



저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)

공학석사학위논문

Sensor-less, Event-Driven and Fine-Grain Instantaneous Power Estimation of Android Smartphones

센서리스 및 이벤트 반응형 기법을 이용한 안드로이드
스마트폰의 순간 전력소모량 추정

2014년 2월

서울대학교 대학원
전기컴퓨터 공학부
김기태

공학석사학위논문

Sensor-less, Event-Driven and Fine-Grain Instantaneous Power Estimation of Android Smartphones

센서리스 및 이벤트 반응형 기법을 이용한 안드로이드
스마트폰의 순간 전력소모량 추정

2014년 2월

서울대학교 대학원
전기컴퓨터 공학부
김기태

Sensor-less, Event-Driven and Fine-Grain Instantaneous
Power Estimation of Android Smartphones

센서리스 및 이벤트 반응형 기법을 이용한 안드로이드
스마트폰의 순간 전력소모량 추정

지도교수 장래혁

이 논문을 공학석사 학위논문으로 제출함

2013 년 11 월

서울대학교 대학원

전기컴퓨터 공학부

김기태

김기태의 공학석사 학위논문을 인준함

2013 년 12 월

위 원 장	_____	하순희	_____	(인)
부위원장	_____	장래혁	_____	(인)
위 원	_____	이창건	_____	(인)

Abstract

Multi-core processors and displays, which have a large-size panel over five-inches in diameter and high resolution close to 300 dot-per-inch, have become commonplace in smartphone designs, and they make long battery life a very challenging task even with a high capacity (about 3,000 mAh) Li-ion battery. Such mobile platforms are comprised of several major components including a multi-core processor, display with its controller, baseband signal processor, different types of memory, radio frequency communication module, various sensors, etc., and achieving power savings in each of those components is very important to reduce total system power consumption. Accurate power estimation for each of these components is the first step required to realize system-wide power optimization in smartphones. Moreover, accurate power estimation without the need for having physical sensors is crucial because most commercial smartphones do not have current sensors for their major components to measure its power consumption. Several researches present sensor-less power estimation methods that predict the power dissipation of a smartphone without any power measurement sensors. However, previous sensor-less power estimation techniques suffer from various problems such as the aliasing, inaccurate logging timestamp, and unobservable devices.

This paper introduces a novel sensor-less, event-driven power analysis framework for providing highly accurate and nearly instantaneous estimates of power dissipation in an Android smartphone. The key idea is to collect and correctly record various events of interest within a smartphone as applications are running on the application processor within it. This is in turn done by instrumenting the Android operating system to provide information about power and performance state changes of various

smartphone components at the lowest device driver layer of the kernel to avoid time stamping delays and component state observability issues. This technique then enables one to perform fine-grained (in time and space) power estimation in the smartphone. In addition, the proposed method takes account of the multi-core processor and organic light emitting diode (OLED) display in its power model while the previous power estimation techniques do not consider. Experimental results show significant accuracy improvement compared to previous approaches. The estimation error of the proposed method is lower by a factor of two with a good fidelity to actual current measurements.

Keywords: Android, Smartphone, Event-driven, Power estimation

Student Number: 2012-20744

Contents

Abstract	i
Contents	iii
List of Figures	v
List of Tables	vii
Chapter 1 Introduction	1
Chapter 2 Background	6
2.1 Sensor-less Power Estimation for Android Smartphones	6
2.2 Previous Approaches	7
Chapter 3 Fine-grained Event-driven Power Estimation Based on Device Driver Layer Event Monitoring	11
3.1 Device Driver Layer Event Monitoring	11
3.2 Power Modeling of Major Components	14
3.2.1 Multi-core Processor	14
3.2.2 Wi-Fi Module	16
3.2.3 Cellular Module	17

3.2.4	GPS Module	17
3.2.5	Display-related Modules	18
3.3	Time Granularity	20
Chapter 4	Implementation	21
4.1	Parameter Extraction	21
4.2	Online Power Estimation	23
Chapter 5	Experiments	26
5.1	Experimental Setup	26
5.2	Power Estimation Results	26
5.3	Overhead Analysis	30
Chapter 6	Conclusion	31
	Bibliography	32
	요약	35
	감사의 글	37

List of Figures

Figure 1.1	Measured and estimated system power consumption while browsing a website using the Google Chrome browser. The system power consumption is measured by a DAQ at 1 kHz sampling frequency, and the estimated power is predicted by a sensor-less and sampling-based power estimation method that has 1 Hz sampling rate. The 1 Hz estimation does not detect GPU on/off during 400 ms and 600 ms and perceive the Wi-Fi packet transmission start at 950 ms and the end at 1200 ms.	3
Figure 1.2	Power spectrum of a smartphone load current and the energy distribution by the frequency.	5
Figure 2.1	V_{batt} profile of a Li-ion battery during average 1 C discharging with pulsed load and constant load.	8
Figure 2.2	Overhead and measurement error of the activity profiling software.	9
Figure 3.1	Device block diagram of Qualcomm MDP MSM8660.	13

Figure 3.2	Measured OLED power, conventional LCD power estimation based on the screen brightness, and the average pixel intensity of the OLED display.	18
Figure 4.1	Overview of the proposed event-driven power estimation method. 22	
Figure 4.2	Structure of the proposed ANN display power model.	24
Figure 5.1	Long-term measured power by the DAQ and estimated power by the proposed method while playing a music video using the Android movie player.	28
Figure 5.2	Short-term power measurements by the DAQ and estimated power by the proposed method for video playback using the Android movie player.	28

List of Tables

Table 5.1	Coefficients of the proposed power model. The coefficient index of the processor denotes the operating frequency, and the coefficient index of display-related units denote the node number and index in the ANN model.	27
Table 5.2	RMS errors of estimated power consumption by the baseline and the proposed method compared with the golden reference.	29

Chapter 1

Introduction

Power consumption of smartphones is increasing with each generation of new devices. This power increase is caused by the need to provide more functionality, higher performance, ultra high resolution displays, high-speed wireless communication, etc. The capacity of batteries that power up such devices is also increasing, albeit at a much lower pace. This gives rise to the need to reduce power consumption of smartphones without limiting the functionality or curbing the performance. This is a very challenging undertaking considering the functional and performance requirements.

In this paper, we will use the term smartphone component to refer to all onboard modules in a commercial smartphone. These components include, for example, the application processor (AP), Wi-Fi, Bluetooth, global positioning system (GPS), cameras, display, flash memory and/or SSD storage, etc. The AP (e.g., Qualcomm Snapdragon S4 or Apple A6 APs) provides the computational and multimedia capabilities, and it also runs the operating system (OS) such as Google Android and Apple iOS. The APs in state-of-the-art smartphones are complex system-on-chip (SoC) devices containing multiple processor cores, several graphics processing units, large on-chip cache

memories, multimedia codec, etc. Clearly, it is essential for the success of any system-wide power management solution to have an accurate accounting of which components within the smartphone are consuming power at a given time instance by how much.

Power minimization in smartphones is typically done through some kind of power management solution whereby the unused components are power gated (turned off) while the power and performance level of active components is reduced to meet performance requirements. Other techniques such as scheduling tasks to maximize component idle time so that they can be put to sleep, offloading compute-intensive tasks to the cloud, etc. have also been suggested [1, 2, 3].

Using current sensors for each component can provide highly accurate power measurements, which show how much power each component is consuming instantaneously, in real time. Unfortunately, the stringent form factors, weight, and cost constraints for smartphones prevent a manufacturer from employing current sensor for each component. Instead, indirect power measurements, which estimate the power consumption of smartphones at some appropriate level of space and time granularity, are widely used. These power estimation techniques typically rely on offline, but state-dependent power characterization of components augmented by dynamically obtained information about the state of each component (e.g., active, idle, and sleep) and activity level (e.g., activity factor in a CPU and packet transmission rate for wireless link). In order to obtain the latter information, it is important to have the right system software support.

