Journal of Network and Computer Applications 44 (2014) 106–119

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Calibrating parameters and formulas for process-level energy consumption profiling in smartphones

Ying-Dar Lin [a], Ekarat Rattagan [a], Yuan-Cheng Lai [c], Li-Pin Chang [a,*], Yun-Chien Yo [a], Cheng-Yuan Ho [b], Shun-Lee Chang [a]

[a] Department of Computer Science, National Chiao Tung University, No. 1001, Ta Hsueh Road, Hsinchu 300, Taiwan
[b] Advanced Research Institute, Institute for Information Industry, 1F, No. 133, Section 4, Minsheng E. Road, Taipei 105, Taiwan
[c] Department of Information Management, National Taiwan University of Science and Technology, No. 43, Section 4, Keelung Road, Taipei 106, Taiwan

ARTICLE INFO

Article history:
Received 6 October 2013
Received in revised form
20 April 2014
Accepted 29 April 2014
Available online 23 May 2014

Keywords:
Energy profiling
Energy estimation calibration
Embedded system
Android

ABSTRACT

Battery-powered mobile devices substantially constrain energy resources. Process-level energy profiling tools can identify the most energy-consuming process and detail the energy usage of hardware components. With the help of energy profiling tools, programmers can fine-tune the energy consumption of processes to extend battery lifetime. However, profiling tools are highly dependent on hardware and must be calibrated for each hardware platform. Furthermore, for any new hardware components, new energy estimation formulas must be created. To solve these two problems regarding off-the-shelf products, this work proposes a two-phase calibrating approach. The first phase reconstructs the power table with a power meter, while the second creates new energy estimation formulas using linear regression analysis. The accuracy of the calibrated tool was evaluated in five scenarios and its error ratio is proven to be below 10%, occasionally less than 5%. Hence, this proposed approach to energy consumption profiling represents a major step in off-the-shelf devices.

1. Introduction

The capability of digital electronic devices doubles every two years (reflecting Moore's law; Intel), while battery capacity has only doubled in the past decade (Parmar). Modern battery-powered mobile devices are so powerful but they suffer from limited energy budgets. For example, a smartphone, a currently popular battery-powered cellular phone, has plenty of computing-intensive applications; for example, music programs, GPS navigation, and power-hungry peripherals such as a screen and Wi-Fi module. However, energy profiling tools cannot provide programmers with accurate information of energy usage on products and devices. This causes programmers to be proficient at performance optimization, but relatively deficient at energy fine-tuning. Hence, energy profiling tools have been in demand and studied over the past several years.

There are three types of approaches for profiling a system's energy consumption: *measurement*; *simulation*; and *estimation*. Measurement-based methods directly measure energy consumption with digital power meters. This can either sample total power considering different configuration factors (for instance, processor frequency) to identify the influence of such factors on system power consumption (Assim, 2006), or probe all hardware components simultaneously to obtain detailed energy consumption data for each component (Bai and Lin, 2005; Mahesri and Vardhan, 2005; Ruan and Lai, 2006). Furthermore, mapping the measured energy consumptions to running processes helps programmers detect energy-hungry code regions (Flinn and Satyanarayanan, 1999; Chang et al., 2003; Baek et al., 2004; Xian et al., 2007). Simulation-based approaches create virtual platforms to gather statuses and utilizations of hardware resources during simulation, and then transform the gathered information to energy profiling and energy consumption. Furthermore, there are two simulation methods. One is tracking every hardware component's power state during simulation so that the energy consumption of the whole simulated platform can be calculated (Cignetti et al., 2000; Gurumurthi et al., 2002). The other is simulating power consumption of every processor during the execution of a program. This approach monitors every register bit of a function unit, such as an adder and an ALU, in resister-transfer level (RTL), and then transforms the monitored information to corresponding power consumption (Ye et al., 2000). Estimation-based approaches are

* Corresponding author.
E-mail addresses: ydlin@cs.nctu.edu.tw (Y.-D. Lin), pokekarat@gmail.com (E. Rattagan), laiyc@cs.ntust.edu.tw (Y.-C. Lai), lpchang@cs.nctu.edu.tw (L.-P. Chang), tonynctu@gmail.com (Y.-C. Yo), tommyho@iii.org.tw (C.-Y. Ho), changsl@cs.nctu.edu.tw (S.-L. Chang).

http://dx.doi.org/10.1016/j.jnca.2014.04.014
1084-8045/

similar to those of simulation except that, during the profiling time, they collect resource utilization information from kernel and log daemons. Some studies have been based on this type of approach to achieve software energy profiling (Banerjee and Agu, 2005; Kansal and Zhao, 2008; Xiao et al., 2013) and online power-saving adaptation (Do et al., 2009; Zeng et al., 2002) without the need for power meters.

The measurement-based method is the most intuitive and accurate among the three types of approaches described above. However, in the real world, vendors usually remove the reserved pins, which are required for power meters, from products because of cost and size reduction. In contrast, the simulation-based approach is impractical for developing an energy-aware simulator for every different hardware platform. Thus, we believe the estimation-based approach is a complementary solution for software energy profiling on product devices.

The concept of the estimation-based energy profiling approach is depicted in Fig. 1(a). In Fig. 1(a), there are two components: *energy estimation formulas* and a *power table*. The former estimates hardware components' energy consumptions, while the latter contains power weight coefficients of formulas. During the profiling time, the estimation-based approach collects necessary resource utilization records of energy estimation formulas from log daemons and the kernel to calculate the energy consumption of each process on hardware components. This is why the estimation-based approach could detect the process level of product devices without using power meters. For example, in Fig. 1(a), the CPU utilization, *CPU_time*, is 5 ms for the profiled process. According to the CPU energy estimation formula and power weight coefficient, *CPU_power*, the CPU energy consumed by the profiled process, is 50 nJ. However, the energy estimation formulas and power table are heavily dependent on the hardware, and thus the estimation-based approach may derive an inaccurate estimation result. Therefore, the default power table in the primitive source code of an energy estimation program must be customized for a device under test (DUT). Furthermore, in a porting procedure, faulty energy estimation formulas must be addressed because they seriously harm the estimating accuracy. For example, Fig. 1(b) shows a remarkable discrepancy between the energy estimation of Android Battery Use (BU), an energy estimation program for Android-based systems, and the energy measurement of data acquisition (DAQ; NI) of file downloading over Wi-Fi. Here, *X* and *Y* axes denote the time in minutes and the energy consumption per minute, respectively. The reason for the inaccurate estimation results of BU in this example is the faulty energy estimation formula of the Wi-Fi module. Therefore, the calibration of parameters and formulas are necessary.

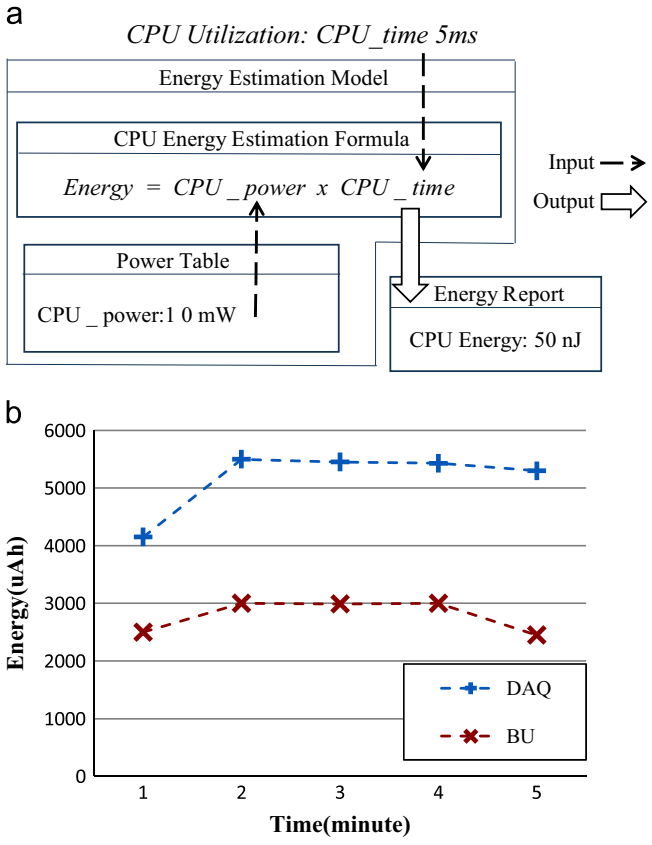

Fig. 1. Estimation-based approach: concept and faulty estimation. (a) Estimation-based energy profiling approach concept and (b) faulty energy estimation of file download.

This work introduces a two-phase calibration approach to calibrate the default power table parameters and faulty energy estimation formulas of off-the-shelf products and devices. The first phase focuses on reconstructing the power table, while the second is on creating estimation functions for the target hardware components using regression analysis. The difference between the proposed and previous approaches is that the proposed method not only improves the estimation accuracy for one process with a DUT, even for that with different DUTs, but also reduces the ill effects of DUTs. Furthermore, to achieve these improvements, the proposed method measures the total power from the battery only and is hardware independent. This two-phase calibration approach can be used for any device's energy consumption estimation. We implemented the proposed approach in a real Android device, and compared it against the built-in Battery Use (BU). We choose Android because it is based on the open-source Linux kernel along with middleware and key applications.

The rest of this paper is organized as follows. Section 2 investigates the spectrum of energy consumption studies, introduces the energy consumption behaviors of the WLAN interface, and overviews the experiment involving the Android platform and its built-in energy estimation program, BU. Section 3 states terminology and our problem statement. Section 4 describes the concept of the proposed approach, *two-phase calibration*, along with an example. Section 5 shows detailed operation procedures and system implementation of an Android product, *Android Dev Phone 1*. Section 6 presents the evaluation results, and Section 7 concludes this work.

