V-edge: Fast Self-constructive Power Modeling of Smartphones Based on Battery Voltage Dynamics

Fengyuan Xu* Yunxin Liu† Qun Li* Yongguang Zhang†
*College of William and Mary** *Microsoft Research Asia*†

Abstract

System power models are important for power management and optimization on smartphones. However, existing approaches for power modeling have several limitations. Some require external power meters, which is not convenient for people to use. Other approaches either rely on the battery current sensing capability, which is not available on many smartphones, or take a long time to generate the power model. To overcome these limitations, we propose a new way of generating power models from battery voltage dynamics, called V-edge. V-edge is self-constructive and does not require current-sensing. Most importantly, it is fast in model building. Our implementation supports both component level power models and per-application energy accounting. Evaluation results using various benchmarks and applications show that the V-edge approach achieves high power modeling accuracy, and is two orders of magnitude faster than existing self-modeling approaches requiring no current-sensing.

1 Introduction

Energy consumption is a paramount concern in all battery-powered mobile devices, including smartphones. Power modeling is a key technology and an effective way to understand the power consumption of applications running on mobile devices. Thus, it has attracted much research effort. With a power model, users can identify the power-hungry applications and better manage battery life of their smartphone [1]. Developers are able to profile, and consequently optimize, the energy consumption of their mobile applications [2].

Existing approaches for power modeling have several limitations. First, an accurate power model heavily depends on individual smartphone's hardware/software configuration, battery age, and device usage [3]. Most existing work [4, 5, 6, 7, 8, 9] relies on external power measurement equipment to generate accurate models. This is labor-intensive and requires experts' knowledge. Since the power model of individual smartphone is different and slowly changing [3, 10], it is expensive to apply this approach to build models tailored to every phone. The "self-metering" approach [3, 11, 12] has been proposed to build individualized power models if a smartphone can read the online voltage and current values from its built-in battery interface. While most smartphones have voltage-sensing capabilities, many smartphones today, including popular models like Nexus S and some Samsung Galaxy series, do not have the ability to sense current. Therefore, the previous approach based on current sensing is not applicable to many smartphones. The State-of-Discharge (SOD) approach [13] sidesteps this problem by using the SOD information in battery interface. It does not require current-sensing but has very long model generation time (days) due to the very slow changing nature of SOD. This makes it impossible to have fast power model construction, which is often required to rebuild power models to adapt to various changes in hardware and software, the battery aging, and usage pattern changes (Section 3).

In this paper we propose a new approach for power modeling, called V-edge, to address the limitations of existing approaches. V-edge is self-constructive, does not require current-sensing, and most importantly, is fast in model building. V-edge is based on the following insight to voltage dynamics on battery-powered devices: when the discharge current of a battery is changed, the instant voltage change, caused by the internal resistance, has a reliable linear relationship with the current change. Therefore, from the voltage change, we can determine the change of current and consequently the power information (see more details in Section 4). The V-edge power modeling requires only voltage-sensing and thus works for most smartphones, and is able to generate power models much faster than SOD-based approaches.

We have designed and implemented a power modeling prototype based on V-edge. Our implementation supports both component-level power models and per-application energy accounting. Experimental evaluation results, using various benchmarks and real applications, show that V-edge is able to generate accurate power models, comparable to the power-meter-based approach. The building time is much shorter than SOD-based approaches.

To the best of our knowledge, V-edge is the first work to model smartphone power consumption by leveraging the regularity of instant battery voltage dynamics. Prior to our exploration, these instant dynamics are treated as irregular fluctuations during slow supply voltage dropping (i.e. SOD decreasing) [13]. Our key contributions are as follows.

- We are the first to observe that the current information can be inferred from instantaneous changes in a battery's voltage. We demonstrate that inferring current in such a manner is fast, reliable, and accurate.
- Based on this observation, we propose V-edge to facilitate the self-constructive power modeling on most smartphones. V-edge is much faster than the existing solution, making it efficient to (re)build power models for timely adapting of hardware and software configurations with minimum interruption to users.
- We present the design and implementation of the power modeling system that applies V-edge on popular smartphones, including power models of major hardware components and the per-application energy accounting.
- We evaluate our V-edge-based implementation using a diverse set of benchmarks and applications. The results demonstrate that, given the same model, the error range of the energy estimations of V-edge is within 4%, on average, compared with those of power-meter-based approaches. The model generation is two orders of magnitude faster than SOD-based approaches.

The rest of the paper is organized as follows. In Section 2, we introduce how power modeling works as background. In Section 3, we survey the related work and motivate V-edge. In Section 4, we describe our observation on battery voltage dynamics and demonstrate how to infer current information from voltage readings of battery interface. We present the V-edge energy measurement system in Section 5 and the power models in Section 6. We describe the design and implementation of a system built upon the V-edge power modeling in Section 7 and evaluation results in Section 8. We discuss limitations of V-edge and future work in Section 9 and conclude in Section 10.

2 Background: Power Modeling

A power model estimates the power consumption of a system, such as a smartphone, based on more readily observable system statuses. Typically, the model is generated through a *training phase*. A set of well-designed programs are run to explore various system states in this phase and corresponding power values are measured at the same time. Provided system states and their power measurements as inputs, various modeling techniques like Linear Regression (LR) can derive the relationship between these two sets of information, i.e., a power model.

It is common to simply take resource utilization as the system status, such as screen brightness, CPU usage and so on. As an example of such a *utilization-based* power model, consider a system consisting of only a CPU. To build a power model for this system, one would first design several training programs generating different CPU loads. Then one would run each training program and exploit some measurement tool, like Monsoon Power Monitor [14], to provide corresponding power value P.

Assuming that power consumption of the CPU has a linear relationship with CPU utilization, a power model can be formulated as $P_{cpu} = a * U_{cpu} + b$, where a and b are constant, U_{cpu} is CPU utilization, and P_{cpu} is the estimated power consumption. Here, U_{cpu} is called a *predicator*, as it is used to indicate the power consumption of the CPU. There can be multiple predicators in a power model. For example, if Dynamic Frequency Scaling (DFS) is enabled on the CPU, one may use two predicators, the frequency F_{cpu} and U_{cpu}, to estimate the power consumption. Besides LR, other techniques (e.g., non-linear regression) can also be used to build alternative (often more complicated) power models.

Once a power model is generated, it can be used in a *power estimation phase* to predict the power consumption of the system without requiring additional power measurements. For example, if the CPU utilization of a program is 25% for a duration of T, the the energy consumption of this program is $E_{total} = (a * 25 + b) * T$. More generally, one can monitor and calculate the total energy consumption of a program with dynamic CPU usages as $E_{total} = \sum_i P^i_{cpu} * \Delta T$, where P^i_{cpu} is the i-th measurement of CPU utilization and ΔT is the time interval of the measurement.