Critical to the success of the aforesaid approach is the ability to correctly and quickly collect information about the state of each component such as the operating state of the Bluetooth module, utilization level of the AP, or data communication rate of the Wi-Fi chipset in an asynchronous (event-driven) manner. However, this ability is hard to come by because it requires OS kernel support and fine-grained data collection of individual components with high timing accuracy in the smartphone. If these

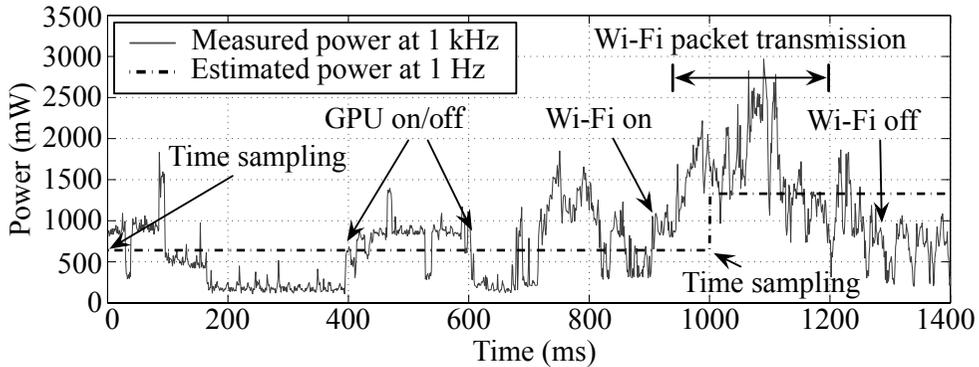


Figure 1.1 Measured and estimated system power consumption while browsing a website using the Google Chrome browser. The system power consumption is measured by a DAQ at 1 kHz sampling frequency, and the estimated power is predicted by a sensor-less and sampling-based power estimation method that has 1 Hz sampling rate. The 1 Hz estimation does not detect GPU on/off during 400 ms and 600 ms and perceive the Wi-Fi packet transmission start at 950 ms and the end at 1200 ms.

conditions are not met, indirect sensor-less power estimation will become inaccurate.

Figure 1.1 shows actually measured (1 kHz sampling) and estimated (1 Hz sampling) power dissipation profiles of a smartphone. The figure and its caption explain the power estimation challenge. In contrast, we will show a fine-grained event-driven indirect power estimation to capture most of the important events in a timely manner without aliasing. A further benefit of the fine-grained event-driven power estimation is that it can be used to guide dynamic power management (DPM) techniques for a smartphone. This is motivated by the fact that modern smartphone systems exhibit a rather short idle period as illustrated in Figure 1.1. Using the sampling-based approach for determining the state of a component does not capture the device state change in a timely manner, which can inhibit proper deployment of effective system-wide DPM policies.

Nevertheless, previous smartphone power estimation methods sample the component states in the order of once every second or so [2, 4, 5, 6]. This is an extremely low sampling rate for modern digital systems. Even if the authors knew about this shortcoming, they could not help it because using a sampling rate of even 10 Hz in their power estimation methods would not be practical due to rapid increase in the profiling overhead as well as timing errors in recording the events of interest.

The key to overcome limitations of previous approaches is twofold: 1) Avoid a regular sampling strategy and instead adopt event-driven recording of component state changes [7, 8]. This is a well-known technique not to lose time granularity while maintaining reasonable profiling overhead. However, there is a critical condition that should be satisfied in the event-driven approach. Events should be captured at a correct position in the event propagation flow of the operating system. 2) Capture and record events of interest at the lowest level of the OS kernel so that one can avoid time stamping delays and state observability issues caused by capturing events at improper positions.

We remind the reader that there are many different positions to capture device state change events in the propagation flow. Modern smartphones are equipped with a full-featured OS that has lots of software locks and memory buffers from a user program to hardware devices. An event (say a command to transmit data packets via the Wi-Fi module) that arrives at the OS kernel entry point can be delayed before it actually gets to the hardware component due to the locks and buffers, and vice versa. Hence, if the state changes of smartphone components are monitored at the highest level of the OS kernel, the time recording of these events can be far from reality of what happens in the device itself.

In theory and practice, it is not possible to achieve highly accurate instantaneous power estimation by sampling at a low rate or event monitoring at improper positions in the event propagation flow. Moreover, a comparison of estimation accuracy should be made with a golden reference that correctly captures the actual power dissipation

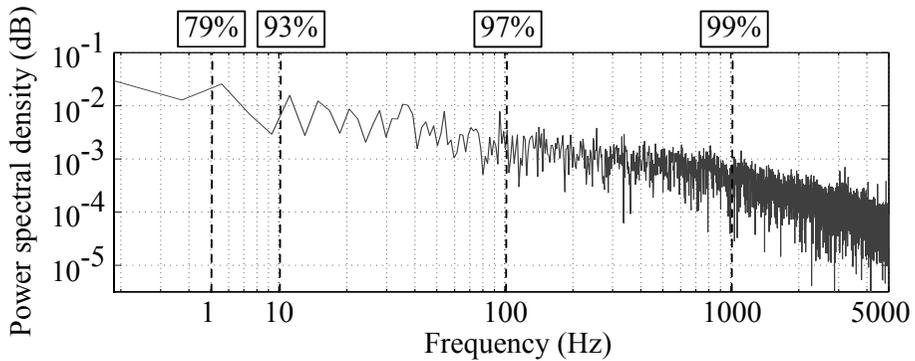


Figure 1.2 Power spectrum of a smartphone load current and the energy distribution by the frequency.

profile. In order to do this, we measure the load current of a commercial smartphone (Samsung Galaxy Nexus) using a high-speed data acquisition system (DAQ) at a 50 kHz sampling frequency and visualize its power spectrum in Figure 1.2. This figure shows that even a 200 Hz sampling frequency has a *temporal aliasing* because 3% of frequency components in the power spectrum are distributed above it. Since the golden reference should contain most important part of the power spectrum of smartphone current variation without excessive aliasing, we conclude that the sampling rate of the golden reference should be at least 1 kHz.

This paper i) first discloses the drawbacks in previous power estimation methods with theoretical evidences, ii) identifies a proper event capture position to alleviate the drawbacks, iii) presents the power model including the multi-core processor and organic light emitting diode (OLED) display as well as other major components, and iv) verifies the proposed method compared with an existing state-of-the-art event-driven method using the golden reference.

The experiments show that the proposed method reduces the instantaneous power estimation error up to 51.7% compared with the existing method, and the computation and energy overheads are only 3.1% and 1.5% (22 mW on average), respectively.

Chapter 2

Background

2.1 Sensor-less Power Estimation for Android Smartphones

The power consumption of a smartphone largely depends on the application and user behavior, which in turn determine component-level hardware activities and power modes. In other words, the power consumption highly correlates with the hardware component activities of the smartphone. Power modes of a component also affect the power consumption. Sensor-less power estimation methods read the components activities and power modes from the system to predict the system power consumption because there is a correlation among them as described above. Android includes the Linux kernel, which is specially customized for the Android framework, to serve its core services such as the memory, process, and hardware managements. The Linux kernel provides plentiful information about the smartphone including the component activities and power mode changes. Therefore, sensor-less power estimation methods can predict the power consumption based on the information, which is collected from the kernel, without additional hardware sensors. This is why recent researches pay

attention to the sensor-less power estimation method instead of the hardware sensor-based methods because it can be easily adopted on Android smartphones without any additional cost and extra space in the smartphones.

2.2 Previous Approaches

Previous works on sensor-less power estimation for smartphones can be classified into two categories: i) sampling-based methods, ii) event-driven methods.

Sampling-based methods periodically collect the information of activities and power modes from the Linux kernel and predict the system power consumption based on these information. The regression power model [2] estimates power consumption by logging user activities. The authors generate their power model from user activities using the linear regression analysis. PowerBooster [4] provides an online power model generation technique using the battery discharge curve and presents an estimation method based on the battery state-of-charge (SOC) estimation by battery output voltage, V_{batt} , monitoring. A feedback power model uses an adaptive modeling method based on the battery monitoring [9]. The model molding technique [5] is introduced to enhance the battery monitoring approaches. The on-chip bus power estimation method uses an energy-state model and cycle-accurate characterization [10]. In particular, the authors estimate the power consumption of off-chip hardware components based on the on-chip bus transaction measured by a performance monitoring unit (PMU). The PMU provides detailed information of bus transactions. The authors design the PMU by using a variable reduction technique to minimize required counters without considerable accuracy degradation.

