Automated Power Model Generation Method for Smartphones

Jemin Lee, Hyunwoo Joe, and Hyungshin Kim, *Member*, IEEE

Abstract — *This paper presents an automated power model generation method for smartphones. Having an accurate power model of mobile devices can help identify the sources of fast discharge of the battery and provide opportunities for optimizing battery use. However, current methods rely on either external measurement equipment or internal current sensor support. This study presents a novel method for power modeling that exploits the usage patterns stored on the phone. This method can be applied to any mobile device regardless of the manufacturer and the model of the device. A power consumption analysis tool, and an artificial usage pattern generator tool have been developed for power modeling. The proposed method is able to generate an accurate power model of various smartphones even without detailed knowledge of the phone hardware*[1].

Index Terms — power model, smartphone, power estimation, green computing.

I. INTRODUCTION

Smartphones are widespread and the number of smartphone users is continuously increasing. One of the most nerve-wracking concerns of users is the fear of full battery discharge. Many users do not have a proper understanding of the relationship between power consumption and currently running applications. That understanding often depends on monitoring and analyzing the power consumption of each hardware component. Accurate knowledge of the power consumption of the target device is also beneficial to application developers. Many power saving efforts in consumer electronic devices have been reported [1], [2]. In addition to the performance and resource monitoring tools available from the current development environment, energy consumption data can guide developers to energy efficient applications.

The power model is one of the key elements for estimating and analyzing the power consumption of the system. Research on power modeling has been reported not only for mobile consumer electronics (CE) devices but also for sensor networks [3]. In the mobile CE devices, various approaches for generating power models have been reported. They can be classified as either manual or automated methods.

Taking the manual approach, researchers reported accurate power model generation methods in a mobile device [4], [5]. However, their methods required very long times for model generation due to their complex manual procedures. They had to install external equipment for current measurement and to monitor all the system state changes. Since each smartphone vendor adopts different hardware components for their products, different power models need to be generated for each phone. Even smartphones having the same processor and LCD show power variations of up to 62% [6]. So, mobile devices such as smartphones, which have a wide range of vendors and products, all require separate power models. Each of them needs to go through all the manual modeling procedures and therefore this approach becomes infeasible.

Automated model generation methods execute software on the target to profile the power consumption of each hardware component. They provide a less accurate model than the manual ones but they have the advantage of faster model generation time. Zhang *et al* [6] proposed using a phone's internal voltage sensor and training software. They defined states for each hardware component and then executed a set of training programs for each defined state. The power consumed during the execution period was sampled to generate the power model. Having this separate operation to generate a model is obtrusive to average phone users. Due to the complexity of the model generation process, it is cumbersome to apply the method to different smartphones having different hardware components inside. Also, this method requires deep understanding of the hardware components to design the training program set, which is another main reason for the limited applicability of the tool. Additionally, this method needs a discharge voltage curve to obtain the result. This discharge voltage curve varies by external factors such as battery aging effect and temperature changes. Therefore, it takes a recurring effort to update the curve.

In this paper, a method is proposed which exploits a user's usage pattern, which is logged during the phone's normal operation. Here, usage pattern means an applications' hardware access profile, such as CPU utilization and display ON time, stored on the system log file. The proposed method uses only the battery level indicator and voltage sensor on the phone. These are standard functionalities supported by every mobile phone and hence, this method is widely applicable.

The present paper extends previous work [7]. Compared to the previous work [7], the proposed method improves accuracy by additionally modeling the power consumption characteristics of the Wi-Fi component. Moreover, a usage pattern generator has been developed for collecting many valid data. This paper provides further evaluation results with six applications on two smartphones.

[1]This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2011-0013115).

Jemin Lee, Hyunwoo Joe, and Hyungshin Kim are with department of computer science and engineering, Chungnam National University, Daejeon, 305-764, South Korea. E-mail: {leejaymin, jhwzero, hyungshin }@cnu.ac.kr

Contributed Paper
Manuscript received 04/01/14
Current version published 06/23/14
Electronic version published 06/23/14.

The proposed method divides logged data into a number of small time windows called "chunks", as was used in other research [8]. A chunk is a usage pattern segment and power consumption data that occurs during 1% battery discharge. Chunks are evaluated and collected for model generation. If more valid chunks are collected, a more accurate model is achieved. The proposed method can be applied without detailed knowledge of the underlying hardware of the phone. A mobile application named PowerDoctor has been developed. It collects and analyzes the user's usage pattern. Also, a tool named UPG (usage pattern generator) is developed for generating usage patterns. It makes it possible to easily collect many valid chunks by synchronizing the hardware component behavior with the update rate of the battery level. Therefore, users can generate a power model much more effectively.

For demonstrating the applicability of the proposed method, power models have been generated for two well-known smartphones. For this initial study, the power model consists of three components: CPU, display and Wi-Fi. Experimental results show that the power model achieved an average accuracy of 92% for five applications. This proves that the proposed approach is feasible.

The contributions of this paper can be summarized as follows:

- A new automatic power modeling method that exploits the usage patterns stored on the phone is proposed. This pattern analysis can provide similar results to those achieved from a training program approach, such as PowerTutor.
- The proposed method does not require either external equipment or an additional internal current sensor. It uses only common hardware on the phones to generate the model.
- The proposed method has better applicability than previous works. Power models were generated for two well-known phones showing accurate results.
- This paper provides two tools, named PowerDoctor and UPG. These tools can generate power models for diverse smartphones and analyze the power consumption of various mobile applications.