Similar to CPU, a power model can be built for other hardware components, such as the screen, Wi-Fi, GPS and so on. After power models of all the components are generated, a power model of the whole system (e.g., a smartphone) can be built on top of the component power models. It is also possible to perform the energy consumption accounting of individual applications or processes, as we will describe in Section 6.

While a power model can give absolute values of energy cost, in practice relative values are often more meaningful to end users. It is usually hard for most users to map absolute energy values (e.g., 10 Joules) to what they concern, such as what percentage of energy has been consumed by screen or an application. As a result, most power monitoring tools on smartphones show power consumption information to users in terms of percentages rather than absolute values [1].

From the above example, we can see that power measurement is the foundation and an essential part of power modeling. As we will see in Section 3, however, the ways in which power measurement is currently done introduces limits to the power modeling's usability and applicability.

3 Related Work and Motivation

System power modeling has been an active research topic and many approaches have been proposed. Based on how power consumption is measured, existing literature can be divided into two categories: *external metering* and *self-metering*. Once power consumption is measured, various *training techniques* to generate models have been studied.

External metering. Most existing work on smartphone power modeling relies on external and expensive power meter to build power models [4, 5, 6, 7, 8]. Those approaches are very accurate because a dedicated power meter can precisely measure power consumption. However, they are labor-intensive and can be done only in a lab. Due to hardware and software diversity of smartphone, each type of smartphones may have a different power model. Any new configuration requires rebuilding the model back in the lab again. Therefore, these in-lab methods are very inflexible and thus not suitable to use in the wild across a large number of users.

Recently, BattOr [9] extended the external meter to mobile settings with a lightweight design. Nevertheless, it is not easy for a layman to operate BattOr because it is not deployed on smartphones. In fact, more and more smartphones use non-replaceable batteries to optimize layout, so attaching any external equipment on them becomes difficult and even dangerous to end-users.

Self-metering. Self-metering approaches [3, 11, 13, 12] collect energy information from smartphones' built-in battery interfaces to generate power models without requiring a power meter. The battery interface consists of battery status registers that the fuel gauge integrated circuit exposes to smartphone operating systems, including voltage, temperature, State-Of-Discharge (SOD), and sometimes current information. The power can be calculated if both voltage and current are provided by the battery interface.

However, many smartphones, including popular ones like the Nexus S and the Samsung Galaxy S2, provide battery interfaces that are only capable of sensing voltages. This means existing self-metering approaches, except [13], are unable to work on a large amount of smartphones (the number is still increasing) already in use.

Zhang et al. [13] proposed building power models based on SOD readings of batteries, which does not require current-sensing. However, the SOD-based approach has a very long model generation time and is inaccurate due to its SOD-based nature. The approach measures the remaining battery capacity (a number from 0% to 100%) to estimate the energy consumption. The granularity of energy measurement is as coarse as 1% of the whole battery capacity. It not only takes tens of minutes to observe a change of battery capacity but also introduces large errors due to the coarse energy granularity.

Motivation of fast power model construction. Fast power model construction is desirable because there are many cases requiring model rebuilding. Besides hardware and software changes, rebuilding is also necessary for changes of software configurations as a simple CPU policy modification may lead to up to 25% differences in power estimation [3]. The battery aging problem [10] also affects power modeling as battery capacity drops significantly with battery age. Thus, a power model needs to be rebuilt after a battery has been used for some time. Furthermore, Dong et al. [3] showed that power models also depend on device usage and demonstrated that a power model should be continuously refined based on usage. In addition, the complexity of modern hardware may require many training cases to generate accurate power models. For example, Mittal et al. [2] used 4096 training cases (for different R, G, B color combinations) to generate a power model for AMOLED display. If it were to take 15 minutes to observe a change of SOD (the minimal time used by Zhang et al. [13]), then it would take the SOD-based approach more than 1,000 hours to generate a single display model, making it almost impossible for end-users to build or rebuild power models on their smartphones.

In addition, the power measuring of a training program need to be performed in a controlled environment. In fast power modeling, short measuring time largely reduces the chance that the user takes the system control back during the running of a training program. Thus, the fast power modeling is more robust because of the tolerance of users' interruptions. Also, the fast one is more flexible because it is able to quickly suspend construction after the completion of a training program and resume later.

Ideally, besides accuracy, a good power model approach should be self-modeling (i.e., it should not depend on external power meters), work for most smartphones (i.e., it should not require current-sensing), and be able to generate models quickly. As shown in Table 1, no existing approach can meet all three requirements. This motivates us to look for a better power modeling approach.

Training techniques for power model construction. Besides LR, other training techniques can also be used for power model construction. For example, Dong et al. [3] used Principal Component Analysis (PCA) to improve the accuracy of a power model by identifying the most effective predicators. Pathak et al. [4] proposed to construct power models using system call tracing. They created Finite State Machines (FSM) for power states of system calls, thus achieving fine-grained power modeling. Our work is complementary to those advanced (and more complicated) model construction techniques. They can be used on top of our battery voltage dynamics based power measurement approach. In this paper, we show that accurate power models can be generated using our new power measurement approach even though we only

Table 1: Comparison of power modeling approaches

	Self-modeling?	Support most phones?	Fast model adaptation?
External metering approaches	✗	✓	✗
Self-metering approaches except SOD	✓	✗	✓
SOD approach	✓	✓	✗
Ideal approach	✓	✓	✓

use basic training techniques for model construction.

4 Sensing Current from Battery Voltage Dynamics

A smartphone is powered by the battery, where supplied voltage is not constant. The voltage dynamics of the battery are exploited here to achieve all desired objectives of the ideal power modeling approach. We show that it is possible to infer discharging current information from instantaneous voltage dynamics of a battery. This inference is reliable enough to be used for power estimation. We also demonstrate that it is practical to detect instantaneous voltage changes by using battery interfaces on smartphones. Based on this, a new energy measurement system is introduced in the next section.

4.1 Battery Voltage Dynamics

The left part of Figure 1 shows the equivalent model of battery electrical circuit [15]. It indicates that at a certain point in time, the voltage reading V of the battery interface can be obtained using

$$V = OCV - V_c - R_b * I$$

where OCV is the open-circuit voltage determined mainly by the remaining capacity of battery, V_c is the voltage drop on the capacitance, R_b is one of the two internal resistors, and I is the discharge current.

