DevScope: A Nonintrusive and Online Power Analysis Tool for Smartphone Hardware Components

Wonwoo Jung, Chulkoo Kang, Chanmin Yoon, Dongwon Kim, Hojung Cha
Department of Computer Science
Yonsei University
Seoul, Korea
{wwjung, ckkang, cmyoon, dwkim, hjcha}@cs.yonsei.ac.kr

ABSTRACT
Smartphone power modeling is an important technique for users, application developers, and hardware manufacturers to build energy-aware systems. Compared to traditional offline techniques that use external power measurement devices, the online approach uses a built-in BMU (Battery Monitoring Unit), which has the advantage of generating a dynamic power model. However, the very low update rate of a BMU is an issue for online power modeling. In this paper, we introduce an autonomous power modeling tool for smartphones called DevScope, which overcomes the limitations of BMU-based online power modeling. DevScope controls components according to the BMU update rate, and derives the component power model automatically by analyzing the changes of power state. With DevScope, we construct an online and automatic power modeling that reflects both the diversity of users' environments and smartphone hardware complexity. By evaluating the scheme with various smartphones and configurations, we show that DevScope, indeed, generates a flexible and accurate power model.

Categories and Subject Descriptors
C.4 [Performance of Systems]: Modeling techniques

General Terms
Measurement, Experimentation.

Keywords
Smartphone Power Analysis, Power Modeling, Battery Monitoring Unit

1. INTRODUCTION
With the increasing number of smartphones, the energy usage of applications and hardware components has become a major issue for smartphone users. In addition to the users' concern, researchers and application developers are interested in optimizing their applications in terms of energy consumption. There has been a great deal of research to deal with this problem. Previous work is generally categorized into two groups: modeling and estimation of power consumption [1, 2], and policy development to reduce the power consumption of smartphones [3, 4]. Accurate power estimation helps end users construct a personalized energy-efficient smartphone system. The estimation result is used as a guideline to choose the applications to install, or to determine the usage patterns of smartphones. Application developers use the result to estimate the energy consumption of their programs and help developing energy efficient applications. Smartphone vendors use the estimation result as a guideline to select hardware components and to optimize the system software in an energy efficient way. Moreover, researchers can use the results to find the energy bottleneck in the system and develop solutions [3, 4]. For these reasons, we say that power modeling and estimation is an important technique that is necessary for constructing energy efficient smartphone systems.

Active work has recently been devoted to constructing an accurate power model. The research usually uses the offline method to measure and analyze power consumption patterns [1, 5-7]. Power consumption patterns are even different for the same device due to operating temperature or some other reasons [1]. The operating system or updating device driver may also affect the patterns. The power models for smartphones should therefore be constructed for each device individually, and updated with the changes of the external factors. Meanwhile, an offline method is often used under laboratory conditions and uses external measurement tools to understand power usage. With this method it is practically very hard to construct individual power models for each device. Additionally, dynamic update of the model is not possible.

The limitation of the offline method can be overcome by using an online approach that employs a Battery Monitoring Unit (BMU) [1, 2]. The BMU is a hardware component, equipped in most recent smartphones, that provides information such as supply voltage, current, and battery temperature. These features would enable the implementation of an online power model that automatically constructs a power model for each device. The scheme enables the reconstruction of the model dynamically at any time, providing flexibility to adapt to changes of external factors, such as software updates. Zhang et al. [1] used an offline method as default, and an online method to compensate for the limitation of offline method. However, they did not consider the detailed properties of a BMU that affect the accuracy; thus, we may not expect accuracy of their online model although their offline model is very accurate. Dong et al. [2] analyzed the properties of BMU and constructed an online power consumption model. Their model did not, however, provide information for each hardware component. In general, to employ a BMU as an online power measurement tool for accurate power modeling, we

CODES+ISSS'12, October 7–12, 2012, Tampere, Finland.

should consider two factors that are inherent in the properties of BMU. First, online measurement may not be precise compared to offline measurement. This is because the information update rate of a BMU is noticeably lower than external measurement tools. Second, since user is not able to intervene in the process of constructing a model, it is difficult to figure out the exact relationship between system activities and power consumption.

In this paper, we propose the solutions for each problem and validate them with experiments. The low update rate of the BMU is overcome by a well-refined measurement scenario. In our solution, the scenario is determined after going through an automatic analysis on the update rate of the BMU as well as on the characteristics of each hardware component equipped in the device. To understand the relationship between system activity and power consumption in online power models, we control the hardware activities in the scenario being synchronized with an update rate of the BMU. In other words, we set aside the relationship in advance by using the method to trigger the hardware activities related to power consumption only at the point that we can measure the device.

The technical contribution of our work is summarized as follows:

- We propose a detailed solution to overcome the limitations of using BMU and dynamically generate accurate power model. The solution is validated with the construction of an accurate power model for smartphone hardware.
- We provide an individual power model for each hardware component of smartphone. To the best of our knowledge our work is the first to construct a power model for each component automatically only using BMU.

The rest of this paper is constructed as follows. In Section 2, we describe the technical challenges of online power modeling. In Section 3, we propose our solution for overcoming these challenges. We describe the detailed mechanism for each hardware component and evaluate it with some smartphones in Section 4. Section 5 presents related works, and we conclude the paper in Section 6.

2. ONLINE POWER MODELING: THE CHALLENGES

2.1 Battery Monitoring Unit (BMU)

A BMU monitors discharge voltage curve, battery temperature, discharging current, and so on. The unit periodically stores the values in registers. Two methods are available to measure the power consumption of a smartphone: (1) using voltage curve and (2) sensing the value of discharging current. Using voltage value is an estimating technique that is available for every BMU but the scheme is prone to generate a significant amount of errors. Sensing the current is an accurate scheme, but the scheme is only available for BMU that is equipped with the current sensors. In our research, we consider BMUs that can measure the discharging current directly. The accuracy of a current value provided by a BMU is reasonable enough to be used for the current measurement of smartphone components. In the case of DS2784 [8], a BMU that is equipped on Google Nexus One (N1) [9], the sensing resolution is about 104 uA, and the measurement range is between -3430 mA and 3430 mA, with the measurement error of ±1%. The BMU monitors the discharging current levels with high sampling rate, and reports its average with low update rate. In the case of DS2784, the sampling rate is 18.6 KHz and the update rate is lower than 0.28 Hz (1 time per 3.52 s).

