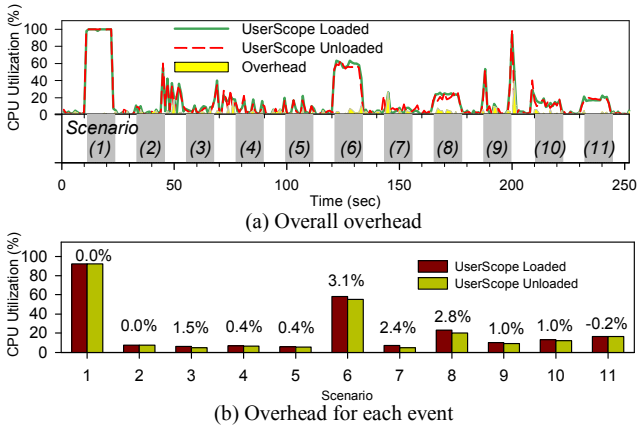
We noticed that UserScope detected changes in some parameters that were not explicitly associated with the events in the scenario. *GPU_State* is a good example; according to our scenario, the GPU events were generated at time section 120~132 (event 6), and they were accurately detected by UserScope. In addition to this scheduled event, however, UserScope captured the activation of the GPU at other sections. That is, at time section 142~154, UserScope detected the GPU usage when the GPS component was turned on (event 7); hence, the GPS icon was displayed on the screen. In addition, at time section 208~220, UserScope captured GPU activity by detecting camera usage. Apart from this, UserScope found that the GPU was intermittently activated with other events that changed the displayed content.

We observed similar behavior in regard to network traffic. Our scenario transmitted packets at time section 34~46 via a cellular network connection (event 2). This activity was observed accurately, but UserScope reported additional network traffic generated by other events. For example, at about 50 seconds, network traffic was monitored, yet this was not caused by our application. At that moment, we had switched the connected network type from cellular to Wi-Fi. Similarly, the reverse change of network connection from Wi-Fi to cellular was observed at around 70 seconds. In fact, these were caused by the packets generated by certain applications in the system, which intended to restore their connections.

C. Overhead

Next, we analyzed the overhead of UserScope in terms of energy consumption. Energy consumption is generally proportional to the utilization of each hardware component [3-7]. The energy overhead of UserScope is mostly dependent upon CPU usage, as UserScope primarily consumes the CPU resource.

To analyze the overhead of UserScope, we obtained the difference in CPU utilization when running the application with and without UserScope. In both cases, we used the same scenario, which excluded the manual part of the scenario used for accuracy evaluations. Fig. 4 shows the overhead of UserScope, and Fig. 4(a) shows the CPU utilizations and the overhead during the entire experiment. Overall, UserScope consumed 0.8% more CPU resource (15.7% average utilization with UserScope and 14.9% without UserScope). This extremely small value means that UserScope does not practically require additional CPU utilization running the entire scenario in our experiment. However, it does not mean that UserScope can operate without any additional cost. Fig. 4(b) shows the overhead of each scenario, and UserScope indeed consumed additional CPU resources in several cases; in particular, the overheads of events 6 and 8 are greater than the others. This overhead depends on the frequency of event occurrence. Event 6 triggered GPU operations and frequently changed the displayed contents, and event 8 generated SD card access



(a) Overall overhead

(b) Overhead for each event

Figure 4. Overhead Analysis of UserScope

operations; these types of operations could occur ten or more times per second. In a practical situation, however, we still expect that UserScope would exhibit acceptable overhead, as it runs in an event-driven manner, and hence, is not activated when the CPU is in an idle state; in addition, the GPU is only activated when the user actively interacts with the smartphone. Considering the typical usage of smartphones, the device is idle and the screen is off most of the time. We also may expect that the usage rate of SD card is generally low (i.e., about one hour a day, according to the case study in Section 6). In summary, the operational overhead of UserScope is acceptable; therefore, it can be operated efficiently as part of the OS kernel.