The remainder of this paper is organized as follows. Section II explains related works. In Section III, the proposed modeling framework and the generated power model are described. Section IV explains the usage pattern generator that augments the real usage pattern collected on the phone. The experimental results addressing the accuracy and the applicability of the proposed method are presented in Section V. The power analysis and usage pattern generation tools are described in Section VI. Finally, Section VII concludes this paper.

II. Related Works

Methods for generating power models for smartphones are categorized into manual and automated methods. Several prior works using manual methods involved the use of external equipment [4], [5]. Shye *et al* [4] proposed a utilization-based power model. Pathak *et al* [5] proposed a finite state machine (FSM) based power model using system call to overcome the limitation of the utilization based power model. With these methods, however, it is practically very hard to construct individual power models for each smartphone because of the need to use external equipment.

In contrast, automated methods have been researched [6], [9], [10], [11]. Zhang *et al* [6] proposed an application power estimation approach that uses utilization based power model, which is generated by using an internal fuel gauge sensor and discharge voltage curve. Kim *et al* [11] overcame the limitation of PowerTutor [6] which considered only one CPU core and linear power consumption for the display. They proposed a more improved power estimation technique than PowerTutor, considering a multi-core smartphone and the non-linear power consumption characteristics typical of displays and 3G modules. These approaches depended on a battery discharge curve and training programs to generate the power model. Dong *et al* [9] proposed an accurate self-modeling mechanism without external equipment. This method is able to achieve high accuracy and rate (100 Hz). However, their power model targets whole smartphone systems. Jung *et al* [10] suggested an autonomous power modeling tool for smartphones, which overcomes the limitations of internal sensors. Their method uses a special scenario for generating the power model, which controls components according to the update rate of internal sensors. However, they use a current sensor that is included in only a few smartphones. Thus, this method is hardly applicable to a diversity of smartphones.

The proposed method does not use additional equipment or an internal current sensor to generate a model generation, yet it does not sacrifice accuracy. It only uses the smartphone's system-provided data, data which is common to every smartphone. Usage patterns logged on the phone are analyzed for automated power model extraction. Also, to reduce the time needed to generate the power model, it uses a usage pattern generator which allows the easy collection of collect valid chunks from the phone.

III. Power Model Generation

The power modeling framework is shown in Fig. 1. It uses the PowerDoctor application which collects usage pattern data and analyzes them by chunks.

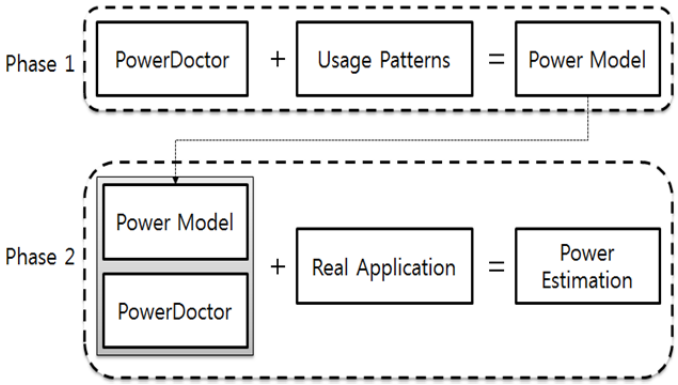


Fig. 1. Power modeling and estimation framework. Note that the proposed framework consists of two phases: one for model generation and the other for power estimation of applications using the generated model.

PowerDoctor is executed during the normal use of the phone. It profiles activities of other applications and a power model of the hardware components is derived. Then, the power consumption of applications can be estimated using the generated power model and PowerDoctor.

PowerDoctor executes two mobile services. First, it samples a phone user's usage pattern every second. When it detects a battery level decrease, it assembles a current chunk and stores it in the database. The second service is to estimate the power consumption of the phone by correlating the power model and profile data. This enables a real time power estimation of the phone. The estimation values are stored in a file system for further analysis.

The power model is derived from three steps. At step 1, PowerDoctor monitors the operating system's power states and maintains the data structure for computing the power consumption of the hardware component. It records the data structure only when it detects a battery level change. The recorded data structure during a 1% capacity decrease in battery level is a chunk. During step 2, PowerDoctor filters out invalid chunks from the database. At step 3, the power model of each hardware component is computed from a regression analysis of the filtered chunks. Fig. 2 shows the proposed power modeling steps. Detailed operation for each step is explained below.

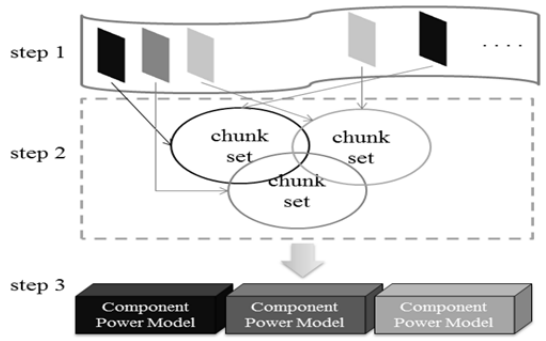


Fig. 2. Power model generation steps. Chunks are sampled from the phone's log file in step 1. The collected chunks are further filtered out to remove invalid samples in step 2. Regression analysis is performed in step 3 to generate a power model for each hardware component.