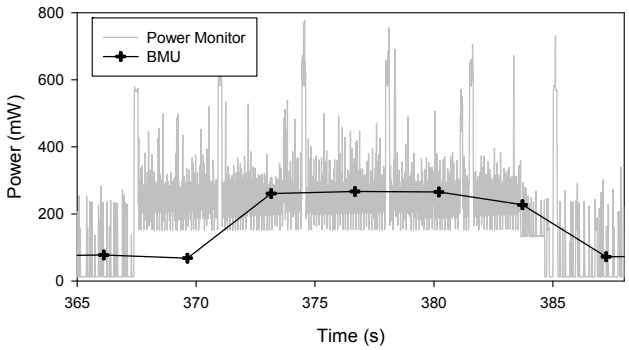


Figure 1. **Offline measurement Vs. online measurement.**

2.2 Challenges

Due to the low update rate (i.e., reporting rate) of BMU, it is difficult to determine the power states of system components. While the update rate of the external measurement tool is 5 KHz [11], the update rate of a BMU (in the case of DS2784) is lower than 0.28 Hz. That is, the external tool is more precise than the internal one in the same measurement scenario. In the case of power state changes within a very short time, we cannot pick up these changes with a BMU. Figure 1, for example, shows the result of power consumption of a WiFi interface card that is measured with external and internal tools. We transmitted 50 MTU sized packets per second in this experiment with N1. The sampling rate of the BMU is high enough (18.6 KHz) to trust the measurement result. However, its update rate is very low, so the power consumption pattern (i.e., power state changes) of the WiFi interface card is hard to grasp with BMU. Note that the pattern can easily be obtained with the use of external measurement tools. Since a power model should provide the power consumption pattern as well as the power consumption value, we should somehow overcome the limitation of the BMU to maximize the advantages of internal measurement.

The online method has to perform the analysis, as well as the measurement, online. The system should therefore recognize the relationship between power consumption and system activity in order for accurate analysis. For example, the first rise in Figure 1 represents the start of sending packets. For humans this is simple to recognize, but the fact is hard to be detected automatically by the system. We need a mechanism to recognize the relationship autonomously and accurately.

3. NONINTRUSIVE POWER MODELING

In this paper, we define the nonintrusive power modeling as a system modeling technique that is conducted automatically and at run-time without external devices or any artificial intervention. Without any modification of the operating system, we developed an application, called DevScope, which analyzes and models the smartphone power consumption.

In line with the previous research [1, 2, 6, 7], the system power consumption model (P) is presented as a first-order polynomial regression function:

$$P = \Sigma_i(\beta_i \times u_i),$$

where u_i is individual component utilization and β_i is power coefficient value of component i.

Some hardware components have multiple terms (i.e., combinations of β_i and u_i) in this equation. For example, most of modern CPU has multiple operating frequencies. To determine the necessary terms for the power model equation, we pre-analyzed each component by investigating the general characteristics of power state transitions. In a practical situation, the detailed operating characteristic of each component depends on hardware and configuration. Therefore, for a flexible power model, the number of terms should be determined dynamically during the modeling process.

DevScope is an automatic and online training tool to generate a power model. The tool first probes the operating systems to find information about the individual component types and its configuration (e.g., available CPU frequency levels). Additionally, by monitoring the update activity of BMU, DevScope figures out the update rate automatically. According to individual component types, system configuration, and BMU update rate, DevScope dynamically creates a component control scenario to perform power analysis. Hence, even though the device (i.e., smartphone) model is identical, the scenario might be different by each configuration. DevScope performs the test scenario, and automatically classifies the results into each terms of the power model. Finally, DevScope analyzes the classified data to find power coefficients that are required in modeling the system power.

Figure 2 shows the overview of the DevScope framework. The tool consists of four parts: Component Controller, Timing Controller, BMU Event Monitor, and Power Model Generator. The BMU Event Monitor is a module to monitor the discharging current data from the BMU. The Timing Controller calculates the BMU update rate and generates a control signal which triggers the actions of the Component Controller. The Component Controller constructs test scenarios for individual components and performs the actions according to control triggers, which are generated by the Timing Controller. All the test results are analyzed by the Power Model Generator to obtain power coefficients. These four modules collaborate to overcome the challenges discussed in the previous section.

3.1 Update Event-based Component Control

The solution to overcoming the very low update rate of BMU is to control individual components thoroughly, according to the update rate while conducting test scenarios. This solution has two problems. First, the update rates of the built-in BMU depend on the type of hardware. Second, the BMU update rate cannot be controlled at run-time. Due to these problems, it is hard to synchronize the component behavior with BMU update rates

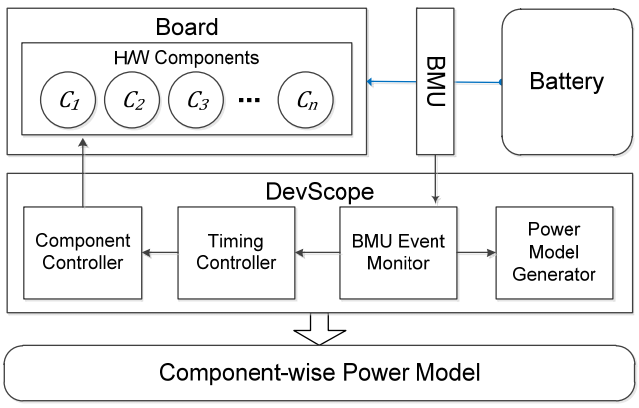


Figure 2. The overview of DevScope framework.

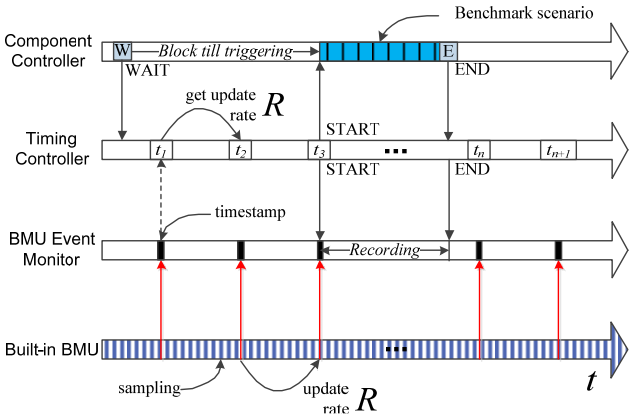


Figure 3. BMU update event based timing controlled component job processing to benchmark its power consumption.

dynamically. To handle this, DevScope adopts an event-driven method that regards the BMU update as an event and monitors the update events continuously. Test scenario for individual component should be performed depending on the BMU update events. Figure 3 describes how DevScope performs the test scenarios based on the event-driven method. The BMU Event Monitor puts a timestamp on each discharging current data whenever an update occurs. Every test scenario has WAIT and END messages. If a scenario is ready to perform, the Component Controller sends WAIT message to the Timing Controller, then blocks every operation until it is triggered by the Timing Controller. A WAIT message is received; the Timing Controller reads the timestamp multiple times to calculate the BMU update rate R. The BMU update rate in the application level has a little delay compared to the actual update rate in the hardware level; hence, R should be re-calculated before the test scenario is performed to redeem the delay between actual update and triggering time. After re-calculating, the Timing Controller triggers the Component Controller to release a blocked scenario when the next update event occurs. Accordingly, this event-driven component control method can synchronize individual test scenarios with the BMU update.

