# Mobile Application and Device Power Usage Measurements[1,2]

Rahul Murmuria, Jeffrey Medsger, Angelos Stavrou
*Computer Science Department*
*George Mason University*
*Fairfax, VA 22030*
*Email:{rmurmuri, jmedsger, astavrou}@gmu.edu*

Jeffrey M. Voas
*Computer Security Division*
*National Institute of Standards and Technology*
*Gaithersburg, MD 20899*
*Email:jeff.voas@nist.gov*

***Abstract*—Reducing power consumption has become a crucial design tenet for both mobile and other small computing devices that are not constantly connected to a power source. However, unlike devices that have a limited and predefined set of functionality, recent mobile smartphone devices have a very rich set of components and can handle multiple general purpose programs that are not a-priori known or profiled.**

**In this paper, we present a general methodology for collecting measurements and modelling power usage on smartphones. Our goal is to characterize the device subsystems and perform accurate power measurements. We implemented a system that effectively accounts for the power usage of all of the primary hardware subsystems on the phone: CPU, display, graphics, GPS, audio, microphone, and Wi-Fi. To achieve that, we make use of the per-subsystem time shares reported by the operating system's power-management module. We present the models capability to further calculate the power consumption of individual applications given measurements, and also the feasibility of our model to operate in real-time and without significant impact in the power footprint of the devices we monitor.**

***Keywords*-smart handheld devices; energy model; power usage measurements;**

## I. Introduction

A recent smartphone market study suggests that free mobile applications on the Android Market which display advertisements sometimes spend up to 75% of their total power consumption on powering these third-party advertisement services [1]. According to researchers, there is a clear trend that suggests that third party applications in the mobile application markets are not designed or engineered with power consumption as a priority [2]. Furthermore, software engineers and practitioners have pointed out some of these design and implementation deficiencies and even consider them as bugs for mobile devices because battery is such a constrained resource [3].

Being able to generate a device-specific scalable power consumption model is therefore crucial for understanding, designing, and implementing better mobile application software. Of course, the consumption measurement and modeling system must have a low footprint on its own while it collects system metrics; otherwise, it will affect the validity of the measurements. A system that significantly alters the power consumption of the device can cause changes that will invalidate the accuracy of the measurements and of the produced consumption model.

However, we believe that a proper energy accounting infrastructure will assist both application developers and smartphone users to extend the battery life of their devices and to make informed decisions about where to spend the remaining device power, potentially in real-time. For the purpose of this study we focused on Android devices but plan to look at other platforms such as the iPhone or Blackberry, in the future.

Earlier research in power analysis for smartphone devices has produced power models with differing features and focuses. We differ from these research efforts in the following key aspects.

1) We perform measurements and data collection of each device subsystem's power consumption by leveraging existing system hooks in the Android kernel wakelocks driver. This enables us to acquire both accurate measurements and at the same time reduce the power and computational requirements of model generation on the phone. Furthermore, we can perform analysis at different time-scales because the measurements happen cumulatively and in real-time.
2) We take into account operation states for individual subsystems in order to construct the model. However, our calculations are independent of the time interval used in creating the model and do not depend on assigning fixed power usage ratings to the operation states considered.
3) We perform measurements of behaviour for all the primary device subsystems under different conditions which provide an in-depth understanding of how the power is consumed on a smartphone.

[1]Disclaimer: We identify certain products in this document, but such identification does not imply recommendation by the US National Institute of Standards and Technology, nor does it imply that the products identified are necessarily the best available for the purpose.

[2]Disclaimer: This paper was co-authored by Jeff Voas; it reflects his personal opinion and does not necessarily reflect the opinions of the Department of Commerce or NIST.

## II. Related Work

There is a wealth of research studies on power models in existing literature. Some of them specifically target smartphones but depend on external hardware to measure the actual current charge consumed by individual components of the smartphone. For instance, PowerScope [4] is an oft-cited research tool which uses hardware instrumentation to measure system activity and thereby allows for the determination of the power consumption of mobile applications.