However, the output voltage of the battery, V_{batt} , varies significantly depending on its load. Figure 2.1 presents V_{batt} profiles of a Li-ion battery with the 1 C discharging on average. The voltage profile under pulsed load varies significantly compared with

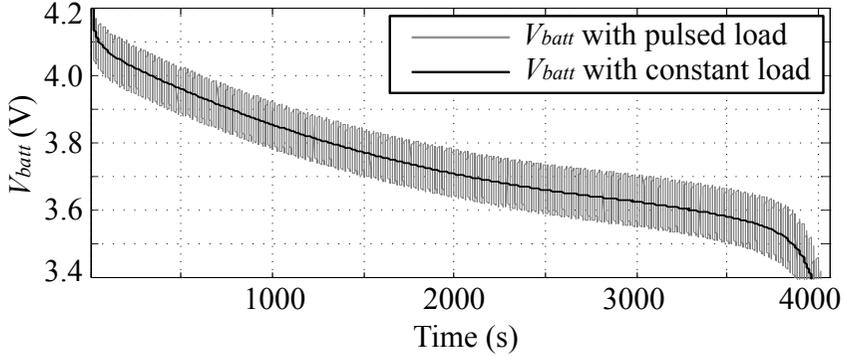


Figure 2.1 V_{batt} profile of a Li-ion battery during average 1 C discharging with pulsed load and constant load.

constant load in spite of the same average discharging current. V_{batt} difference is more than 10% in Figure 2.1. Furthermore, in reality, we are not able to obtain the energy difference result instantaneously because sufficient time is required to make meaningful V_{batt} difference compared with measurement error. The battery model in [4] is similar to the analytical model used in the energy estimation with Peukert’s law. According to the previous researches, Peukert’s formula shows an average error of 14% and maximum error of 43% compared with the Dualfoil simulation, which is an accurate simulator based on a physical model. Peukert’s formula works well for light and constant load, but the errors become very large at heavy or pulsed loads [11].

Moreover, a common issue that all sampling-based techniques face is the excessive performance and energy overhead due to frequent accessing the Linux kernel and reading large amount of data from the kernel. Figure 2.2 shows the profiling overhead and power measurement error of sampling-based approaches. In this experiment, we implement activity profilers that periodically collect system activities on user-space and kernel-space, respectively. One of the major error sources for the sampling-based techniques is the mismatch between the sampling period of the power measurement and system activity profiling. We can sample the current or voltage with up to 100 kHz

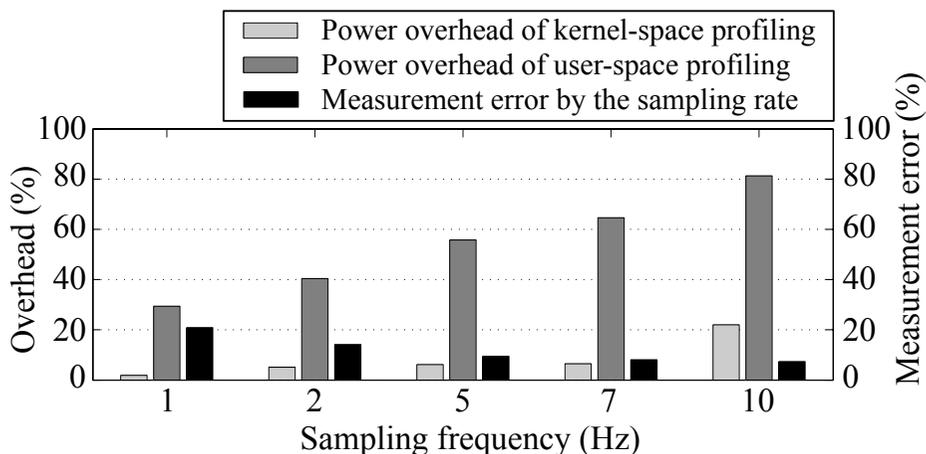


Figure 2.2 Overhead and measurement error of the activity profiling software.

without any problem by using an advanced measurement equipment like a DAQ. However, the system activity profiling frequency is merely several Hz in user-space and even kernel-space. If we increase the sampling rate of the system activity profiling to more than 10 Hz, the number of observable parameters drastically decreases or an application stops working properly due to the excessive profiling overhead. As shown in Figure 2.2, the measurement error reduces gradually from 2 Hz while the profiling overhead increases significantly. Although the profiling overhead of the kernel-space profiler is lower than the user-space profiler because it has less system calls from the user-space, it is still unacceptable for an online power estimation framework due to its high overhead. In addition, the authors in [5, 6] rely on the smart battery interface with a fuel gauge that can measure battery voltage, temperature, and even current sometimes. However, the sampling rate of battery fuel gauges in commercial smartphones is still several Hz that is too low to provide accurate measurement results. Moreover, the measurement error of the sampling-based methods is highly exaggerated and underestimated because they compare their power estimation result with the average power consumption of the smartphone measured in several Hz.

In contrast, event-driven methods collect system activities only when an event, which affects the system power consumption, occurs. This nature drastically alleviates the profiling overhead, which makes it more suitable for the sensor-less power estimation method. The simplest way for implementing the event-driven method is modifying the source codes of Android or the Linux kernel to record some information of the events. A fine-grained energy profiler called Eprof [7] predicts the power consumption by modifying the kernel code and accounting system call events. The authors in [8] develop a non-disruptive method called AppScope without any modification of existing source codes. They use a debugging tool for the Linux kernel to collect the information by setting a breakpoint on an arbitrary kernel function.

However, these previous event-driven approaches also have drawbacks such as inaccurate event timestamps, unobservable devices, and low estimation granularity. Moreover, the previous approaches cannot be extended to consider the multi-core processor and OLED display in its power model. These drawbacks hinder the accurate estimation and thus limit the DPM for smartphones.

Chapter 3

Fine-grained Event-driven Power Estimation Based on Device Driver Layer Event Monitoring

3.1 Device Driver Layer Event Monitoring

The Linux kernel consists of three important parts: the system call interface, the abstracted generic kernel layer, and the hardware-dependent device drivers. As the name implies, the system call interface is an interface between user-space applications and the generic kernel layer. The abstracted generic kernel layer is comprised of kernel routines that are independent from specific hardware components. The device drivers under the generic kernel layer are hardware-dependent routines for each hardware component that the kernel supports. Previous researches [7, 8] monitor system events on the abstracted generic kernel layer. However, their approaches are not accurate as they miss some events or collect inaccurate events. This is due to the fact that the abstracted generic kernel layer does not directly communicate with the hardware components.

In contrast, we directly monitor the events in the device driver layer. In practice, we

adopt a non-disruptive debugging tool and define customized callback handler functions for the tool in the device driver layer. For each device component, we program the callback handlers to monitor events of interest such that system activities or the power mode change. We also record the timestamp of each event as well. Based on these events, we obtain the necessary information such as operating frequency, utilization time, status (i.e., whether the device is being power gated or clock gated), and so on. These information are used in corresponding power model to obtain the power consumption for each component. Because power modes and utilization for each component are all controlled by their driver, monitoring the events in the device driver layer has two advantages as follows against previous approaches.