3.2 Update Rate-aware Workload Allocation

Even if each component behavior is synchronized with the BMU update, the power state transition cannot be recognized online, without considering the duration of each power state and the BMU update period. Moreover, some components (e.g., WiFi interface) have different power states depending on the workload size for the same operation. In this case, the test scenario should be designed with a proper workload size to keep power state stable long enough to cover the BMU update period. Consequently, DevScope should consider not only the power state duration but also the workload size to successfully conduct the test scenario.

Figure 4 represents the examples of BMU measurement results with update rate R according to the power states change. Note that the updated value from the BMU is an average value during one update period. The top two graphs show the case of power state change from base to active, and the bottom two graphs show the opposite case. In Figure 4(a), it is easy to recognize power state changes because the active power state is longer than R. Note that the start of component operation is synchronized with the BMU update by DevScope (See Section 3.1). However, if the duration

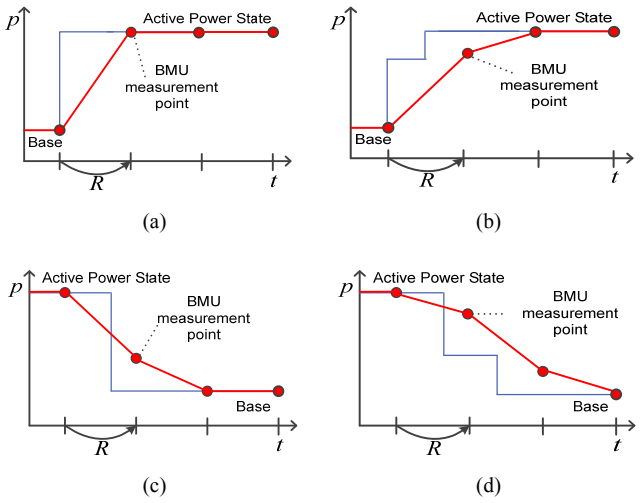


Figure 4. Examples of BMU measurement result by component power state transition. The blue line represents real power states and the red line represents the measurement result of BMU.

of state is shorter than R, the state cannot be defined and often called a hidden state. With hidden states, the BMU measurement result would have multiple slopes, as illustrated in Figure 4(b). Thus, we need at least $3R$ duration to recognize the hidden state by observing slopes of the BMU measurement result.

On the other hand, it is not possible to halt component behavior intrusively, at the exact update moment of the BMU, without any effect on power state stability (See Figure 4(c) and (d)). Accordingly, we cannot predict a timing of power state change from active to base state. In Figure 4(c) and (d), similar to the above method, DevScope recognizes the power state changes with at least $4R$ duration for both cases. DevScope assigns sufficient duration for component control and allocates workload to keep every power state longer than $5R$, as a default, to recognize power states more accurately.

Unfortunately, the duration of some power states cannot be controlled. For instance, the WiFi interface has multiple power states, depending on workload size, during the data sending operation; hence, DevScope finds the threshold workload size, which triggers the power state change, by conducting the tests repeatedly by increasing the workload size gradually. Indeed, the corresponding workload for WiFi interface is the number of packets transmitted per second (See Section 4.4).

As discussed above, DevScope recognizes the transition in component power state by adjusting the duration of power states. Nevertheless, for some components, duration adjustment is insufficient or impossible to recognize because of its fundamental power characteristics. Considering this, DevScope conducts a preliminary test to find the adequate factors, such as the packet rate of the WiFi interface.

In the following section, we describe the preliminary testing and detailed power consumption analysis method for each component, and discuss the power measurement result. We also show the evaluation results with experimentation.

4. COMPONENT POWER ANALYSIS

With DevScope, we analyze the power characteristic of major hardware components in a smartphone: CPU, Display, WiFi, Cellular, and GPS. Previous research [6] directly measured the power consumption of hardware components. Their evaluation showed that most of the power consumption of smartphones is caused by CPU, Display, and Network. Considering that many applications are based on location-based services, we additionally analyzed the GPS. In this section, we describe the specific policy for these five hardware components.

Unlike other hardware components, the CPU cannot be turned off unless the smartphone is off. Therefore, we cannot independently measure the power consumption of the other four components. We calculate the power consumption of hardware components by subtracting the power consumption of the CPU, as illustrated in (1).

$$P_{Component} = P_{Measured} - P_{CPU}, \quad (1)$$

where $P_{Component}$ is the power consumption of hardware component, $P_{Measured}$ the total power consumption of smartphone that is measured by the BMU, and P_{CPU} the power consumption of the CPU. For estimating P_{CPU}, we used the CPU power model computed by DevScope. The CPU power model should therefore be generated ahead of the other hardware components. To minimize the error caused by the CPU, we set the CPU frequency to the highest performance, and the CPU state to idle.

In the following, we describe the technique for measuring and analyzing power consumption of hardware components. We evaluate the model using two smartphones: Google Nexus One (N1) and Sony-Ericsson Xperia Arc (Arc) [10]. Note that either device has different types of BMUs, that is to say their update rates are different. To validate the analysis, we also conducted the analysis with the Monsoon Power Monitor [11] in off-line using the same scenario.

4.1 CPU

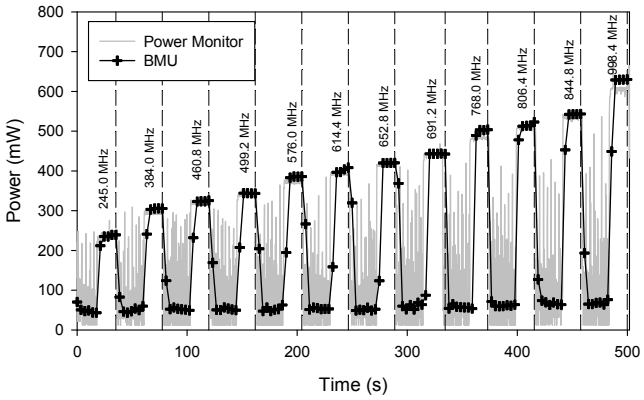
Modern processors of mobile devices are able to adjust the frequency automatically to conserve power. This technique is commonly known as DVFS (Dynamic Voltage and Frequency Scaling). According to the CPU frequency governor, DVFS sets the CPU frequency, depending on the predefined frequency-voltage table. The frequency-voltage table is defined by the policy of kernel, thus the power consumption of identical CPUs can differ as a result of each system policy.