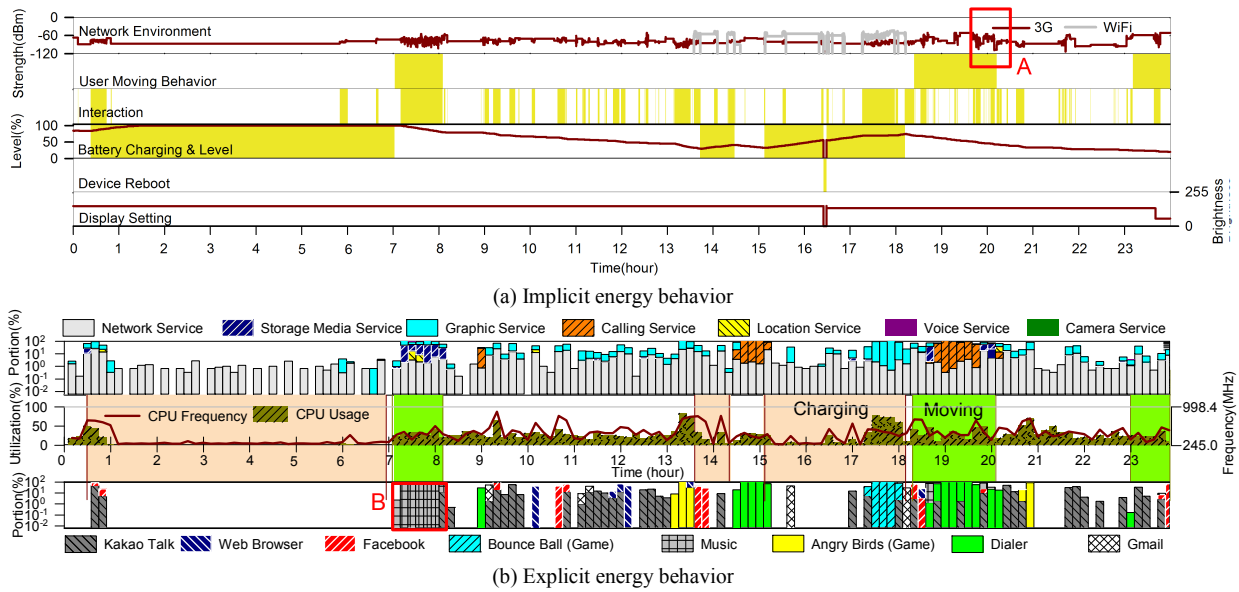
VI. Case Study: UserScope Application

We developed a UserScope-based application, which is similar in functionality to other tools [13, 15, 16] for collecting smartphone user behavior. In order to analyze the energy behavior of users in real life, we tracked the actual usage of a smartphone user who is a male graduate student using the application. The participant was traced for five working days, from Monday to Friday.

A. Results

Fig. 5 presents the overall energy behavior of the participant on a typical day. Note that the data collected for five days revealed that the participant maintained relatively regular life patterns during the week; hence, we only show the trace for one day in the figure.

Fig. 5(a) shows the implicit energy behavior of the participant. The participant was using his smartphone in network environments that were generally seamless and stable, except for some specific times (the red box A in Fig. 5(a)). He was also connected to Wi-Fi networks wherever possible. A notable point is that, even while connected via Wi-Fi, the device maintained a connection with a cellular network and collected information, such as signal strength. This is probably because the device was accessing the cellular network for some services, such as MMS. We also found that the participant frequently interacted with the device and actively utilized applications during most of the interaction time (94.8% of the total interaction time).



(a) Implicit energy behavior

(b) Explicit energy behavior

Figure 5. 24-hour Trace of User Energy Behavior

Regarding battery charging behavior, the participant always charged the battery at night, probably while he was sleeping, and intermittently attempted to charge the phone during the day, regardless of battery level.

Fig. 5(b) shows the explicit energy behavior of the participant. For the analysis, we averaged the usage of each service and application over ten minutes. During an entire day, the graphic service was frequently activated during the daytime, whereas the network service was persistently activated, even during sleeping hours. Because no application was in the foreground at that time, the network activity was probably caused by the traffic automatically generated by some background services. Compared to these, other services were intermittently used for a few specific times during the day. Regarding application usage, the participant activated various types of applications during the course of the day. An interesting observation is that the participant's preference for services and applications changed as the spatiotemporal situation around him changed, as shown in Fig. 5(b).

B. Discussion: Potential of UserScope

Our findings regarding this case study imply that context-aware and personalized energy management may be, indeed, possible. In the following, we discuss the possibility of energy management policy using the UserScope framework, based on our observations.