First, monitoring the events in the device driver layer provides the most accurate utilization time for hardware components. For example, in the network-related components, a significant time lag exist between a logged event timestamp at the generic kernel layer and actual time of the event at the device driver layer. This actually becomes one of main reasons why previous approaches did not yield accurate results in fine granularity. In fact, there are several buffers between the TCP/IP protocol stack of the generic kernel layer and the device driver of a network interface card (NIC). During a packet transmission, transmission-related events are captured immediately after sending packets are inserted into a socket buffer in the TCP/IP stack if we monitor the events at the generic kernel layer. However, the packets are not sent out until they move to the top of a queue in the NIC device driver. Therefore, packets in the buffer and the queue can be delayed during the transmission depending on the current system status and Linux kernel's policy. The delay increases under the following scenarios: i) there is only a small amount of packets to be sent in the buffer, ii) the Wi-Fi signal strength is not good enough, iii) or the remote host does not respond quickly. These transmission lags are pretty random depending on the current status of the system. Our experiment shows that the average and standard deviation of the time lags are 1 ms and

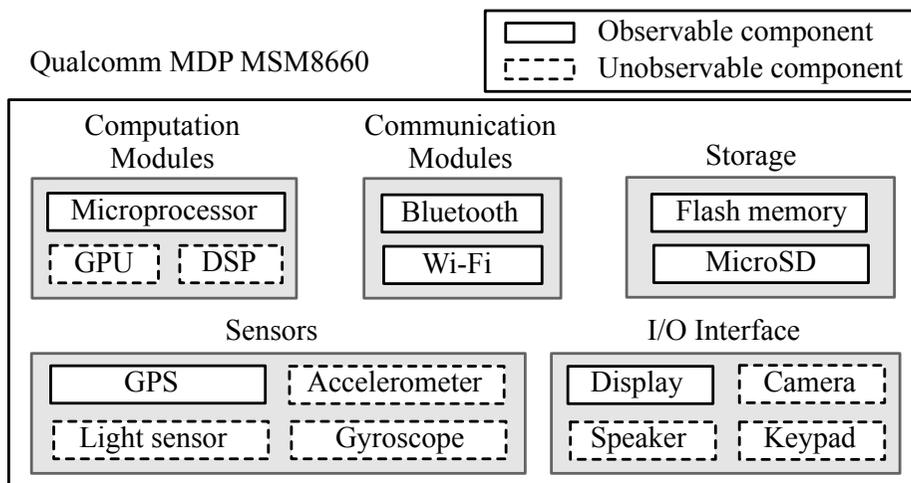


Figure 3.1 Device block diagram of Qualcomm MDP MSM8660.

3.2 ms, respectively. These values are large enough to cause significant inaccuracy in fine-grained power estimation.

Second, monitoring the events in device driver layer allows us to access more devices. As the abstracted generic kernel layer communicates with the underlying device drivers through pre-defined interfaces, it does not know the detail of specific hardware operations in the device drivers. Therefore, the existing techniques [7, 8] are unable to capture some events from several hardware components such as the GPU, digital signal processor (DSP), and GPS. For example, there is no source code to control activity of the GPS module in the generic kernel layer. Although the authors of [8] manage to overcome the GPS issue by collecting events from the Android framework, it is less accurate compared to doing it in the device driver layer. By monitoring the events in the device driver layer, we have access to all of these components aforementioned. Figure 3.1 shows observable and unobservable components from the abstracted generic kernel layer in MDP MSM8660, which is a development platform made by Qualcomm.

3.2 Power Modeling of Major Components

In this paper, we use Samsung Galaxy Nexus, which is an Android smartphone developed by Google and Samsung in October 2011, as the target smartphone platform. We identify several power consuming components in the target platform including the multi-core processor, Wi-Fi, cellular, GPS, and display modules. For each component, we present a power model based on the accessible information such as the frequency, utilization, power modes, and so on. Note that these components are "logical" in the sense that the CPU and GPU may be physically integrated on the same AP chip, but they are typically activated at different time when running different applications. Thus, we are able to report the power consumption of CPU separately from that of the GPU although they are on the same physical chip. In addition, the power consumption of other supporting circuitry when the CPU is active will be reported as the CPU power. For example, memory power is included in the CPU power report because these two physical components are logically working together when the CPU is active.

3.2.1 Multi-core Processor

The CPU is one of the major power consumers in the smartphone [12]. Common APs include multiple processor cores in the SoC design of an AP to provide the computational and processing power of state-of-the-art smartphones. Unfortunately, previous researches focusing on power characterization and modeling of smartphones [2, 4, 6, 8] have only considered a simple single-core processor. As a result, these techniques become inaccurate when multiple applications and/or multiple threads of the same application are running on the multi-core processor in the state-of-the-art smartphones.

An intuitive way to build a multi-core power model is to linearly scale up the single-core power model by the number of cores. However, we observe that the power

consumption of a dual-core processor in the target platform does not increase linearly with the number of active cores. Thus, we characterize the dual-core power consumption as a function of the operating frequency, utilization, and normalized co-utilization rate of each core. The utilization of each core is the amount of time that the core has spent in user and kernel mode for handling a process. The normalized co-utilization rate of each core that lies between 0 and 1 is a ratio that the core has been concurrently used with another core in the utilization time, and it is calculated through dividing the concurrent utilization time by the total utilization time of the core.

The proposed method monitors events related with a change of the operating frequency, utilization, and suspend state of each core. We apply different power models depending on the current status of the dual-core processor as follows: i) if the dual-core processor is not in suspend mode, and the total utilization of all cores is higher than zero,

$$P_{CPU} = \frac{1}{\Delta t} \sum_{i=1}^n \left[r_i u_i^{cpu} \beta_{dual[freq]}^{cpu} + (1 - r_i) u_i^{cpu} \beta_{single[freq]}^{cpu} \right], \quad (3.1)$$

ii) if the dual-core processor is not in suspend mode, but the total utilization of all cores is zero,

$$P_{CPU} = \beta_{idle[freq]}^{cpu}, \quad (3.2)$$

iii) if the dual-core processor is in suspend mode,

$$P_{CPU} = \beta_{suspend}^{cpu}, \quad (3.3)$$

where Δt is the time difference between two successive events, and n is the number of cores, which is two in this paper. u_i^{cpu} and r_i are the utilization and the normalized co-utilization rate of the i -th core during Δt . We figure out the utilization of each core from the usage statistics variable called *cpu_usage_stat* that cumulates CPU execution and idle time in the Linux kernel. β_{single}^{cpu} , β_{dual}^{cpu} , β_{idle}^{cpu} , and $\beta_{suspend}^{cpu}$ denote power coefficients (with the unit of W) for the single-core, dual-core, idle, and suspend states,

respectively, at $freq$ MHz operating frequency. We obtain values of all power coefficients through a characterization process that we will describe later. The CPU power model can be improved to account for the thermal effect caused by elevated temperature since the temperature sensors in the smartphone are all accessible from the Linux kernel. However, it is beyond the scope of delivering the concept of this paper.

3.2.2 Wi-Fi Module

As we explain in Section 3.1, we monitor the transmission events in the NIC device driver for the Wi-Fi module. We obtain the accurate event timestamp because the events happen immediately before and after the actual transmission occurs. In addition, the Wi-Fi module has a power saving mode. This mode allows the Wi-Fi module turns off its transmitter and receiver if it does not have any packet to transmit. In general, the power consumption of the Wi-Fi module depends on its power mode and the packet transmission rate [13]. Thus, we monitor the clock on/off event of the module and model the Wi-Fi module as follows:

$$P_{WIFI} = \begin{cases} \alpha^{wifi} \left(\beta_{ht}^{wifi} + u^{wifi} \cdot \beta_{weight}^{wifi} \right) & u^{wifi} > u_{threshold}^{wifi}, \\ \alpha^{wifi} \cdot \beta_{lt}^{wifi} & u^{wifi} \leq u_{threshold}^{wifi}, \end{cases} \quad (3.4)$$

where u^{wifi} is the packet transmission rate during the past 500 ms. α^{wifi} is 0 if the clock of the Wi-Fi module is disabled, and 1 otherwise. β_{ht}^{wifi} and β_{lt}^{wifi} denote power coefficients for the high and low packet transmission rates, respectively, and β_{weight}^{wifi} is the weight factor of the Wi-Fi module based on the packet transmission rate. $u_{threshold}^{wifi}$ is the threshold rate of the packet transmission. We pick the optimal threshold from our experiments.