TABLE I
CHUNK DATA STRUCTURE

Chunk element	Detailed description
average_power	average power consumption of the phone
average_CPU_util	average CPU utilization
std_dev_CPU_util	standard deviation of CPU utilizations
display_on_time	display ON time during the chunk
average_brightness	average display brightness
std_dev_brightness	standard deviation of display brightness values
Wi-Fi_on_rate	Wi-Fi ON time during the chunk
average_packet_rate	average number of packets sent and received
std_dev_packet_rate	standard deviation of number of packets sent and received
average_battery_voltage	average battery voltage during the chunk period
capacity_range	capacity level when the chunk is recorded
chunk_length	time spent for the 1% capacity drop

A. Step 1: Generation of Chunks

Chunks are generated from runtime hardware states such as CPU utilization, display on/off status, and the number of transmitted or received Wi-Fi packets. Those runtime state data can be defined as the usage pattern of the phone during the chunk generation phase. PowerDoctor periodically checks if the battery capacity has changed. Until that change is detected, it saves the current hardware status as the current usage pattern. When a battery capacity change is detected, a chunk data structure is created and saved onto the database on the phone. PowerDoctor continues generating chunks until it has enough chunks to generate a power model. Table I shows the collected chunk data structure. A chunk consists of runtime usage pattern and power-related data during the chunk period.

The average consumed power in a chunk is computed as follows,

$$\text{Power}_{\text{avg}} = \frac{\text{Battery Capacity(Ah)}/100}{\text{Chunk Length}} \times \text{Voltage}_{\text{avg}}. \quad (1)$$

A battery capacity of (1) is the nominal battery capacity value of the phone.

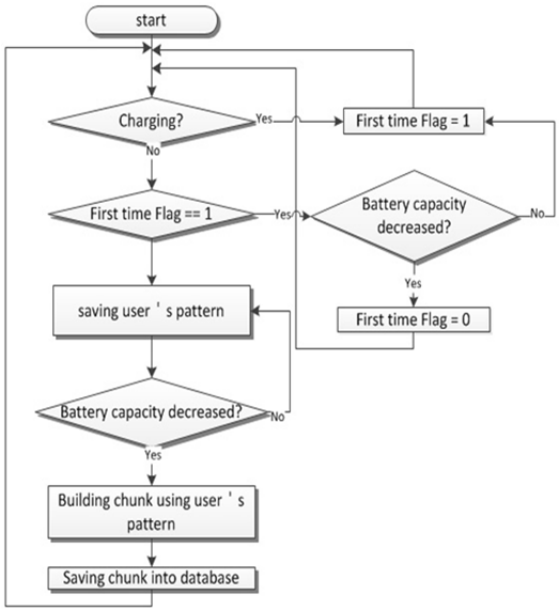


Fig. 3. Flow chart of chunk generation step. This chart contains early operations to skip the erroneous first chunk.

Fig. 3 shows the actual flow chart for chunk generation. It contains an initial stage for avoiding an erroneous first chunk. After bypassing the first chunk, PowerDoctor starts collecting chunks from the capacity change detection phase. When the phone is charging, a usage pattern is not recorded.

B. Step 2: Filtering out invalid chunks

Chunks are the hardware status and power data set collected during a 1% battery capacity drop. Smartphones consist of various hardware components such as CPU, display, and Wi-

Fi. The power model of the CPU is generated ahead of the other hardware components because it can be operating in standalone mode. Once this is completed, other hardware components can be modeled from the CPU model and chunks. For this measurement, the Dynamic Frequency Scaling (DFS) governor is set in performance mode to simplify the CPU power model. With that, the CPU is run at constant frequency during the modeling process.

From the chunks collected at step 1, this step selects chunks where only the CPU is active and other hardware components are turned off. Fig. 4 shows the CPU chunks and their power consumption. More cohesive distribution for the same utilization value has been expected. However, power consumption of the chunks varies even for the same utilization. Since this paper does not use a well-carved training suite, the chunks may contain results from a wide variation of processor activity during the chunk length. So, those chunks are filtered out by computing their noise. Noisy chunks are identified by computing the standard deviation of usage patterns stored in the chunks. If the value is larger than a threshold, the related chunks are classified as invalid. Then, those chunks are removed from the set. This result is shown in Fig. 5.

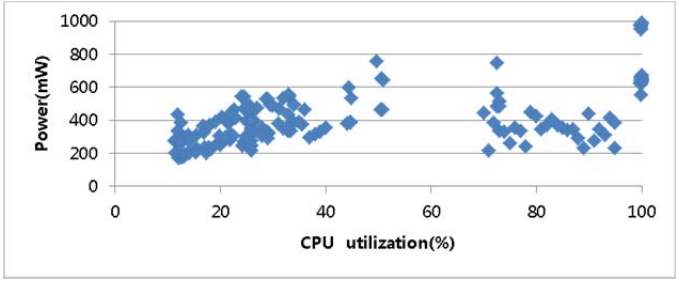


Fig. 4. Chunks with only CPU active. Chunks show less condensed distribution.

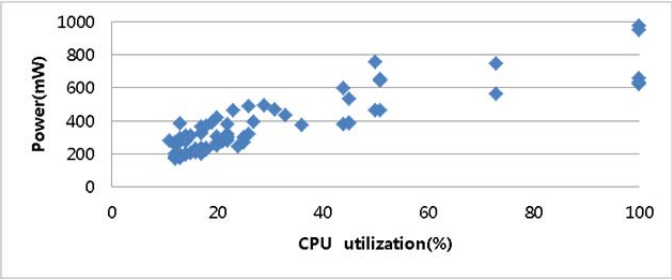


Fig. 5. Chunks with cohesiveness. Chunks having large variation in CPU utilization are filtered out.