When encountering a notable amount of current change, OCV and V_c remain roughly the same value in a short time frame, but the multiplication of R_b and I is sensitive to this current change. As illustrated in the right part of Figure 1, we can observe a sharp edge of voltage readings from the battery interface immediately after the current change. This is known as *internal resistance effect*. After the instantaneous change, the voltage then slowly decreases due to the current discharging on the battery. We define this instantaneous voltage change, $R_b * \Delta I$, as *V-edge*, which is in volts. Clearly, the value of V-edge has a linearly proportional relationship with the change of current. If we measure the V-edge values with the same baseline current I_0 (this can be achieved by starting all the training programs from the same baseline when generating a power model), V-edge has a one-to-one mapping with the current.

$$V_{edge} = R_b * \Delta I = R_b * I - R_b * I_0$$

Or,

$$I = \frac{1}{R_b} * V_{edge} + I_0$$

Via this relationship, we can quickly determine the current value given the V-edge. Next we show that this linear relationship is reliable (Section 4.2) and V-edge can be detected accurately (Section 4.3). Thus, we can use V-edge to further estimate the power consumption and construct power models (Section 5).

4.2 Reliable Relationship between V-edge and Current

The linear relationship between V-edge and current is evident in theory, but because it requires a simplifying assumption about the battery, we seek to understand whether the relationship holds in practice. To this end, we design a set of test trials that run various tasks with different stable workloads on the smartphone. Five batteries for a Google Nexus S phone and three for a Samsung Galaxy Nexus phone were picked for experiments with consideration of different aging stages and manufactures [1]. We ran all tests on these batteries and measured their V-edge values (in μV) respectively. The corresponding current levels (in *mA*) of these tests were obtained on a Monsoon Power Monitor at a constant voltage level. We then modeled the relationship between V-edge and current using LR for each battery.

Table 2 shows the regression results of eight batteries. The first five batteries are for the Nexus S and the last three are for the Galaxy Nexus. R^2 is the *Coefficient of Determination*, a widely used measure of how well the LR is [16]. We can see that R^2 values of these eight fittings are all above 0.99, indicating very good fitting results. More concretely, Figure 2 shows how well the regression fits the data of battery 2, of which the R^2 value is smallest.

Those real-world experimental results demonstrate that the relationship between V-edge and current is indeed reliable. This provides the foundation of our proposed fast and accurate power modeling approach.

[1]new to one-year-old batteries from four manufactures

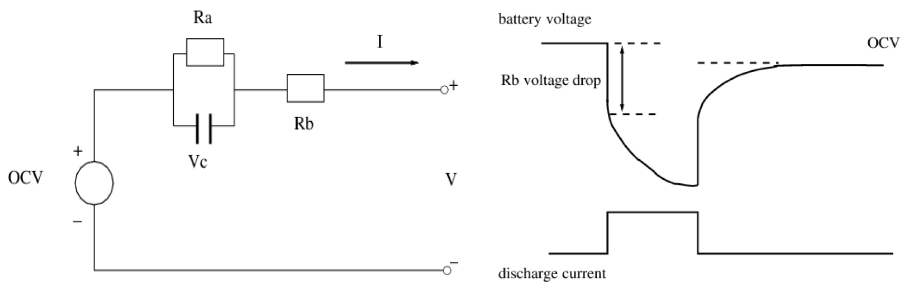

Figure 1: Battery voltage dynamics. Left: Equivalent electrical circuit model for batteries. Right: Battery voltage curve when discharge current is changed. The deep drop of voltage is caused by current increasing on resistor R_b.

Battery	Slope α	Intercept β	R^2
1	0.0048	103.4	0.9987
2	0.0054	100.9	0.9945
3	0.0050	103.3	0.9992
4	0.0054	101.8	0.9991
5	0.0054	102.3	0.9985
6	0.0057	158.9	0.9978
7	0.0056	154.5	0.9979
8	0.0051	157.0	0.9976

Table 2: Linear mapping between V-edge and current on eight batteries of two different smartphones, in the form of current $I = \alpha * V_{edge} + \beta$. R^2 is the metric indicating the goodness of fitting.

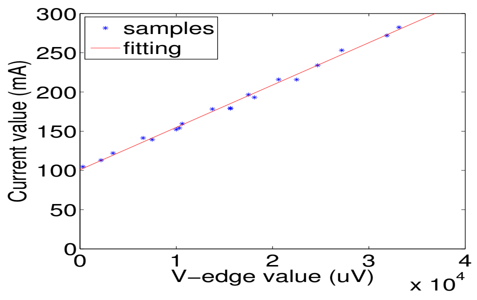

Figure 2: Sampling vs. fitting on battery 2.

4.3 Detecting V-edge

We show in this subsection that V-edge can be easily and accurately captured by battery interfaces on smartphones. Figure 3 illustrates the curve of voltage readings from the battery interface of a Nexus S, when CPU utilization was increased from idle to 95%. We can see a clear voltage drop immediately after CPU utilization (thus the current) was increased. After the instantaneous drop, the voltage decreases very slowly, even with the high discharging current of 95% CPU utilization (the slope will be even gentle if the current draining is smaller). By sampling voltage values from the battery interface before and after the instantaneous voltage change, we can calculate the value of V-edge. Depending on how soon we sample the voltage value after the instantaneous voltage drop, the calculation leads to certain error. Table 3 shows the errors when the sampling happens at different times (i.e., sampling delay) after the instantaneous voltage drop.

We can see that the error is zero if the sample is taken within three seconds of the instantaneous voltage drop. The error increases when the sampling delay becomes larger. If the sampling delay is 10 seconds, the error is 11.76%. Clearly, to reduce error, we should sample voltage value as soon as possible after the instantaneous voltage drop.

The battery interface of smartphones typically updates the voltage value periodically but the updating rate may vary drastically across different phones. For example, the Galaxy S2 updates ten times less frequently than the Nexus S (one update every ten seconds versus one per second). In the case of a low update rate, we should align our voltage sampling with the voltage updating. To achieve this, we employ the following procedure to detect battery interface parameters - the updating interval and time.