2. Background

2.1. Spectrum of energy consumption studies

This section introduces five measurement-based and two estimation-based approaches. *Total energy measurement* (TEM), *subtractive method* (SM), *component energy measurement* (CEM), *sampling CPU occupation* (SCO), and *concurrent measurement* (CM) are measurement-based while *counting resource utilization* (CRU) and *hybrid estimation* (HE) belong to estimation-based approaches. Moreover, TEM, SM, and CEM involve hardware energy consumption, while the other four entail software energy consumption. For distinction, hardware and software energy consumptions are discussed separately. The relationship between software and hardware energy consumption is analogous to commerce in a market, as shown in Fig. 2. Device under test (DUT) and hardware components are analogous to a market and stores in the market, separately. Processes running on the DUT are likened to persons spending in the market.

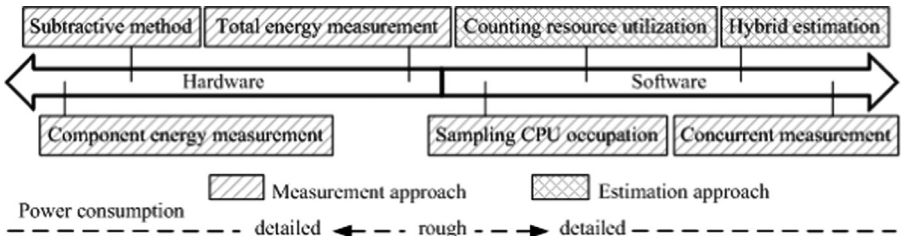

Fig. 2. Spectrum of energy consumption studies.

2.1.1. Hardware energy consumption

Three methods, TEM, SM, and CEM, are generalized based on the resolution of energy breakdown. Total energy measurement (TEM) observes energy consumption behaviors from total energy consumption similar to examining business status from total revenue of a market. On handheld computers, Assim (2006) measured the total energy consumption in different system configurations, and showed that energy consumption of a display, processor, and Wi-Fi module plays a vital role in determining battery lifetime. The SM roughly identifies the component-wide energy consumption by comparing energy consumption differences in variant hardware statuses. Component energy measurement (CEM) simultaneously probes all hardware components to sample the energy consumed by the components. This measurement can be illustrated by programming invoice machines to record incomes for each store.

Based on portable computers, Bai and Lin (2005) presented the energy measurement method for hardware components using small resisters. Mahesri and Vardhan (2005) measured energy consumption of main hardware components in IBM laptops. They concluded that the power consumed by the CPU and display dominates the power consumption of a laptop. Hardware components, such as disk drives, expend considerable energy only in the working state. Carroll and Heiser (2010) considered OpenMoko, which is an open smartphone design and whose circuit schematics are available to the research community. They studied the energy consumption of each individual hardware component and evaluated the efficacy of dynamic voltage–frequency scaling of OpenMoko.

2.1.2. Software energy consumption

According to the accuracy of energy attribution, the methods for examining software power consumption are classified into four methods: SCO; CM; CRU; and HE.

In practice, SCO samples total energy with an external power meter. To map the measured energy to the running processes in the DUT, the meter triggers the logs of a program counter (PC) and a process identifier (PID) for each energy record. During the analyzing time, the energy record can be associated with a specific process using the PC and PID. Because the energy-to-process mapping depends only on CPU utilization, inaccuracy in energy attribution could be increased by concurrent energy consumption contributed by multiple hardware components. PowerScope (Flinn and Satyanarayanan, 1999) is a popular tool for discovering energy-hungry code regions. Chang et al. (2003) replaced the time-driven sampling approach of PowerScope with an energy-driven sampling approach to improve accuracy of energy attribution. ePRO (Baek et al., 2004) integrates energy and performance profiling into a convenient tool with well-defined user interface. Shye et al. (2009) derived a system-level linear power model for Android platform smartphones. Zhang et al. (2010) also improved the accuracy of a linear power model by designing a training suite to exhaustively train all related hardware components.

Concurrent measurement (CM) embeds energy sensors on hardware components to simultaneously trace energy utilization. For accurate energy attribution, it uses a sophisticated method of synchronizing the multisource energy samples and the system events. This measurement is analogous to the budget for each shopper in a particular store. The amount a shopper spends can be calculated by summing up the price of his or her bills. M-Sync (Xian et al., 2007) measures an accurate energy attribution that raises the accuracy by up to 90% of sampling CPU utilization. Jung et al. (2012) proposed using the built-in battery voltage sensor to construct the power models of the components in a smartphone. Dong and Zhong (2011) introduced a linear power modeling method with a high sampling rate for software energy estimation.

To trace software energy consumption contributed by each hardware component, the CRU method counts resource requests for each process. The counts of resource requests can be translated into energy consumption using the energy estimation model mentioned in Section 1. Using a daily life analogy, the method works similarly to counting various types of receipts for estimating personal spending. pTop (Do et al., 2009) estimates component-wide energy consumption for each process on a laptop and provides a programming interface for designing energy-aware applications. ECOSystem (Zeng et al., 2002) counts resource utilizations and carefully allocates energy budgets among competing tasks to extend battery lifetime.

The HE method is a derivation of counting resource utilizations and a sampling CPU utilization. In practice, some of the resource utilizations can be counted easily, e.g., disk I/O, while the other resource utilization is difficult to be counted on a DUT, e.g., memory access. The energy consumed by the countable resources is easy to estimate, while the residual energy escaping from estimation is proportionally shared according to the CPU occupation. Based on the hybrid estimation method, PowerSpy (Banerjee and Agu, 2005) distinguishes the battery energy consumed by threads from that consumed by hardware components. Pathak et al. (2012, 2011) recently studied the asynchronous behavior of application power consumption, and proposed a system-call-driven finite-state-machine approach for accurate power accounting in smartphones. Snowdon (2009) proposed Koala, an OS-level power measurement/management platform, to take into account the processes' response to power state transitions in terms of performance and energy consumption. They found that memory-bound processes can gain a significant power saving with a small performance reduction.

Table 1 summarizes the comparisons of tools for software energy profiling. This work focuses on calibrating the energy estimation tools, which belong to the CRU method, and solves two hardware dependent drawbacks of tools.

2.2. Energy consumption of wireless network interface

The energy efficiency of a WLAN interface is a critical concern especially for battery-powered mobile devices. For a clean image of power consumption behaviors, Stemm and Katz (1997) measured the power consumption on four distinct interfaces.

Table 1
Comparison of software energy profiling tools.

Method	Tool	Level	Features	Drawbacks
Sampling CPU occupation (SCO)	PowerScope	Function	• Energy hotspots detection	• High overhead when high sample rate • Rough energy attribution • No component-wide energy breakdown
	ePRO	Function	• Performance and energy profiling • Fine energy debugging UI	
Concurrent measurement (CM)	M-Sync	Function	• Track of programs' usage of multiple hardware components • Accurate energy synchronization	• High overhead when high sample rate
Counting resource utilization (CRU)	pTop	Process	• Energy-sensitive process management	• Hardware dependent, e.g. power table • Hardware component dependent, e.g. estimation formulas
	Battery Use	Process	• Many hardware components consideration	
Hybrid estimation (HE)	PowerSpy	Thread	• Energy detection from battery	

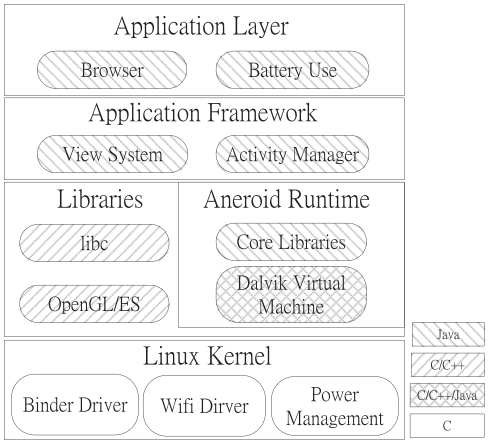

Fig. 3. Architecture of Android system.

They concluded that power consumption of receiving packets is nearly equivalent to that of an idle network interface, and sending packets consumes more power than receiving packets. Ebert et al. (2002) further studied the influence of packet size, transmission rate, and RF power level on power consumption of 802.11 standard network interfaces. In an identical RF channel, Ebert et al. showed that a higher RF power level results in more power consumption on the interface. In contrast, the packet size and transmission rate were less significant in power consumption. For mathematical analysis, Feeney and Nilsson (2001) characterized the energy consumption of an IEEE 802.11 wireless network interface operating in an *ad hoc* mode using simple linear formulas. However, the work did not include discussion of power saving mechanisms, which contain two distinct operation modes: *power saving mode* (PS) and *active mode* (AM). In PS, the interface periodically awakens from the sleeping state only for beacon packets. In contrast, in AM, the interface remains active in the idle state for handling instantaneous packet transmission. Therefore, idling in AM consumes more power than in PS. Rohl et al. (1997) simulated the power saving mechanism of an IEEE 802.11 *ad hoc* wireless network, and they also identified the figures for optimal beacon intervals for ATIM window sizes.

2.3. Android systems

At the end of 2008, Google released Android, a new software platform for smartphones. Market share of Android is expected to continue growing for the next few years (MIC, 2013).

2.3.1. Android framework

Android adopts a Linux kernel as a hardware abstraction because it provides numerous proven hardware drivers and sophisticated core operating system infrastructures, especially in the networking layers. Above the OS, lightweight libraries, such as Bionic libc optimized for embedded system, are exposed for native programming. The Dalvik virtual machine and core libraries, which provide most of the functionality for Java programming, construct the runtime environment for Android applications. The application framework contains several services to serve the user applications. Taking the advantage of the Dalvik virtual machine and a unified application framework, applications can migrate between platforms seamlessly. According to the functionality and performance requirements, the components in Android is written in different programming languages, as shown in Fig. 3.