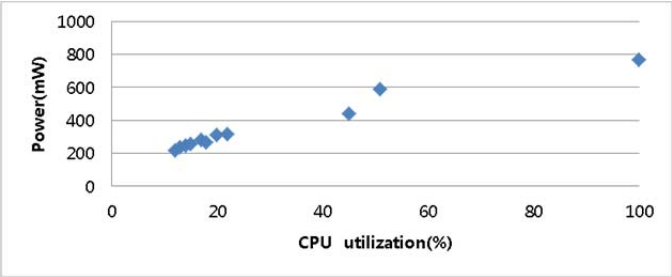


Fig. 6. Power consumption of chunks after merging 3 valid chunks. The number of chunks is reduced but now the linear behavior of the power value can be identified.

Chunk distribution in Fig. 5 is still not satisfactory. That is mainly because the measurement framework being used relies on the accuracy of the battery level indicator on the phone. This limits the precision and accuracy of the power model. Each 1% of battery level should contain exactly the same amount of battery capacity value. However, this cannot be achieved from the battery capacity data on the phone.

Errors in the chunks can be further reduced by merging multiple chunks having similar utilization. Fig. 6 shows the chunk distribution after merging 3 valid chunks.

C. Step 3: Power Model Computation

The previous steps have identified valid chunks that contain the power profile of a single hardware component, and which have only a small standard deviation among the stored power profiles. This step computes a power model from the filtered chunk set. The model is derived from the regression analysis of valid chunks. Table II shows the computed power model of two different smartphones. The detailed procedure of power model generation is explained in the following.

Most application processors support DFS, and for an accurate power model, operating frequency should be considered. However, the proposed model intentionally did not include CPU frequency, in order to simplify the modeling procedure. This model only considered processor utilization as the dependent parameter; nonetheless, it still could achieve a reasonably accurate power model. Fig. 7 shows the regression analysis of the filtered chunks from step 1 and 2. The linear regression fits well with the CPU power data.

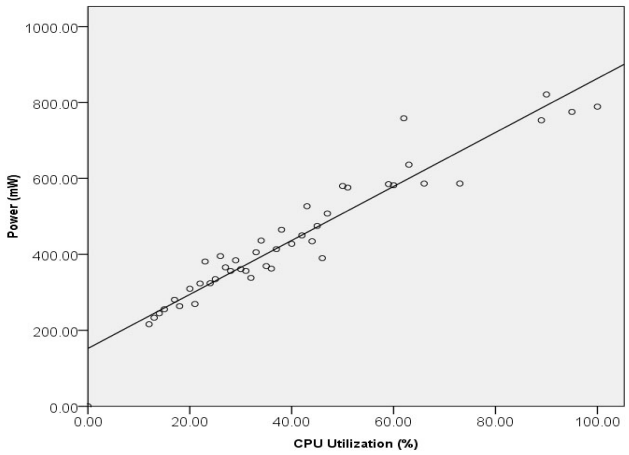


Fig. 7. Regression analysis of the CPU on a phone. Dots on the graph show the valid chunks after steps 1 and 2.

Recent smartphones use OLED technology as their display device. As reported in Chameleon [12], the power consumption of an OLED display varies up to 5 times depending on the image color distribution. The proposed method did not consider color parameter for power model generation. The decision was made to simplify, and to support real time power profiling. Instead, for the experiment to measure accuracy, the display brightness was set to maximum and this significantly reduced the color contribution to power consumption.

TABLE II
GENERATED POWER MODELS OF TWO DIFFERENT SMARTPHONES (P1, P2)

Target	HW Unit	Parameter	Description	Value	Power Coefficient
P1	CPU	CPU_on	Fraction of time interval with the cpu ON	0, 1	$\beta_{\text{cpu}} = 152mW$
		CPU_util	Average cpu utilization	0~100	$\beta_{\text{util}} = 7.1mW$
	Display	Display_on	Fraction of the time interval with the screen ON	0, 1	$\beta_{\text{display}} = 148.3mW$
		Brightness_level	Screen brightness	0~255	$\beta_{\text{br}} = 3.194\text{mW}$
	Wi-Fi	WiFi_on	Fraction of time interval Wi-Fi connection is ON	0, 1	$\beta_{\text{wifi}} = 87.048mW$
		Packet	Number of Rx/Tx packets with Wi-Fi during interval	0~∞	$\beta_{\text{wifi\_b1}} = 0.054mW$
					$\beta_{\text{wifi\_b2}} = 0.001mW$
		Threshold	Number of packets for changing power state	243	$\beta_{\text{wifi\_max}} = 159.171mW$
P2	CPU	CPU_on	Fraction of time interval with the cpu ON	0, 1	$\beta_{\text{cpu}} = 131.495mW$
		CPU_util	Average cpu utilization	0~100	$\beta_{\text{util}} = 7.080mW$
	Display	Display_on	Fraction of the time interval with the screen ON	0, 1	$\beta_{\text{display}} = 285.034mW$
		Brightness_level	Screen brightness	0~255	$\beta_{\text{br}} = 2.281mW$
	Wi-Fi	WiFi_on	Fraction of time interval Wi-Fi connection is ON	0, 1	$\beta_{\text{wifi}} = 23.764mW$
		Packet	Number of Rx/Tx packets with Wi-Fi during interval	0~∞	$\beta_{\text{wifi\_b1}} = 2.231mW$
					$\beta_{\text{wifi\_b2}} = 0.053mW$
		Threshold	Number of packets for changing power state	109	$\beta_{\text{wifi\_max}} = 890.704\text{mW}$

