Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones

Lide Zhang† Birjodh Tiwana† Zhiyun Qian† Zhaoguang Wang†
Robert P. Dick† Z. Morley Mao† Lei Yang⋆

†EECS Department, University of Michigan
Ann Arbor, MI, USA
{lide,tiwana,zhiyunq,zgw,dickrp,zmao}@umich.edu

⋆ Google Inc.
Mountain View, CA, USA
leiyang@google.com

ABSTRACT

This paper describes PowerBooter, an automated power model construction technique that uses built-in battery voltage sensors and knowledge of battery discharge behavior to monitor power consumption while explicitly controlling the power management and activity states of individual components. It requires no external measurement equipment. We also describe PowerTutor, a component power management and activity state introspection based tool that uses the model generated by PowerBooter for online power estimation. PowerBooter is intended to make it quick and easy for application developers and end users to generate power models for new smartphone variants, which each have different power consumption properties and therefore require different power models. PowerTutor is intended to ease the design and selection of power efficient software for embedded systems. Combined, PowerBooter and PowerTutor have the goal of opening power modeling and analysis for more smartphone variants and their users.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling techniques

General Terms

Measurement, Design

Keywords

Power modeling, mobile phones, battery

1. INTRODUCTION

There is tension between the interest in potentially power-hungry smartphone application features and the requirement for low power consumption necessary for long battery lifespans. Designers of smartphone hardware–software platforms have incorporated power-saving features, allowing components to dynamically adjust their power consumptions based on required functionality and performance. However, using these features wisely (or at least avoiding undermining their benefits) requires that software developers understand the implications of their design decisions. Unfortunately, many software developers have limited experience with energy-constrained portable embedded systems such as smartphones. As a consequence, many smartphone applications are unnecessarily power-hungry.

End users have difficulty determining which applications are energy-efficient, and which squander energy; as a result, application users may blame short battery lifespans on the operating system or hardware platform instead of unfortunate and unintentional software design decisions. Fortunately, application designers have an incentive to develop energy-efficient smartphone software. Their main barrier is the difficulty of determining the impact of software design decisions on system energy consumption, but that barrier can be overcome.

Researchers have proposed a number of power models for portable embedded systems including Palm [1] and HTC Dream [2]. These power models were derived manually by using a power meter attached to one specific embedded system instance. As a result of the model derivation process, the generated power model is at best accurate for one type of embedded system and at worst accurate only for the specific embedded system instance for which it was built. It would require great effort and time to manually generate power models for the wide range of phones now available.

This paper describes online power model generation techniques. The models produced by these techniques provide accurate, real-time power consumption estimates for power-intensive Android platform smartphone components including CPU, LCD, GPS, and audio, as well as Wi-Fi and cellular communication components. The proposed power model is based on the influence of the power management and activity states of hardware components on system power consumption. Hardware components are associated with system variables, e.g., LCD brightness, that are subject to introspection and allow estimation of component power consumptions. We also provide an on-line power estimation tool, called PowerTutor, that uses a function of these variables to determine system-level power consumption. The power estimation and model generation techniques described in this paper can be applied to a variety of platforms. PowerTutor has been evaluated on the Android HTC Dream (ADP1) and HTC Magic (ADP2) phones.

In this paper, we show that phones of different types have significant differences in power consumption properties and

This work was supported in part by Google; in part by NSF under awards CNS-0720691, CCF-0702761, and CNS-0347941; in part by SRC under award 2007-HJ-1593; and in part by DARPA under award HR0011-08-1-0021.

CODES+ISSS'10, October 24–29, 2010, Scottsdale, Arizona, USA.

provide evidence that power consumption differences between individual phones of the same type are negligible for HTC Dream and HTC Magic phones. Motivated by this observation and the difficulty of generating power models manually, we propose a battery-based automatic model construction technique. This technique uses the built-in battery voltage sensors common to modern smartphones. Instead of using a power meter, we use this voltage sensor, and a somewhat complex but automated characterization procedure, to generate a power model.

The work described in this paper makes three main research contributions.

1. We provide manually generated power models for HTC Dream and HTC Magic phones. These comprehensive system-level models consider CPU, LCD, GPS, Wi-Fi, cellular, and audio components (see Section 3). *This is the first time a GPS power model has been described.* For components with significant power consumption, we find that power consumption is independent of the states of other components. See Section 3 for details.
2. *We measure variation in power consumption properties among phones.* In particular, for the phones we studied, we quantify the (small) variation among multiple instances of the same type of phone, and the (large) variation among different types of phones. See Section 4 for details.
3. We describe a novel automated power model construction technique. This technique uses built-in battery voltage sensors and knowledge of battery discharge behavior to monitor power consumption while explicitly controlling the power management and activity states of individual components. *It requires no external measurement equipment.* See Section 5 for details.

In addition, our work makes a practical contribution: we describe an easy-to-use on-line power estimation technique that uses the power models described above to determine component-level power consumption during application execution. A software implementation of this estimator has been released on the Android market. This software tool, which we refer to as PowerTutor, has been used by more than 6,000 people. See Section 7 for details.

2. RELATED WORK

In this section we summarize related work on online power modeling and model construction techniques.

Power modeling has been well studied by many researchers, not only for mobile embedded systems, but also for general-purpose computers. For example, some power modeling techniques [3, 4] require deep knowledge of the relationship between processor functional unit activities and their resulting power consumptions. Run-time functional unit activities are monitored using built-in hardware performance counters. In contrast, other researchers developed "black-box" microprocessor power models [5, 6] that require no knowledge of hardware component implementation. These models are based on the assumption of linear relationships between processor power consumption and several hardware performance counters, e.g., instructions executed and TLB misses. Flinn and Satyanarayanan [7] developed a workstation power modeling technique that assigns energy consumption to processes or procedures within a process. Such models are simple, fast, and impose low overhead. However, they only model power consumed by the CPU and therefore provide only part of the solution for embedded system power estimation.

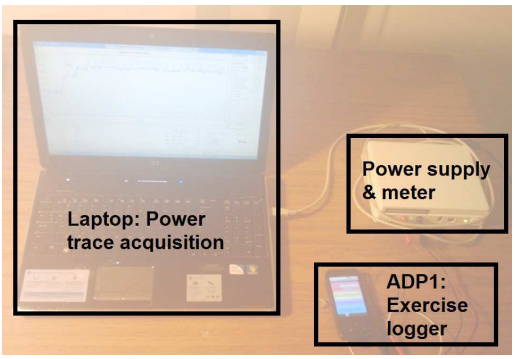


Figure 1: Experimental setup for power measurement.