Some more recent models divide the system into states and associate a fixed power consumption value to each state. In [5], the authors describe the device as a finite-state-machine. They trace system-calls in order to determine which state the system is in. On the other hand, PowerBooter [6] employs a seemingly comprehensive set of training and characterization applications in order to construct a model with power usage costs assigned to predefined states. They further claim that there is negligible difference in the usage patterns for two phones of the same manufacturing. However, there is sufficient literature to suggest that smartphone usage among different human users can lead to different energy models [7], [8]. Further, software updates, indoor or outdoor usage, geographical region and climate, and wear-and-tear are some of the other factors which may cause significant changes in the model over time. Therefore, characterizing the system with a fixed set of benchmarks and a model that has fixed states is not scalable, even if calculated comprehensively.

In Sesame [9], authors present a statistical power model that is adaptive and can be deployed for general-purpose computers as well as Linux-based smartphones including Nokia and Android. Their focus is on achieving high rate power estimations by performing an estimation every 10 ms. We discuss the differences between our model and Sesame in Section IV.

Pathak et al. in [2], [5] present a model based on tracing system calls. Their claim is that hardware component utilization-based models do not address the power behavior of devices, such as camera and GPS, where a process might use the subsystem for a few milliseconds, but in consequence, the operating system keeps that subsystem alive for much longer. They further claim that certain operations on the smartphone have an indirect impact on subsystem power usage. For example, file I/O can cause a change in the power state of the sdcard module. While these findings are true, it does not invalidate the use of utilization-based models. Our system receives utilization information from the power suspend driver, and therefore does not depend on sampling all device calls made from userspace activities. Moreover, our system uses a sampling rate to perform the regression while creating the model, but thereafter to actually evaluate power consumption in real-time, we do not need a sampling rate. Our subsystem constants, once calculated, are time-period independent. We explain this concept further in Section III-A. An additional problem with system call trace-based power models [5] is that system calls do not address the power consumed by sensory inputs, such as the proximity sensor, compass, and capacitive touch-screen.

Other researchers have attempted to model individual activities on the smartphone for better energy management. Iyer et al. demonstrated a technique to save overall energy consumption by darkening parts of the mobile organic light-emitting diode (OLED) display, which were not the focus of a user's attention [10]. They further experimented with greenscales, but did not cover a full-scale analysis of the power consumption based on the average pixel color of the screen. Our study in Section V further complements their study. CoolSpots [11] is another oft-cited research that demonstrates the use of quantitative measurements to automatically switch between multiple radio interfaces, such as Wi-Fi and Bluetooth, in order to save power. Castignani et al., in [12], conducted a similar study for newer Wi-Fi and 3G interfaces.

## III. Model

On mobile systems, the battery discharge is reported by the kernel as a global value. There is no existing commercially available method to divide the measurements into per-subsystem or per-application readings on any platform. Therefore, we have created a statistical model which breaks the power consumption into subsystems.

$$power\ used = cpu + display + graphics + \\ + gps + audio + mic + ... + wifi. \quad (1)$$

We accomplish this by collecting subsystem usage measurements from the operating system. In our sample implementation, the usage information is obtained from the power suspend driver, which was implemented by Google for the Android operating system. In addition, we record the CPU utilization, screen brightness, pixel color strength on the screen, and other such parameters which are not controlled by a subsystem or device driver, but by the operating system itself.

We consider the power used by each subsystem as a function of the time that the subsystem was used and the state in which the subsystem was operating. We express this in terms of usage *U* of the subsystem. This can be represented mathematically.

### A. Regression model

Let *C* be the constant for the relative impact of a subsystem *i* on the power consumption *P*, and *U* be the total usage of the subsystem *i*. This equation is applied for each time interval *t*.

$$P(t) = \sum_i C_i U_i(t). \quad (2)$$

Table I
WAKELOCKS CORRESPONDING TO EACH SUBSYSTEM

| Subsystem | Wakelock | Subsystem | Wakelock |
|---|---|---|---|
| Display | main | Graphics | kgsl |
| Wi-Fi | wlanrxwake | GPS | GPS |
| Audio | AudioHardwareQSDOut | Touchscreen | event3 |
| Microphone | AudioHardwareQSDIn | Compass | event2 |

We use Android's power suspend driver to get most of our subsystem usage data. Therefore, it is possible for us to collect data cheaply in the required format as per this model. The Android operating system requires that every device driver register a wakelock with this suspend driver. The operating system calculates the amount of time that this driver was in an active state. The locks we chose for each subsystem are listed in Table I. The performance of our logger is discussed in Section VI.