The CPU cannot be independently turned off; therefore, the power consumption of the CPU must be subtracted in the power consumption of other components. For this, DevScope analyzes the frequency-voltage table ahead of anything else, and generates the discrete power model. The CPU power model is as follows:

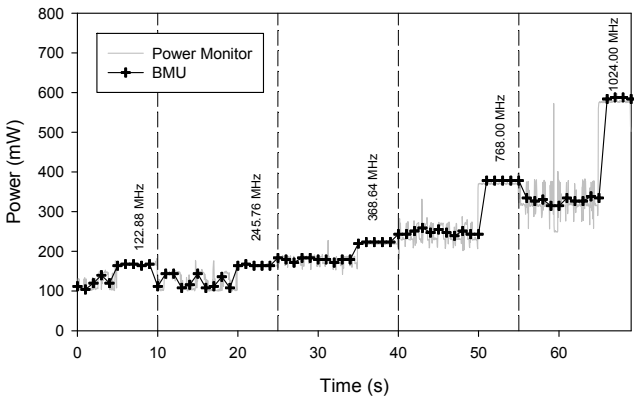
$$P_{CPU} = \beta_{freq_i} \times u + \beta_{idle_i}, 0\% \le u \le 100\%, \ i = 0,1,2 \cdots, n, \quad (2)$$

where i is the index of the frequency-voltage table, β_{freq_i} is the power consumption of the CPU in the i-th CPU frequency, u is the CPU utilization, and β_{idle_i} the idle power consumption of the i-th frequency. The number of available frequency is dynamically determined by DevScope which looks up the frequency-voltage table.

To determine β_{freq_i} and β_{idle_i} with frequency-voltage table, DevScope measures the power consumption after setting the



(a) Google Nexus One



(b) Sony-Ericsson Xperia Arc

Figure 5. The power consumption measurement result for CPU by DevScope.

frequency to every value in the table. To compute β_{idle_i}, DevScope first induces the utilization of the CPU to zero. Second, DevScope leads the utilization of the CPU to the maximum value, then computes β_{freq_i}. DevScope repeats this operation with every frequency in the table. In this operation, the Component Controller must recognize the state transitions of the CPU to precisely measure the power consumption. To generate the precise coefficients, DevScope keeps the CPU power states stable for $5R$ which we consider an adequate period.

Figure 5(a) shows the power consumption traces of CPU with N1 by the Monsoon Power Monitor and BMU. Since the frequency-voltage table of N1 is classified with twelve (i.e., 12 different settings), the figure shows the power consumption within twelve types of frequency. In each frequency level, the low power consumption is in an idle state, the high power consumption is in the maximum working status. The idle state defined is the complete idle because of the CPU usage by kernel. In our work, the necessary power consumption required by the system is considered as the base power, and is included in β_{idle}. Figure 5(b) shows the power trace with Arc. The device has five types of frequency level, and the power consumption of the base power is sharply changed at different frequencies. Since R of Arc is 1 second, the scale of the x-axis is different.

4.2 Display

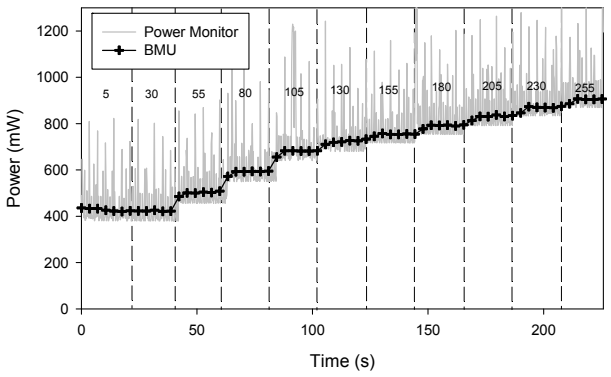
The display type of recent smartphone is usually LCD or OLED. The power consumption of OLED is changed by the color of the screen [3]. Since the analysis on color change is not in the scope of this paper, we only analyze the LCD display in the current work.

The power consumption of LCD varies with the brightness level of the screen. The relationship is not completely linear [6], hence, DevScope generates the power model by considering every possible brightness level.

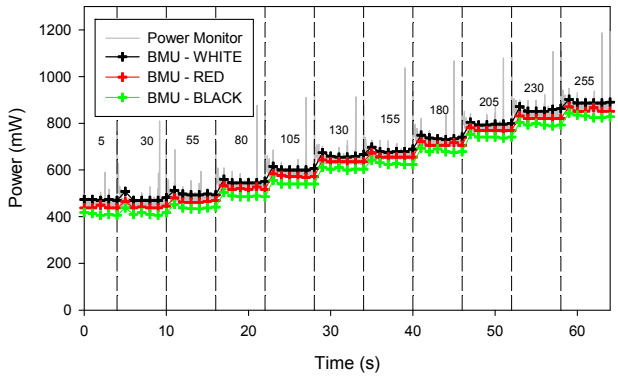
$$P_{Display} = \beta_{brightness_B}, 0 \le B \le 255, \quad (3)$$

where B means the brightness level of display backlight, and $\beta_{brightness_B}$ is the power consumption of display when the brightness level is B.

DevScope dynamically generates the table which contains the coefficients for each brightness level. To reduce the overhead, DevScope does not measure the power consumption of every brightness level; instead, a set of brightness level is measure and the rest are interpolated. Like other components, DevScope measures the **p**ower consumption at update timing of the BMU with a fixed brightness level, and the duration of the state is $5R$. At each brightness level, DevScope averages the sampled data and considers this value as the power consumption of display.

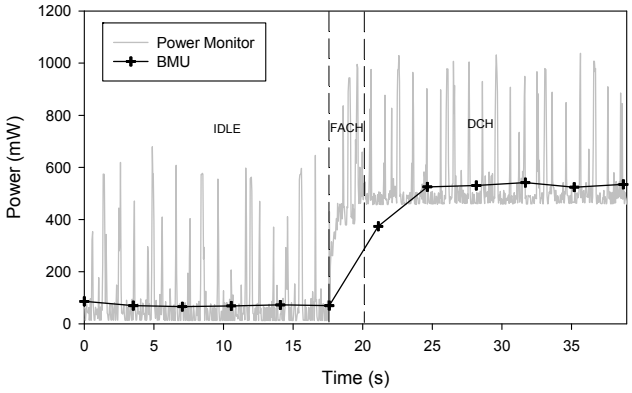


(a) Google Nexus One

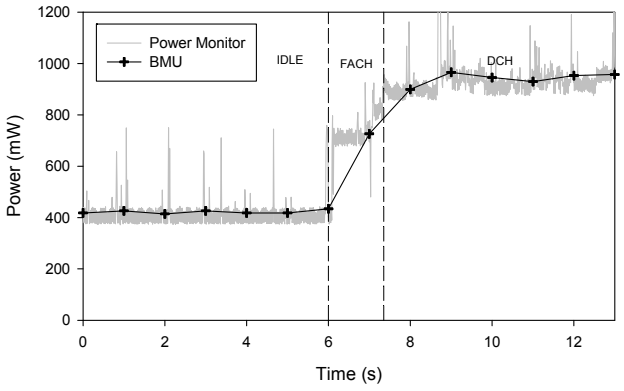


(b) Sony-Ericsson Xperia Arc

Figure 6. The power consumption measurement results for LCD display by DevScope.



(a) Google Nexus One



(b) Sony-Ericsson Xperia Arc

Figure 7. The power consumption measurement graph for Cellular by DevScope.