Mobile embedded system power models are generally component-based. Cignetti et al. proposed a full-system power model for Palm PDAs [1] and Shye et al. [2] derived a system-level power model for Android platform smartphones. Both power models were constructed by correlating operating system visible state variables with power consumption while running a range of normal software applications. This modeling technique is sometimes accurate. However, it suffers from a potential drawback: the accuracy of the resulting model relies on the training applications exercising the full set of component activity and power management states that may be encountered during the use of model. We suggest, instead, that training and characterization applications be designed to explicitly exercise all relevant system states, so that the resulting model is appropriate for use with arbitrary applications. Section 3.2 describes the selection of components and states to consider.

The above power models are constructed using external power meters. To the best of our knowledge, only two papers have proposed battery behavior based power model construction techniques. The concurrent work from Dong and Zhong [8] proposed a automatic construction of a power model using a smart battery interface, while Gurun and Krintz [9] proposed an adaptive power estimation model that uses the built-in Battery Monitor Unit (BMU). Both techniques require knowledge of the discharging current and remaining battery capacity, which are not available for most phones. Our technique relies only on knowledge of the battery discharge voltage curve and access to a battery voltage sensor, which is available on most smartphones.

3. POWER MODEL

We first describe the specific smartphone platform we model, followed by explanations of our measurement setup. We then explain the development of training software to exercise the relevant power management and activity states of hardware components with significant power consumptions.

3.1 Smartphone Hardware Components and Experimental Setup

This section describes power modeling of an Android Dev Phone 1 (ADP1), a version of the HTC Dream mobile phone that permits superuser access. Its hardware components are shown in Table 1. We used the Android 1.6 software development kit, which supports both Java and C program development. In Section 4, we will describe the power model for the Android Dev Phone 2 (ADP2).

Table 1: Hardware Components for HTC Dream

Hardware component	Detailed description
Processor	MSM7201A chipset, including ARM11 application processor, ARM9 modem, and high-performance DSP
LCD Display	TFT-LCD flat glass touch-sensitive HVGA screen
Wi-Fi interface	Texas Instruments WL 1251B chipset
GPS	A-GPS and standalone GPS
Cellular	Qualcomm RTR6285 chipset, supporting GSM, GPRS/EDGE, Dual band UMTS Bands I and IV, and HSDPA/HSUPA
Bluetooth	Bluetooth 2.0+EDR via Texas Instruments BRF6300
Audio	Built-in microphone and speaker
Camera	3.2-megapixel camera
Battery	Rechargeable lithium-ion battery with capacity: 1,150 mAh
Storage	microSD card slot

We use a Monsoon FTA22D meter [10] for power measurement. The measurement instruments are illustrated in Figure 1. The Monsoon meter supplies a stable voltage to the phone and samples the power consumption at a rate of 5 KHz. During characterization, the ADP1 runs two programs simultaneously. One is a power state exerciser that controls characteristics influencing phone power consumption such as CPU utilization and LCD brightness. This control is not always perfect or precise; we therefore also run a second program to log readings at sufficiently high frequency to capture most changes of variables indicating the system state. By using these two programs, we can exercise all relevant power states in a relatively short time, and determine the precise system state at any particular time.

3.2 Selecting Hardware Components and System Variables

To determine which components need to be considered, we carry out the following experiment for each.

1. Hold the power and activity states of all other components constant.
2. Vary the activity state to extreme values for the components of interest, e.g., set CPU utilization to its lowest and highest values or configure the GPS state to extreme values by controlling activity and visibility of GPS satellites.

For each component, determining the setting that results in extreme power consumption requires some experimentation and knowledge of the component implementation.

Based on these initial experiments, we exclude the components with insignificant impact on the system power consumption, e.g., the SD card. The following components are modeled: CPU and LCD display as well as GPS, Wi-Fi, cellular, and audio interfaces. By measuring the power consumption of the phone when it is at different cross products of extreme power states (e.g., for LCD and CPU, the cross products can be [Full brightness, Low CPU] and [Low brightness, High CPU]), we found that the maximum error resulting from assuming that individual components are independent is 6.27%. This suggests that a sum of independent component-specific power estimates is sufficient to estimate system power consumption.

3.3 Training Suites to Derive Power Model

It is necessary to determine the relationship between each state variable and power consumption for each relevant hardware component. The main idea is to use a set of training programs to change one activity state variable at a time, while keeping all others constant. In each training program, we periodically vary each state variable over its full range. Fixing power states of all other components when exercising one component can reduce measurement noise resulting from state transitions by other components. For example, to determine the relationship between CPU utilization and total power consumption, we fix the CPU frequency and disable the LCD display, as well as the cellular, Wi-Fi, and GPS interfaces. We then use a program to gradually vary the CPU utilization from 0% to 100%. Note that some component power state variables cannot be independently controlled. For example, Wi-Fi and CPU power consumptions are interdependent. To take the influence of interdependent components into account, we also monitor all component power states while exercising the target component. During regression, the power states of all components are considered. In the following subsections, we discuss the implementation of the training programs and the relationship between the power consumption and the corresponding state variables.

CPU: CPU power consumption is strongly influenced by CPU utilization and frequency. Varying dynamic, leakage, and peripheral circuit power consumptions invalidate simple cubic frequency–power relationship approximations. In this work, we measure the dependence of CPU power consumption on utilization and frequency–voltage settings.

The HTC Dream platform supports two CPU frequencies: 385 MHz and 246 MHz. The corresponding power coefficients are shown in Table 2. We consider only the application processor (ARM11); system variables are hidden for the other processor (ARM9), which is dedicated to cellular data and voice services [11]. We model the cellular processor as a part of the cellular interface. The variable β_{CPU} shown in Table 2 indicates the power difference between active and idle states of the application processor [12].

The CPU training program is composed of a CPU use controller, which controls the duty cycle of a computation-intensive task, and a frequency controller, which writes the system frequency file in the /sys filesystem.

LCD: The LCD display power model is derived using a training program that turns the LCD on and off and changes its brightness. To simplify modeling, we used 10 uniformly distributed brightness levels.