In our experiments, we used a *training stage* to calculate the relative constants *C*. For this, we performed Ordinary Least Squares (OLS) regression on equations where usage *U* and power *P* were already known.

The constants *C* are the relative ratios of each subsystem's impact on the overall power consumption. These constants are subject to external factors, like wear and tear, weather changes, network signal strength, and device driver updates. It is possible to perform this regression in real-time, so that the constants *C* update when the environment changes. However, our goal is to demonstrate a possibility of creating such a model and therefore we make certain calculations in a lab setting, even though the same can be performed on the device, in real-time.

Once these constants are calculated, they can be used in the same regression equations, in order to predict the power usage of a live system, irrespective of whether the power supply source was USB, Alternating Current (AC) adapter or battery.

### B. Application power analysis

The model described so far has used subsystem usage values for the entire operating system. By extension, we replace these values with subsystem usage statistics of a given application or process, and we calculate per-application power consumption.

That is, for a given application *j* :

$$P^j(t) = \sum_i C_i U_i^j(t). \tag{3}$$

In this analysis, we do not need to recalculate the constants $C_i$. The same constants work for the per-application analysis and for any new chosen time-interval for measurements.

### C. Ordinary Least Squares (OLS) regression analysis

The least squares linear regression is one of the most basic statistical models [13]. When expressing our system in statistical terms, the 'dependant variable' or 'output variable' is the power *P* and the 'input variables' or 'features' are subsystem usages $U_i$. Since there are *i* subsystems, we have an *i*-dimensional feature space for the regression, which can be plotted as an *i*-dimensional plane. For every time interval, we get one set of input variables and an output variable.

The goal of the *training stage* is to find the best choice values for the constants $C_i$ which are fixed coefficients for each subsystem. The accuracy of these chosen constants are determined by the method of least squares approach. During the training period, we have data for the subsystem usages $U_i$ and the current discharge *P*, for *n* time intervals. The residual *R* is a function of the amount of error for each of the *n* intervals. The error is calculated by a simple formula:

$$\sum_n (actual\ P_n - calculated\ P_n)^2. \tag{4}$$

In (4), the calculated $P_n$ is the $\sum C_i U_i$ for all *i*, where the $C_i$ are chosen coefficients. By minimizing the value of (4), we achieve the best values for the constants $C_i$. Thereafter, these derived constants $C_i$ can be used with subsequent input readings U, at the time interval $n+1$, to calculate $P_{n+1}$.

This system can be adaptive if we perform the least squares analysis periodically on the device to find better chosen constants, C. In our implementation, we simply provide the subsystem usage data *U* in the form of a matrix of *n* rows (one for each time interval) and *i* columns, along with a vector of all $P_n$ values, into a built-in OLS function in GNU Octave software [14]. If we performed this computation on the phone, we could do the same in real-time with very little overhead. However, such an implementation is not in the scope of this paper.

It is to be noted that we need certain constraints on the best chosen values for $C_i$. Often, the OLS function would over-fit the data by setting constants to zero or negative values. We restrict the function from returning such a result and assume the next best fit as our correct choice of constants.

### D. Non-linearity of subsystems

It is common knowledge that most subsystems are non-linear, such that the power consumed at a given time interval strongly depends on the state in which the device was operating. For example, different pixel color and brightness states have a strong effect on the amount of power consumed by the display subsystem.

PowerBooter [6] reduces the system into a fixed set of states for which the power usage is pre-calculated. Thereafter, they employ a multi-variable regression on the data. In contrast, we demonstrate how we translate the non-linear power consumption of individual subsystems into a linear equivalent, without compromising the continuous nature of our data. In order to be able to make this argument, we first attempted to understand the power consumption of each

subsystem under the various states in which they operate. These operating states are analyzed in Section V.

## IV. Discussion

In this section, we compare our model implementation with that of Sesame [9]. Sesame also employs a regression technique to calculate subsystem constants in a training mode and later use them with system parameters in order to compute the expected output or response.

They use the Total Least Squares (TLS) method in order to create their statistical model. This model was chosen to reduce errors in both predictors (input) as well as responses (output). We argue that TLS has limitations in many situations.