Figure 6 shows the measurement results according to the brightness level. Similar to other components, the scale of the x-axis is different by the BMU update rate. The indicated value at the top is the brightness level. Figure 6 shows that the power consumption of the display is not linear with the brightness level. Arc in Figure 6(b) uses the LCD display, but uses the LED type as backlight. This display adjusts the backlight by the color of the frame buffer. Thus, when the screen color is black, red, or white, the power consumption differs in Figure 6(b). The difference in the power consumptions for each color is, however, very small. In this paper, DevScope fixes the color as white and considers the brightness level only.

4.3 Cellular

The Cellular has three types of RRC (Radio Resource Control) states, which are IDLE, FACH, and DCH, according to energy-awareness policy in mobile networks [12]. The power consumption of Cellular interface does not depend on the packet rates as opposed to WiFi (See Section 4.4), but only depends on the power state transitions. We model the Cellular power consumption, $P_{Cellular}$ as follows:

$$P_{Cellular} = \beta_{rrc}, rrc \in \{IDLE, FACH, DCH\}, \quad (4)$$

where rrc is one of the RRC state, and β_{rrc} is the power coefficient.

To analyze the power state transition, the test scenarios for Cellular are performed by controlling data traffic. First of all, DevScope waits until the RRC state becomes IDLE without any data transmission. After detecting the IDLE state, DevScope sends HTTP requests immediately. At this time, the IDLE state transits to DCH. There indeed exists FACH state in this transition. However, the FACH state cannot be recognized directly by the BMU. We define the unrecognizable states, such as the FACH state, as the hidden state (See Section 3.2). In the DCH state, DevScope sends the HTTP requests periodically to keep the DCH state stable for $5R$.

Figure 7 shows the measurement results of N1 and Arc while running DevScope for Cellular. DevScope clearly distinguishes IDLE and DCH states. To find the power coefficients of the FACH state, DevScope uses a heuristic method. In Figure 7(a), the FACH state duration is shorter than R(=3.52 s), thus the FACH state could not be monitored by BMU. If there exists a FACH state as a hidden state, such as Figure 7(a) (Refer to Figure 4(b)), DevScope estimates the power coefficient, β_{FACH}, by approximation between IDLE and DCH state power consumption. We represent the β_{FACH} function as follows:

$$\beta_{FACH} = \frac{P_{component} \times R - \beta_{DCH} \times \Delta t_{DCH}}{\Delta t_{FACH}}, \quad (5)$$

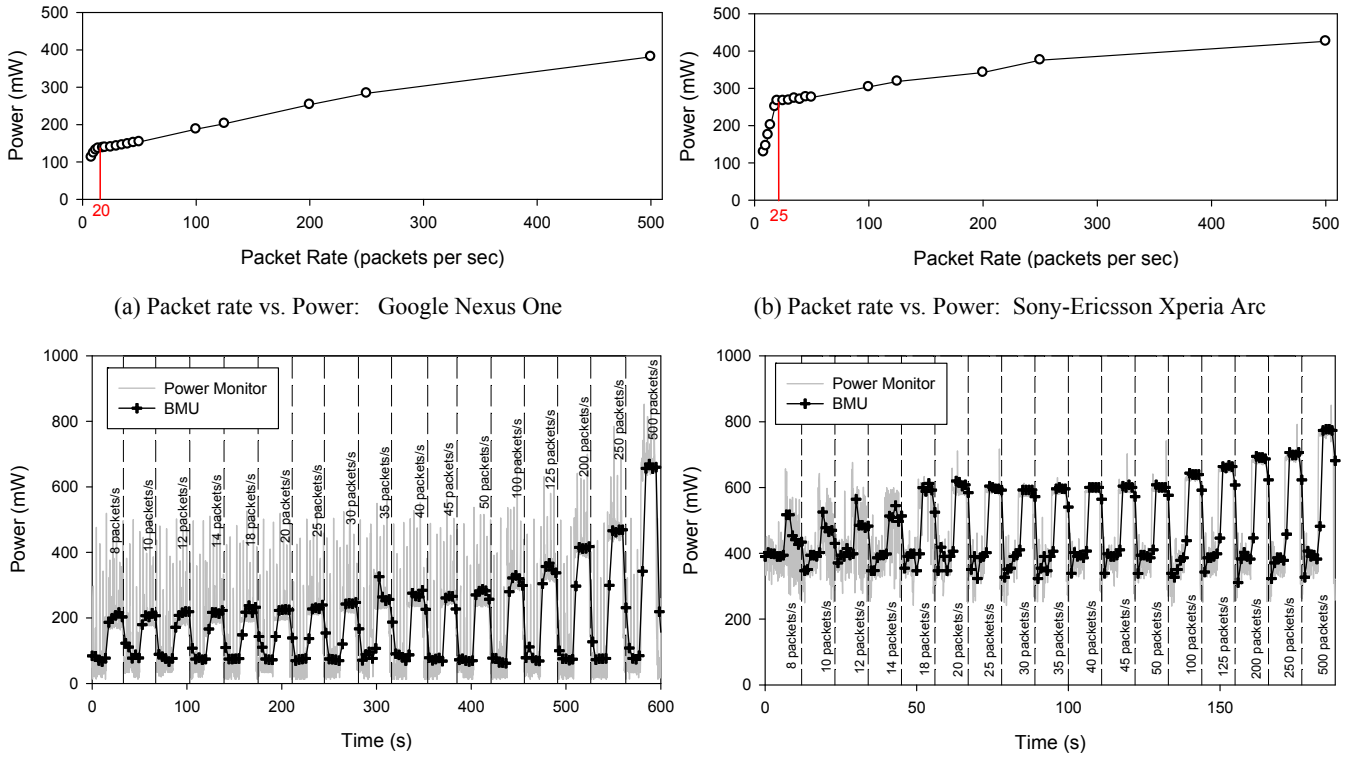
where $P_{component}$ is the result of subtracting CPU power consumption from the measured power consumption at the first-detected DCH state by BMU, and Δt_{DCH} and Δt_{FACH} are the amounts of time staying in the corresponding power states within R. On the other hand, in Figure 7(b), the FACH state duration is similar to Figure 7(a), but the update rate of Arc (R =1.00 s) is much shorter than N1. Thus, the FACH state is not a hidden state but its duration is insufficient to estimate the power consumption automatically. DevScope figures out β_{FACH} of Arc in the same way as N1.