GPS: We consider the influence of the following GPS-related variables on power consumption: mode (e.g., active, sleep, or off), the number of satellites detected, and the signal strength of each satellite. All these variables are logged using the Android Software Development Kit API. To control the GPS state, we use the *requestLocationUpdate* method [12], to make the GPS component switch between sleep and active states. It was necessary to change the physical environment of the smartphone to control the number

Table 2: HTC Dream Power Model

Model	$(\beta_{uh} \times freq_h + \beta_{ul} \times freq_l) \times util + \beta_{CPU} \times CPU\_on + \beta_{br} \times brightness$ $+ \beta_{Gon} \times GPS\_on + \beta_{Gsl} \times GPS\_sl + \beta_{Wi\text{-}Fi\_l} \times Wi\text{-}Fi_l + \beta_{Wi\text{-}Fi\_h} \times Wi\text{-}Fi_h$ $+ \beta_{3G\_idle} \times 3G_{idle} + \beta_{3G\_FACH} \times 3G_{FACH} + \beta_{3G\_DCH} \times 3G_{DCH}$						
Category	System variable	Range	Power coefficient	Category	System variable	Range	Power coefficient
CPU	util	1–100	β_{uh}: 4.34 β_{ul}: 3.42	LCD	brightness	0–255	β_{br}: 2.40
				GPS	GPS_on	0,1	β_{Gon}: 429.55
	$freq_l$,$freq_h$	0,1	n.a.		GPS_sl	0,1	β_{Gsl}: 173.55
	CPU_on	0,1	β_{CPU}: 121.46	Cellular	data_rate	0–∞	n.a.
Wi-Fi	npackets, R_{data}	0–∞	n.a.		downlink_queue	0–∞	n.a.
	$R_{channel}$	1–54	β_{cr}		uplink_queue	0–∞	n.a.
	$Wi\text{-}Fi_l$	0,1	$\beta_{Wi\text{-}Fi\_l}$: 20		$3G_{idle}$	0,1	β_{3G_idle}: 10
	$Wi\text{-}Fi_h$	0,1	$\beta_{Wi\text{-}Fi\_h}$: Equation 1		$3G_{FACH}$	0,1	β_{3G_FACH}: 401
Audio	Audio_on	0,1	β_{audio}: 384.62		$3G_{DCH}$	0,1	β_{3G_DCH}: 570

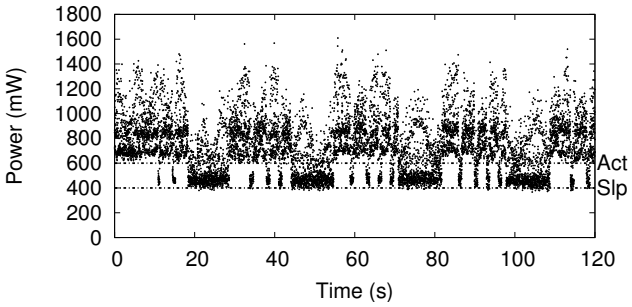


Figure 2: Power profile for the current GPS policy.

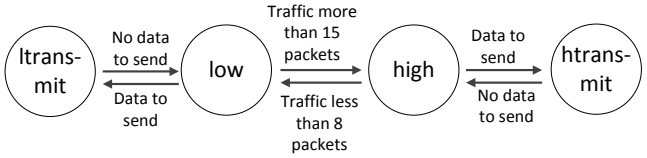


Figure 3: Wi-Fi interface power states.

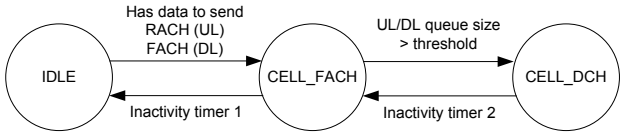


Figure 4: 3G interface power states.

of satellites available and their signal strengths. To this end, we use a conductive hemisphere (i.e., a Faraday Wok) that attenuates radio frequency signals, allowing us to exercise coarse-grained control over the GPS environment. We considered three states: active with many satellites available, active with few satellites available, and sleep. Our measurements indicate that the power consumption depends strongly on whether the GPS component is active or in sleep mode (see Figure 2), but has little dependence on the number of satellites available or the signal strength. Considering only the GPS operating mode is sufficiently accurate.

Wi-Fi: To derive the Wi-Fi model we consider two network parameters: data rate and channel rate. The Wi-Fi power model is derived by exchanging fixed size (1 KB) TCP packets between the smartphone and a local server. We control the data rate by varying the delay between transmissions from 0 s to 2 s in steps of 0.1 s. These experiments are repeated at uplink channel rates of 11 Mbps, 36 Mbps, 48 Mbps, and 54 Mbps. Repeating the experiment with UDP produced similar results.

The Wi-Fi power model depends on four system variables: number of packets transmitted and received per second (*npackets*), uplink channel rate ($R_{channel}$), and uplink data rate (R_{data}). Figure 3 shows the Wi-Fi power model. The Wi-Fi interface has four power states: *low-power*, *high-power*, *ltransmit*, and *htransmit*. *Ltransmit* and *htransmit* are states the network card briefly enters when transmitting data. After sending the data, the card returns to its previous power state. When transmitting at high data rates, the card is only briefly in the transmit state, i.e., approximately 10–15 ms per second. The time for low-power transmit state is even shorter. The Wi-Fi component power consumption in either transmitting state is approximately 1,000 mW. The *low-power* state is entered when the Wi-Fi interface is neither sending nor receiving data at a high rate. Power consumption in this state is 20 mW. Transition from *low-power* state to *high-power* state happens when more than 15 packets are transmitted or received per second. Interestingly, packet rate, not bit rate, determines the power state. The value of *npackets* must drop to 8 per second to return to the *low-power* state, i.e., the system has hysteresis. In the *high-power* state, the power consumption is 710 mW.

To verify our claim that at a particular channel rate and packet rate, Wi-Fi interface power consumption is independent of bit rate, we repeat the experiments with packet size varying from 0 B to 1 KB, in 100 byte intervals. We observe that the packet size does not influence power consumption given fixed channel and packet rates. However, when the channel rate is low, more time is spent in the very high power consumption transmitting state, given the same amount of data transmitted. The Wi-Fi interface power consumption in *high-power* state is modeled as follows:

$$\beta_{Wi\text{-}Fi\_h} = 710\,\text{mW} + \beta_{cr}\left(R_{channel}\right) \times R_{data} \text{ and} \quad (1)$$

$$\beta_{cr}\left(R_{channel}\right) = 48 - 0.768 \times R_{channel}. \quad (2)$$

Cellular: The cellular interface model is derived by sending UDP packets between a smartphone and a local server via the T-Mobile UMTS 3G network. Packet sizes vary from 10 B to 1 KB. For each packet size, we vary the delay between transmissions from 0 s to 12 s in 0.1 s intervals. Results are similar for TCP packets. The following model does not consider signal strength, but this is a focus of our current work.