1) In order to calculate error from the residual in our OLS-based model, we needed the expected result for the power (output). However, if both the output and input were to be adjusted, like they are in TLS, then relative "error" for each of the input variables needs to be known as well. This is usually not available in a real-time scenario.
2) Second, it might be fair to assume that some of the input from predictors can be noisy, or its update time period can be infrequent, but an assumption that all the data points need to be adjusted can lead to dilution of the data. By doing this, we hurt the system's capacity to identify anomalies.
3) Third, using TLS automatically requires you to scale the data to common terms. Running the regression on the data where one of the predictors has either 50 seconds, or 50000 ms or 0.83 of the total time window, will each give very different results. Scaling one predictor has a strong impact on the constants calculated for other predictors. On the other hand, because we use OLS, our subsystem data does not need to be carefully normalized to the same common terms.

Further, the component analysis and linear regression performed by Sesame are expensive. Their paper suggests that it can take about 10 minutes on mobile devices to collect the predictor data from the operating system and perform the regression. They perform this calculation when the device is connected to a power source with an AC adapter or a USB cable.

Our technique primarily uses the wakelocks to determine subsystem usage (predictors). This interface covers all the device drivers, and as a result, most of our data comes from a single interface. Our data collection therefore is instantaneous. Our regression calculations also take only a few seconds off-line and when performed on a mobile device, we are confident that this CPU time will not increase significantly.

Table II
Components of Google Nexus One

| Component | Nexus One |
|---|---|
| Processor | 1 GHz Qualcomm QSD 8250 Snapdragon ARM |
| LCD Display | SLCD capacitive touchscreen % |
| Wi-Fi | Wi-Fi IEEE 802.11b/g/n |
| GPS | aGPS |
| Cellular | T-Mobile USA: GSM/UMTS/HSPA |
| Audio | Built-in microphone and speaker |
| Battery | Internal Rechargeable Li-ion: 1400 mAh |
| Operating System | Android 2.3.3 (Gingerbread) Cyanogenmod |

## V. Subsystem Analysis

The subsystems are each dependent on various factors that determine their power consumption. This includes the amount of time they are kept alive, but more importantly, the state of the hardware and other environmental factors. We performed various experiments in order to tailor our regression model to fit each of the subsystems. Table II details the Nexus One device we used for our primary results.

### A. Display subsystem

Our model calculates a 'usage' value, $U_{disp}$, for the display component of (2). Each pixel of the screen consumes a different amount of power depending upon its color and screen brightness. In this paper, we call this 'pixel strength'. The display subsystem usage for a given time interval *t* is determined by

$$U_{disp}(t) = avg.\ pixel\ strength\ *\ display\ uptime. \quad (5)$$

The *display uptime* is recorded from the power suspend driver by parsing the time that the wakelock 'main' was active. In Table I, we list the locks for each subsystem.

To calculate *pixel strength*, we studied the effect of the display brightness level and pixel color on the rate of current discharge. The results of our experiments concluded that red, green, and blue pixels each have a different effect on the rate of current discharge. Blue pixels cause a higher rate of current discharge than green pixels and green pixels cause a higher rate of current discharge than red pixels. We found that the effect of the remaining colors can be determined by performing calculations on their rgb components.

*1) Experimental Setup:* We created an Android application which sets the screen color to the programmed rgb value. The screen is further set to full screen mode, so that the entire screen is filled with that one color, eliminating the menu bar at the top of the normal screen view. We then record the rate of current discharge in milliampere-hour per minute (mAh/m) for brightness levels varying from 0% to 100%.

*2) Parameters chosen for measurements:* In Fig. 1, we plotted white and green colors for different measurement environments. The plot shows that these parameters do not influence the current discharge readings.