First, we found that smartphone usage patterns depend on the user's spatiotemporal situation. The user typically prefers a few specific services and applications in a particular situation. That is, user preference significantly affects the energy consumption of the device, as the energy-related properties of the device depend on the situation. For example, the participant in our case actively used music applications during moving hours. Utilizing this personal behavior, the energy management policy could optimize a memory-management scheme [20], and hence, obtain better energy efficiency, by assigning more memory space to the application to buffer streamed data.

Second, the participant usually accesses the network in stable environments, yet encounters sporadic difficulties in connection, especially in a transit period (the red box A in Fig. 5(a)). Accessing the network in this situation would cause a significant overhead in the system (i.e., energy waste), due to the severe level of packet retransmission. Using the network-related information obtained with UserScope, we can detect this situation and possibly delay the transmissions until the user enters a network-friendly environment. This personalized management would certainly reduce the energy consumption of the device.

Third, kernel-level information can be used to develop a customized power governor. The Linux kernel presently makes use of processor utilization to implement the CPU frequency governor. The utilization-based policy is often inefficient, depending on the workload type or other system parameters [21]. The diverse types of context information obtained with UserScope can be used to implement a customized frequency governor for CPU power management. For instance, when a user is accessing storage media service (the red box B in Fig. 5(b)), the CPU clocks could stay low. If the device is found accessing the CPU resource intensively, the CPU clock should stay high, regardless of its utilization. The decision should be based on understanding the user's energy behavior.

Last, apart from direct energy management with user context, the system-level information acquired with UserScope can be used to detect abnormal behavior of the device, and even energy bugs [22]. UserScope continuously monitors the system status and builds up user context. The information is then utilized to develop an advanced system-monitoring tool. In our case study, the participant hardly interacted with the device at night. Any noticeable energy-related event detected with UserScope should then be further analyzed for its legitimacy.

VII. CONCLUSIONS

An advanced energy management for smartphone should employ a strategy that reflects energy-related behavior specific to the individual user. To achieve this personalized scheme, the energy behavior of the user should be acquired efficiently and accurately. In this paper, we presented a kernel-level framework that extracts the energy-related user context for Android-based smartphones. Based on the classification of energy-related usage types, we determined the parameter set to be extracted from the smartphone. UserScope is then implemented as a kernel module to collect information in an event-driven manner at the kernel level. According to our experiments, the proposed scheme guarantees monitoring accuracy, as well as low system overhead, implying that UserScope is effectively introduced as part of the OS kernel. UserScope includes an information-sharing mechanism through which software components can easily be interfaced. We evaluated the efficiency of the sharing mechanism. We believe that a kernel-level framework such as UserScope, which extracts user energy behavior, is essential for the development of a personalized energy management scheme for smartphone users. In our future work, we plan to develop personalized energy management policies based on the results acquired with our experience with UserScope.

ACKNOWLEDGMENT

This work was supported by a grant from the National Research Foundation of Korea (NRF), funded by the Korean government, Ministry of Education, Science and Technology under Grant (No.2011-0015332).

REFERENCES

[1] iOS 5 Battery Problems, http://www.huffingtonpost.com/2011/11/11/ios-5-battery-problems-apple-iphone_n_1088691.html

[2] Motorola Droid RAZR/RAZR MAXX ICS Bug Fix To Release in Mid-August, http://www.fun47.com/motorola-droid-razrrazr-maxx-ics-bug-fix-to-release-in-mid-august/

[3] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha, "AppScope: Application Energy Metering Framework for Android Smartphone Using Kernel Activity Monitoring," Proc. USENIX Annual Technical Conference (USENIX ATC'12), USENIX, Jun. 2012, pp. 387–400.

[4] W. Jung, C. Kang, C. Yoon, D. Kim, and H. Cha, "DevScope: A Nonintrusive and Online Power Analysis Tool for Smartphone Hardware Components," Proc. the 10th IEEE/ACM/IFIP Int. Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS'12), Tampere, Finland, Oct. 2012.

[5] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones," in Proc. the 8th IEEE/ACM/IFIP Int. Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS'10), Scottsdale, AZ, USA, Oct. 2010, pp. 105–114.

[6] A. Pathak, Y. C. Hu, and M. Zhang, "Fine-grained Power Modeling for Smartphones Using System Call Tracing," Proc. the 6th European Conf. Computer Systems (EuroSys'11), Salzburg, Austria, Apr. 2011, pp. 153–168.