2.3.2. Battery Use

Battery Use is an energy consumption estimation program of Android. It is based on the counting resource utilization method, and has been integrated into the Android system since, version 1.6. During system booting, the application framework starts a special service, battery info, which takes the responsibility for the counting resource utilization. For calculating energy consumption, BU raises the inter process communication (IPC) to obtain the data from the battery info service. Table 2 summarizes the default energy estimation formulas and power weight coefficients in BU.

In Table 2, the basic energies are the energy consumed to keep the hardware components active. In the basic energy estimation formulas, P_R^b is the basic power consumption of the hardware component R, and T_R^b is the time duration wherein the component is active. Because there are many brightness levels, backlight energy estimation is formulated in a summation form of multiple energy consumption instances. In the estimation formulas of working energy, P_R^w is the working power and $T_{R,p}^w$ is the duration wherein hardware component R are used by the process p.

Table 2
Default energy estimation formulas in Battery Use.

Energy item	Estimation formulas	Power weight coefficient
Wi-Fi basic	$P^b_{wifi} \times T^b_{wifi}$	P^b_{wifi}
Radio basic	$P^b_{radio} \times T^b_{radio}$	P^b_{radio}
Screen basic	$P^b_{screen} \times T^b_{screen}$	P^b_{screen}
Screen backlight	$\sum_{i=i}^{l} P^{bt}_i \times T^{bt}_i$	P^{bt}_i
CPU idle	$P^b_{cpu} \times T^b_{cpu}$	P^b_{cpu}
CPU working	$P^w_{cpu} \times T^w_{cpu,p}$	P^w_{cpu}
GPS working	$P^w_{gps} \times T^w_{gps,p}$	P^w_{gps}
Phone call	$P^w_{radio} \times T^w_{radio}$	P^w_{radio}
Networking	$E^{net}_{byte} \times \left(V^{rcv}_p + V^{snd}_p\right)$	E^{net}_{byte}

The networking energy estimation of process p is defined as a product of per byte transmission energy E^{net}_{byte} and total traffic volume including receiving packets V^{rcv}_p and sending packets V^{snd}_p.

The formulas in Table 2 are extracted from the source code of BU. Basically, these formulas assume that the power consumption are stable when components are working in the same state. However, different DUTs may require calibrating the coefficients of the formulas for accurate power consumption estimation. The energy estimation method described above remains the same until the latest Android version when we wrote this paper.[1]

3. Problem statement

3.1. Terminology

Table 3 defines the terminology of this work. The real and estimated energy consumption of a target hardware component T are E^T and $\hat{E}^T$, respectively. $\hat{E}^T$ is formulated by the energy predictor variable f_k and the power weight coefficient c_k corresponding to f_k. For linear regression analysis, $\mathbf{I}_n$ is the nth input data set containing k values (from $f_{n,1}$ to $f_{n,k}$) for k predictor variables (from f_1 to f_k). Because E^T is difficult to be measured directly from real devices, E^T_{apx}, the approximate value, is adopted in the regression analysis for creating a new estimation formula. $E_{total}(T_1, \cdots, T_m)$ and $\hat{E}_{est}(T_1, \cdots, T_m)$ denote the real and estimated total energy of hardware components from T_1 to T_m, respectively.

3.2. Problem description

Because there would be no reserved pins for power probing in off-the-shelf products, it is nearly impossible to measure the energy consumption of a hardware component in the final products. Even if we could measure the energy consumption using the development prototypes, the estimation results would be biased because of the difference between the prototypes and the final products. Therefore, we propose measuring the total system energy consumption and cross-matching the results of different test scenarios to identify the energy consumption of individual components. The rest of this section is on the statements of the problems for power formula calibrating without hardware component power measurement.

Let the energy E^T consumed by a component T be modeled as a linear equation $\hat{E}^T = c_0 + c_1 f_1 + \cdots + c_k f_k$ with k energy predictor variables and $k+1$ power weight coefficients (Do et al., 2009; Economou et al., 2006). C_0 is the constant parameter and c_k is the corresponding coefficient to energy predictor variable f_k. This work calibrates parameters and formulas for process-level energy consumption profiling by updating power weight coefficients and modifying faulty energy estimation formulas.

Let the consumed energy E^T for a component T be modeled as a linear equation $\hat{E}^T = c_0 + c_1 f_1 + \cdots + c_k f_k$ with k energy predictor variables and $k+1$ power weight coefficients (Do et al., 2009; Economou et al., 2006). c_0 is the constant parameter and c_k is the corresponding coefficient to energy predictor variable f_k. This work calibrates parameters and formulas for process-level energy consumption profiling by updating power weight coefficients and modifying faulty energy estimation formulas.

3.2.1. Problem statement 1: updating power weight coefficients

The default values in the power table may be specific to some particular DUTs, or their default values are arbitrarily set and can cause unacceptable errors in the estimation results E^T. Therefore, updating the correct value to the power weight coefficients, c_k, and replacing them to the power table are necessary for a new hardware component in the DUT to obtain a correct E^T. Hereafter, we call this problem "power table reconstruction".

3.2.2. Problem statement 2: modifying faulty energy estimation formulas

$E_{total}(T_1, \cdots, T_m)$ can be correctly measured on an off-the-shelf device, but for a hardware component T, its energy estimation formula may be wrong because of the totally different energy consumption behavior of T in the new DUT. Furthermore, it is difficult to discover the faulty energy estimation formula of a particular component T from a set of hardware components' energy estimation formulas. Therefore, identifying a faulty energy estimation formula and create a correct one for a specific hardware component T is necessary.

4. Two-phase calibration approach

4.1. Calibration approach overview

Most of the DUTs share a common set of hardware components. For example, Android smartphones have an LCD screen, a Wi-Fi module, radio hardware, and a system-on-chip (SoC) processor with two cores. The first core is for radio signal processing and the second is for user applications. Because energy consumption heavily depends on hardware design, equipping similar hardware components usually causes the DUTs to resemble energy consumption behaviors. For example, as shown in Fig. 4, the backlight power of four smartphones maintains the similar linearity between brightness levels and consumes stable energy within the brightness levels. The main difference of the four lines is the slope related to the power weight coefficient, P^{bt}_i QUOTE , of the screen energy estimation formula mentioned in Section 2. The above observation suggests that most energy estimation models can adapt to the similar DUTs by updating power weight coefficients for each estimation formula.

During the reconstruction of the power table, power consumption can be classified into two categories: *system-basic* and *process-related*. The former considers the whole system, while the latter is process-level and is consumed by the hardware components related to specific processes. In other words, the system-basic category represents the basic power consumption of system-wide hardware components, while the process-related one is defined as the extra power consumption consumed by the activity of running processes. In this work, the energy consumption from the system-basic power belongs to the entire system, while that from the process-related power is charged from the corresponding processes.

[1] ⟨http://source.android.com/devices/tech/power.html#⟩.

Table 3
Terminology definitions.

Term	Definition
$\hat{E}^T$	The energy estimation for the target hardware component T
E^T	The real energy consumed by the target hardware component T
E^T_{apx}	The approximate energy consumption of E^T
f_k	The energy predictor variable of $\hat{E}^T$
c_k	The corresponding power weight coefficient to f_k
$\mathbf{I}_n$	The n-th input data for the k energy predictor variables $f_1 \cdots f_k$
$f_{n,k}$	The variable value of f_k in input data $\mathbf{I}_n$
$E_{total}(T_1, \cdots, T_m)$	The total energy measured by the power meter for the system where hardware components $T_1 \cdots T_m$ are active or working
$\hat{E}_{est}(T_1, \cdots, T_m)$	The sum of calibrated energy estimation for $\hat{E}^{T_1} \cdots \hat{E}^{T_m}$

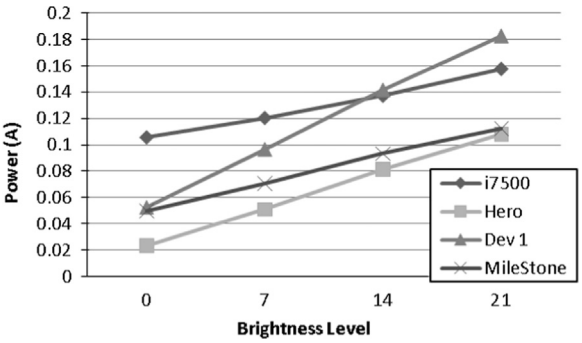

Fig. 4. Backlight energy consumption behaviors of smartphones.

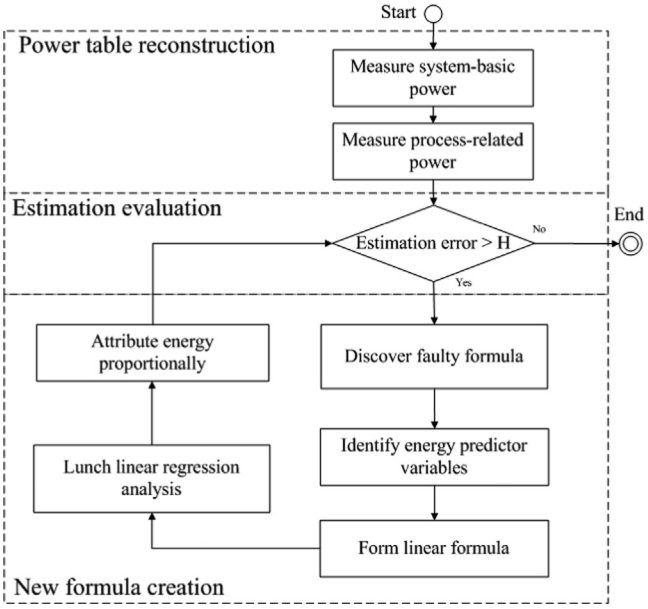

Fig. 5. Tow-phase calibrating flowchart.