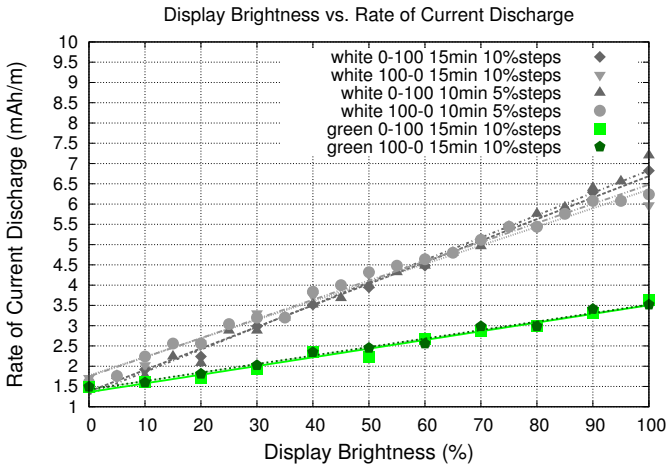


Figure 1. Effect of measurement parameters on current discharge readings

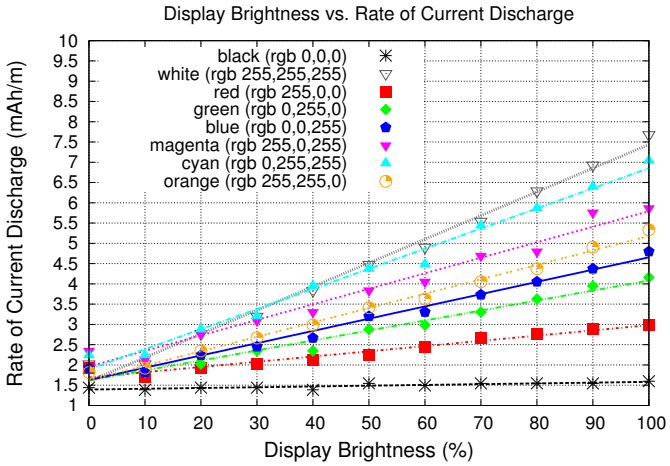


Figure 2. Effect of Pixel Color and Display Brightness on the Rate of Current Discharge

Therefore, our measurements for the data presented in Fig. 2 were taken at 10% increments of brightness levels, with each level measured for 10 minute intervals. The readings were recorded starting from 0% brightness, moving upwards until 100%.

*3) Pixel Strength Analysis:* Fig. 2 displays the results for different combinations of red, green, and blue colors. We observe that the black pixels had the smallest effect on the rate of current discharge and white pixels had the highest effect on the rate of current discharge. Further, the blue component had a greater impact on current discharge than the green and the red components.

Black, which has an rgb value of (0,0,0), consumed some baseline amount of current charge. We performed a linear regression on the area between the curves for each color and the black curve, for the various rgb combinations. We conclude that the red, green, and blue components have a linear impact on the overall current discharge. Further, for

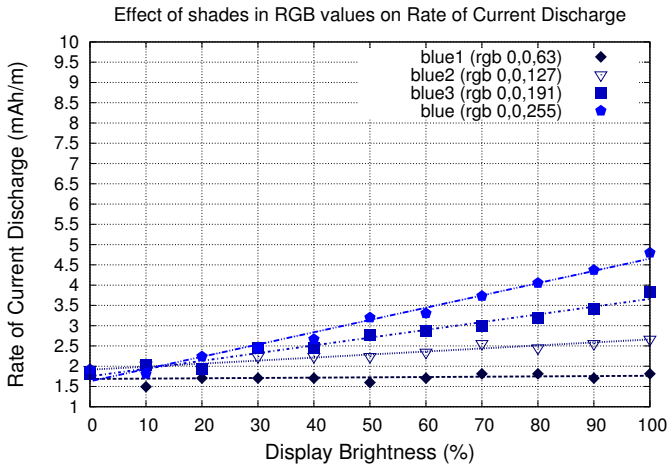


Figure 3. Effect of color shade on the rate of current discharge

any color combination, given the rgb value, we can calculate this impact on current discharge by using (6).

$$pixel\ strength = 0.4216 * r + 0.7469 * g + 1 * b. \quad (6)$$

We can verify this by comparing the ratio of pixel strengths of two colors with the ratio of the area under the curves of these two colors.

For Orange vs. Blue at 100% brightness:

- Orange: 0.4216 * 255 + 0.7469 * 255 + 1 * 0 = 297.968.
- Blue: 0.4216 * 0 + 0.7469 * 0 + 1 * 255 = 255.0.

Comparing 297.968 : 255.0 to the areas obtained from actual data from the device, 191.499 : 165.315, we find that the error in our calculation is 0.87%.