4.4 WiFi

Depending on the amounts of outgoing packets, the WiFi power states are classified into four states [1, 2, 7]: L_IDLE, H_IDLE, L_TRANSMISSION, H_TRANSMISSION. Both L_IDLE and H_IDLE states are the states in which outgoing packets do not exist, whereas outgoing packets are available in L_TRANSMISSION and H_TRANSMISSION. The power consumption pattern of WiFi differs according to the specific packet rate (i.e., threshold workload size) [1, 2]. Figure 8(a) and Figure 8(b) represent the WiFi power consumption. The results do not include the power consumption of the CPU. The power pattern of WiFi is separated by the threshold. The threshold of N1 and Arc is 20 packets/sec and 25 packets/sec, respectively. If the packet rate is less than threshold, WiFi is in the L_TRANSMISSION; otherwise in the H_TRANSMISSION. Based on this fact, we model the power consumption of WiFi as follows:

$$P_{WiFi} = \begin{cases} \beta_{LT} \times p + \beta_{LT\,Base} & if\ p \le Threshold \\ \beta_{HT} \times p + \beta_{HT\,Base} & if\ p > Threshold \end{cases}, \quad (6)$$

where β_{LT} and β_{HT} are the rate of increase in power consumption in the L_TRANSMISSION and H_TRANSMISSION, p is the packet rate, and $\beta_{LT\,Base}$ and $\beta_{HT\,Base}$ are the base power consumptions of WiFi in the L_IDLE and H_IDLE, respectively. The parameters are constant with packet rate. The H_IDLE state is a hidden state that cannot be monitored by the BMU. Considering that the BMU provides power consumption of the WiFi as the average power consumption for the BMU sampling period, we assume that the power consumption of the H_IDLE is included in the computation of H_TRANSMISSION and it is not computed separately.



(a) Packet rate vs. Power: Google Nexus One

(b) Packet rate vs. Power: Sony-Ericsson Xperia Arc

(c) Time vs. Power: Google Nexus One

(d) Time vs. Power: Sony-Ericsson Xperia Arc

Figure 8. The power consumption measurement result for WiFi by DevScope.

DevScope automatically calculates the threshold, the coefficients, and the constants mentioned in (6). Figure 8(c) and Figure 8(d) show the overall change in power consumption with the increasing packet rate. As the packet rate continuously increases, we observe that the power obtained with DevScope also increases. The pattern of increase in power consumption slightly differs due to the devices used in the experiment. The major difference in base power, when outgoing packets do not exist, is caused by the CPU power consumption. Because the CPU frequency is set to the highest, the value of the base power is different. The starting time of the workload is the same as the update timing of the BMU, and the state duration is $5R$. Since the device's BMU sampling frequency is different, DevScope obtains a different number of samples in a given time. Therefore, the scale of x-axis is different in Figure 8(c) and Figure 8(d). Because the threshold is generally a small value, in order to increase the accuracy of threshold detection, DevScope runs with higher granularity in the lower packet rates.

4.5 GPS

GPS consumes most of the power in the process of communicating with satellites. The power states of GPS are defined into three states: OFF, SLEEP, ACTIVE. GPS does nothing in OFF state. In SLEEP state, GPS waits to connect with satellites. In ACTIVE state, GPS communicates with satellites. Even though ACTIVE consumes more power than SLEEP, the duration of ACTIVE is very short; hence, it is a hidden state to the BMU. Since it has a small effect in power consumption, we regard SLEEP and ACTIVE states as ON states. Based on these two states, we model the power consumption of GPS as follows:

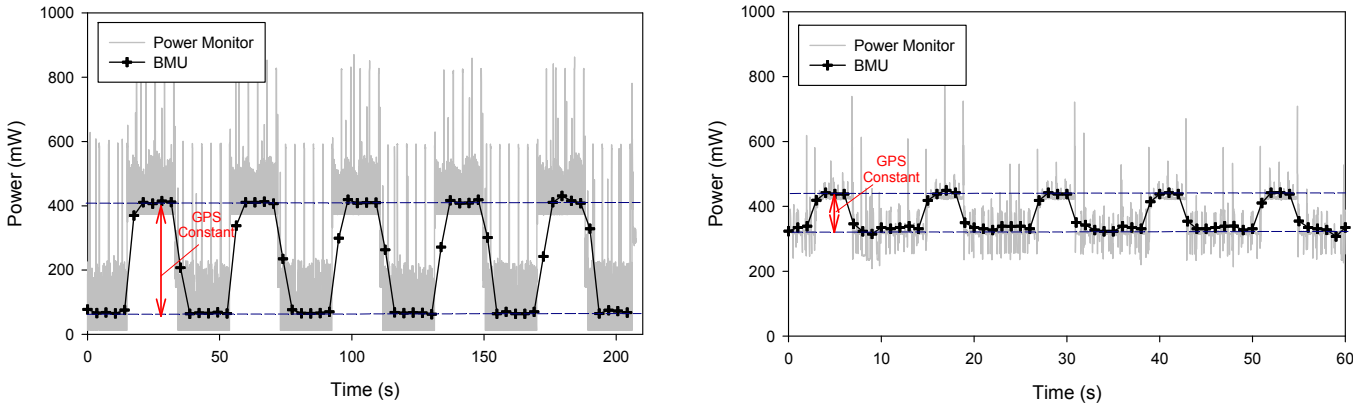
$$P_{GPS} = \beta_{GPS}, \tag{7}$$

where β_{GPS} is the power coefficients.

If GPS receives the request for updating location, it keeps in the ON state until GPS fixes the current location or the request is canceled. DevScope should maintain the ON state until receiving a sufficient number of samples. Thus we designed a method to keep GPS in the ON state. If GPS fixes the current location in the time shorter than the sampling interval of BMU, GPS once again updates its current location. Figure 9(a) and Figure 9(b) represent the power change by GPS. The difference of base power for each device is caused by different power consumption of each CPU. After repeating to turn ON and OFF state, DevScope calculates the difference of power consumption as β_{GPS}.

4.6 Summary

We empirically characterized the component power characteristics and described power analysis methods to derive power models for each component. Based on the component power characteristics, DevScope dynamically devises the test scenario in accordance with the update rate-event based control methods and the update rate-aware workload allocation rules.



(a) Google Nexus One (b) Sony-Ericsson Xperia Arc

Figure 9. The power consumption measurement graph for GPS by DevScope.

Table 1 summarizes the results of the power coefficients and other constants, which are found by DevScope on two smartphones. For the relative comparison between online modeling results with offline one, we have also derived offline power models using Monsoon instead of built-in BMU. As shown in Figure 5, 6, 7, 8, 9, the BMU power measurement results were reliable compared to external measurement results. The online results which have 5 samples total are the average of 3 nearest values (i.e., KNN with k=3). Accordingly, the results show that the BMU measurements filter out fluctuations on each power state by the update rate-aware workload allocation mechanisms. This filtering effect makes it easy to analyze the component power states automatically. DevScope keeps observing slopes of the result to detect the power state transitions. Obviously, the hidden power states exist. Finding accurate power coefficients of the hidden states is difficult. DevScope classifies the hidden power states into two types. The FACH state of Cellular, for instance, exists at the beginning of the power state transition from base to active power state. Depending on the BMU update rate, the FACH state could be hidden. In this case, DevScope detects the existence of the hidden state and estimates its coefficient by approximation. Comparing to the FACH state, the High Idle state of WiFi exists at the end of power transition from active to base. This case is, in fact, a remaining issue in modeling component power characteristics with online measurement.