Sometimes, although two DUTs have similar hardware components, the energy estimation formulas still cannot be successfully applied from a DUT to the other DUT because of the variant specification of hardware component T. Furthermore, when the DUT is a product, there are no longer reserved pins for measurement. Therefore, the real energy E^T consumed by T is unable to be directly measured. Thus, to create a new energy, estimation formula to replace the faulty or inaccurate one and obtain the energy consumption E^T, this work proposes using a linear regression analysis and an approximate energy consumption E^T_{apx}, obtained by subtracting $\hat{E}_{est}(T_1, \cdots, T_m)$ from $E_{total}(T_1, \cdots, T_m)$, to help modeling power consumption behaviors of T. Moreover, to facilitate programmers in obtaining proper data, the energy estimated using new formulas is mapped onto running processes for process-level energy profiling.

Figure 5 depicts the main flow of the two-phase calibration process, i.e., the power table reconstruction and new formula creation. The first phase, power table reconstruction, involves updating power weight coefficients (as explained in Section 3), and can be divided into two function blocks of measuring the system-basic power and process-related power. The second phase, new formula creation, involves identifying the faulty formulas, replacing the faulty formulas with new, correct ones, and proportionally distributing the estimated energy. Between the two phases, the estimation evaluation is used to feedback calibrated results. There are predefined test scenarios and a threshold (H) for the evaluation procedure. The calibrated tools are tested by estimating energy consumption of each test scenario. If one of the estimation errors is larger than the threshold H, the new formula creation procedure is performed for further calibration. Note that a stricter threshold value will cause more runs of calibration, but more accurate result will be calibrated. In this work, the error ratios are chosen as the criteria and defined in Section 6.

4.2. Power table reconstruction

4.2.1. System-basic power coefficient measurement

It is easy to obtain the system-basic power weight coefficient, such as P^b_{wifi}, by measuring the difference of total power consumption when the hardware component operates in different states. The most common case is to measure the power difference between on and off states of hardware components. In other cases, multiple power measurements are performed if the target component provides multiple operating states. For example, a screen backlight can work at different brightness levels and results in different power consumptions at the each level, as shown in Fig. 4.

4.2.2. Process-related power coefficient measurement

Because the process-related power, for example $T^w_{cpu,p}$, depends on the hardware resource utilization, a dedicated process is used to stress the hardware component. In practice, the dedicated process should be chosen or designed carefully for stressing only one hardware component during a time period. For instance, the infinite *for-loop* can be used to stress a CPU hardware component with minimized memory accesses. The process-related power can be refined by subtracting the total system-basic power from the measured total power consumption. This is why the process-related power measurement is after the system-basic power measurement.

4.3. New formula creation

4.3.1. Discovering faulty formula

From the feedback of estimation evaluation, there may be a faulty estimation formula with incorrect energy estimation results because of wrong or missing energy predictor variables. Because old and new DUTs have different energy consumption behaviors, they use different energy predictor variables. Therefore, the faulty estimation formula always causes the faulty results on T, regardless of how the power weight coefficients are modified. For example, the following is a simple method that discovers one single faulty formula on the DUT. Let there be m estimation formulas, each for a hardware component, and assume that there is only one faulty formula among m estimation formulas. In such a situation, there can be C_k^m test scenarios involving k hardware components, where k is an arbitrary variable and is smaller than m. During the evaluation in test scenarios, two false evaluation scenarios can be found and both of them involve the faulty estimation formula. With the cross-matching algorithm, the m' joint formulas of both false evaluation scenarios can be extracted. Based on the m' formulas, the next loop is continued with $C_{k'}^{m'}$ test scenarios until only one joint formula is discovered. At the end of the discovering procedure, only one joint formula is the faulty estimation formula.

4.3.2. Identify energy predictor variables

To determine the energy predictor variables for T, a series of experiments are performed to observe the remarkable effect of each candidate energy predictor variable f_k on energy consumption. In each experiment, only one energy predictor variable can be examined with different variable values. For example, we observed that, during network transmissions, if the size of transaction packet changes, the power consumption of networking components varies as well. Thus, we identified that the transaction packet size is a predictor variable of energy consumption for networking components. Another possible method to determine the energy predictor variables is to take the advantage of proven results from other papers that extensively report on the power consumption behavior of T on similar hardware platforms. However, this approach is highly difficult and requires domain knowledge of T for the energy predictor variable choice. Hence, this work uses the experimental approach.

4.3.3. Forming linear formulas

We assume that the linear property between hardware resource utilization and energy consumption of T holds (Do et al., 2009; Economou et al., 2006). Therefore, the estimation equation can be formulated in a summation form,

$$\hat{E}^T = c_0 + c_1 f_1 + \cdots + c_k f_k$$

where f_k is the new energy predictor variable and c_k is f_k's corresponding power weight coefficient.

To obtain the unknown $k+1$ coefficients, the observation input data $\mathbf{I}_n$ is collected with the instance of real energy consumption E_n^T, consumed by T. For the convenience of regression analysis, the relationship of input data set $\mathbf{F}$, the real energy consumption set $E^T = (E_1^T \quad E_2^T \quad \cdots \quad E_n^T)^T$, and the power weight coefficient set $\mathbf{C} = (c_0 \quad c_1 \quad \cdots \quad c_k)^T$ are compiled in a vector form

$$\mathbf{E}^T = \mathbf{FC},$$

where the matrix of input data set $\mathbf{F}$ is shown as

$$\mathbf{F} = \begin{pmatrix} \mathbf{I}_1 \\ \mathbf{I}_2 \\ \vdots \\ \mathbf{I}_n \end{pmatrix} = \begin{pmatrix} 1 & f_{1,1} & \cdots & f_{1,k} \\ 1 & f_{2,1} & \cdots & f_{2,1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & f_{n,1} & \cdots & f_{n,k} \end{pmatrix}.$$

4.3.4. Launching linear regression analysis

The probing pins for component energy measurement would have been removed from DUTs. Therefore, the real energy $\mathbf{E}^T$ cannot be measured directly from hardware component T with power meters. An alternative is to apply the approximate energy consumption E_{apx}^T for $\mathbf{E}^T$. Assume there are a total of $m+1$ active or working components while measuring the total energy, and the energy estimation of m components performs effectively after the first calibrating phase (Do et al., 2009; Economou et al., 2006). Therefore, the E_{apx}^T is calculated by a subtracting operation,

$$E_{apx}^T = E_{total}(T_1, \quad \ldots, \quad T_m, \quad T) - \hat{E}_{est}(T_1, \quad \ldots, \quad T_m),$$

where $\hat{E}_{est}(T_1, \cdots, T_m)$ is the estimated total energy of m hardware components and $E_{total}(T_1, \cdots, T_m, T)$ is the measured total energy of $m+1$ hardware components.

In practice, the calibrating procedure of T can only be performed after the energy consumption of other involved hardware components are estimated. For example, the CPU energy consumption is estimated before the energy consumption of the networking module is calibrated. This is because networking packages always consume the energy on the CPU for network protocol processing. Therefore, instead of $\mathbf{E}^T$,

$$E_{apx}^T = (E_{apx,1}^T \quad E_{apx,2}^T \quad \cdots \quad E_{apx,n}^T)^T$$

is adopted as the energy consumed by T for regression analysis, where $E_{apx,n}^T$ is the approximate value for E_n^T.

4.3.5. Attributing energy proportionally

he last procedure involves mapping the energy consumption to the related processes. Based on per-process information logging, each resource request on hardware component T is counted in the profiling time. During the analysis time, the estimated energy of the new created formula is proportionally and fairly shared among the corresponding processes according to the request counts.

4.4. Using two-phase calibration: an example

Assume there are three hardware components, T, T_1, T_2, on a DUT without probing pins for component energy measurement, and an energy estimation application is being calibrated for correct energy estimation on the DUT. The application contains three estimation formulas, $\hat{E}_{foult}^T$, $\hat{E}^{T_1}$, $\hat{E}^{T_2}$, and a default power table with null values. $\hat{E}^{T_1}$ estimates the system-basic energy of T_1, while $\hat{E}_{foult}^T$ and $\hat{E}^{T_1}$ predict the process-related energy of T and T_2, respectively. In the first phase of the calibration procedure, the system-basic power weight coefficient of $\hat{E}^{T_1}$ is retrieved by switching T_1 on and off. For the case of $\hat{E}_{foult}^T$ and $\hat{E}^{T_2}$, the desired process-related power weight coefficients are measured with two dedicated programs stressing T and T_2 individually.

After the first phase, the calibrated tool is evaluated in three (C_2^3) scenarios. Each scenario involves different hardware components, and the evaluation results are shown in Fig. 6(a). With the cross-matching manner, because T appears in both two failure scenarios, the $\hat{E}_{foult}^T$ is identified as a faulty energy estimation formula. The correct energy predictor variables of T are discovered from a series of experiments and form a new formula $\hat{E}_{new}^T$. In the regression analysis, the energy consumption of T is in demand and

approximated by $\hat{E}^T_{apx}$. In Fig. 6(b), the energy consumption of T_1 and T_2 are correctly estimated by $\hat{E}_{est}(T_1,T_2)$ after the first phase calibration, and the total energy $E_{total}(T_1,T_2,T_3)$ can also be measured directly from the battery or the power supply. Finally, to map the estimated energy to processes, if the energy estimated by $\hat{E}^T_{new}$ is e^r and the usage count of hardware component T is 3 and 4 time units within a time interval for the processes P_1 and P_2, respectively, then the energy consumed by P_1 on T is estimated as $e^T 3/(3+4)$.

5. Implementation on Android Dev Phone 1

In this work, the Android Dev Phone 1 (Dev 1) is used as the DUT. The goal of this section is to obtain the correct energy estimation of Battery Use on Dev 1 by executing the two-phase calibration procedures.