We first put the smartphone into idle for a time period longer than its battery interface updating interval (e.g., tens of seconds), then (at time t_0) we increase CPU utilization to a high level and immediately start sampling voltage values at a rate of 1Hz. Once we detect a voltage value change larger than a threshold (i.e., the instantaneous voltage drop caused by increased CPU utilization) at time t_1, we put the CPU into idle again and continue to sample voltage values at 1Hz. When we detect a voltage value change larger than the threshold again (i.e., the instantaneous voltage increase caused by decreased CPU utilization) at time t_2, we stop sampling. Figure 4 de-

Table 3: Sampling error of V-edge in different sampling delays

Sampling delay (s)	1	2	3	4	5	6	7	8	9	10
Sampling error (%)	0%	0%	0%	2.94%	5.88%	5.88%	8.82%	11.76%	11.76%	11.76%

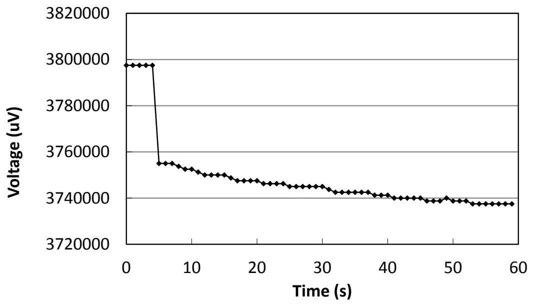

Figure 3: Voltage curve on a Nexus S smartphone when CPU utilization is increased from idle to 95%. Sampling rate is 1Hz.

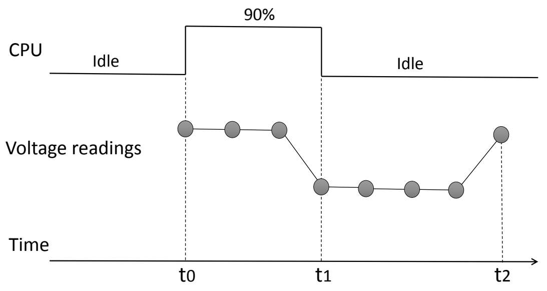

Figure 4: Estimate the voltage updating interval and time of battery interface.

scribes this procedure. Then we treat $\Delta t = t_2 - t_1$ as the updating interval of the battery interface where t_1 and t_2 are the times when voltage updates are triggered. With the sampling rate of 1Hz, the estimation error is within two seconds. Once we know the updating interval and time of the battery interface, we can align V-edge detection with the voltage updating so that the delay of V-edge detection is limited to two seconds. Thus, we can accurately measure the value of a V-edge.

The value of a V-edge is decided by the corresponding current change. If the current change is very small, it is hard to detect the V-edge. To study how likely we can detect a V-edge, we conducted a set of tests with different current changes. Table 4 shows the results. For each current change, we repeated the test 50 times and report the probability that the change was detected. We can see that there is about a 64% chance that the V-edge is detected along with just a 7.5 *mA* increment of current value. With a 30 *mA* change we achieve up to 98% and 100% with 37.5 *mA*. On a Nexus S smartphones, 37.5 *mA* can be caused by a small change of only 4% CPU utilization. To build a power model, we can easily design training programs with a current change much larger than 37.5*mA*. We conclude that V-edge is sensitive enough for component level power modeling.

Current increment (*mA*)	Probability (%)
7.5	64%
15	90%
22.5	98%
30	98%
37.5	100%

Table 4: Probability of capturing current changes

5 V-edge Energy Measurement System

Once we derive the current information from V-edge, it is feasible to calculate the power information and further generate power models on top of it. Thus, in this section, we show how to build an alternative energy measurement system based on V-edge that is equivalent to the traditional energy measurement systems. In traditional energy measurement systems, the energy cost E of a task is measured by power P and time T, $E = P * T$. Power is decided by current I and voltage V, $P = I * V$. For simplicity, we assume that a task has a constant power consumption during its execution time. The same analysis below can be easily extended to a task with dynamic power consumption by dividing the whole execution time into small time slots with a constant power consumption and using $\sum_i (P^i * T^i)$ to replace $P * T$. That is, the total energy cost $E = \sum_i (P^i * T^i) = \sum_i (I^i * V^i * T^i)$, where i indicates the ith slot.

In our new energy measurement system, we introduce a new term, the V-edge power P_{edge}, to replace the traditional power. The V-edge power is defined as

$$P_{edge} = V_{edge} * V$$

where V_{edge} is the V-edge at the corresponding voltage level V and the unit of P_{edge} is square volts. It does not matter whether the value of V is the voltage value before the instant voltage drop or after the instant voltage drop (see Figure 1) because the difference between the two voltage values is fixed as V_{edge}. That is, the two voltage values are interchangeable. In our implementation, we choose the voltage value before the instant voltage drop.

Similarly, we define the V-edge energy as

$$E_{edge} = P_{edge} * T = V_{edge} * V * T$$

to replace the traditional energy.

As we show in Section 4.2, V-edge and current have a linear relationship $I = \alpha * V_{edge} + \beta$. Thus, we have

$$\begin{aligned} P &= I * V = (\alpha * V_{edge} + \beta) * V \\ &= \alpha * P_{edge} + P_{edge_0} \end{aligned}$$

where P_{edge_0} is a constant value denoting the baseline V-edge power. That is, we can calculate the real power consumption of a task from the V-edge power of the task. Similarly, we have

$$\begin{aligned} E &= P * T = (\alpha * P_{edge} + P_{edge_0}) * T \\ &= \alpha * E_{edge} + E_{edge_0} \end{aligned}$$

where E_{edge_0} is the baseline V-edge energy.

During power model generation, we can directly measure P_{edge} but not P_{edge_0}. To calculate P_{edge_0}, we employ the following procedure in the power model training phase. We first design two tasks with constant but different stable workloads. We then run the two tasks to consume the same amount of energy in terms of percentage of battery capacity (e.g., 2% of battery capacity which can be achieved by reading SOD information provided by the battery interface). The V-edge power of the two tasks is P^1_{edge} and P^2_{edge}, their traditional power is P^1 and P^2, and their execution time is T^1 and T^2. Without loss of generality, we assume that T^1 is smaller than T^2. As the tasks consume the same amount of energy, we have

$$\begin{aligned} P^1 * T^1 &= P^2 * T^2 \\ (\alpha * P^1_{edge} + P_{edge_0}) * T^1 &= (\alpha * P^2_{edge} + P_{edge_0}) * T^2 \\ P_{edge_0} &= \alpha * \Theta \end{aligned}$$

where $\Theta = \frac{P^1_{edge} * T^1 - P^2_{edge} * T^2}{T^2 - T^1}$ is a known constant value determined by running the two tasks. Note that here we only need SOD readings to derive the value of baseline power, which is done only once. In SOD-based power modeling approaches, every model training program depends on SOD readings, making the model generation time unacceptably long as shown in Section 8.

In fact, even the determination of P_{edge_0} can be skipped if we are only interested in the energy profile excluding baseline, as is usually the case for end users and application developers. Thus, the V-edge energy system is a linear transformation of the corresponding traditional method.