The proposed method collected chunks where Wi-Fi was used. Regression analysis was performed to generate the power model. The power consumption of the Wi-Fi mainly depends on packet rate and it is classified into four states: L_TRANSMISSION, H_TRANSMISSION, L_IDLE, H_IDLE [6]. The proposed method cannot identify these four states due to the sampling rate of the battery level indicator and coupling usage patterns. As in the FSM model, this method does not distinguish them during the regression analysis [4], [6], [10]. If the packet rate is greater than a threshold, the Wi-Fi power consumption is estimated as a constant value to reduce the error of the regression model. This threshold is defined as the maximum packet rate in collected chunks. Therefore, for Wi-Fi power states, the collected chunks follow two models: quadratic and constant.

$$\mathrm{P} = P_{\text{cpu}} + P_{\text{display}} + P_{\text{wifi}} \tag{2}$$

$$P_{\text{cpu}} = \beta_{\text{util}} \times \text{CPU\_util} + \text{CPU\_on} \times \beta_{\text{cpu}} \tag{3}$$

$$P_{\text{display}} = \beta_{\text{br}} \times \text{Bright\_level} + \text{Display\_on} \times \beta_{\text{display}} \tag{4}$$

$$P_{\text{wifi}} = \begin{cases} \beta_{\text{wifi\_b1}} \times \text{Packet} + \beta_{\text{wifi\_b2}} \times \text{Packet}^2 \\ +\text{WiFi\_on} \times \beta_{\text{wifi}}, \text{ if Threshold} \geq \text{Packet} \\ \beta_{\text{wifi\_max}}, \text{ if Threshold} < \text{Packet} \end{cases} \tag{5}$$

The smartphone power consumption is estimated by using the generated power coefficients and utilization of each hardware component. Equation (2) shows that the proposed method modeled the phone's power consumption with the CPU, display, and Wi-Fi. The power consumption of each hardware component is expressed as (3), (4), and (5). In (3) and (4), the power consumption of the CPU and display are computed from the linear regression model. The proposed method models the power consumption of Wi-Fi as (5). If the number of packets transmitted or received per second is greater than a threshold, it has a constant power model, and otherwise, a quadratic model. The power coefficients β_i in (3), (4), and (5) can be found in Table II. Other values also can be found in Table II.

IV. ARTIFICIAL USAGE PATTERN GENERATION

This section introduces a method to increase the number of valid chunks. Basically, the proposed method acquires valid chunks from a mobile application's activity log. However, users need an enormous number of usage patterns to generate a valid chunk. This is mainly because the proposed method obtains chunks from the user log file instead of a benchmark suite, as used in PowerTutor [6]. Accordingly, users can not perform regression analysis with just a few usage patterns.

So that users can collect valid chunks efficiently a usage pattern generator is proposed, similar to a training suite [6]. Fig. 8 shows the overview of the usage pattern generator. It controls the usage of hardware components according to the update rate of the battery level indicator. It consists of two modules, a workload allocator and battery capacity level monitor.

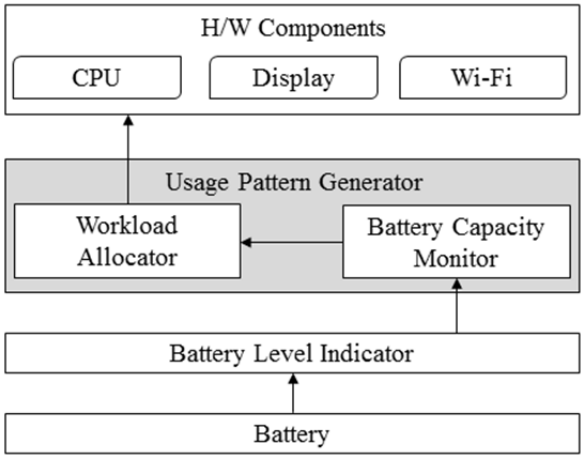


Fig. 8. Overview of the usage pattern generator.