3.2.3 Cellular Module

The cellular module has a similar issue to the Wi-Fi module because Android uses the cellular module as a network adaptor if the Wi-Fi connection is unavailable. We monitor the cellular events in the transmission routines of the cellular module device driver. The cellular module repeatedly changes its power mode between an awake and a sleep mode depending on its load, and it goes into the suspend mode when the user explicitly disables the cellular connection. The power consumption of the cellular module is related with its power mode and packet transmission rate [13]. Therefore, we monitor the clock on/off events of the cellular module as well as the packet transmission rate. Our cellular power model is given as

$$P_{CELL} = \begin{cases} \beta_{awake}^{cell} + u^{cell} \cdot \beta_{weight}^{cell} & \text{if power mode is awake,} \\ \alpha^{cell} \cdot \beta_{sleep}^{cell} & \text{otherwise,} \end{cases} \quad (3.5)$$

where β_{awake}^{cell} and β_{sleep}^{cell} denote power coefficients for the awake mode and the sleep mode, respectively, and β_{weight}^{cell} is the weight factor of the cellular module based on the packet transmission rate. u^{cell} is the packet transmission rate during the past 500 ms, and α^{cell} is 0 if the cellular module is in the suspend mode, and 1 otherwise.

3.2.4 GPS Module

The GPS power consumption depends to its power mode. The Android OS enables the GPS module when an application requests the current location of the smartphone and disables the GPS immediately after the location service is done. We monitor the GPS clock on/off event in the driver layer, and the power model is defined as

$$P_{GPS} = \alpha^{gps} \cdot \beta_{on}^{gps}, \quad (3.6)$$

where α^{gps} is 0 if the GPS clock is disabled, and 1 otherwise. β_{on}^{gps} is the power coefficient for the GPS module.

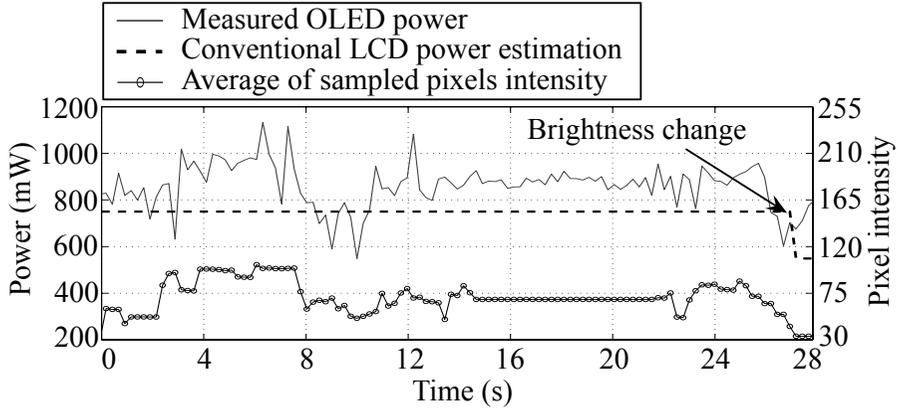


Figure 3.2 Measured OLED power, conventional LCD power estimation based on the screen brightness, and the average pixel intensity of the OLED display.

3.2.5 Display-related Modules

The power consumption of an OLED display depends on the pixel color that the display shows [14, 15], so the conventional power estimation methods that only use a screen brightness for a liquid-crystal display (LCD) are not appropriate for the OLED display. However, previous researches [2, 4, 6, 8] only present the LCD power model, so they significantly underestimate the power consumption when the OLED display in the smartphone frequently changes its screen contents. Figure 3.2 shows the importance of considering the pixel color for OLED power estimation. In this experiment, we monitor the brightness change events and periodically calculate an average pixel intensity from the OLED display. The figure shows that there is a strong correlation between the OLED power consumption and the average pixel intensity of the screen. This implies that we should consider the average pixel intensity for more accurate OLED power estimation.

In this paper, we present a contents-aware low-overhead OLED display modeling. Android draws its video frame to the display device through the framebuffer of the

Linux kernel, which is a memory area containing the pixel data of a current video frame that the display shows. For taking into account of the OLED display, we read the pixel data and calculate the average pixel intensity from the framebuffer.

In addition, we find that there are correlations in terms of the utilization among several display-related hardware components. We analyze the interaction among the hardwares using the correlation coefficient analysis calculated from the utilization of each component. The correlation coefficient among two components' utilizations X and Y is given by

$$\rho(X, Y) = \frac{cov(X, Y)}{\sigma_X \sigma_Y}, \quad (3.7)$$

where $cov(X, Y)$ is the covariance of X and Y . σ_X and σ_Y denote the standard deviations of X and Y , respectively. The analysis shows that the correlation coefficients among the OLED display, GPU, and DSP are higher than 0.75. This means these components are highly correlated to each other. Therefore, we model these components all together using the artificial neural network (ANN), which is a non-linear regression method. ANN is more robust to the correlation in the input variables and shows better result than linear regression analysis in various domains [16, 17, 18]. We characterize the OLED power consumption as a function of the average pixel intensity and the utilization of the GPU and DSP modules, so we monitor the GPU and DSP clock on/off events in each device driver. The OLED display power model is given as

$$P_{DISP} = \sum_{k=1}^p \left[\theta \left(\sum_{j=1}^n u_j^{disp} \beta_{iw[j][k]}^{disp} + \beta_{ib[k]}^{disp} \right) \beta_{hw[k]}^{disp} \right] + \beta_{ob}^{disp}, \quad (3.8)$$

where p and n are the number of nodes in the hidden layer and the input layer of the ANN model, and we use $p = 5$ and $n = 3$ in this paper. θ is the sigmoid function that is given by

$$\theta(x) = (1 + e^{-x})^{-1}. \quad (3.9)$$

u_1^{disp} , u_2^{disp} , and u_3^{disp} are the average of sampled pixels intensity, DSP utilization, and GPU utilization, respectively. β_{iw}^{disp} , β_{hw}^{disp} , β_{ib}^{disp} , and β_{ob}^{disp} denote power coefficients

for the input node weights, hidden node weights, input node biases, and output node bias of the ANN model, respectively.

3.3 Time Granularity

As we described in Section 2.2, the existing sampling-based techniques cause excessive energy overhead depending on its sampling frequency. According to our experiment, the energy overhead of the sampling-based method gradually increases from 1 Hz sampling frequency and saturates at about 10 Hz. On the other hand, the event-driven method does not suffer from the overhead thanks to its asynchronous nature, so it can perform fine-grained estimation whose granularity is higher than the sampling-based methods. However, the existing event-driven technique [8] still make the power estimation in a low time granularity, i.e., it gives one estimation result per second no matter how many events are logged during this second. Thus, we refer its power estimation granularity as 1 Hz, and our experimental results show that such coarse-grained power estimation results in significant power estimation error. Therefore, we employ the fine-grained approach with 1 kHz power estimation granularity because it is enough to cover most of the power spectrum in the smartphone load currents, as shown in Figure 1.2. We observe that the granularity levels higher than 1 kHz considerably increase the overhead and do not provide much more gain though.

Chapter 4

Implementation

4.1 Parameter Extraction

We take the non-disruptive event-driven approach using a debugging tool called KProbes [19], which is developed by IBM. KProbes gives a non-destructive way to set a breakpoint to an arbitrary function in the kernel and specify a callback handler function that is called whenever the breakpoint hits. In this paper, we use Samsung Galaxy Nexus as a target platform. We implement an event profiler as a kernel module to set breakpoints to several device driver routines and define customized callback handler functions. The customized callback handlers monitor events of interest related to the system activities and the power modes. In addition, we record the timestamp of each event as well as other useful information (e.g., function parameters and global variables). Figure 4.1 shows the architecture overview of the proposed method.