After knowing P_{edge_0}, in the power estimation phase, when a set of tasks run together (e.g., multiple components or processes), we can obtain energy percentage consumed by each task, even without knowing the value of α. For simplicity, let us assume that there are only two tasks, i and j. We can calculate their power percentage from their V-edge power as follows. The energy consumption of task i is

$$\begin{aligned} E^i &= P^i * T^i = (\alpha * P^i_{edge} + P_{edge_0}) * T^i \\ &= \alpha * (P^i_{edge} + \Theta) * T^i \end{aligned}$$

The calculations of task j are similar to task i, so they are omitted due to space limitations.

Thus, the energy percentage of task i is

$$\%E^i = \frac{E^i}{E^i + E^j} = \frac{(P^i_{edge} + \Theta) * T^i}{P^i_{edge} * T^i + P^j_{edge} * T^j + (T^i + T^j) * \Theta}$$

In addition, we can also estimate how long the remaining battery will last. If $X\%$ of battery has been used by tasks i and j, we have

$$\begin{aligned} X\% * C &= E^i + E^j \\ (100 - X)\% * C &= (P^i + P^j) * T^{ij}_L \end{aligned}$$

where C is the battery capacity and T^{ij}_L is the remaining time of the battery if we continue to run both task i and task j at the same time. By solving the above equations, we can get

$$T^{ij}_L = \frac{100 - X}{X} * \frac{P^i_{edge} * T^i + P^j_{edge} * T^j + (T^i + T^j) * \Theta}{P^i_{edge} + P^j_{edge} + 2 * \Theta}$$

If $T^i = T^j = T$, we simply have $T^{ij}_L = \frac{100-X}{X} * T$.

And if we run only task i in the future, the remaining battery time will be

$$T^i_L = \frac{100 - X}{X} * (1 + \frac{P^j_{edge} + \Theta}{P^i_{edge} + \Theta}) * T$$

In summary, the V-edge energy system is able to measure and estimate the power consumption of a system. In the following section, we will develop a system based on V-edge that can address common user concerns such as how much energy a particular application consumes, or how long the battery will last if the user continues running one or more applications.

6 Power Modeling Based on V-edge

We model the power consumption of four major hardware components of smartphones: CPU, screen, Wi-Fi and

GPS. The main purpose is to demonstrate the usability of the V-edge energy measurement system underlying, so we do not introduce new power models. Instead, we use existing or modified ones which are simple and able to capture the main power characteristics of the hardware. In addition, we describe how to do per-application accounting based on generated component-level power models. Power value is provided in the V-edge power $V_{edge} * V$.

CPU Model. DFS is available on most CPUs and often enabled to save power. Thus, for the CPU power model, we consider both CPU frequency and CPU utilization. For each possible CPU frequency, we model the power consumption of the CPU as a linear function of

$$P_{cpu} = a * U_{cpu} + b$$

where U_{cpu} is the CPU utilization.

Screen Model. The power consumption of a screen is decided not only by the brightness of backlight but also by the pixel colors. For example, at the same backlight level of 255, the power consumption of full-screen white is almost three times that of full-screen red.

Dong et al. [17] used RGB values to create a linear model of OLED (Organic Light-Emitting Diode) type displays' power consumption. However, because this model is not suitable for AMOLED (Active-Matrix OLED) types, Mittal et al. [2] proposed another model to also capture the non-linear properties of AMOLED. However, the full model requires 4096 colors, leading to high training overhead. This neutralizes the advantage of self-metering approaches, timely model adaptation. Therefore, we provide a simplified yet effective alternative using the function

$$P_{screen} = f(L) * (c_r * R + c_g * G + c_b * B)$$

where P_{screen} is the screen power consumption, $f(L)$ is a quadratic function of the brightness level L, (R, G, B) is the average RGB value of all pixels, and c_r, c_g and c_b are the coefficient of R, G, and B.

The goal of this screen model is to reduce the number of colors tested. Based on the above function, we derive a preliminary model from only 216 measured RGB colors ($6 \times 6 \times 6$) by first assuming the linear relationship between the power and RGB color. Obviously, this preliminary model does not work well on AMOLED. Additionally, we measure the power of another 125 samples uniformly distributed in the RGB color space. Then we can obtain the power differences of these 125 colors between measured and modeled values. Because the power of AMOLED gradually changes among similar colors, the difference of one in these 125 colors can roughly represent the average offset between measured and modeled power values for all colors nearby. Therefore, the final estimated power value of a RGB color is the calculation of the above function plus the difference of one of the 125 color samples that is closest to this estimated color. In this way, the modeling error decreases to a low level, while a lot of training time is saved.

Note that when we train a screen power model, the power consumption of training programs will include the power consumption of the CPU because the CPU cannot be turned off to run any training program. Thus, we need to remove the power consumption of the CPU from the total power consumption of screen training programs. This is done by generating the CPU power model first and applying it in training screen power model.

Wi-Fi Model. We employ a simple model that considers the data throughput of both directions. A linear power function

$$P_{wifi} = d * D + e$$

is selected where D is the application data, incoming and outgoing, through the Wi-Fi interface. Similar to the screen model, we also remove the power consumption of the CPU in training the power model of Wi-Fi.

GPS Model. We model the power consumption of GPS based on the ON/OFF states, following the work [11, 18]:

$$P_{GPS} = f_{GPS} * S$$

where f_{GPS} is the the power coefficient and S is 1 when GPS is enabled or 0 otherwise.

Per-application Accounting. Users often want to know the power consumption of each individual application so that they can identify where the energy was spent. This per-application power accounting can be done on top of the component-level power models. We can monitor the activities of a process on each component (CPU, screen, Wi-Fi and GPS) and account corresponding power consumption as a function of

$$P_{process} = \sum_i P^i_{cpu} + \sum_j P^j_{screen} + \sum_k P^k_{wifi} + \frac{1}{N} \sum_l P^l_{GPS}$$

where i, j, k, l are the ith, jth, kth and lth time when the process uses CPU, screen, Wi-Fi and GPS, respectively. N is the total number of processes using GPS at the same time. Zhang et al. [13] found that the sum of all component estimates is sufficient to estimate the whole system consumption. Thus, we also adopt this assumption. The power consumption of an application is the sum of the power consumption of all its processes.