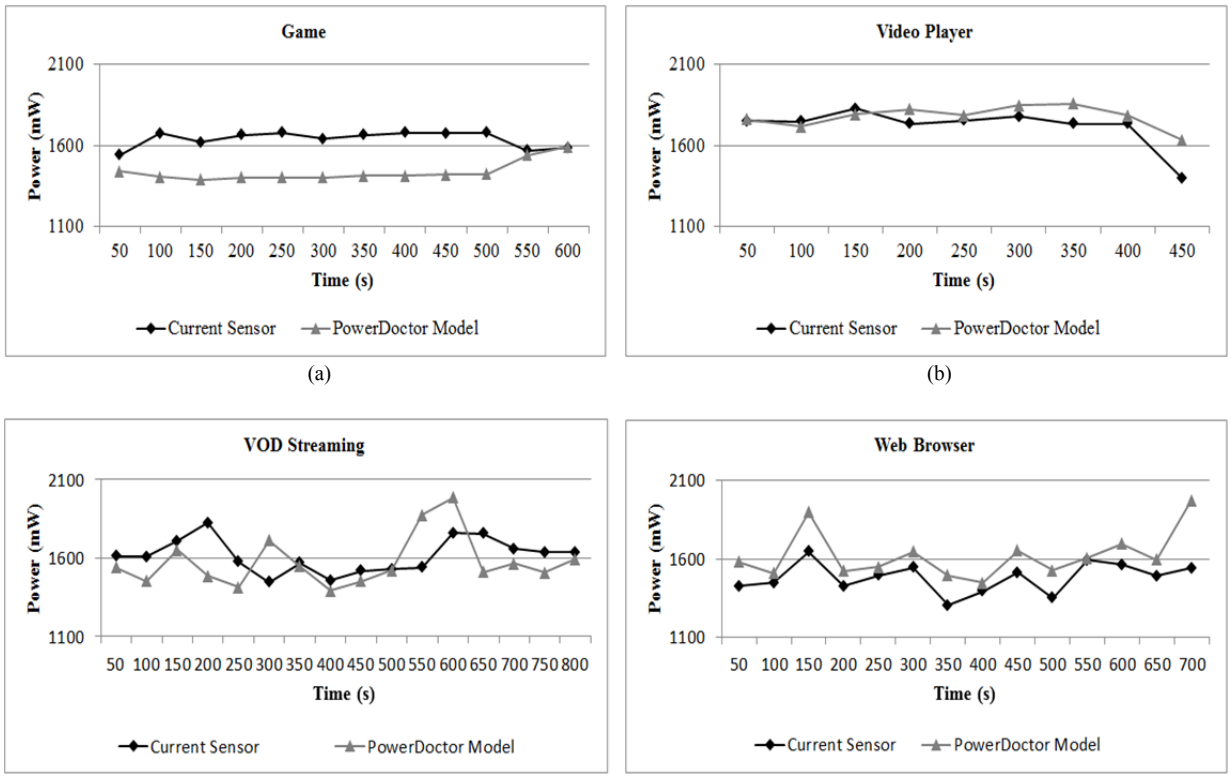


Fig. 9. Power estimation of mobile applications on a smartphone.

The workload allocator keeps the usage state of components at the same battery level to minimize coupling and any variation of usage patterns. The battery capacity monitor reads the current battery state. Also, it notifies the workload allocator if the battery level is decreased. In that case, the workload allocator assigns a new job, to generate another usage pattern. Eventually, the usage pattern generator dynamically synchronizes the component behavior with the update rate. Hence, users can efficiently collect valid chunks. Unlike the method described in Section III, it does not need an extra filtering processes.

The following explains in detail how to control each hardware component according to the battery level update rate.

- **CPU:** CPU power consumption depends on CPU utilization. In this work, the usage pattern generator keeps the same CPU utilization for gathering valid chunks. To collect other valid chunks, it changes the duty cycle of a computation intensive task when it detects a battery level decrease.
- **Display:** The proposed method only considers an LCD power model in the current work. The LCD power model is strongly influenced by the brightness level of the screen. The relationship is not completely linear, according to a recent work [11]. Therefore, the display power model is generated by considering every possible brightness level. Like the CPU, it keeps a fixed brightness level at the update timing of the battery level indicator. If the battery level has dropped, it is changed to another brightness level, either increasing or decreasing.
- **Wi-Fi:** To derive the Wi-Fi model, the proposed method only considers one network parameter, which is the packet rate. It sends fixed size (1MB) TCP packets to the local server while collecting valid chunks. To control the packet rate, the proposed method changes to a transmission period corresponding to update rate of the battery level indicator.

V. Evaluation of Power Model

The accuracy of the generated power model can be verified by comparing the power consumption measured from a current sensor and the estimation from the generated power model. To compute true power consumption, the phones that were used for this experiment have internal current sensors. PowerDoctor is used for the real time power estimation. It contains its own power model which is generated from PowerDoctor. Hence, PowerDoctor applications generate estimated power consumption data, which are stored onto the SD card. The current sensor values are logged as the kernel's debug message (d_{mesg}).

Six popular applications were selected as the benchmark applications. Hardware components which were not included in the proposed power model were turned off during the measurement. Fig. 9 shows some of the measurement results. To compare the estimated results to the current sensor, 50 estimated data sets, collected from PowerDoctor at 1 second intervals, were averaged, because the current sensor samples at a 50 second period. A video file was played for 450 seconds with a Video Player. As shown in Fig. 9(b), the PowerDoctor model estimates the power consumption accurately, with the CPU being used continuously. However, as shown in Fig. 9(a), the power consumption of a game

application showed the highest error among the popular applications. The cause of the error is assumed to be caused by the GPU operation, because the proposed method did not consider a GPU power model [13].

Estimation error is computed with the difference between measured and estimated power consumption values, as (6), and from that, accuracy is computed as (7).

$$\text{error} = \left| \frac{\text{actual(current sensor)} - \text{estimated}}{\text{actual(current sensor)}} \right| \times 100\% \quad (6)$$

$$\text{accuracy} = 100\% - \text{error} \quad (7)$$

As shown in Fig. 10, the generated power model has computed estimation errors of between 4.7% and 12.2%.

The result above shows that this proposed method is able to achieve accuracy similar to the power model built by PowerTutor [6], although it shows a greater estimation error compared with PowerTutor's results, which has an accuracy within 4.1% of error estimation. However, since this is still a prototype method, in future updates it is expected to overcome present accuracy limitations. In addition, the previous work [7] demonstrates that this novel methodology is capable of providing better accuracy when more valid chunks are collected. Therefore, if this proposed modeling framework can be extended to a collaborative environment for model generation, the collection of valid chunks will improve the accuracy of the proposed method exponentially, because the power model generation will provide representative power consumption for the application that can be run on the smartphone.