Table 2 shows the error rates. We calculated the error rate E_{β_x} which is related to coefficient β_x as follows:

$$E_{\beta_x} = \left| \frac{\beta_{x\_monsoon} - \beta_{x\_bmu}}{\beta_{x\_monsoon}} \right|, \quad (8)$$

where $\beta_{x\_monsoon}$ is the coefficient value obtained with Monsoon, and β_{x_bmu} the one with BMU. As shown in the table, the measurement results of DevScope are similar to the offline results.

Table 1. Power Coefficient Values.

Comp.	Index	N1-Monsoon		N1-BMU (R=3.52 s)		Arc-Monsoon		Arc-BMU (R=1.00 s)	
CPU	i	β_{freq_i}	β_{idle_i}	β_{freq_i}	β_{idle_i}	β_{freq_i}	β_{idle_i}	β_{freq_i}	β_{idle_i}
	0	193.8	41.5	197.1	39.2	47.9	118.9	50.8	115.9
	1	254.5	41.9	258.3	39.6	42.5	122.4	41.8	122.9
	2	276.4	43.8	278.9	43.5	42.9	176.8	46.3	177.1
	3	296.2	43.6	298.1	43.9	127.9	242.9	130.7	247.7
	4	327.4	45.0	330.5	43.6	251.8	325.6	264.6	323.0
	5	347.1	46.0	351.7	43.1	N/A			
	6	365.7	49.9	370.6	46.3				
	7	391.6	47.4	390.0	50.2				
	8	436.6	46.9	438.4	47.1				
	9	461.1	47.7	459.9	49.9				
	10	488.2	47.6	491.6	46.0				
	11	553.6	53.7	552.6	55.6				
GPS		β_{GPS}		β_{GPS}		β_{GPS}		β_{GPS}	
	ON	346.9		330.0		108.0		113.0	

Comp.	Index	N1-Monsoon		N1-BMU (R=3.52 s)		Arc-Monsoon		Arc-BMU (R=1.00 s)	
Display	B	$\beta_{brightness_B}$		$\beta_{brightness_B}$		$\beta_{brightness_B}$		$\beta_{brightness_B}$	
	5	351.6		359.9		147.9		148.4	
	55	427.6		435.1		172.1		171.1	
	105	628.1		614.3		276.0		278.5	
	155	686.3		686.8		354.1		357.3	
	205	770.9		764.8		469.6		473.2	
	255	835.3		837.5		559.5		564.5	
Cellular	rrc	β_{rrc}		β_{rrc}		β_{rrc}		β_{rrc}	
	IDLE	76.9		69.2		85.0		96.4	
	FACH	461.5		321.8		393.0		403.0	
	DCH	551.1		531.4		619.0		623.4	
WiFi		β_{LT}	β_{HT}	β_{LT}	β_{HT}	β_{LT}	β_{HT}	β_{LT}	β_{HT}
	Transmit	2.0	0.7	2.0	0.6	4.1	0.3	3.8	0.3
	Base	110.0	140.7	110.0	140.7	166.2	250.3	173.1	251.4
	Threshold	20		20		25		25	

Table 2. Error Rate (%)

Comp.	Index	N1		Arc		Comp.	Index	N1		Arc	
CPU	i	β_{freq_i}	β_{idle_i}	β_{freq_i}	β_{idle_i}	Display	B	$\beta_{brightne}$		$\beta_{brightne}$	
	0	1.8	5.5	6.1	2.7		5	2.4		0.3	
	1	1.5	5.7	1.7	0.4		55	1.8		0.6	
	2	0.9	0.7	8.0	0.2		105	2.2		0.9	
	3	0.7	0.7	2.2	2.0		155	0.1		0.9	
	4	1.0	3.1	5.1	0.8		205	0.8		0.8	
	5	1.3	6.1	N/A			255	0.3		0.9	
	6	1.4	7.3			Cellular	rrc	β_{rrc}		β_{rrc}	
	7	0.4	6.0				IDLE	10.1		13.4	
	8	0.4	0.4				FACH	30.3		2.5	
	9	0.3	4.5				DCH	3.6		0.7	
	10	0.7	3.4			WiFi		β_{LT}	β_{HT}	β_{LT}	β_{HT}
	11	0.1	3.6				Transmit	0.3	7.4	6.5	0.4
GPS		β_{GPS}		β_{GPS}			Base	0.1	0.0	4.2	0.4
	ON	4.9		4.7			Threshold	0.0		0.0	

In the case of Cellular, the power coefficient β_{FACH} of N1 has bigger difference between online and offline models than other cases. This is because the FACH state is a hidden state as we described in Section 4.4. In summary, the results, indeed, validate the accuracy and effectiveness of the proposed scheme.

5. RELATED WORK

Power modeling on smartphones is an important research topic. The power model for each component is used to analyze the application power consumption [1, 18]. The model is used as a guideline for power management research for hardware component [3, 4, 13, 14]. The technique also provides important information for energy-efficient operating systems [15, 16].

Power modeling is accomplished with the information about the hardware activities, and also with information on the actual power consumption related to the activities. The information on hardware activities is generally provided by the operating system. The OS provides information about utilization for each hardware component, and power modeling is based on this. This approach is generally called utilization-based approach [1, 6, 7]. One difficulty of this approach is that every power state of a component is represented only with this information. That is, it is not possible to construct an accurate power model for the hardware components (e.g., WiFi) that have several power states based on the size of the workload. Pathak et al. [5] proposed an approach, called the syscall-based approach, based on the system call to overcome the limitation of utilization-based approach. They analyzed the system calls to get the status and the workload of each component, then constructed finite state machines based on them. This approach is interesting, but in-depth study and measurements on target devices should be required *a priori* to obtain detailed power states.

Power measurement methods are categorized into offline and online methods. Carroll and Heiser [6] measured the actual power consumption with a specific device of which electric schematic was open. However, it is not practical to apply this approach to commercial smartphones that have complex and closed circuit structures. Many researchers proposed one-time modeling with an external measurement tool in a laboratory environment [1, 5, 7]. However, the power consumption pattern of a smartphone is changing dynamically by external factors such as operating temperature or system software updates. One-time power models based on offline methods are static, so they cannot be adapted to the dynamic properties. There has been research that overcomes the weaknesses of offline methods with online methods. Dong and Zhong [2] proposed an accurate self-modeling mechanism based on online measurement method. However, the model targets the whole smartphone systems, and unlike ours, each component of smartphone cannot be analyzed with their approach. Zhang et al. [1] proposed an online measurement method using discharge voltage curve to make up for the weaknesses of offline measurement. However, the discharge voltage curve can be altered by external factors such as operating temperatures. Kj□rgaard et al. [17] proposed a power profiling technique using an online measurement method based on a genetic algorithm. However, the scheme requires additional resources to execute the algorithm, and the utilization factor which governs the power consumption of some hardware components (e.g., CPU) is not considered.