The measured data are consistent with the finite state machine based power model shown in Figure 4. The model depends on transmit and receive rate (*data_rate*) and two queue sizes. It contains three important states for the communication channel between base station and cellular interface [13].

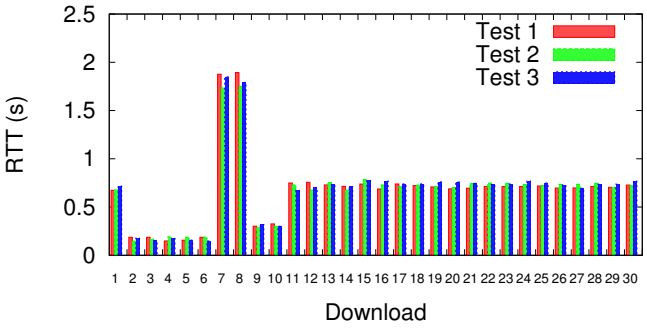


Figure 5: TCP handshake RTT.

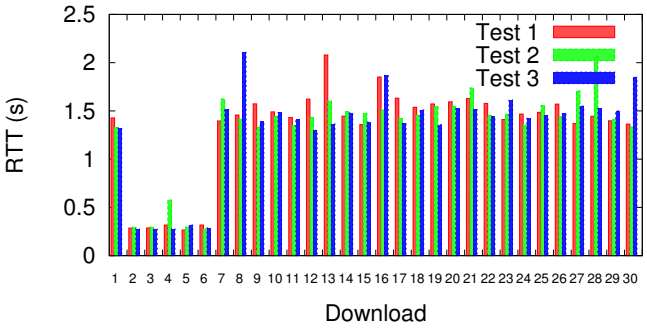


Figure 6: HTTP GET RTT.

CELL_DCH: In this state, the cellular interface has a dedicated channel for communication with the base station. It can therefore use high-speed downlink/uplink packet access (HSDPA/HSUPA) data rates, resulting in a power consumption of 570 mW for the cellular interface. When there is no activity for a fixed period of time (inactivity timer 2), the cellular interface enters the *CELL_FACH* state.

CELL_FACH: In this state the cellular interface shares a communication channel to the base station. It can access the random/forward access (*CELL_RACH*/*CELL_FACH*) common channels. Its data rate is only a few hundred bytes per second. *CELL_RACH* is an uplink channel and *CELL_FACH* is a downlink channel. Cellular interface power consumption in this state is 401 mW. If there is a lot of data to be transmitted, the cellular interface enters the *CELL_DCH* state. Transition from *CELL_FACH* to *CELL_DCH* is triggered by changes in the downlink/uplink queue sizes maintained for these two states in the radio network controller. Our measurements indicate state transition thresholds of 151 bytes for the *uplink_queue* and 119 bytes for the *downlink_queue*. Once either queue size exceeds its threshold, *CELL_DCH* is entered. If idle for a sufficient duration (inactivity timer 1), the IDLE state is entered.

IDLE: In this state the cellular interface only receives paging messages and does not transmit data. The power consumption is 10 mW.

In order to infer the inactivity durations resulting in state transitions for T-Mobile's UMTS 3G network, we repeatedly download an 80 KB file using HTTP 30 times with a period that increases from 1–29 seconds in one-second intervals, recording the timestamp for each packet. The experiment is repeated 3 times. Before each transmission, the connection is left idle for 30 seconds to allow the cellular interface to enter the idle state. We calculate two Round Trip Times (RTTs) at the beginning of each download. The first RTT is the time between sending out a SYN packet and receiving a SYN-ACK packet during TCP connection set up. The second RTT is the time between sending the HTTP Get request and receiving the first data packet. Figures 5 and 6 show the first and second RTTs calculated for each download. Based on these figures, we can infer the times at which state changes occur due to inactivity.

In Figure 5, the RTTs of download 11, as well as those of subsequent downloads, are equal to the RTT of the first download, which starts from the idle state. Hence the sum of the two inactivity timers is 10 seconds. Figure 5 indicates that the RTTs of downloads 7 and 8 are larger than those of downloads 2–6. This is due to the delay of the state demotion from *CELL_DCH* to *CELL_FACH*. Figure 6 shows that downloads 2–6 have smaller RTTs than the others; the other downloads experienced delay in state promotion from *CELL_FACH* to *CELL_DCH*. We conclude that inactivity timer 1 is initialized to 6 seconds and inactivity timer 2 is initialized to 4 seconds.

Audio: We modeled the audio interface by measuring the power consumption when it is not used, and when an audio file is played at different volumes. The measured data (see Table 2) indicate that audio interface use influences power consumption but speaker volume does not. We hypothesize that the increased power consumption during audio output is due to activating a digital signal processor and/or speaker amplifier.

3.3.1 Regression-Based Approach

After collecting power traces for hardware components under control of our training software, we use multi-variable regression to minimize the sum of squared errors for the power coefficient.

$$\begin{pmatrix} P_0 \\ P_1 \\ \cdots \\ P_n \end{pmatrix} = \beta_1 \cdot \begin{pmatrix} U_{01} \\ U_{11} \\ \cdots \\ U_{n1} \end{pmatrix} \ldots + \beta_m \cdot \begin{pmatrix} U_{0m} \\ U_{1m} \\ \cdots \\ U_{nm} \end{pmatrix} + c. \quad (3)$$

In this equation, U_{ij} represents system variable i in the jth state. P_j is the power consumption when all system variables are in the jth state. The inputs for regression are the system variables and the outputs are power consumptions and power coefficients β_i. Constant c is the minimum system power consumption. Note that Equation 3 only represents a linear relationship between system variables and power consumption. However, this is insufficient for some system variables, e.g., processor frequency. In Table 2, we use a zero-one indicator associated with a power coefficient to represent the non-linear relationship between frequency and power consumption.