7 System Design and Implementation

We have designed a general V-edge-based architecture, illustrated in Figure 5, to run on typical smartphone operating systems. In our design, V-edge runs as a system service in the background, collects data on system resource utilization and activities, generates power models

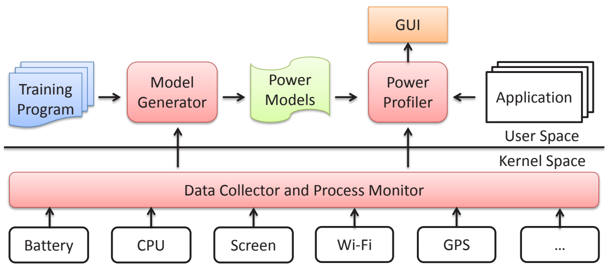

Figure 5: System architecture based on V-edge.

and uses them for power consumption estimation. It also provides a tool with a GUI for users to review the power consumption information of each component and application.

Data Collection. The data collection part is designed to run in the kernel due to two considerations. First, running in the kernel gives us more flexibility and less latency compared with the user space. Second, it avoids the expensive user-kernel mode switching, thus introducing less system overhead. We collect three types of data: voltage readings from the battery interface, utilization information of each hardware component (CPU, screen, Wi-Fi and GPS), and process execution and switching information. For Wi-Fi utilization, we capture the data packets transmitted over Wi-Fi by intercepting the network stack. For process execution and switching information, we hook the kernel scheduler to collect thread scheduling information. We add a new system call to fetch the collected data from the kernel where power model generation and power estimation are done. Voltage information is only used in generating power models, while process information is only used in estimating per-application power consumption. Hardware utilization information is used for both power model generation and power estimation.

Power Model Generation. The system, on top of V-edge, automatically generates component-level power models for CPU, screen, Wi-Fi, and GPS as formulated in Section 6. This is done by running a set of training programs for each component in a controlled way. For example, to build the power model of the CPU, we run CPU training programs with other components in their baseline power states. All training programs of a component run from the same initial state to ensure their measured V-edge values are consistent. For example, each CPU training program starts when the CPU is idle. The model generator also aligns runs of training programs with the voltage updating of the battery interface as we described in Section 4.3, to reduce errors of the voltage sampling. The modeling procedure is done without user awareness when the smartphone is idle and not plugged in. If the user suddenly interrupts the generation by using the phone, the procedure can suspend and resume later with little time penalty, thanks to the short estimation time of V-edge. We also allow power model updates adaptively or through a GUI tool described later in this section.

Power Consumption Estimation. Power estimation is done by tracking hardware resource usage and applying generated models. When users use their smartphones as normal, the data collector keeps running in the background to collect the usage information of each component (frequency and utilization percentage for CPU, brightness level and pixel colors for screen, packet size and number for Wi-Fi and usage of GPS). Thus, the power profiler is able to calculate the power consumption of each component. By tracking process switching, we can know which process is using the resources at a given time. Therefore we can associate resources usage and thus power consumption to the corresponding process, for per-process and per-application accounting.

Power Profiling GUI Tool. On top of the power profiler, we design a GUI tool to show the percentage of the energy consumed by each hardware component and provide a rebuilding option to users.

We have implemented the V-edge-based power modeling and monitoring system on the Android platform. Our implementation in total consists of 2k lines of code for the core components (data collection, model generation and power estimation) and 4k+ lines of code for the training programs.

8 Evaluation

We evaluate our implementation of the V-edge-based system by answering the following questions. 1) How fast can power models be generated? 2) How accurate is the power estimation using the generated models, both at component-level and in per-application accounting? 3) How much system overhead does the implementation introduce in terms of CPU and memory usage?

8.1 Experimental Setup

Devices. We conduct all experiments on a Nexus S smartphone running Android 4.0. We use a Monsoon Power Monitor to measure the actual power consumption of the experiments as the ground truth and for comparison.

Training programs for model generation. We develop a total of 412 training programs to generate power models for CPU, screen, Wi-Fi, and GPS. For each CPU frequency (there are five configurable CPU frequencies on a Nexus S), we use eight training programs with CPU usages randomly picked from eleven possible values (idle to full). Similarly, for the screen we use 347 training programs with different brightness levels and RGB colors of different pixel blocks. For the Wi-Fi, we use 24 training programs with different packet sizes and transmission rates. Finally, we use one training program for the GPS module.

Benchmarks. We design a set of benchmarks to evaluate the accuracy of our implementation on component-level power estimation. For the CPU we use four benchmarks running for 60 seconds at a CPU frequency of 200 MHz, 400 MHz, 800 MHz and 1000 MHz. For the screen, we use 15 benchmarks. Each of them displays a different picture, as shown in Figure 7, for 10 seconds. For Wi-Fi, we use a benchmark which sends UDP packets with a randomly selected packet size of 50, 100 or 1000 bytes and a random packet inter-arrival time from 1 to 50 milliseconds. The total run time of the Wi-Fi benchmark [2] is 60 seconds. For the GPS, we use a benchmark which uses the location service for 60 seconds.

Applications. We use six real applications to evaluation the accuracy of our implementation on power estimation of real world applications. These applications are *Gallery* where we use the default photo viewer on Android to show 20+ photos (randomly taken by the camera on Nexus S) in slide show mode, *Browser* where we use the default Android browser to read news on Bing News, *Angry Birds* where we play this free version game with commercials, *Video* where we watch a homemade video clip on the default player, *Skype* where we make a VoIP call through Wi-Fi, and *GPS Status* where we run a popular GPS-heavy app from Google Play [19]. Each test performs for one minute.

8.2 Model Generation Time

Model generation time is the time period to run all the training programs and construct power models. It is mainly decided by how quickly power consumption can be measured. In the V-edge approach, as shown in Section 4.3, we can detect the instant voltage changes and consequently measure power consumption in several seconds. However, in a SOD-based approach, power measurement time is much longer because it measures power consumption by observing changes of SOD, at least 15 minutes [13]. Given our 412 training programs, it takes the V-edge-based system for 1.2 hours in total (including the stabilization time between the training cases which can be further optimized) to generate the power models. However, it would take more than 100 hours for the SOD approach to generate the same power models. Our proposed approach is two orders of magnitude faster than the SOD approach.

More importantly, the long model building time of the SOD-based approach demands multiple rounds of the battery recharging, thereby requiring the user intervention. As such, the SOD-based approaches are difficult to automate. With the short modeling time, our approach can be easily done without the user involvement. For the sake of optimization, we can further split the whole procedure into small pieces and manage to complete them one by one. Each piece of modeling tasks just takes minutes of the smartphone idle time and consumes little energy. Thus, the V-edge-based system is transparent to end users.