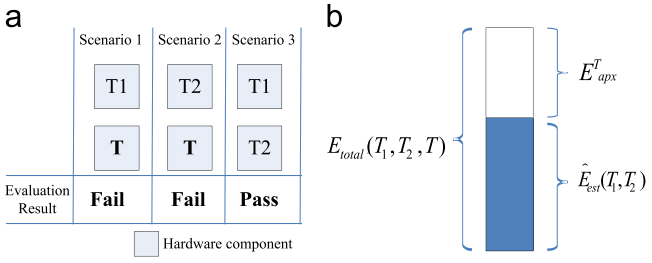

Fig. 6. Example run of two-phase calibration: (a) evaluation results under three scenarios and (b) approximate energy consumption of T.

Table 4
Power table values.

Category	Energy item	Power weight coefficient	Default (mA)	Reconstructed (mA)
System-basic	Wi-Fi basic	P^b_{wifi}	0.1	22.68
	Radio basic	P^b_{radio}	0.1	3.27
	Screen backlight	P^{bt}_{max}	0.1	134.87
	Screen basic	P^b_{screen}	0.1	38.48
	CPU idle	P^b_{cpu}	0.1	1.45
	Phone call	P^w_{radio}	1	204.94
	GPRS.basic	P^w_{gprs}	0.1	103.32
Process-related	CPU working	P^w_{cpu}	0.2	84.08
	GPS working	P^w_{gps}	1	104.99
	Networking	E^{net}_{byte}	0.1	N/A

5.1. Power table reconstruction on Dev 1

In this work, the power consumption of eight hardware components is measured and further classified into two categories: system-basic and process-related. The measurement procedures for each hardware component are summarized as follows.

5.1.1. System-basic category

- Wi-Fi basic: the Wi-Fi basic power is easy to obtain by switching the Wi-Fi module on and off with the Android *Setting* application.
- Radio basic: the *Airplane mode* closes every wireless interface including the phone radio. It is utilized to switch the radio between on and off states.
- Screen backlight: in the Android Linux, the *sysfs* file system (Mochel, 2005) is able to set values into the kernel variables and is used to configure the screen brightness. Because of the linearity of backlight power, as shown in Fig. 4, the power difference between the maximal and minimal screen brightness is measured and denoted as P^{bt}_{max}. Power consumption of other brightness levels is calculated using linear interpolation.
- Screen basic: the power consumed by an LCD panel is obtained by measuring the power difference between the on and off states of the screen. In practice, there are two tricks of the screen basic power measurement. First, the screen backlight is turned off to distinguish the screen basic power from the screen backlight power. Second, to prevent the system from entering suspend mode when the screen is closed, it should make use of a power management feature of Android system such as *wakelock* (Google).
- CPU idle: is defined as the power consumed in Battery Use while the screen is closed. With the power button on the Dev 1, turning off the screen is easy.
- Phone call: is the power of making a call. It can be measured by the average power during the call excluding other basic power.

5.1.2. Process-related category

- GPS working: To obtain the working power of the GPS device, we created a simple Android application, called GPStest, to activate the GPS without processing the GPS data.
- CPU working: CPU_busy contains the infinite for-loop and is used to stress the CPU hardware resource.

In this study, both the CPU and the hardware components that get activated synchronously with the CPU, such as memory, contribute to the working power consumption of the CPU. The idle power consumption of the CPU is defined accordingly. Table 4

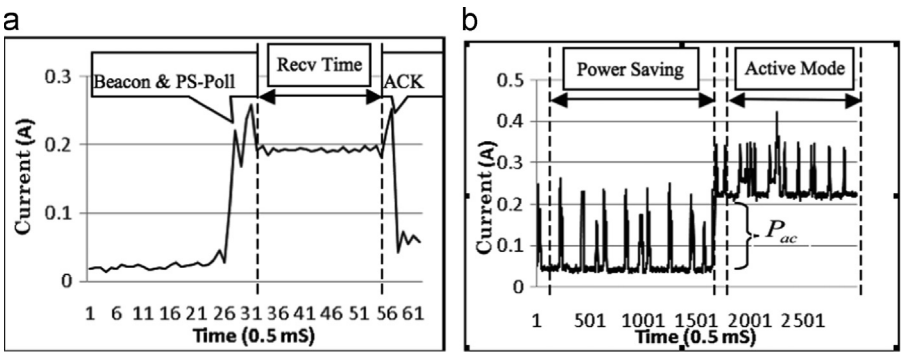

Fig. 7. Power consumption behaviors of the Wi-Fi interface: (a) power consumption while receiving a packet in PS and (b) power consumption in PS and AM.

shows power tables with categories, energy items, power weight coefficients, default value, and reconstructed value. The default and reconstructed values are respectively the raw data from the Android source code and rebuilt using the proposed approach. Most energy consuming hardware is the radio component operating during phone calls. Moreover, the screen backlight with the maximal brightness level and working GPS consume more power than does the busy CPU. In new Android versions, the default values are left to vendors for their own definitions.

5.2. Wi-Fi formulas creation on Dev 1

From the evaluation results, the estimation errors of test scenarios with Wi-Fi networking always exceed the predefined error ratio threshold, 10%. As a result, the original Wi-Fi estimation formula is recognized as the faulty formula. Accordingly, our method will create a new estimation formula for the Wi-Fi module on Dev 1.

From observations on Dev 1, the energy consumed by the Wi-Fi module correlates closely with operation modes and packet transmission time, T_{tx}. T_{tx} stands for receiving time and sending time. In Fig. 7, the power consumption is captured from the power line of battery by DAQ when the Dev 1 receives packets. For every second, DAQ records 2000 power samples and its resolution of current probe is ± 1 mA. The length of power pulse, in Fig. 7(a), closely depends on the time receiving a packet. Figure 7(b) depicts dramatic difference in power consumption between power saving mode (PS) and active mode (AM) at the same data rate. The

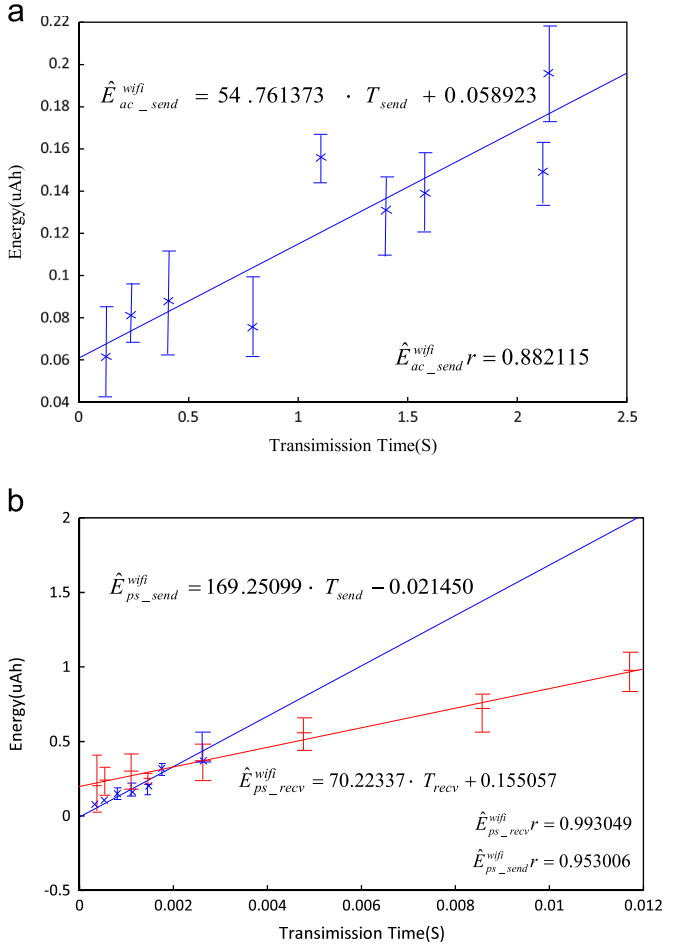

Fig. 8. Linear regression of Wi-Fi module in operation modes: (a) regression of receiving and sending in PS and (b) regression of sending packet in AM.

Table 5
Linear regression results summarization.

Mode		Energy estimation function (uA h)
PS	recv	$\hat{E}^{wifi}_{ps\_recv} = 70.22337 \times T_{recv} + 0.155057$
	send	$\hat{E}^{wifi}_{ps\_send} = 169.25099 \times T_{send} - 0.021450$
AM	recv	$\hat{E}^{wifi}_{ac} = P_{ac} \times T_{ac}$
	send	$\hat{E}^{wifi}_{ac} = P_{ac} \times T_{ac}$
		$\hat{E}^{wifi}_{ac\_send} = 54.761373 \times T_{send} + 0.058923$

$\hat{E}^{wifi}_{ps\_send}/\hat{E}^{wifi}_{ps\_recv}$: the sending/receiving energy estimation in PS.
T_{recv}/T_{send}: the sending/receiving time of a packet.
$\hat{E}^{wifi}_{ac}$: the estimated energy consumed by P_{ac}.
T_{ac}: the time duration while Wi-Fi module works in AM.
$\hat{E}^{wifi}_{ac\_semd}$: the sending energy estimation in AM.

obviously difference is that, in AM, the Wi-Fi module consumes an active energy from active power (P_{ac}) inherently.

In the experiment, T_{tx} is adopted as the energy predictor for the Wi-Fi module. The general regression equation of the module is formulated as

$$\hat{E}^{wift} = c_0 + c_1 \times T_{tx}.$$

To produce different lengths of packet transmission time, user datagram protocol (UDP) packets in different sizes are transmitted with several data rates; for example, 54 Mbps and 11 Mbps. The linear regression analysis is launched for sending and receiving packets in the two operation modes, individually. The results are shown in Fig. 8 and summarized in Table 5. In Fig. 8, the high correlation coefficient (r) of each line proves the liner property between transmission time and energy consumption of a Wi-Fi module.