[7] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the energy spent inside my app? Fine Grained Energy Accounting on Smartphones with Eprof," Proc. the 7th European Conf. Computer Systems (EuroSys'12), Bern, Switzerland, Apr. 2012, pp. 29–42.

[8] M. Dong and L. Zhong, "Chameleon: A Color-Adaptive Web Browser for Mobile OLED Displays," IEEE Transaction on Mobile Computing (TMC), vol. 11, May 2012, pp. 724–738.

[9] D. C. Snowdon, E. L. Sueur, S. M. Petters, and G. Heiser, "Koala: A Platform for OS-level Power Management," Proc. the 4th European Conf. Computer Systems (EuroSys'09), Nuremberg, Germany, Apr. 2009, pp. 289–302.

[10] D. Chu, A. Kansal, J. Liu and F. Zhao, 2011, "Mobile Apps: It's Time to Move Up to CondOS," Proc. the 13th Workshop on Hot Topics in Operating systems (HotOS XIII), Napa, CA, USA, May 2011.

[11] H. Zeng, C. Ellis, A. Lebeck and A. Vahdat, "ECOSystem: Managing Energy as a First Class Operating System Resource," ACM SIGPLAN Notices, vol. 37, Oct. 2002, pp. 123–132.

[12] A. Roy, S. Rumble, R. Stutsman, P. Levis, D. Mazières and N. Zeldovich, "Energy Management in Mobile Devices with Cinder Operating System," Proc. the 6th ACM European Conference on Computer Systems (EuroSys'11), Salzburg, Austria, Apr. 2011, pp. 139–152.

[13] H. Falaki, R. Mahajan, and D. Estrin, "SystemSens: A Tool for Monitoring Usage in Smartphone Research Deployments," Proc. the 6th ACM International Workshop on Mobility in the Evolving Internet Architecture (MobiArch'11), Washington, D.C., USA, Jun. 2011. pp. 25–30.

[14] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin, "Diversity in Smartphone Usage," Proc. the 8th Annual International Conference on Mobile Systems, Applications and Service (MobiSys'10), San Francisco, CA, USA, Jun. 2010, pp. 179–194.

[15] J. Froehlich, M. Y. Chen, S. Consolvo, B. Harrison and J. A. Landay, "MyExperience: A System for in situ Tracing and Capturing of User Feedback on Mobile Phones," Proc. the 5th International Conference on Mobile Systems, Applications and Services (MobiSys'07), San Juan, Puerto Rico, Jun. 2007, pp. 57–70.

[16] C. Shepard, A. Rahmati, C. Tossell, L. Zhong and P. Kortum, "LiveLab: Measuring Wireless Networks and Smartphone Users in the Field," ACM SIGMETRICS Performance Evaluation Review, vol. 38, Dec. 2010, pp. 15–20.

[17] E. Oliver and S. Keshav, "An Empirical Approach to Smartphone Energy Level Prediction," Proc. the 13th International Conference on Ubiquitous Computing (UbiComp'11), Beijing, China, Sep. 2011, pp. 345–354.

[18] Kprobes, http://www.kernel.org/doc/Documentation/kprobes.txt.

[19] D. Kim, W. Jung and H. Cha, "Runtime Energy Estimation of Mobile AMOLED Displays," Proc. 2013 Design, Automation & Test Europe (DATE'13), Grenoble, France, Mar. 2013.

[20] T. Yang, D. Chu, D. Ganesan, A. Kansal and J. Liu, "Fast App Launching for Mobile Devices using Predictive User Context," Proc. the 10th International Conference on Mobile systems, applications and services (MobiSys'12), Low Wood Bay, Lake District, UK, Jun. 2012.

[21] E. L. Sueur and G. Heiser, "Slow down or sleep, that is the question," Proc. USENIX Annual Technical Conference (USENIX ATC'11), Portland, OR, USA, Jun. 2011.

[22] A. Pathak, Y. C. Hu and M. Zhang, "Bootstrapping Energy Debugging on Smartphones: A First Look at Energy Bugs in Mobile Devices," Proc. the 10th ACM Workshop on Hot Topics in Networks (HotNets'11), Cambridge, MA, USA, Nov. 2011.