4. INTRA- AND INTER-PHONE POWER CONSUMPTION VARIATION

We previously explained the construction of a power model for an ADP1 phone. In order to determine how general the power model generated for one phone is, we compared models for different instances of the same type of phone, and models for different types of phones. In this section, we characterize two ADP1 phones and four ADP2 (HTC Magic) phones. HTC Magic has the same processor and LCD specifications as the ADP1 (HTC Dream), but a different cellular interface [14].

Table 3 shows the inter- and intra-type power model variation for ADP1 and ADP2 phones. The intra-type variation is the standard deviation normalized by the mean of the sample phones of the same type. The inter-type variation is the difference between the means of the samples for the two types of phones. Note that the power model parameters in

Table 3: Variation of Power Models Among Phones

Variation (%)		Intra-ADP1	Intra-ADP2	Inter-type
CPU	β_{uh}	1.46	9.6	-23.16
	β_{CPU}	9.05	9.20	33.28
LCD	β_{br}	1.56	2.5	-28.13
Wi-Fi	$\beta_{Wi\text{-}Fi_h}$	1.31	3.55	2.86
	$\beta_{Wi\text{-}Fi_l}$	4.89	4.86	-31
Cell	$3G_{DCH}$	1.03	1.73	62.01
	$3G_{FCH}$	2.80	2.94	27.42
GPS	GPS_{on}	1.35	3.01	-5.12
	GPS_{sl}	2.48	3.82	-11.50
Audio	β_{audio}	3.31	2.57	-59.37

the table can also be seen as power measurements for a particular workload, i.e., variation in power model parameters is linearly related to prediction error. For example, for an application using the audio device, we expect to see less than 4% prediction error from using the power model derived for an ADP1 to predict audio device power consumption for another ADP1. These data provide some support for the following conclusions.

First, inter-type variation is significant. Among all the hardware components, the power models for cellular interfaces differ the most, with variation of 62% between ADP1 and ADP2. This result is consistent with data from the Environment Working Group [15], which shows that the ADP2 has greater cellular interface radiation than the ADP1. Interestingly, although ADP1 and ADP2 have the same LCD display specifications, the power model parameters differ by more than 20%. We speculate that the LCD display in the ADP2 is a more energy-efficient part with the same display quality specifications.

Second, intra-type variation is small. We presently have few intra-type power model samples. To draw a tentative conclusion, we calculated the confidence interval for the intra-type sample variance under the assumption that the distribution of power consumption differences between phones has a Gaussian distribution. With 95% confidence, the maximum intra-type variance exceeds 10.4% for no component. This conclusion is tentative because we have not demonstrated that the power consumption difference distribution is Gaussian.

5. BATTERY STATE BASED AUTOMATED POWER MODEL GENERATION

We have shown that the variation between power models for different types of phones is significant. This necessitates building a new power model for each type of phone. Current manual measurement based modeling techniques are time-consuming and require access to power measurement instruments. Ideally, it would be possible to quickly and conveniently generate accurate power models for new types of phones without access to special equipment.

We propose a power model generation technique that uses knowledge of battery discharge behavior, and the built-in battery voltage sensors in many embedded systems, to determine the average power consumption resulting from placing components into different power states. This power characterization technique does not require external power measurement equipment. We now give a brief tutorial on the properties of lithium-ion batteries, which will provide a foundation for explaining the proposed power model generation technique.

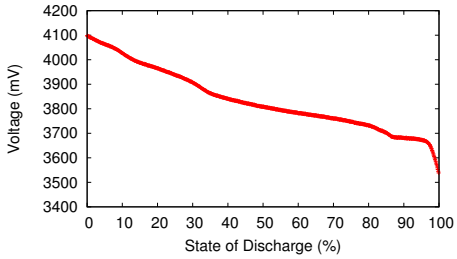


Figure 7: Discharge curve of ADP2 lithium-ion battery.

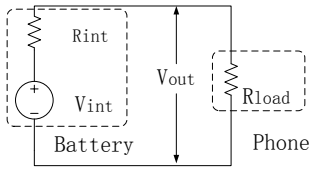


Figure 8: Equivalent circuit for battery.

5.1 Battery Basics

Lithium-ion batteries are popular for portable embedded systems due to their high energy-to-weight ratios, long service lifetimes, and low self-discharge currents.

The voltage of a lithium-ion battery changes during discharge, allowing energy depletion rate (power consumption) to be estimated based on changes to observed voltage. We now explain the reasons for and properties of this voltage change. During discharge, current within the battery is carried by lithium ions (Li+) moving from negative to positive electrodes, through the non-aqueous electrolyte and separator diaphragm [16]. Figure 7 shows the discharge curve of the lithium-ion battery in an ADP2 phone with a (relatively low) discharge current of 64.5 mA. The state of discharge (SOD) is the percent of the rated battery energy that has been discharged. As shown in the figure, the discharge curve is monotonically decreasing. Note that both the energy capacity and the discharge curve change with discharge current, temperature, and battery age [17], which may potentially influence the accuracy of the proposed technique, as we will discuss in Section 5.2.

The internal impedance of a battery and its load (i.e., a phone) influence its output voltage. A battery can be modeled as a variable resistor in series with a variable voltage source, as shown in Figure 8. R_{load} is the equivalent resistance of the phone, R_{int} is the internal resistance of the battery, and V_{int} is the internal voltage of the battery. Due to the voltage drop across R_{int}, the terminal voltage (V_{out}) is lower than V_{int}. V_{int} and R_{int} can be modeled as functions of SOD.

5.2 Battery State Based Model Generation

The main idea of the battery state based power model generation technique is to use the training software described in Section 3 to control phone component power and activity states. The phone components are held in a particular state for a significant period of time and the change in battery SOD is determined using the built-in battery voltage sensor, allowing an estimate of the power consumption for that power management and activity state. At this point, the regression technique described in Section 3 can be used to build a power model. One question remains: how can the battery voltage readings be converted into power consumption values? To achieve that, we need to determine the

SOD (i.e., total consumed energy) variation within a testing interval based on the sensed voltages:

$$P \times (t_1 - t_2) = E \times (SOD(V_1) - SOD(V_2)), \quad (4)$$

where P is the average power consumption in time interval $[t_1, t_2]$, E is the rated battery energy capacity, and $SOD(V_i)$ is the battery state-of-discharge at voltage V_i (i is 1 or 2). The following challenges remain for the proposed technique.