For Magenta vs. Cyan at 100% brightness:

- Magenta: 0.4216 * 255 + 0.7469 * 0 + 1 * 255 = 362.508.
- Cyan: 0.4216 * 0 + 0.7469 * 255 + 1 * 255 = 445.46.

Comparing 362.508 : 445.46 to the areas obtained from actual data from the device, 239.011 : 288.462, we find that the error in our calculation is 1.7%.

These measurements were for the strongest shade of 255 for each component. We performed further measurements with different shades of blue. These readings are plotted in Fig. 3. Based on the readings, it is clear that the shade also impacts the current discharge linearly and (6) is true for weaker shades of each component as well.

These results indicate that our model is effective in determining the pixel strength, given the rgb values. In order to calculate the average pixel strength of the entire screen, we find the average rgb values at any given instant from a snapshot of the framebuffer.

*4) Implementation:* We incorporated these observations into our logger which creates the input data for (2).

The pixel strength is calculated using native code which samples the framebuffer for rgb values at random points,

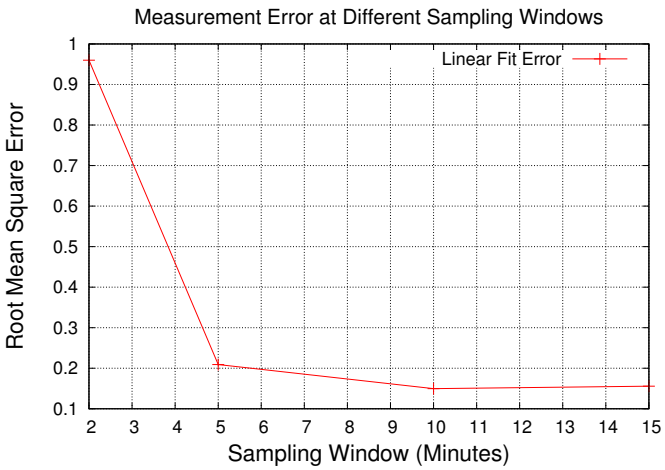


Figure 4. Measurement error in rate of current discharge for CPU utilization increasing in steps of 5% at a fixed frequency

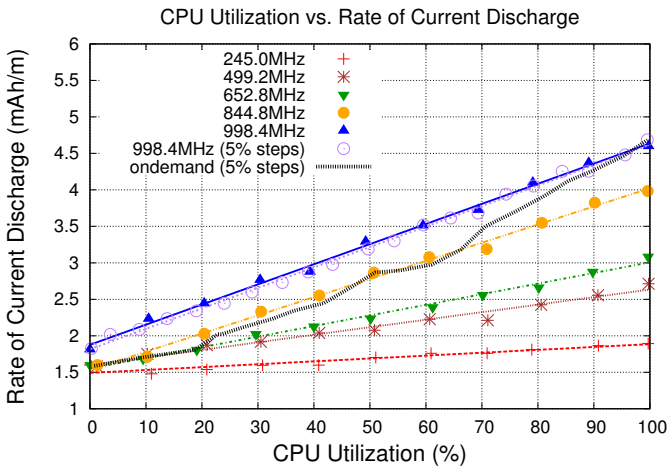


Figure 5. Effect of CPU Frequency and Percentage CPU Utilization on the Rate of Current Discharge

and averages them. We used Glenn's formula [15] in order to determine the right sampling size for the framebuffer for a confidence level of 99% and an error allowance of 2%. We fed these averaged values into (6) to get the corresponding pixel strength of the screen at the time of recording the measurements.

Multiplying this value with the wakelock uptime gives us the best estimate of the display subsystem usage, $U_{disp}$, for the corresponding logging time interval $t$.

This usage value is recorded into a matrix, which is the input for our OLS regression model described in Section III.

### B. CPU subsystem

We studied the effects of CPU utilization and CPU frequency on the rate of current discharge. The CPU utilization can have large effects on the overall rate of current discharge depending on the frequency at which the CPU is maintaining. The higher the CPU frequency is, the greater the rate of current discharge. We ran experiments to find a pattern.

*1) Experimental Setup:* We created an Android application which stabilizes the CPU utilization at a target percentage using an arbitrary CPU load and maintains that percentage CPU level for a fixed interval while we record the rate of current discharge. We set the desired CPU frequency in the operating system for each test run. This setting can be found under the *sys* filesystem.