In AM, because the receiving power nearly equals P_{ac}, receiving the packet consumes no additional energy on the Wi-Fi module, excluding active energy. Therefore, the receiving energy of AM is omitted from the regression analysis and estimated simply with Wi-Fi active energy $\hat{E}^{wifi}_{ac}$, shown in Table 5. The value of P_{ac} is retrieved as 165.81 mA by measuring the power difference between the two modes.

In Fig. 8(a), because the sending slope is two times steeper than the receiving one, it implies that sending one byte consumes more energy than receiving two bytes in PS. Moreover, additional energy for listening to the beacons and sending the polling control packet (Ebert et al., 2002) (PS-Poll) causes the constant coefficient of the receiving line to be larger than that of the sending line. Moreover, from sending lines in Fig. 8(a) and (b), we can observe that sending packets consumes less energy in AM than in PS.

With per-process traffic logging, the estimated transmission energy (for example, $\hat{E}^{wifi}_{ps\_send}$) easily relates to the corresponding processes when transmitting the packets. Because the policy of switching into AM is related to packet count, the estimated active energy $\hat{E}^{wifi}_{ac}$ is shared propositionally according to the sending and receiving packet counts of processes.

5.3. Wi-Fi power daemon implementation on Dev 1

Regarding logging networking traffic, the socket layer of the Android Linux kernel is modified slightly. In the layer, a list of recorders counting traffic volume for each process is created. Note that some networking traffic does not consume the resources of the Wi-Fi module because of inter process communication (IPC). Therefore, to exclude the IPC traffic, from statistics, the packets with the local address such as 127.0.0.1 are filtered by a black list. The translation between the traffic volume and desired

transmission time is achieved through the data rate information retrieved directly from the Wi-Fi driver.

In the Wi-Fi driver, the policy of switching the operation modes involves the count of the packets transmitting through the Wi-Fi module. If the packet count value is greater than 15 per second, the Wi-Fi module automatically switches into AM. By contrast, if the packet count value is smaller than eight per second, the module enters PS from AM. Therefore, to predict the operation modes for the Wi-Fi module, the packet count information should be captured from the data link layer.

In the implementation, as shown in Fig. 9, the *proc* (Mouw) files are created for shipping the information from kernel space to user space. Furthermore, to reduce the performance overhead of the Java virtual machine, the Wi-Fi power daemon is created in C language. The Wi-Fi power daemon calculates the energy consumption of the Wi-Fi module for each process and logs the results into a file once per second. To integrate the logs of Battery Use and the Wi-Fi power daemon, a log parser is created to combine these two logs with the timestamps labeled on each energy record.

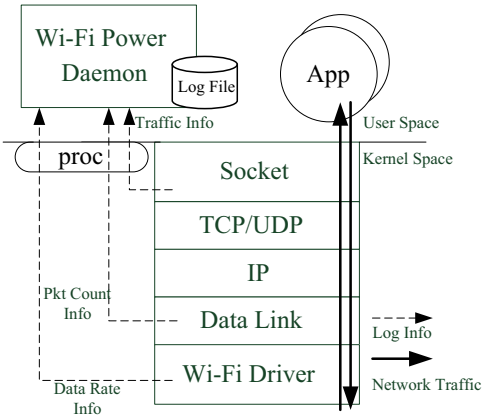

Fig. 9. Wi-Fi power daemon implementation.

5.4. GPRS power formulas

Because Battery Use does not include the power model of the GPRS interface, we built a new formula for it. We considered the power consumption of the downloading and uploading activities using the GPRS interface. The power formulas for the GPRS inherits the form of the Wi-Fi power formulas, as follows:

$$\hat{E}_{ac}^{gprs} = c_0 + c_1 \times T_{ac},$$

where *ac* is either *dw* (downloading) or *up* (uploading), T_{ac} is an time duration of the downloading or uploading activity, c_0 is the basic energy (in uA h), and c_1 is the power weight coefficient. In our experiment, we downloaded a 100 kB file and uploaded the same file via GPRS using the AndFTP client, and measured the power consumption of the GPRS interface. Figure 10(a) and (b) shows the power consumption behaviors of uploading and downloading the file. Notice that the GPRS active periods in Fig. 10(a) and (b) are [10.8 s, 41.5 s] and [20.9 s, 68.6 s], respectively. We used the same method described in Section 5.2 to compute the power weight coefficients, and the power formulas are shown in Table 6.

6. Evaluation studies

In this section, an evaluation framework is designed to verify the correctness of the energy estimation results. The energy consumption of five scenarios, described in Sections 6.2 and 6.3, are profiled at the process level for case studies. We must

Table 6
GPRS formulas.

Activity	Energy estimation function (uA h)
Downloading file	$\hat{E}_{dw}^{gprs} = 64.63 \times T_{dw} + 12.03$
Uploading file	$\hat{E}_{up}^{gprs} = 105.94 \times T_{up} + 0.96$

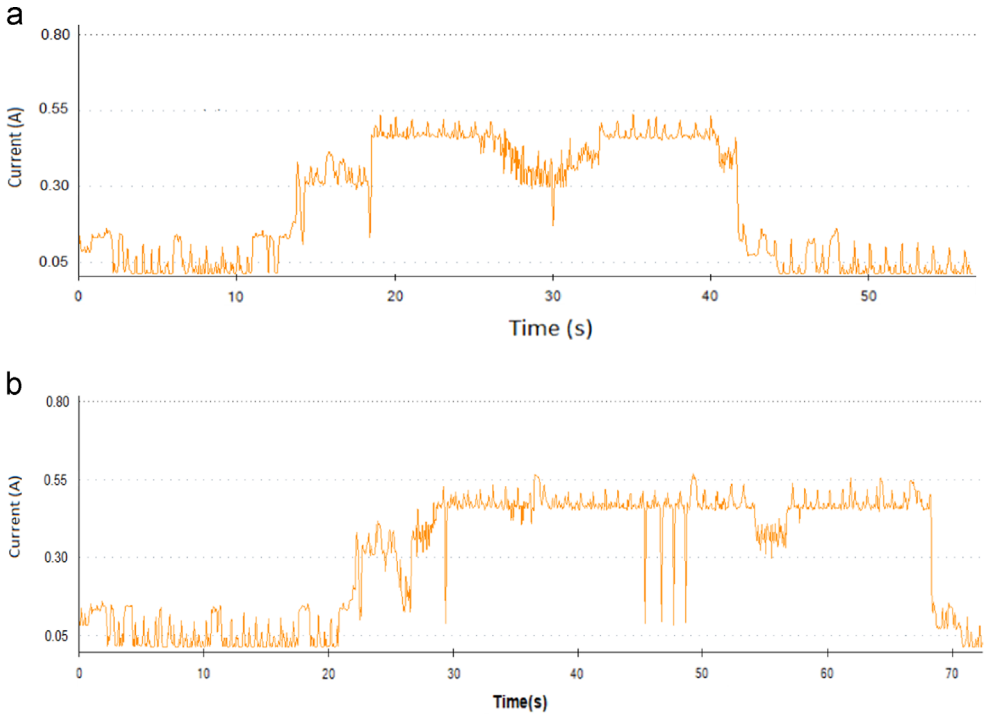

Fig. 10. Power consumption behaviors of uploading/downloading a 100 kB file via GPRS: (a) GPRS power consumption behavior of downloading an 100 kB file and (b) GPRS power consumption behavior of uploading a 100 kB file.

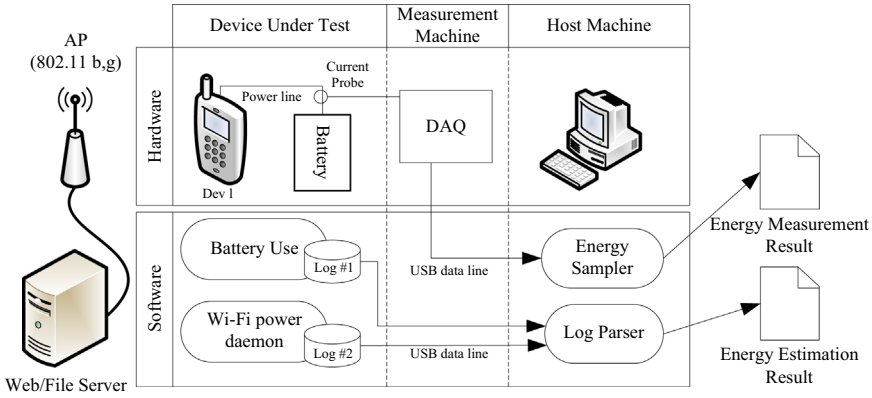

Fig. 11. Power consumption evaluation framework.

Table 7
Error ratio comparison between full-system and two-phase under five scenarios.

		Without networking		With networking		
		System idle	CPU intensive	Web browsing	FTP download	FTP upload
Full-system	Mean (%)	24.60	9.04	25.80	6.52	34.78
	Standard deviation	2.31	2.28	1.78	1.49	0.46
Two-phase	Mean (%)	4.79	7.39	9.16	2.82	4.74
	Standard deviation	2.47	1.10	1.84	1.07	0.72

emphasize that the data used by the calibration process shown in prior sections are different from that in the following experiments.

6.1. Evaluation framework

In the experiments shown in Fig. 11, Dev 1 is configured at the minimal brightness level and the SIM card is removed. Furthermore, to experiment with a clean networking environment, the Wi-Fi access point (AP) and Web/FTP server are within a local network. Dev 1 is configured to transmit traffic data through the Wi-Fi module only. We evaluated the proposed method described in Sections 6.1–6.3, for system-level and process-level energy estimation. We choose the full-system modeling solution presented by Economou et al. (2006) to measure the system-level energy consumption and a DAQ (NI) to measure the process-level energy consumption. The estimated results are compared against the measured results. Notice that the full-system modeling solution cannot provide the energy consumption information of processes and hardware components, while DAQ is used to measure the hardware component energy consumption with one single hardware component.