Determining the SOD based on voltage: As shown in Section 5.1, the present voltage can be used for an inverse lookup of SOD based on the discharge curve. However, there is a potential problem with this idea. The discharge curves of different batteries may vary. Using the same look-up table for all batteries may be inaccurate. We therefore characterize the discharge curve for each battery separately. During characterization, the training software discharges the battery from fully charged to completely discharged states using a constant discharge current, thereby maintaining a linear relationship between SOD and discharging time. A logger runs in background to record the battery output voltage. Note that even for the same phone, the discharge curve may vary with temperature and aging. To eliminate the effect of these external factors, we recommend that characterization be conducted at room temperature (i.e., 73–78°F), in which range the average variance of discharge curve is 4.3%. We acquire the variance by comparing the corresponding voltages at uniformly distributed samples of SOD on the SOD curves under the highest (112.2°F) and lowest temperature (89.6°F) reached by our training suites.

We use a piece-wise linear function to model the non-linear relationship between SOD and battery voltage. To derive the function, we traverse $\{(SOD_1, voltage_1), \cdots, (SOD_n, voltage_n)\}$, which is ordered by increasing voltage values. Each additional SOD and voltage tuple is grouped with the data points on the most recent line segment and linear regression fitting is used. If the maximum error returned by regression is larger than an error threshold, in our case 0.1%, we start a new line segment at the current point.

Determining the energy capacity E: As shown in Equation 4, E is needed to determine power consumption. However, nominal energy capacity may change due to aging and discharge rate. There are two potential solutions to the aging problem. A user with a new battery (e.g., an early adopter wanting to characterize a new type of phone for the first time) can read E from the battery label. For a user with old battery, it would instead be necessary to determine the battery energy based on knowledge of system power consumption in some state, e.g., the maximum CPU power consumption. Note that it would be possible to build a power model for all the components in a new phone without knowing the battery initial energy or the power consumption of any component. However, the power consumptions in this model would be relative to the battery energy, i.e., knowing absolute component power consumptions requires knowledge of the absolute value of some energy or power consumption number within the model.

Different discharge rates result in different battery energy capacities. We quantified the error resulting from assuming that the battery energy capacity is independent of discharge rate by using the lowest and highest discharge rates the phone can produce during discharge curve characterization. For our ADP2 battery, this results in less than 2.4% error in power consumption estimates.

The impact of internal resistance: Due to internal battery resistance, the shape of the discharge curve depends on the discharge current. As a result, even if the internal voltage source has constant voltage, i.e., the battery has the same SOD, the terminal voltage depends on discharge current. Worse yet, the internal resistance depends on SOD. To eliminate the impact of internal resistance, we switch all components of the phone to their low-power modes to minimize discharge current when taking a voltage reading. This minimizes voltage drop across R_{int} in Figure 8, thereby minimizing the difference between V_{out} and V_{int}. We used a voltmeter to verify that V_{int} is within 0.03 V of V_{out} under these conditions.

Some phones that have recently started to appear on the market are equipped with built-in current sensors. This simplifies determining the power consumption for each component power state. However, built-in current sensors are not yet common, so relying on their presence reduces the generality of a power modeling technique.

6. POWER MODEL VALIDATION

In this section, we evaluate the accuracies of the meter-based (see Section 3) and battery-based (see Section 5) power models. We first evaluate the meter-based model when running popular applications. Then, we explain the implementation of the proposed battery-based model construction technique and evaluate the resulting model by comparing it with the meter based model.

6.1 Accuracy Analysis for the Meter-Based Power Model

We validated the power model on six popular applications.

- Break the Block: A game that uses CPU, LCD, and Audio.
- Google Talk: An instant message application that uses CPU, LCD, Wi-Fi/3G, and Audio.
- Google Maps: A web mapping application that uses CPU, LCD, Wi-Fi/3G, and GPS.
- The Weather Channel: A weather forecast application that uses CPU, LCD, Wi-Fi/3G, and GPS.
- YouTube: A web-based video sharing application that uses CPU, LCD, and Wi-Fi/3G.
- Browser: The default web browser on Android that uses CPU, LCD, and Wi-Fi/3G.

We repeated the experiment under the following conditions. We used full brightness for Google Maps, Break the Block, and Gtalk; brightness level 102 for YouTube; brightness level 210 for The Weather Channel; and brightness level 36 for Browser. We enabled 3G in Gtalk and used Wi-Fi for all other applications.

To evaluate the accuracy of the model, we used two error metrics. *abs avg* is defined as the average of the absolute values of the errors, i.e.,

$$\left| \frac{\text{measured} - \text{predicted}}{\text{measured}} \right|. \quad (5)$$

To estimate the accuracy of the model when estimating the impact of software design on phone battery lifespan, we used another metric, *avg*, i.e.,

$$\frac{\text{measured} - \text{predicted}}{\text{measured}} \quad (6)$$

This metric better gauges accuracy predicting the power consumption over long time spans.

Figure 9 shows the modeled and measured power consumptions for each application. Error histograms are also