8.3 Accuracy

We evaluate the accuracy of the V-edge approach by comparing its energy consumption estimations with both ground-truth measurements and estimations from power-meter-based models. These power-meter-based models are built by measuring power consumption of training programs using an external power meter in the model generation phase. This external-metering approach represents the highest accuracy that one model can achieve because its inputs are precise. Note that the energy comparison is stricter than direct model parameter comparison because model errors can be magnified.

Accuracy of CPU modeling. Figure 6 shows the energy consumption of the CPU benchmarks, including the ground truth and the estimated results of the V-edge approach and power-meter-based approach. Compared to ground truth, the errors of the V-edge approach are 1.45%, 7.89%, 9.71% and 4.18% (5.79% on average). The corresponding numbers of the power-meter-based approach are 1.32%, 5.28%, 5.92%, and 1.54% (3.51% on average). The average difference between our approach and the power-meter-based approach is only 3.65%.

The stable relationship between CPU usage and power consumption introduces small errors to both V-edge-based and power-meter-based approaches.

Accuracy of screen modeling. Figure 7 shows the results of the screen benchmarks. Compared to ground truth, the average error of the V-edge approach is 5.77% (max 15.32%, min 1.22%) and the power-meter-based approach is 5.55% (max 15.81%, min 0.11%). The average difference between our approach and power meter approach is only 3.49%. Note that Figure 7 shows normalized results. The absolute energy consumption of the pictures are very different, as large as 3.3 times.

Our screen model is one of the most sophisticated smartphone screen models considered in self-metering approaches. Nonetheless, our experiments show that it is of limited accuracy (relatively wide error range). The reason is that this model relies on a small number of reference colors to correct initial estimations and provides final answers. Therefore, if a photo has an average pixel color similar to one reference, its estimation error is low. Otherwise, it is a bit high. The model could be optimized,

[2]For the experiment purpose, a stable wireless environment is expected in order to remove the influence of outside factors

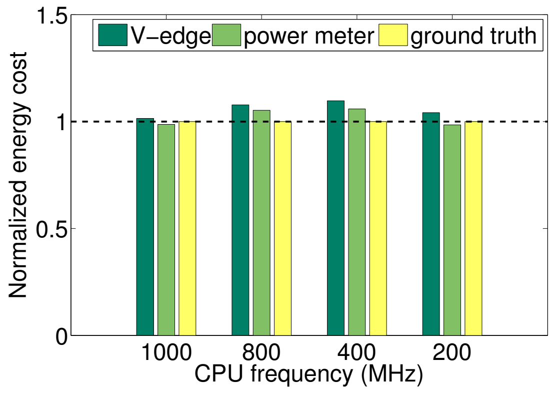

Figure 6: Energy consumption of CPU benchmarks. Results are normalized relative to ground truth values.

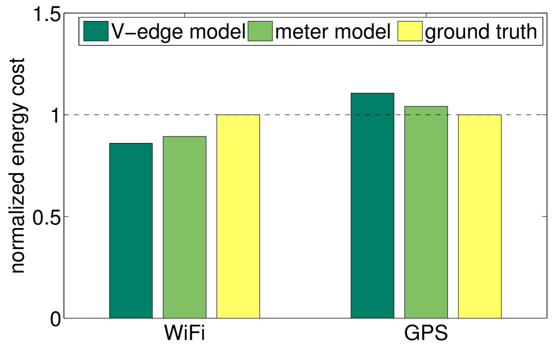

Figure 8: Energy consumption of Wi-Fi and GPS benchmarks. Results are normalized relative to ground-truth values.

but it is out of the scope of this paper.

Accuracy of Wi-Fi modeling. Figure 8 shows the results of the Wi-Fi benchmark. Compared to ground truth, the error of the V-edge approach is 14% and the power-meter-based approach is 10.65%. The difference between two modeling approaches is only 3.75%.

The error of Wi-Fi benchmark is relatively large. This is because our model is simple and served as the comparison platform of two modeling approaches. More predicators like packet numbers per second may improve the accuracy of modeling. Additionally, there are more CPU activities involved in both building and using the Wi-Fi model, compared with other components. Thus, some error is contributed from the CPU model.

Accuracy of GPS modeling. Figure 8 also shows the results of the GPS benchmark. Compared to the ground truth, the error of the V-edge approach is 10.6% and the power-meter-based approach is 4.1%. The difference between our approach and power meter approach is 6.5%.

Accuracy of real applications. Figure 9 shows the results of the six real applications. Compared to the ground truth, the errors of the V-edge approach are 19.5%, 8.6%, 0.2%, 1.6%, 14.7% and 15.5% (10% on average). The corresponding numbers of the power-meter-based approach are 15.6%, 12.5%, 4.2%, 2%, 18.3% and 12.2% (10.8% on average). The average difference between our approach and the power-meter-based approach is only 3.8%.

The accuracy of each component model has an impact on application experiments. For example, the Wi-Fi estimation errors are accumulated quickly in the communication-intensive applications like *Skype*, leading to the relatively large difference between modeled and real results. So is the case of *GPS Status* that has a lot of interactions with the GPS module. As to estimation errors of *Galley*, displayed photos are randomly picked, so it is possible that many of them have average colors not similar to any of 125 references. Another reason is that the average RGB color over all pixels may not be a good predictor. We will investigate this in future. In addition to the individual model accuracy, errors are also introduced by the assumption that the linear combination of all component energy consumption is equal to the whole system consumption. Besides, we do not include power models for other minor energy consumers such as disk I/O.

Summary. All experimental results show that the accuracy of our approach is very close to the power-meter-based approach. The total average difference is only 3.7% for all component-level and application-level power estimations. This demonstrates V-edge's strength in facilitating the self-constructive power modeling.

8.4 System Overhead

Our implementation introduces a very small system overhead in terms of the usage of the CPU and memory. To evaluate, we measured the system CPU and memory usage when V-edge is enabled and disabled for monitoring system energy consumption. With V-edge enabled, the smartphone used only 2 MB more memory to run background V-edge code and store the collect data in memory. Such a small memory footprint is negligible compared with the large memory size of 512MB or 1GB on today's smartphones. We did not observe any noticeable difference on CPU usage because of its event-driven implementation like the work [4]. Thus, our implementation is lightweight and introduces low system overhead.