For example, volunteer phone users can be enlisted to install PowerDoctor to collect valid chunks from their phone. The collected chunk data can be transferred to a server and the extended framework can then obtain a larger number of chunk data, from multiple users having the same phone model. Then, those chunks can be applied to the power model generation steps, and a more accurate power model can be computed.

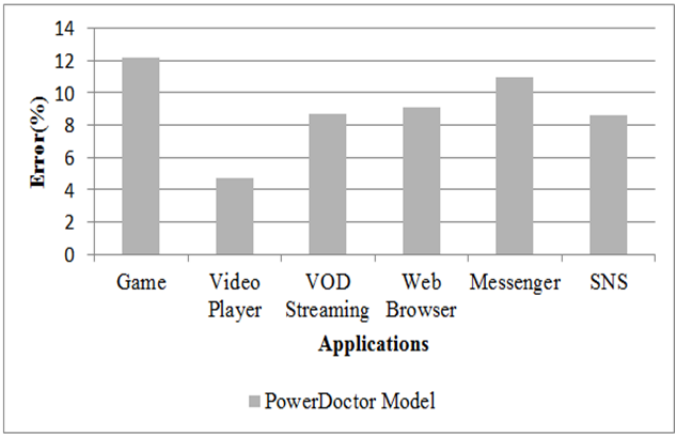


Fig. 10. Estimation errors of power model.

To further demonstrate the applicability of the proposed method, it was used to generate a power model for another phone (a P2).

The PowerDoctor estimated the power consumption on the P2. The accuracy of the power model can be derived from (6) and (7) as before. Fig. 11 shows the computed accuracy from the two phones. As it shows, generated power models for the two phones show an accuracy between 93% and 95% for five applications. For the Game application, it shows 79% accuracy on the P2. This is because the number of valid chunks collected was not enough for model generation, and because the proposed power model did not include the GPU hardware component [13].

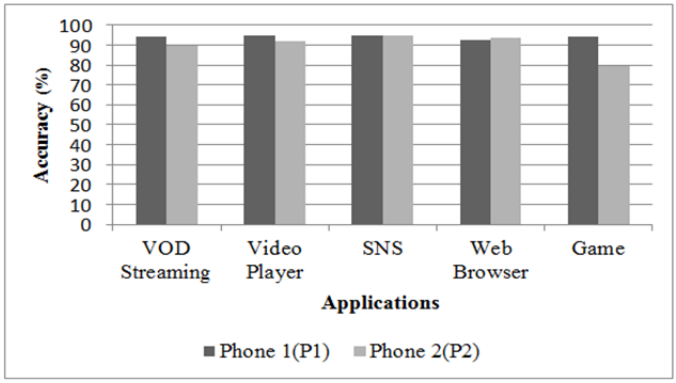


Fig. 11. Estimation accuracy on P1 and P2 phones.

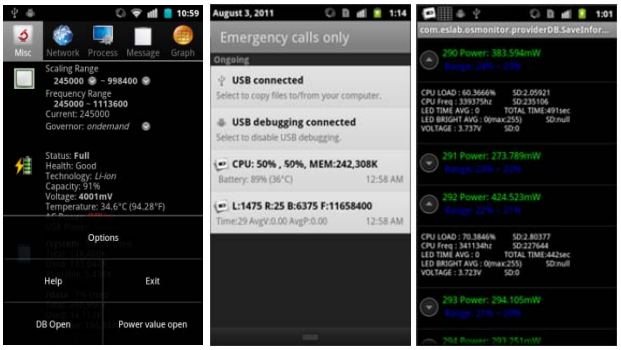
VI. Power Modeling Tools

This section describes the two tools developed to generate the power model.

The first tool is PowerDoctor. It collects usage patterns for generating chunks. Then, as described in Section III, the proposed method can compute a power model from the chunks. The graphic user interface for PowerDoctor is shown in Fig. 12. Fig. 12(a) shows its main interface screen. It displays the usage of the hardware component. Also, as shown in Fig. 12(b), it executes two services that are in charge of collecting the usage pattern and estimating power consumption. Fig. 12(c) shows the screen shot of collected chunks. Chunks are stored in the database and they can be used to generate the power model.

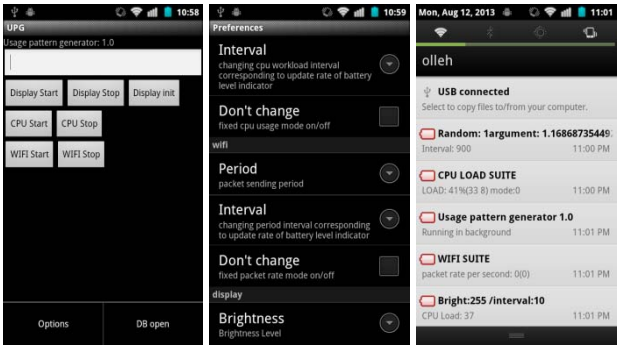
The second tool is the usage pattern generator. As shown in Fig. 13, it consists of three interface screens: a button interface, option interface, and service interface. Fig. 13(a) shows its main interface screen. There are various buttons such as start, stop, and settings. Users can generate an artificial usage pattern. For example, during the 1% battery capacity drop, the tool collects the usage of the hardware component with constant processor utilization. Fig. 13(b) shows its configuration screen, with which usage pattern services can be configured. Fig. 13(c) shows information on the services that are running in the background. Each service is in charge of a single hardware usage and it generates a usage pattern upon preference setting.