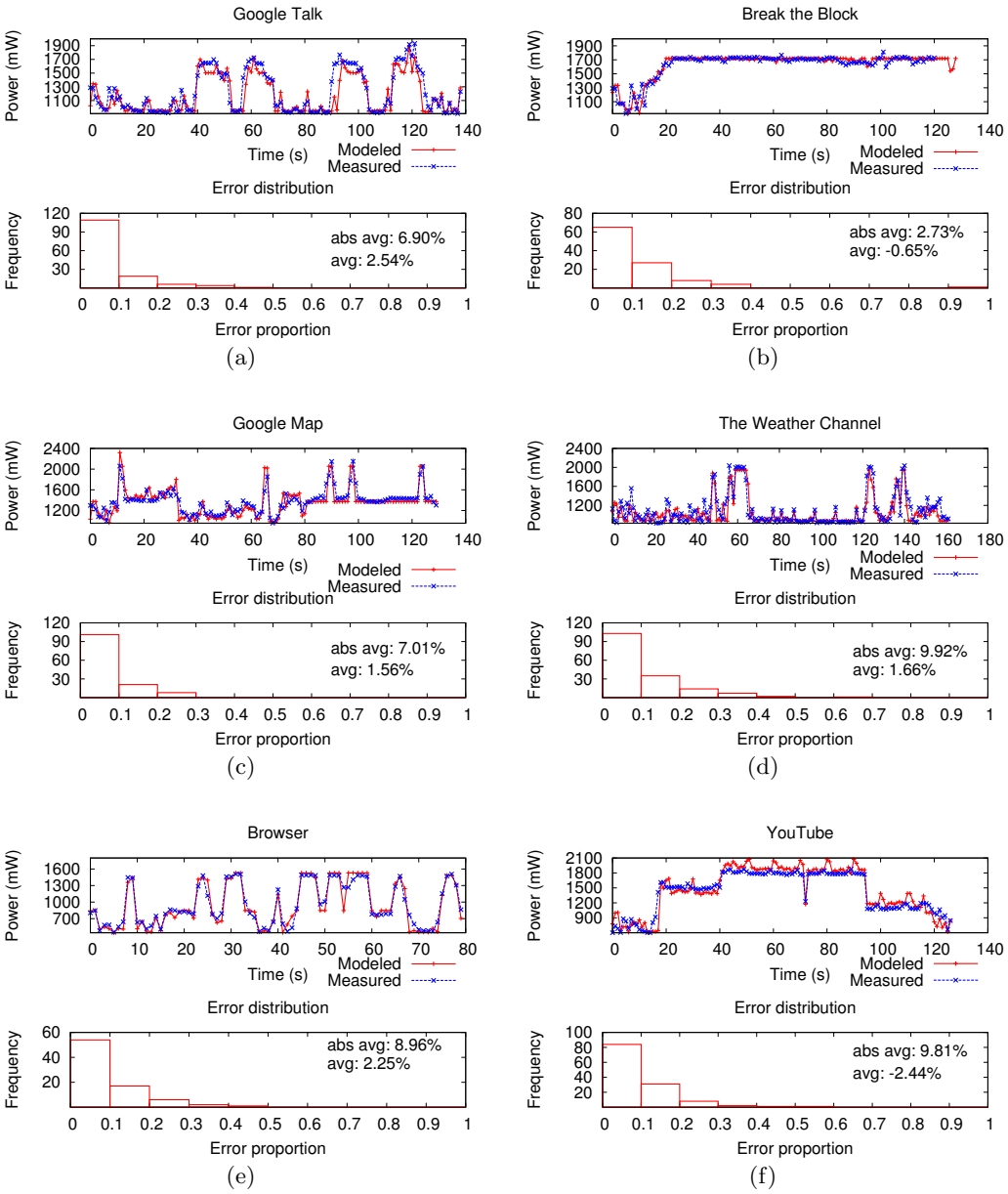


Figure 9: Power profiles for selected applications.

shown. The figures show that the average long-term error (*avg*) is less than 2.5% over the application's lifespan and that average error (*abs avg*) is less than 10% for 1 second intervals.

We also measured the power overhead of the on-line power consumption estimation technique. It is only 80 mW, an order of magnitude lower than the power consumption of high-power states of most smartphone components.

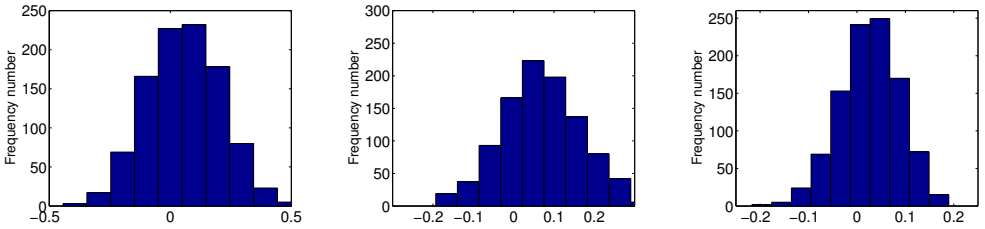
6.2 Implementation of the Automatic Battery-Based Model Generation Technique

The power modeling process may be automated as described in Figure 11. As described in Section 5, constructing the battery discharge curve requires three steps.

1. *Obtain the battery discharge curve for each individual component.* The battery starts in a fully charged state. We characterize the discharge curve individually for each phone.

2. *Determine the power consumption for each component state.* In this step, the state of a single component is varied while other components are kept in low-power states. In order to determine the power consumption for each power state, the battery voltage at the beginning and the end of the power state discharge interval are recorded. The phone is placed in a low-power state immediately before taking a voltage reading to eliminate the impact of the voltage drop across the battery internal resistance. We repeatedly measure voltage for 1 minute, and discharge the battery for 15 minutes between measurement intervals.

3. *Perform regression to derive the power model.* After



(a) 15 minutes (b) 30 minutes (c) 45 minutes

Figure 10: Error distribution for LCD.

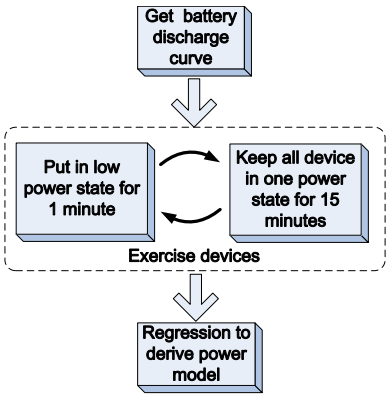


Figure 11: Battery SOD based power model construction.

the battery voltage differences for each power state discharge interval are collected, we use them to calculate average power consumption within the 15-minute intervals. We then use regression to generate the power model

.

The discharge interval for each power state is difficult to select. Minimizing this duration makes the characterization process more convenient. However, the interval must be long enough for the change in battery voltage to exceed noise. In order to determine the optimal interval, we do statistical analysis of the model error distribution as a function of battery discharge interval per power state. Figure 10 shows the error distribution of the LCD model. We did this analysis for all components.

To estimate the error distribution of the battery SOD based technique, one could repeat the entire model construction process many times with different discharge intervals. However, there is a more efficient way to gather the same data. *Bootstrapping* is a technique to treat an initial set of samples as a stand-in for the population and to re-sample from it repeatedly, with replacement. In our experiment, we collect 6 samples at each LCD brightness with a discharge duration of 15 minutes. We then randomly select one sample for each brightness level. By doing regression on these randomly selected samples, we are able to derive an LCD power model. The error is defined as the percentage difference between the newly-derived model and the meter-based model. Repeating this process 1,000 times allows us to determine the error distribution for models generated using 15-minute battery discharge intervals. To determine the distribution for 30-minute intervals, we randomly select and average two 15-minute sample points without replacement at each brightness level. Note that the experiment was designed to minimize correlation between different samples

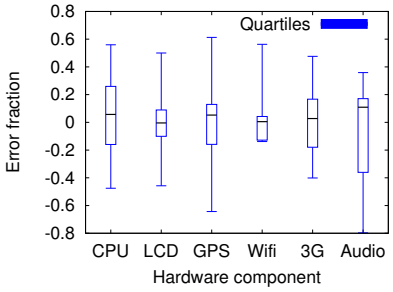


Figure 12: Error distributions for components.

for the same power state; we repeatedly cycled through all power states six times.