We need to obtain the per component power consumption so that we can extract power coefficients. As state-of-the-art commercial smartphones do not provide facilities to let us directly measure power consumptions per each component, we carefully

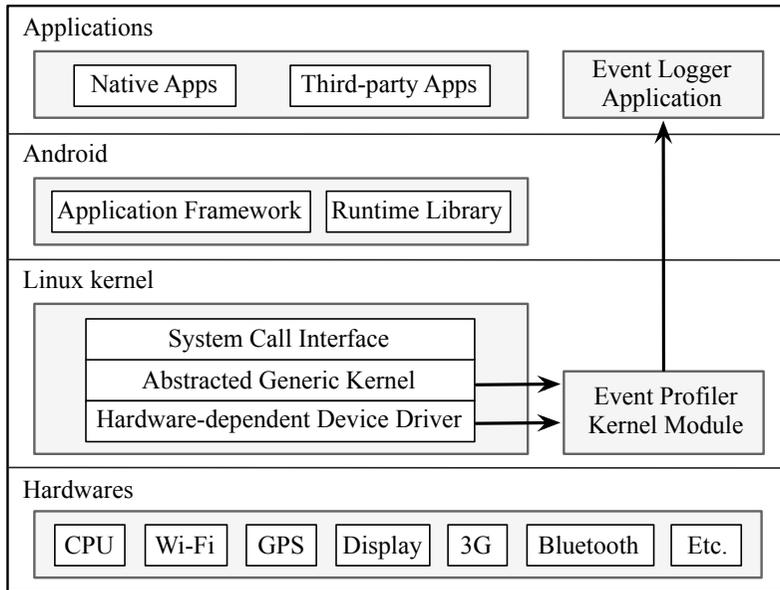


Figure 4.1 Overview of the proposed event-driven power estimation method.

design some usecases to disable and enable one component at a time while others are remained unchanged. We also subtract the CPU power so that the per component power consumptions are mutually exclusive. We compare the total power consumption before and after we run these usecases to obtain the power consumption for each component.

We first extract the multi-core processor model parameters. Our event profiler monitors a change of the operating frequency, utilization, and suspend state on the *cpufreq_notify_transition* and *omap4_cpu_suspend* functions in the Linux kernel. We implement a custom test-bench application to model the dual-core processor in the target platform using various usecases. We measure the CPU power consumption while the test-bench is executing, and all the other hardware components are disabled or at least utilized at a constant usage level if they cannot be disabled. We also log the processor-related events using the event profiler. Once we have done the measurement

and logging procedures, we eliminate the time shift between the two signals using the normalized cross-correlation and extract the model parameters from these aligned signals using the regression analysis.

We characterize other components in a similar procedure. For each component, we run usecases that only make changes to that component and measure the total power consumption accordingly. We first subtract the estimated processor power from the total system power consumptions measured in these scenarios. Then we compare the power consumption results, which already exclude contribution from the processor, to obtain the power consumption of the component of interest. This is because we cannot control the processor as we wish when we characterize other hardware components. Our event profiler monitors the power mode change and the packet transmission start/stop events in the *sdioh_request_packet* and *dhdsdio_sdclk* functions for the Wi-Fi module and the *hsi_ioctl*, *hsi_read*, and *hsi_write* functions for the cellular module. It also monitors the clock on/off events in the *omap2_clk_enable* and *omap2_clk_disable* functions of the OMAP device driver for the GPS, GPU, and DSP modules.

4.2 Online Power Estimation

After we characterize the devices and extract the necessary power coefficients, we input these information to our online power estimation models. Among these models, Figure 4.2 shows the structure of the proposed ANN display power model described in Equation 3.8. We read the screen pixel data and calculate its average pixel intensity from the framebuffer of the Linux kernel to estimate the OLED display power consumption. We repeatedly perform this process because changing the pixels in the framebuffer does not raise any event that the proposed event-driven method can detect. In order to avoid excessive profiling overhead, we read the framebuffer twice per second and divide each video frame into 3-by-3 pixel blocks, and then we use the

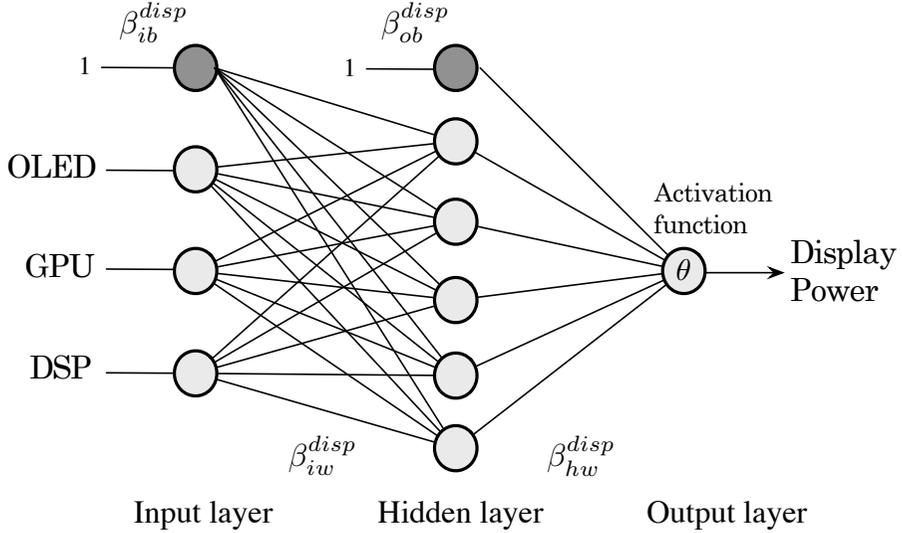


Figure 4.2 Structure of the proposed ANN display power model.

center pixel of each block to calculate the average pixel intensity of the OLED display. Moreover, Android OS provides a feature to adjust the screen brightness even if the smartphone has an OLED display, which does not have a backlight module. A controller of the OLED display adjusts its screen brightness using gamma correction depending on the brightness level that the user sets. Accordingly, the pixel data in the framebuffer remains unchanged even though the brightness level changes, but the brightness change of the screen affects the power consumption of the OLED display. Therefore, we weight the average pixel intensity by the current screen brightness to compensate for the difference between the calculated average pixel intensity from the framebuffer and the actual average pixel intensity that the display shows. We use the weighted average pixel intensity as an input variable of the ANN display power model in Equation 3.8.

For online usage, the event profiler captures the system events and records necessary information such as operating frequencies and power modes. Our power meter

application, which is executed in user-space, reads the event log and predicts the system power consumption based on the presented power model. In addition, the event profiler also gather a process identifier when it captures events. The process identifier shows which process raised the event, and it is used to trace back applications and accumulate the estimated power consumption of each application in the system. Therefore, the power meter can show the energy profiles for each application as well as the total system power consumption.

Chapter 5

Experiments

5.1 Experimental Setup

We use the NI-9227 DAQ from National Instruments to measure the load current of the target platform, which is Samsung Galaxy Nexus, and also use the E3648A DC power supply from Agilent to provide a constant voltage to the target platform instead of its internal battery. We measure the power consumption of the target platform at a 50 kHz sampling frequency from the NI DAQ, which is used as the golden reference, and characterize the power coefficients of our power models for the target platform, as shown in Table 5.1. The coefficient indices for CPU and display components in Table 5.1 denote the CPU operating frequency (MHz) and the node number/index in the ANN display model, respectively.

5.2 Power Estimation Results

We evaluate the proposed method using several famous Android applications and obtain the power consumption results using the proposed method based on the monitored

Table 5.1 Coefficients of the proposed power model. The coefficient index of the processor denotes the operating frequency, and the coefficient index of display-related units denote the node number and index in the ANN model.