6. CONCLUSION

In this paper, we demonstrated automatic and online smartphone power modeling techniques, based on the use of BMU, which provides component power analysis and generates power model. We proposed a synchronization technique between update rate and component control for overcoming the limitations of online power modeling using BMU. Furthermore, we proposed an autonomous method for power state analysis to generate component power model for five components: CPU, LCD, WiFi, Cellular, and GPS. Through an extensive evaluation using the Android smartphones, we showed that DevScope derives component power models dynamically and nonintrusively, regardless of the BMU update rate, device models, and configurations. In summary, we proposed a practical smartphone power modeling technique to reflect both the diversity of the user environment and smartphone hardware complexity by implementation. The result of our research is expected to contribute to the development of energy-aware smartphone systems. For instance, the power model generated by DevScope is currently used for AppScope [18, 19], which is a framework to meter application energy consumption for Android smartphones.

7. ACKNOWLEDGEMENT

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.2012-0005522).

8. REFERENCES

[1] Zhang, L., Tiwana, B., Qian, Z., and Wang, Z. 2010. Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones. In *Proceedings of the 8th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis* (Scottsdale, AZ, USA, October 24 – 29, 2010). CODES+ISSS'10. ACM, New York, NY, 105-114. DOI= http://dx.doi.org/10.1145/1878961.1878982.

[2] Dong, M. and Zhong, L. 2011. Self-Constructive High-Rate System Energy Modeling for Battery-Powered Mobile Systems. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services* (Washington, D.C, USA, June 28 – July 1, 2010). MobiSys'11. ACM, New York, NY, 335-348. DOI= http://dx.doi.org/10.1145/1999995.2000027.

[3] Dong, M. and Zhong, L. 2011. Chameleon: A Color-Adaptive Web Browser for Mobile OLED Displays. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services* (Washington, D.C, USA, June 28 – July 1, 2010). MobiSys'11. ACM, New York, NY, 85-98. DOI= http://dx.doi.org/10.1145/1999995.2000004.

[4] Anand, B., Thirugnanam, K., Sebastian, J., Kannan, P. G., Ananda, A. L., Chan, M. C., and Balan, R. K. 2011. Adaptive Display Power Management for Mobile Games. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services* (Washington, D.C., USA, June 28 – July 1, 2010). MobiSys'11. ACM, New York, NY, 57-70. DOI= http://dx.doi.org/10.1145/1999995.2000002.

[5] Pathak, A., Hu, Y. C., Zhang, M., Bahl, P., and Wang, Y. – M. 2011. Fine-grained Power Modeling for Smartphones Using System Call Tracing. In *Proceedings of the ACM European Conference on Computer Systems* (Salzburg, Austria, April 10 – 13, 2011). EuroSys'11. ACM, New York, NY, 153-167. DOI= http://dx.doi.org/10.1145/1966445.1966460.

[6] Carroll, A. and Heiser, G. 2010. An Analysis of Power Consumption in a Smartphone. In *Proceedings of the 2010 USENIX Annual Technical Conference* (Boston, MA, USA, June 23 – 25, 2010). USENIX ATC'10. USENIX Association, Berkeley, CA.

[7] Shye, A., Scholbrock, B. and Memik, G. 2009. Into the Wild: Studying Real User Activity Patterns to Guide Power Optimizations for Mobile Architectures. In *Proceedings of the 42nd IEEE/ACM International Symposium on Microarchitecture* (New York, NY, USA, December 12 – 16, 2009). MICRO'09. ACM, New York, NY, 168-178. DOI= http://dx.doi.org/10.1145/1669112.1669135.

[8] DS2784 Datasheet, http://pdfserv.maxim-ic.com/en/ds/DS2784.pdf retrieved on 3/20/2012

[9] Google Nexus One Specification, http://www.htc.com/us/support/nexus-one-google/tech-specs/ retrieved on 3/20/2012

[10] Sony Ericsson Xperia Arc, http://www.sonymobile.com/gb/products/phones/xperia-arc/specifications/ retrieved on 3/20/2012

[11] Monsoon Solutions, Inc., http://www.msoon.com/LabEquipment/PowerMonitor/ retrieved on 3/20/2012

[12] Qian, F., Wang, Z., Gerber, A., Mao, Z. M., Sen, S., and Spatscheck, O. 2010. Characterizing Radio Resource Allocation for 3G Networks. In *Proceedings of the 10th ACM Internet Measurement Conference* (New York, NY, USA, November 1 – 3, 2010). IMC'10. ACM, New York, NY, 137-150. DOI= http://dx.doi.org/10.1145/1879141.1879159.

[13] Haverinen, H., Siren, J. and Eronen, P. 2007. Energy Consumption of Always-On Applications in WCDMA Networks. In *Proceedings of the IEEE 65th IEEE Vehicular Technology Conference* (Dublin, Ireland, April 22 – 25, 2007). VTC'07. IEEE, 964-968. DOI= http://dx.doi.org/10.1109/VETECS.2007.207.

[14] Qian, F., Wang, Z., Gerber, A., Mao, Z. M., Sen, S., and Spatscheck, O. 2011. Profiling Resource Usage for Mobile Applications: A Cross-layer Approach. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services* (Washington, D.C, USA, June 28 – July 1, 2010). MobiSys'11. ACM, New York, NY, 321-334. DOI= http://dx.doi.org/10.1145/1999995.2000026.

[15] Roy, A., Sumble, S., and Stutsman, R. 2011. Energy management in mobile devices with the Cinder operating system. In *Proceedings of the ACM European Conference on Computer Systems* (Salzburg, Austria, April 10 – 13, 2011). EuroSys'11. ACM, New York, NY, 139-152. DOI= http://dx.doi.org/10.1145/1966445.1966459.

[16] Vallina-Rodrigues, N. and Crowcroft, J. 2011. ErdOS: Achieving Energy Savings in Mobile OS. In *Proceedings of the ACM 6th International Workshop on Mobility in the Evolving Internet* (Washington, D.C., USA, June 28, 2011). MobiArch'11. ACM, New York, NY, 36-42. DOI= http://dx.doi.org/10.1145/1999916.1999926.

[17] Kj□rgaard, M. B. and Blunck, H. 2011. Unsupervised Power Profiling for Mobile Devices. In *Proceedings of the 8th International ICTS Conference on Mobile and Ubiquitous* (Copenhagen, Denmark, December 6 – 9, 2011). MobiQuitous'11.

[18] Yoon, C., Kim, D., Jung, W., Kang, C. and Cha, H. 2012. AppScope: Application Energy Metering Framework for Android Smartphone using Kernel Activity Monitoring. In *Proceedings of the 2012 USENIX Annual Technical Conference* (Boston, MA, USA, June 13 – 15, 2012). USENIX ATC'12. USENIX Association, Berkeley, CA.

[19] AppScope, http://mobed.yonsei.ac.kr/~appscope/ retrieved on 7/25/2012