Those tools are provided as open-source software and they are released on the official market.



(a) (b) (c)

Fig. 12. Screen shots of PowerDoctor interface.



(a) (b) (c)

Fig. 13. Screen shots of usage pattern generator interface.

VII. CONCLUSIONS

This paper proposed a new method for generating a power model for smartphones without external equipment or explicit training programs. This method is useful since it can generate a power model without detailed knowledge of the underlying hardware or a specific training program design. The proposed method uses a regular user's usage pattern, which is already recorded on the phone. From the usage pattern, it identifies a data segment named a "chunk" which contains the hardware status and power data for a single operating state. Those collected chunks are fed into a regression analysis for each hardware component.

For the feasibility study, this paper performed modeling on the CPU, display, and Wi-Fi interface of a smartphone. The generated power model was verified by estimating the power consumption on the real phone and comparing that with the actual measured values. The result shows that the proposed method is valid. This method was applied to two smartphones and accurate power models were successfully generated. This verifies the applicability of the proposed method.

This paper used a simplified power model to quickly verify the concept. Due to that, the proposed power model contains about 5% to 15% error. Currently, the proposed method is being extended to include every hardware component in the smartphone. Also, a collaboration-based chunk collection system is being developed. Once the proposed framework is installed on users' mobile devices, this method can automatically generate a very accurate power model for virtually all the phones in the market.

REFERENCES

[1] S. B. Jang, and Y. G. Kim, "Power saving and delay reduction for supporting WLAN-based fixed-mobile convergence service in smartphone," *IEEE Trans. Consumer Electron.*, vol. 56, no. 4, pp. 2747-2755, Nov. 2010.

[2] S. H. Lim, S. W. Lee, B. H. Lee, and S. Lee, "Power-aware optimal checkpoint intervals for mobile consumer devices," *IEEE Trans. Consumer Electron.*, vol. 57, no. 4, pp. 1637-1645, Nov. 2011.

[3] H. Joe, J. Park, C. Lim, D. Woo, and H. Kim, "Instruction-level power estimator for sensor networks," *ETRI Journal*, vol. 30, no. 1, pp. 47-58, Feb. 2008.

[4] A. Shye, B. Scholbrock, and G. Memik, "Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures," *in Proc. 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, New York, USA, pp. 168-178, Dec. 2009.

[5] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y. M. Wang, "Fine-grained power modeling for smartphones using system call tracing," *in Proc. 6th conference on Computer systems*, Salzburg, Austria, pp. 153-168, Apr. 2011.

[6] 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. 8th IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, Scottsdale, USA, pp. 105-114, Oct. 2010.

[7] J. Lee, H. Joe, and H. Kim, "Smart phone power model generation using use pattern analysis," *in Proc. IEEE International Conference on Consumer Electronics*, Las Vegas, USA, pp. 412-413, Jan. 2012.

[8] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin, "Diversity in smartphone usage," *in Proc. 8th international conference on Mobile systems, applications, and services*, San Francisco, USA, pp. 179-194, Jun. 2010.

[9] M. Dong, and L. Zhong, "Self-constructive high-rate system energy modeling for battery-powered mobile systems," *in Proc. 9th international conference on Mobile systems, applications, and services*, Bethesda, USA, pp. 335-348, Jun. 2011.

[10] W. Jung, C. Kang, C. Yoon, D. Kim, and H. Cha, "DevScope: a nonintrusive and online power analysis tool for smartphone hardware components," *in Proc. 8th IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, Tampere, Finland, pp. 353-362, Oct. 2012.

[11] M. Kim, J. Kong, and S. W. Chung, "Enhancing online power estimation accuracy for smartphones," *IEEE Trans. Consumer Electron.*, vol. 58, no. 2, pp. 333-339, May. 2012.

[12] M. Dong, and L. Zhong, "Chameleon: a color-adaptive web browser for mobile oled displays," *IEEE Trans. Mobile Comput.*, vol. 11, no. 5, pp. 724-738, May. 2012.

[13] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha, "AppScope: application energy metering framework for android smartphone using kernel activity monitoring," *in Proc. 2012 USENIX Annual Technical Conference*, Boston, USA, pp. 387-400, Jun. 2012.

BIOGRAPHIES

Jemin Lee received his B.S. and M.S. degree in 2011 and 2013, respectively from Chungnam National University, Rep. of Korea. He is currently working toward his Ph.D. in the Department of Computer Science and Engineering, Chungnam National University. His research interests include energy-aware computing and automated testing of mobile applications.

Hyunwoo Joe received his B.S., M.S., and Ph.D. degrees in computer science, in 2005, 2007, and 2014, respectively, from Chungnam National University, Rep. of Korea. His research interests include wireless sensor networks, energy-aware computing and virtualization for spacecraft computer.

Hyungshin Kim (M'04) received his B.S. and Ph.D. degrees in computer science, in 1990 and 2003, respectively, from Korea Advanced Institute of Science and Technology (KAIST). He received the M.Sc. degree in satellite communication engineering from University of Surrey, U.K. in 1990. He joined Chungnam National University in 2004 and he is a Professor in the department of computer science and engineering. His research interests include avionics system, energy-aware computing, and embedded system software.