Component	Coeff.	Coeff. index	Value (W)	Coeff.	Coeff. index	Value (W)
CPU	β_{single}^{cpu}	350	0.95	β_{dual}^{cpu}	350	1.08
		700	1.26		700	1.53
		920	1.58		920	1.98
		1200	1.89		1200	2.61
	β_{idle}^{cpu}	350	0.86	$\beta_{suspend}^{cpu}$	N/A	0.72
		700	1.04			
		920	1.35			
		1200	1.58			
Display	1	1	2.24	β_{hw}^{disp}	1	-0.30
		2	-1.33		2	1.07
		3	0.83		3	0.16
		4	-0.88		4	-1.04
		5	2.45		5	-0.28
	2	1	-0.81	β_{ib}^{disp}	1	2.94
		2	-3.03		2	-9.63
		3	-9.21		3	19.91
		4	-3.62		4	-2.68
		5	16.21		5	-12.62
	3	1	2.65	β_{ob}^{disp}	N/A	0.10
		2	9.51			
		3	-1.99			
		4	2.34			
		5	-4.72			
Cellular	β_{awake}^{cell}	N/A	0.90	β_{weight}^{cell}	N/A	3.24E-4
	β_{sleep}^{cell}	N/A	0.54			
Wi-Fi	β_{ht}^{wifi}	N/A	0.68	β_{weight}^{wifi}	N/A	2.21E-4
	β_{lt}^{wifi}	N/A	0.32	$u_{threshold}^{wifi}$	N/A	850
GPS	β_{on}^{gps}	N/A	0.14			

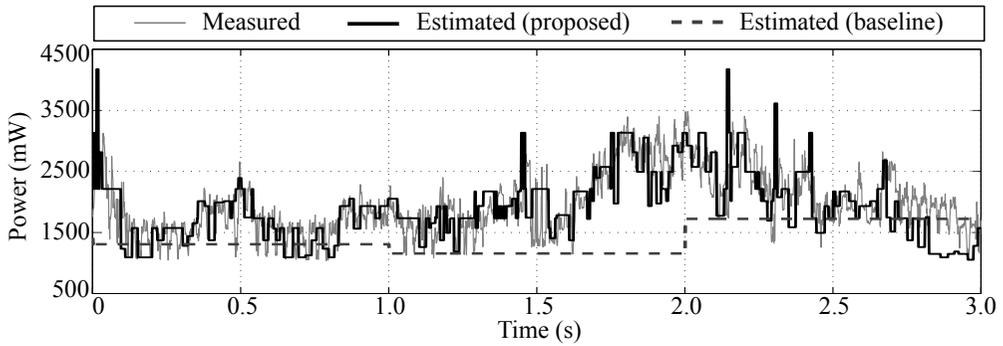


Figure 5.1 Long-term measured power by the DAQ and estimated power by the proposed method while playing a music video using the Android movie player.

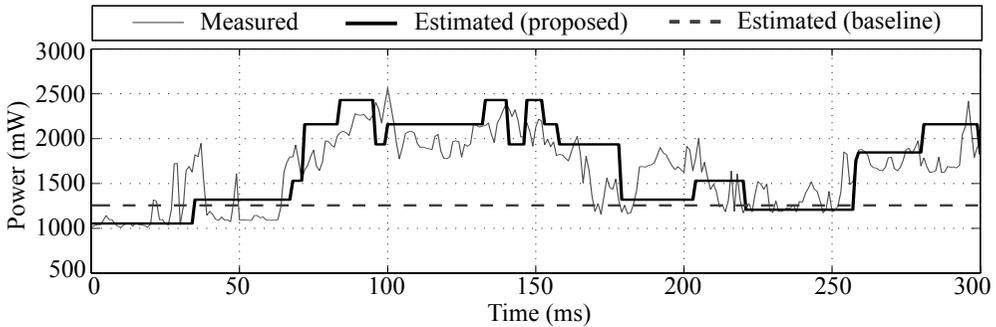


Figure 5.2 Short-term power measurements by the DAQ and estimated power by the proposed method for video playback using the Android movie player.

and measured data with 1 kHz estimation granularity. We also implement the baseline method presented in [8] to the best we can, and then compare our results with the baseline results and the golden reference. In the experiments, each set of comparison lasts for three seconds. This three-second time interval is a long term for the instantaneous power estimation, and it is also enough to analyze the state transition of hardware components and decide a proper DTM strategy for the current condition in smartphones.

Table 5.2 RMS errors of estimated power consumption by the baseline and the proposed method compared with the golden reference.

Application	Baseline (mW)	Proposed (mW)	Error reduction
Movie Player	901	435	51.7%
Angry Birds	1132	584	48.4%
Skype (Wi-Fi)	938	510	45.6%
YouTube	957	564	41.1%
Skype (Cellular)	1010	612	39.4%
Chrome Browser	1113	693	37.7%
Google Maps	920	593	35.5%

Figures 5.1 and 5.2 show the long-term and short-term power estimation results for video playback using the proposed method, baseline, and golden reference, respectively. Compared to the coarse-grained baseline results, our method yields more fine-grained accurate estimation results. The baseline method fails to capture the instantaneous power changes of components because its estimation granularity is too low, and it significantly underestimates the power consumption of some components such as the multi-core processor, OLED display, GPU, and DSP due to the lack of consideration of these components and the unobservable device problem. The root-mean-square (RMS) errors of the proposed method and the baseline in Figure 5.2 are 293 mW and 538 mW, respectively. Table 5.2 provides the long-term evaluation results for various applications. The proposed method reduces the power estimation RMS error by up to 51.7% compared with the baseline.

5.3 Overhead Analysis

The proposed method based on the event-driven approach has less overhead than the previous methods that adopt the sampling-based approaches thanks to the asynchronous nature of the event-driven method. However, although the profiling overhead of the proposed method is not significant, the overhead is not negligible and unavoidable even if we use the event-driven approach.

Thus, we analyze the overhead of the proposed method in terms of computation and energy by measuring and comparing the CPU execution time and the system power consumption of the target platform with and without our event profiler, which is a kernel module, under the idle, normal load, and heavy load conditions, respectively. In this experiment, the analysis shows that the computation and energy overheads of the proposed method are 3.1% and 1.5% (22 mW on average), respectively. On the other hand, the baseline method shows 5.9% computation overhead and 34.9 mW energy overhead on average in the same conditions.

Chapter 6

Conclusion

We introduce a novel sensor-less, event-driven power analysis framework for providing highly accurate and nearly instantaneous estimates of power dissipation in an Android smartphone. We monitor system events in the device driver layer of the Linux kernel to obtain necessary information such as power modes and activities of hardware components. By collecting the data at the lowest layer of the kernel, we manage to obtain accurate information and get access to many devices, which are not accessible in upper layer of the kernel. We present power models for identified components and perform a characterization process to extract the power coefficients for the proposed power models. The online power meter monitors system events and calculates the fine-grained power consumption of the smartphone based on the proposed power models. Experimental results show significant accuracy improvement compared to previous power estimation approaches. The estimation error of the proposed method is lower by a factor of two, and our method shows a good fidelity with respect to actual current measurements in a Samsung Galaxy Nexus smartphone.

Bibliography

- [1] B.-G. Chun and P. Maniatis, “Augmented Smartphone Applications Through Clone Cloud Execution.” in *Proceedings of Hot Topics in Operating Systems*, vol. 9, pp. 8-11. 2009.
- [2] A. Shye, B. Scholbrock, and G. Memik, “Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures.” in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 168-178. ACM, 2009.
- [3] A. Roy, S. M. Rumble, R. Stutsman, P. Levis, D. Mazieres, and N. Zeldovich, “Energy management in mobile devices with the Cinder operating system.” in *Proceedings of the sixth conference on Computer systems*, pp. 139-152. ACM, 2011.
- [4] 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 *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pp. 105-114. ACM, 2010.

- [5] M. Dong and L. Zhong, “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*, pp. 335-348. ACM, 2011.
- [6] W. Jung, C. Kang, C. Yoon, D. Kim, and H. Cha, “DevScope: a nonintrusive and online power analysis tool for smartphone hardware components.” in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pp. 353-362. ACM, 2012.
- [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.” in *Proceedings of the 7th ACM european conference on Computer Systems*, pp. 29-42. ACM, 2012.
- [8] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha, “Appscope: Application energy metering framework for android smartphone using kernel activity monitoring.” in *Proceedings of the USENIX annual technical conference*, 2012.
- [9] S. Gurun and K. Chandra, “A run-time, feedback-based energy estimation model for embedded devices.” in *Proceedings of the 4th international conference on Hardware/software codesign and system synthesis*, pp. 28-33. ACM, 2006.
- [10] Y. Kim, S. Park, Y. Cho, and N. Chang, “System-level online power estimation using an on-chip bus performance monitoring unit.” in *Proceedings of Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, pp. 1585–1598, IEEE, 2011.
- [11] D. Rakhmatov, S. Vruthhula, and D. Wallach, “Battery lifetime prediction for energy-aware computing.” in *Proceedings of the international symposium on Low power electronics and design*, pp. 154-159. ACM, 2002.