We implemented the full-system solution, which performs linear regression method using the total power. As a result, the total system energy can be estimated by

$$P_{dev1} = 0.067 + (8.67 \times 10^{-4}) \times u_{cpu} + (1.28 \times 10^{-7}) \times u_{mem} + (2.15 \times 10^{-5}) \times u_{disk} + (3.56 \times 10^{-4}) \times u_{net}$$

where u_{cpu} is CPU utilization in percentage, u_{mem} is the sum of data and instruction cache miss count in kilo-time per second, u_{disk} is the number of read and write sectors, and u_{net} is the sum of send and receive traffic volume in kilo-bytes per second. In contrast, DAQ measures the energy consumption from the power line between the battery and Dev 1, as shown in Fig. 11.

The evaluation details are as follows: On the host machine, the energy sampler logs two thousands energy records from the DAQ as the energy measurement results every second. In contrast, the energy estimation results of the two-phase calibration approach are produced by the energy estimation daemons, which are BU and the Wi-Fi power daemon running on Dev 1. During the profiling time, the energy sampler and energy estimation daemons generate the measurement and estimation logs concurrently. During the analysis time, the log parser processes the energy estimation logs extracted from Dev 1 for the energy estimation results. The accuracy of the energy estimation result is evaluated by the error ratio (*err*), which is defined as

$$err = |est - mea|/mea \times 100\%,$$

where *est* is the energy estimation result, and *mea* is the energy measurement result. The smaller *err* is, the more accurate *est* will be.

We conducted the full-system modeling solution and our two-phase calibration approach using the five scenarios. Table 7 summarizes the average error ratios and the standard deviations of the total system energy. In the five scenarios, the estimation accuracy of two-phase results is guaranteed and all error ratios are below 10%, although different estimation errors of hardware energy estimation formulas exist. Conversely, the full-system modeling solution produced accurate estimations only in CPU intensive and FTP download scenarios. This may be because the four predictor variables of the full-system estimation formula cannot accurately model the energy consumption behaviors of the embedded devices that have plenty of hardware components.

6.2. Evaluation scenarios without networking

6.2.1. System idle scenario

In this scenario, Dev 1 is put idle and measured for four minutes. Figure 12(a) shows that the energy estimation accuracy of the two-phase result is high and the average error ratio is about 4.79%. Figure 12(b) depicts the energy decomposition in process level in the scenario. From this figure, 97% of the total energy consumption of Dev 1 contributed by the LCD display. Notice that the display energy includes the energy of screen backlight and LCD panel. The other 3% are shared among the idle process, the

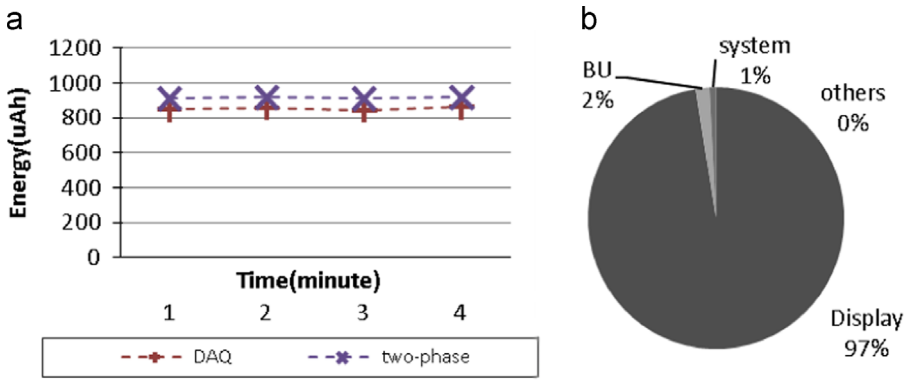

Fig. 12. Energy consumption under system idle scenario: (a) estimation evaluation and (b) energy profiling.

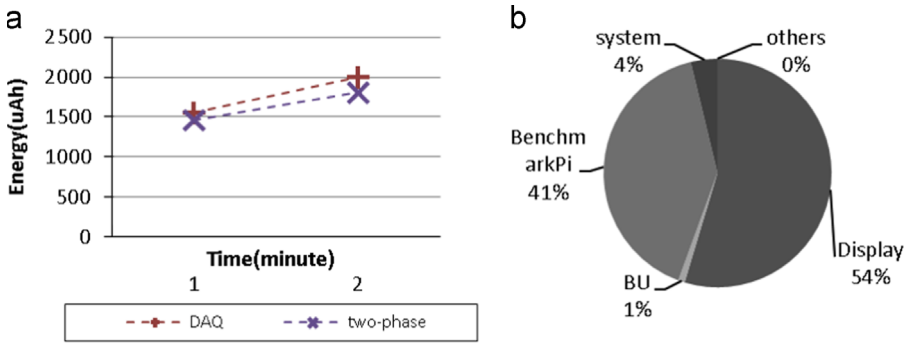

Fig. 13. Energy consumption under CPU intensity scenario: (a) estimation evaluation and (b) energy profiling.

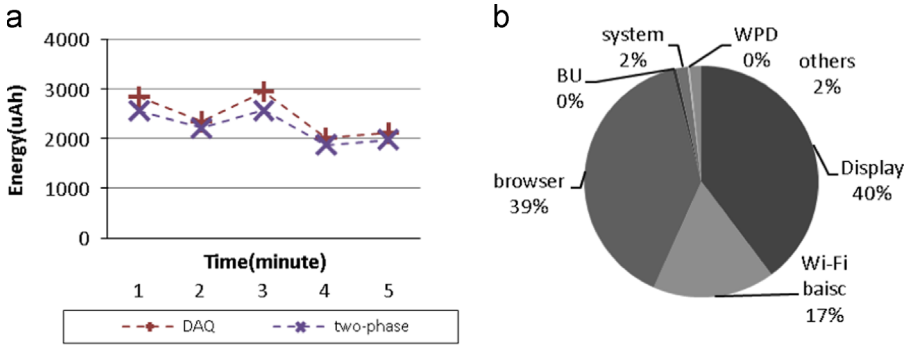

Fig. 14. Energy consumption under web browsing scenario: (a) estimation evaluation in web browsing and (b) energy profiling in web browsing.

profiling tool, and Battery Use (BU). Figure 12(b) indicates that the overhead of BU is extremely small, only 2%.

6.2.2. *CPU intensive scenario*

BenchmarkPi calculates the approximate value of Pi so it is a CPU intensive application. In this experiment, the BenchmarkPi is once in the first minute and twice in the second minute. The error ratio of the proposed energy estimation is approximately 7.39%. The process BencharmkPi uses considerable energy (41%) of Dev 1 and the display uses 54% of energy consumption, as shown in Fig. 13(a) and (b).

6.3. *Evaluation scenarios with networking*

6.3.1. *Web browsing scenario*

We refer to the Wi-Fi power daemon as WPD in this section. For every forty seconds, the Android browser connects to a web server and retrieve web pages.

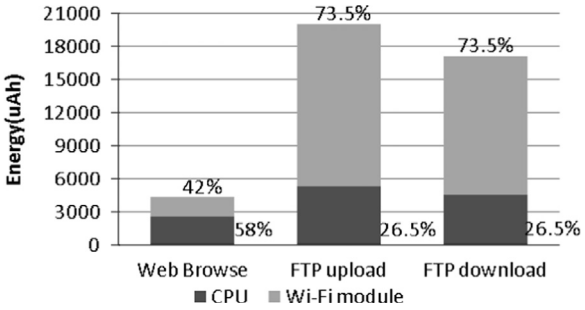

Fig. 15. CPU and Wi-Fi module energy consumption of browser process and AndFTP process.

The experiment results within a five-minute period are shown in Fig. 14. In Fig. 14(a), the estimation results closely match the measured results, and the average error ratio is 9.16%. In Fig. 14(b), the energy profiling tools consume negligible energy as BU and

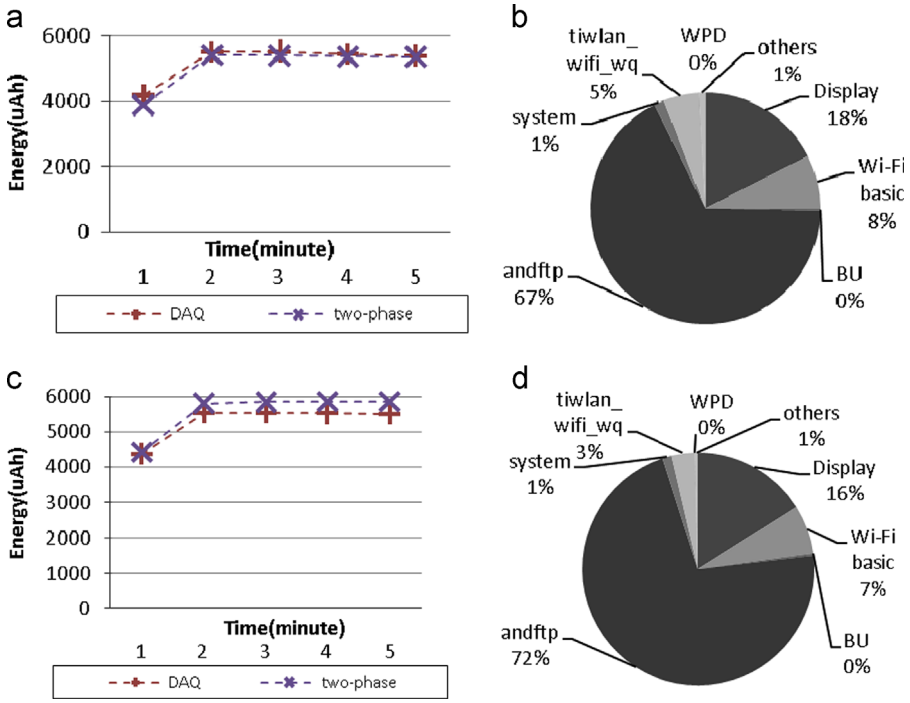

Fig. 16. Energy consumption under file transmission scenarios: (a) estimation evaluation in FTP download; (b) energy profiling in FTP download; (c) estimation evaluation in FTP upload and (d) energy profiling in FTP upload.