9 Discussions and Future Work

V-edge provides prior power modeling techniques an opportunity to work on most smartphones on the market. Our power modeling system is one simple example that

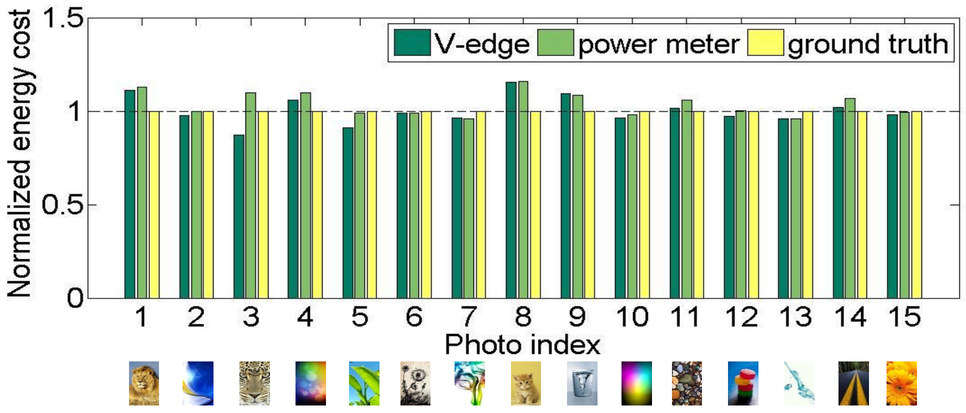

Figure 7: Energy consumption of screen benchmarks. Results are normalized relative to ground-truth values.

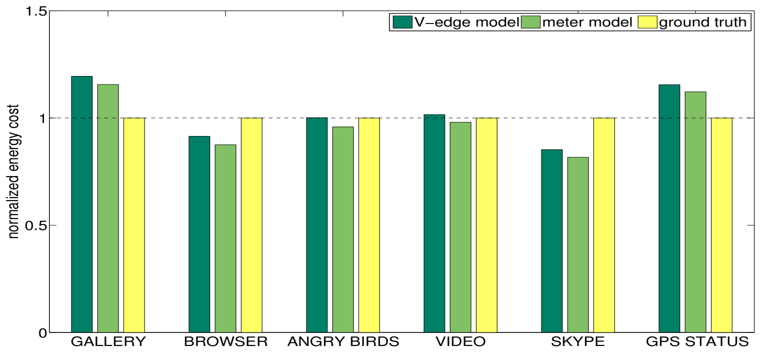

Figure 9: Energy consumption of applications. Results are normalized relative to ground-truth values.

is built upon V-edge. It is intended to demonstrate the implementation feasibility and exhibit benefits that V-edge offers. Therefore, we only cover major energy consumers, such as the CPU and screen. In the future, we plan to complete our models by adding more components, like a 3G module, in order to create a useful system tool.

Another issue worthy of investigation is the model optimization for self-metering approaches. Usually, the more accurate estimations are expected, the more predictors a model need to consider, and thus the more overhead the building procedure has. For example, if we only use the backlight level to model the screen like previous work, 83% building time is saved for our whole system. However, the accuracy is not acceptable. We therefore plan to study how to select more efficient predictors to balance this accuracy and overhead trade-off.

In addition, although our implementation is based on Android platform, the V-edge approach is general enough and not limited to only the Android platform. We plan to implement the V-edge-based system on other mainstream smartphone platforms such as Windows Phone.

10 Conclusions

In this paper, we propose a new approach called V-edge for fast and self-constructive power modeling on smartphones. The V-edge approach is novel because it builds power models by leveraging the regular patterns of the voltage dynamics on battery-powered devices. Different from most existing self-modeling approaches, the V-edge-based approach does not require current-sensing of battery interface so that it works for most smartphones on the market. We have designed and implemented a V-edge-based modeling prototype. It performance demonstrates that V-edge can facilitate fast and accurate power modeling with low overhead.

Acknowledgment. We thank Thomas Moscibroda, Ranveer Chandra, our shepherd Dave Levin, and anonymous reviewers for their valuable comments and insightful feedback. This work was supported in part by US National Science Foundation grants CNS-1117412 and CAREER Award CNS-0747108.

References

[1] Antutu battery saver. https://play.google.com/store/apps/details?id=com.antutu.powersaver&feature=search_result#?t=W251bGwsMSwxLDEsImNvbS5hbnR1dHUucG93ZXJzYXZlciJd.

[2] R. Mittal, A. Kansal, and R. Chandra. Empowering developers to estimate app energy consumption. In *MobiCom*, 2012.

[3] M. Dong and L. Zhong. Self-constructive high-rate system energy modeling for battery-powered mobile systems. In *MobiSys*, 2011.

[4] A. Pathak, Y. Hu, M. Zhang, P. Bahl, and Y. Wang. Fine-grained power modeling for smpartphones us-ing system call tracing. In *EuroSys*, 2011.

[5] A. Carrol and G. Heiser. An analysis of power consumption in a smartphone. In *USENIX ATC*, 2010.

[6] A. Shye, B. Scholbrock, and G. Memik. Into the wild: Studying real user activity patterns to guide power optimizations for mobile architectures. In *MICRO*, 2009.

[7] T. Cignetti, K. Komarov, and C. Ellis. Energy estimation tools for the Palm. In *MSWIM*, 2000.

[8] J. Flinn and M. Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *WMCSA*, 1999.

[9] A. Schulman, T. Schmid, P. Dutta, and N. Spring. Demo: Phone power monitoring with BattOr. In *MobiCom*, 2011.

[10] Battery performance characteristics. http://www.mpoweruk.com/performance.htm.

[11] W. Jung, C. Kang, C. Yoon, D. Kim, and H. Cha. Devscope: A nonintrusive and online power analysis tool for smartphone hardware components. In *CODES+ISSS*, 2012.

[12] S. Gurun and C. Krinz. A run-time, feedback-based energy estimation model for embedded devices. In *WMCSA*, 2006.

[13] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. Dick, Z. Mao, and L. Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *CODES+ISSS*, 2010.

[14] Monsoon power monitor. http://www.msoon.com/LabEquipment/PowerMonitor/.

[15] S. Lee, J. Kim, J. Lee, and B. H. Cho. State-of-charge and capacity estimation of lithium-ion battery using a new open-circuit voltage versus state-of-charge. *Journal of Power Sources*, 2008.

[16] R. Myers. *Classical and modern regression with applications*, volume 2. Duxbury Press Belmont, CA, 1990.

[17] M. Dong and L. Zhong. Chameleon: A color-adaptive web browser for mobile oled displays. In *MobiSys*, 2011.

[18] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha. Appscope: Application energy metering framework for android smartphone using kernel activity monitoring. In *USENIX ATC*, 2012.

[19] GPS status. https://play.google.com/store/apps/details?id=com.eclipsim.gpsstatus2&hl=en.