- [12] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone." in *Proceedings of the USENIX annual technical conference*, pp. 21-21. 2010.
- [13] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: a measurement study and implications for network applications." in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pp. 280-293. ACM, 2009.
- [14] D. Shin, Y. Kim, N. Chang, and M. Pedram, "Dynamic voltage scaling of OLED displays." in *Proceedings of the 48th Annual Design Automation Conference*, pp. 53-58. IEEE, 2011.
- [15] M. Dong, Y.-S. K. Choi, and L. Zhong, "Power modeling of graphical user interfaces on OLED displays." in *Proceedings of the 46th Annual Design Automation Conference*, pp. 652-657. ACM, 2009.
- [16] M. A. Bocco, E. Willington, and M. A. Arias, "Comparison of regression and neural networks models to estimate solar radiation." in *Proceedings of Chilean journal of agricultural research*, no. 3, pp. 428-435, 2010.
- [17] H.-T. Pao, "A comparison of neural network and multiple regression analysis in modeling capital structure." in *Proceedings of Expert Systems with Applications*, no. 3, pp. 720-727, 2008.
- [18] M. Carolin and E. Fernandez, "Analysis of wind power generation and prediction using ANN: A case study." in *Proceedings of Renewable Energy*, no. 5, pp. 986-992, 2008.
- [19] A. Mavinakayanahalli, P. Panchamukhi, J. Keniston, A. Keshavamurthy, and M. Hiramatsu, "Probing the guts of kprobes." in *Linux Symposium*, vol. 6, 2006.

요약

최근에 판매되는 고성능 스마트폰은 멀티코어 프로세서 및 5인치 이상의 고해상도 디스플레이 등과 같이 전력소모량이 매우 높은 부품들을 장착하고 있다. 이로 인해 3,000 mAh 이상의 대용량 배터리를 사용하더라도 스마트폰의 가용시간을 늘리는 것은 점점 더 어려워지고 있다. 이런 고성능 스마트폰은 멀티코어 프로세서 및 고해상도 디스플레이 외에도 베이스밴드 프로세서, 다양한 종류의 메모리, 통신 모듈 및 각종 센서 등의 주요 하드웨어 부품들로 구성되는데, 스마트폰의 가용시간을 늘리기 위해서는 이들 개별 부품의 전력소모량을 줄이는 것이 매우 중요하다. 그리고 개별 부품의 전력소모량을 정확하게 측정하는 것은 전력절감을 위한 첫번째 필수 단계이다. 그러나 시중에 판매되는 대부분의 스마트폰은 전력소모량을 측정할 수 있는 물리적인 센서를 장착하고 있지 않기 때문에 물리적 센서의 도움 없이 전력소모량을 간접적으로 추정할 수 있는 방법이 필요하다. 하지만 기존에 발표된 전력소모량 추정 관련 연구들은 신호왜곡현상, 부정확한 타임스탬프 기록, 일부 하드웨어 부품의 사용량을 관찰할 수 없는 등의 문제를 가지고 있다.

본 논문에서는 물리적 센서를 사용하지 않고 이벤트 구동(event-driven) 방식으로 각종 하드웨어 이벤트를 수집하는 기법을 도입하여 안드로이드 스마트폰의 순간 전력소모량을 간접적으로 보다 정확하게 추정할 수 있는 프레임워크를 제안한다. 본 논문의 핵심 아이디어는 스마트폰의 전력소모량과 관련된 다양한 하드웨어 이벤트들을 수집하고 정확하게 기록한 후, 이를 기반으로 순간 전력소모량을 추정하는 것이다. 본 연구에서 제안하는 방법에서는 부정확한 타임스탬프 기록이나 일부 부품의 사용량을 관측할 수 없는 등 기존 연구들에서 발생하는 다양한 문제를 해결하기 위해 리눅스 커널의 최하위 디바이스 드라이버 계층에서 개별 하드웨어 부품들의 사용량 및 각종 상태 변화를 측정하고 기록한다.

또한 기존 연구보다 더욱 세밀한 전력소모량 추정 결과를 제공한다. 본 연구에서 수행한 실험 결과에 따르면 제안하는 방법이 기존 연구에 비해 보다 우수한 전력소모량 추정 결과를 보여주었다. 삼성 갤럭시 넥서스 스마트폰의 전력소모량을 추정하는 실험 결과, 본 연구에서 제안하는 방법을 통해 간접적으로 추정한 전력소모량의 추정 오차가 기존 연구들에 비해 최대 절반으로 감소하였다.

주요어: 안드로이드, 스마트폰, 이벤트 반응형, 전력소모량 추정
학번: 2012-20744

감사의 글

지난 2년 동안 부족한 저를 이끌어주시고 아낌없는 가르침과 조언을 주신 지도 교수 장래혁 교수님께 깊은 감사의 말씀을 드립니다. 교수님의 지도와 격려 덕분에 석사과정 동안 많은 것을 보고 배우며 공학도로서의 기본적인 소양을 갖추 수 있었습니다. 또한 본 논문의 심사를 맡아주시고 유익한 의견을 주신 하순희, 이창건 교수님, 그리고 대학원 과정 동안 많은 조언을 해주신 조남익 교수님께도 감사의 인사를 드립니다.

그동안 연구실에서 많은 시간을 함께 보냈던 신동화, 박재현, 김영현, 박상용, 김재민, 이주연, 고영신 선배님, 그리고 지금은 다른 곳에서 자신의 길을 열심히 걷고 있는 저의 동기 김영일, 인턴 고범규의 큰 도움이 있었습니다. 특히 제 연구에 많은 조언과 도움을 주신 신동화 박사님께 지면을 빌어 다시 한번 감사의 말씀을 드립니다. 또한 연구실의 중요 업무를 맡아주신 전소연, 송명옥 씨에게도 감사의 인사 드립니다. 여러분들의 도움 덕분에 좋은 결과를 거두며 석사 과정을 잘 마무리할 수 있었습니다.

제 인생의 멘토이시며 대학원 진학과 더불어 그 외 수많은 지원을 아낌없이 해주신 삼정데이타서비스(주) 오충용 대표이사님께 머리 숙여 감사의 인사를 드립니다. 그동안 베풀어주신 은혜에 다시 한번 감사드리며 기대에 부응할 수 있도록 나날이 발전하는 모습 보여드리겠습니다. 그리고 학업에 매진할 수 있도록 많은 도움을 주신 이정엽 이사님, 그 외 모든 임직원분들께 감사드립니다.

특히 오랜 기간 회사를 떠나있는 동안 제 업무를 맡아서 잘 처리해준 임성순 씨에게 감사드립니다.

또한 사랑하는 가족들에게 감사의 마음 전합니다. 먼저 지금까지 저를 건강하게 키워주신 부모님께 진심으로 감사드립니다. 부모님께 자랑스러운 아들이 되고자 앞으로도 계속 노력할 것입니다. 나날이 발전하는 아들의 모습을 계속 지켜봐 주십시오. 오랜 세월 동안 동고동락한 외삼촌, 외숙모, 종오에게도 깊은 감사의 마음 전합니다. 그리고 부족한 저를 항상 너그럽게 이해해주시고 응원해주시는 장인어른, 장모님, 처남에게도 감사의 마음을 표합니다. 또한 제 삶의 원동력이며 힘든 고비마다 항상 함께해준 세상에 둘도 없는 사랑하는 아내 이나영에게 감사의 마음을 전합니다. 아내의 헌신적인 사랑 덕분에 이 모든 것을 이룰 수 있었습니다. 언제나 저를 믿고 따라준 아내에게 다시 한번 감사드리며, 우리 앞날에 늘 기쁘고 행복한 일만 가득하기를 기원합니다.

끝으로 저를 아끼고 도와주신 모든 분께 다시 한번 감사의 인사를 드리며 본 논문을 마치고자 합니다. 감사합니다.

2013년 12월 24일

김기태 드림