WPD are energy-efficient. The energy consumed by the browser process is almost identical to that of the display energy. Furthermore, keeping the Wi-Fi module active uses 17% of the total energy. Because the test web pages contain *JavaScript*, which heavily stresses the CPU, the browser process consumes more energy on CPU (58%) than on the Wi-Fi module (42%) in the scenarios, as shown in Fig. 15.

6.3.2. File transmission scenarios

In this experiment, a 90 MB file is transmitted between Dev 1 and a file server using an FTP client, AndFTP. The FTP application requires more energy for uploading a file than downloading it, because sending packets consumes more power than receiving packets. This can be observed from Fig. 16(a) and (c), which clearly shows that the accuracy of the two-phase results is high, and the average error ratios are below 5%. From Fig. 16(b) and (d), the AndFTP process consumes 67% and 70% energy of Dev 1 during downloading and uploading, respectively. Different from the browser process in the prior experiment,

The AndFTP process consumes more energy on Wi-Fi networking than on CPU, as shown in Fig. 15.

7. Conclusion and future work

This work proposes a two-phase approach to calibrate the faulty energy estimation results at the process level for off-the-shelf devices. The first phase reconstructs the power table for a DUT and the second phase further replaces the faulty energy estimation formulas with correct ones based on linear regression analysis.

Accurate energy consumption estimation with a set of co-existing hardware components is difficult because there are many predictor variables in the estimation formulas for the co-existing hardware components. We propose testing one hardware component at a time, and systematically rebuild the estimation formula for each component. We performed five case studies, and the proposed two-phase method produced more accurate energy consumption estimations than a full-system modeling approach. Furthermore, the average error ratios in energy consumption estimation of the proposed approach are proven below 10%, and in the file transmission scenarios the average error ratios are less than 5%. The proposed approach only imposes limited overheads on the DUT. For example, the Wi-Fi power daemon and Battery Use are proven contributing less than 3% of the total energy consumption.

The current study does not include the energy profiling of video-based applications because of the absence of hardware video decoders. We shall further extend this work to hardware video decoders and video-based applications like YouTube. The problem for discovering multiple faulty formulas should also be addressed in future work. Recent study identified that the power consumption of AMOLED screens is a quadratic function to the LCD brightness (Mittal and Kansal, 2012; Xu et al., 2013). However, we believe that the core idea of this work is not subject to any specific power model and regression method. For example, the proposed two-phase calibration process can include both the linear model and the quadratic model and pick up the model of the minimal error during the calibration process.

References

Assim S. Power consumption in handheld computers. In: Proceedings of the IEEE Asia pacific conference on circuits and systems; 2006. p. 1721–4.

Baek W, Kim Y, Kim J. ePRO: a tool for energy and performance profiler for embedded applications. In: Proceedings of the international SoC design conference. Seoul (Korea); 2004. p. 372–5.

Bai Y, Lin Y. Measurement and improvement of power consumption for portable computers. In: Proceedings of the IEEE international symposium on consumer electronics; 2005. p. 122–7.

Banerjee KS, Agu E. PowerSpy: fine-grained software energy profiling for mobile devices. In: Proceedings of the IEEE WirelessCom; 2005. p. 1136–41.

Carroll A, Heiser J. An analysis of power consumption in a smartphone. In: Proceedings of the 2010 USENIX annual technical conference. Boston (MA, USA); June 2010.

Chang F, Farkas K, Ranganathan P. Energy-driven statistical sampling: detecting software hotspots. Lect Notes Comput Sci 2003;2325:110–29.

Cignetti TL, Komarov K, Ellis CS. Energy estimation tools for the palm. In: Proceedings of the 3rd ACM international workshop on modeling, analysis and simulation of wireless and mobile systems. Boston (Massachusetts, United States); 2000. p. 96–103.

Do T, Rawshdeh S, Shi W. pTop: a process-level power profiling tool. In: Proceedings of the workshop on power aware computing and systems; 2009.

Dong M, Zhong L. 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 (MobiSys); 2011.

Ebert J, Burns B, Wolisz A. A trace-based approach for determining the energy consumption of a WLAN network interface. In: Proceedings of the European wireless; 2002. p. 230–6.

Economou D, Rivoire S, Kozyrakis C, Ranganathan P. Full-system power analysis and modeling for server environments. In: Proceedings of the workshop on modeling, benchmarking, and simulation; 2006.

Feeney LM, Nilsson M. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In: Proceedings of the INFOCOM 2001, twentieth annual joint conference of the IEEE Computer and Communications Societies: vol. 3; 2001. p. 1548–57.

Flinn J, Satyanarayanan M. PowerScope: a tool for profiling the energy usage of mobile applications. In: Proceedings of the second IEEE workshop on mobile computer systems and applications; 1999. p. 2–10.

Google. Power management [Online]. Available: ⟨http://pdk.android.com/online-pdk/guide/power_management.html⟩.

Gurumurthi S, Sivasubramaniam A, Irwin MJ, Vijaykrishnan N, Kandemir M. Using complete machine simulation for software power estimation: the SoftWatt approach. In: Proceedings of the eighth international symposium on high-performance computer architecture; 2002. p. 141–50.

Intel. Moore's law: made real by Intel® innovation [Online]. Available: ⟨http://www.intel.com/technology/mooreslaw/⟩.

Jung Wonwoo, Kang Chulkoo, Yoon Chanmin, Kim Donwon, Cha Hojung. 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 (CODES+ISSS'12); 2012.

Kansal A, Zhao F. Fine-grained energy profiling for power-aware application design. SIGMETRICS Perform Eval Rev 2008;36:26–31.

MIC. Global Android smartphone shipment volume forecast to reach 31.80 million units in 2013 [Online]. Available: ⟨http://mic.iii.org.tw/english/press/en_5_press_room_1_1.asp?selyear5=&doc_sqno=7494⟩.

Mahesri A, Vardhan V. Power consumption breakdown on a modern laptop. Lect Notes Comput Sci 2005;3471:165–80.

Mittal Radhika, Kansal Aman, Chandra Ranveer. Empowering developers to estimate app energy consumption. In: Proceedings of the 18th annual international conference on Mobile computing and networking (Mobicom). 2012.

Mochel P. The Sysfs file-system. In: Proceedings of the 2005 Linux symposium; 2005. p. 313–26.

Mouw E. Linux kernel procfs guide [Online]. Available: ⟨http://kernelnewbies.org/Documents/Kernel-Docbooks?action=AttachFile&do=get&target=procfs-guide_2.6.29.pdf⟩.

NI. Data acquisition (DAQ) [Online]. Available: ⟨http://www.ni.com/dataacquisition/⟩.

Parmar N. Out of juice: the tyranny of the battery [Online]. Available: ⟨http://www.smartmoney.com/spending/technology/Out-of-Juice-The-Tyranny-of-the-Battery-20821/⟩.

Pathak Abhinav, Hu Y Charlie, Zhang Ming, Bahl Paramvir, Wang Yi-Min. Fine-grained power modeling for smartphones using system call tracing. In: Proceedings of the sixth conference on computer systems (EuroSys'11); 2011.

Pathak Abhinav, Hu Y Charlie, Zhang Ming. 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 (EuroSys'12); 2012.

Rohl C, Woesner H, Wolisz A. A short look on power saving mechanisms in the wireless LAN standard draft IEEE 802.11. In: Proceedings of IEEE 802.11, the 6th WINLAB workshop on third generation wireless systems. New Brunswick (NJ); 1997. p. 183–8.

Ruan S, Lai Y. Development and analysis of power behavior for embedded system laboratory. In: Proceedings of the ACM workshop on embedded system education; October 2006. p. 45–50.

Snowdon David C, Le Sueur Etienne, Petters Stefan M, Heiser Gernot. Koala: a platform for OS-level power management. In: Proceedings of the 4th ACM European conference on computer systems (EuroSys'09); 2009.

Shye Alex, Scholbrock Benjamin, Memik Gokhan. 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 (MICRO 42). New York (NY, USA): ACM; 2009.

Stemm M, Katz R. Measuring and reducing energy consumption of network interfaces in hand-held devices. IEICE Trans Commun 1997;80:1125–31.

Xian C, Cai L, Lu Y. Power measurement of software programs on computers with multiple I/O components. IEEE Trans Instrum Meas 2007;56:2079–86.

Xiao Peng, Hu Zhigang, Liu Dongbo, Yan Guofeng, Qu Xilong. Virtual machine power measuring technique with bounded error in cloud environments. J Netw Comput Appl 2013;36(2):818–28.

Xu F, Liu Y, Li Q, Zhang Y. V-edge: fast self-constructive power modeling of smartphones based on battery voltage dynamics. In: Proceedings of the 10th USENIX symposium on networked systems design and implementation; 2013.

Ye W, Vijaykrishnan N, Kandemir M, Irwin MJ. The design and use of simplepower: a cycle-accurate energy estimation tool. In: Proceedings of the design automation conference; June 2000. p. 340–5.

Zeng CSEH, Lebeck AR, Vahdat A. ECOSystem: managing energy as a first class operating system resource. In: Presented at the SIGPLAN Not.; 2002.

Zhang Lide, Tiwana Birjodh, Qian Zhiyun, Wang Zhaoguang, Dick Robert P, Mao, Zhuoqing Morley, et al. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In: Proceedings of the eighth IEEE/ACM/IFIP international conference on hardware/software codesign and system synthesis (CODES/ISSS'10); 2010.