We can draw two conclusions from Figure 10. First, the mean of the errors for all discharge intervals deviates from zero by no more than 0.4%. This suggests that, given an adequate battery discharge interval, the battery-based model is as accurate as the meter-based model. Second, the variance of the distribution decreases with longer intervals. For a 45-minute interval, more than 92% of trials have errors less than 10%.

These two conclusions suggest two alternative model-construction system designs. Users can be allowed to choose a trade-off between model construction time and accuracy. We expect that most users would allow a power model to be automatically constructed while they sleep (6.5 hours for a 15-minute battery discharge interval). Another alternative is to have a central web-based system gradually learn the model from samples collected from multiple users. Each user might characterize a phone using a 15-minute battery discharge interval and submit the data. The data for multiple users of the same type of phone could be combined to produce an accurate model. Note that model construction would only need to be done once for a new type of phone, and that automating this process and removing the need for special power measurement equipment would represent a significant improvement on current conditions.

6.3 Accuracy Analysis for the Battery-Based Power Model

Figure 12 shows the error distributions of the battery-based power model using a 15-minute battery discharge interval. The error distribution is generated as described in Section 6.2. The box boundaries indicate the 25th and 75th percentiles and the line span indicates the maximum positive and negative errors. The line in the middle of the box is the mean of all errors.

(a) Main view (b) Chart view

Figure 13: PowerTutor interface.

7. POWER ESTIMATION TOOL

This section describes PowerTutor [18], an online power estimation system that has been implemented for Android platform smartphones. PowerTutor provides accurate, real-time power consumption estimates for power-intensive hardware components including CPU and LCD display as well as GPS, Wi-Fi, audio, and cellular interfaces. It uses power models generated using the proposed manual (see Section 3) or automated (see Section 5) characterization techniques. The interface for PowerTutor is shown in Figure 13. Figure 13(a) shows its configuration screen, with which the display of on-line power traces can be configured. Figure 13(b) shows an example display of the power consumption traces of various hardware components.

PowerTutor has two main purposes:

- Application developers can use PowerTutor to rapidly, accurately, and conveniently determine the impact of software design changes on power consumption. It provides a time series of power consumption estimates per hardware component, allowing developers to identify power inefficient behavior, much of which results from unintentional but inappropriate use of smartphone hardware devices.
- Smartphone owners can use PowerTutor to determine the power consumption characteristics of competing applications, allowing them to make better informed decisions about which applications to use or buy. Most existing application descriptions and reviews do not mention power consumption. PowerTutor also estimates battery lifespan subject to a particular smartphone owner's actual application usage patterns.

PowerTutor has been released on the Android market and has been used by more than 6,000 people.

8. CONCLUSION

This paper has described an on-line power estimation and model generation framework. The PowerTutor power estimation tool informs smartphone developers and users of the power consumption implications of decisions about application design and use. The power model in PowerTutor includes six components: CPU and LCD as well as GPS, Wi-Fi, audio, and cellular interfaces. For 10-second intervals, it is accurate to within 0.8% on average with at most 2.5% error. A software implementation of the power estimation tool has been publicly released on the Google Android Application Market. This paper has also described PowerBooter, an automatic battery state of discharge based power model generation technique. PowerBooter power model construction without using a power meter. The result indicates that the power model built with PowerBooter is accurate to within 4.1% of measured values for 10-second intervals.

9. REFERENCES

[1] T. Cignetti, K. Komarov, and C. Ellis, "Energy estimation tools for the Palm," in *Proc. of the ACM Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 2000, pp. 96–103.

[2] A. Shye, B. Scholbrock, and G. Memik, "Into the wild: studing real user activity patterns to guide power optimizations for mobile architectures," in *Proc. Int. Symp. Microarchitecture*, 2009, pp. 168–178.

[3] R. Joseph and M. Martonosi, "Run-time power estimation in high-performance microprocessors," in *Proc. Int. Symp. Low Power Electronics & Design*, Aug. 2001, pp. 135–140.

[4] C. Isci and M. Martonosi, "Runtime power monitoring in high-end processors: Methodology and empirical data," in *Proc. Int. Symp. Microarchitecture*, Dec. 2003, pp. 93–104.

[5] F. Bellosa, "The benefits of event-driven energy accounting in power-sensitive systems," in *Proc. Special Interest Group on Operating Systems European Wkshp.*, 2006, pp. 37–42.

[6] G. Contreras, et al., "XTREM: a power simulator for the Intel XScale," in *Proc. Conf. Languages, Compilers, and Tools for Embedded Systems*, June 2004, pp. 115–125.

[7] J. Flinn and M. Satyanarayanan, "PowerScope: a tool for profiling the energy usage of mobile applications," in *Proc. Wkshp. on Mobile Computer Systems and Applications*, 1999, p. 2.

[8] M. Dong and L. Zhong, "Sesame: A self-constructive virtual power meter for battery-powered mobile systems," Tech. Rep., 2010.

[9] S. Gurun and C. Krintz, "A run-time, feedback-based energy estimation model for embedded devices," in *Proc. Int. Conf. Hardware/Software Codesign and System Synthesis*, Oct. 2006, pp. 28–33.

[10] "Monsoon power monitor," http://www.msoon.com/LabEquipment/PowerMonitor/.

[11] "MSM7000 chipset," http://www.qualcomm.com/products_services/chipsets/index.html.

[12] "Android SDK reference," http://developer.android.com/reference/packages.html.

[13] H. Holma and A. Toskala, *HSDPA/HSUPA for UMTS: High Speed Radio Access for Mobile Communications*. John Wiley & Sons, 2006.

[14] "HTC Magic specification," http://www.htc.com/www/product/magic/overview.html.

[15] "Environment working group data," http://www.ewg.org/cellphoneradiation/Get-a-Safer-Phone?&allavailable=1&order=sar.

[16] D. Linden and T. B. Reddy, *Handbook of Batteries*. MacGraw-Hill, 2002.

[17] "Battery and energy characteristics," http://www.mpoweruk.com/performance.htm.

[18] "PowerTutor," http://powertutor.org.