For the set of tests performed in this experiment, the application was configured to start each test by stabilizing itself at 0% CPU. We then changed the CPU utilization in steps of 2%, 5%, or 10% while running for a 10 minute interval at that utilization level. The tests were performed with frequencies set to 245.0 MHz, 499.2 MHz, 652.8 MHz, 844.8 MHz, 998.4 MHz and 'on-demand'.

The 10 minute interval was chosen due to low variance in the collected data. This was the conclusion of our experiments with various time intervals for recording the measurements. We plotted the mean squared error for measurements at different sampling intervals in Fig. 4.

*2) CPU Performance Analysis:* The curves for the selected frequencies are plotted in Fig. 5. At regular increasing levels of CPU utilization, the amount of current discharge per minute was recorded. At any given CPU utilization, the power consumption is linearly comparable depending upon their set frequency.

In summary:

(a) The effect of the percentage CPU utilization on the rate of current discharge can be accurately represented linearly.

(b) The CPU frequency has a large effect on the rate of current discharge and the higher the CPU frequency, the greater the rate of current discharge.

*3) Implementation:* In our logger, we parse the CPU utilization and the current CPU frequency from the *proc* and *sys* filesystems respectively. CPU utilization for a given interval is equivalent to the ratio between the total time that the CPU spent in its various modes and the time that it was idle. Since there is no corresponding wakelock in the power suspend driver, the parsed values are used directly in the regression model for the CPU subsystem usage, $U_{cpu}$.

### C. Wi-Fi Subsystem

The power consumption by the Wi-Fi subsystem during transmission depends on the rate of packets sent or received. We conducted experiments to determine the exact relationship between these factors and to create equations to fit the findings into our model. We were able to generalize the readings to arrive at one single equation to govern all states of transmission. Our study does not include or identify the power consumed to seek a connection with a Wi-Fi access-point. A separate study can be conducted to extend our

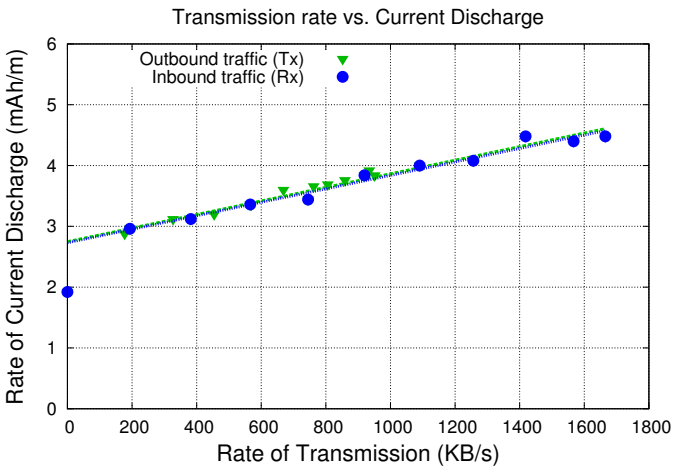


Figure 6. Effect of Wi-Fi transmission rates on current discharge

model to include the power consumed to find and connect to different access points.

*1) Experimental Setup:* The network link used was a Local Area Network (LAN) with a Wi-Fi 802.11G router. For testing outgoing traffic *(Tx)* a program was used which sent User Datagram Protocol (UDP) packets from the mobile device over Wi-Fi at different data transmission rates. The program was run natively on the mobile device. UDP packets were sent from the device at a rate of 0 KB/s for the first 20 minutes and the transmission rate increased by approximately 100 KB/s each 20 minutes until a transmission rate of 950 KB/s was reached. The number of bytes transmitted from the device and the battery charge count were recorded.

A second experiment was conducted to determine if the amount of incoming traffic *(Rx)* had a similar impact on the rate of current discharge. A program was run on the device to receive UDP packets sent to the device at different transfer rates by another client on the network. The client on the network sent UDP packets to the device at a rate of 0 KB/s for the first 20 minutes and then increased the rate at which it sent packets by 100 KB/s each 20 minutes, until a rate of 1700 KB/s was reached.

The device was kept stationary for this entire experiment. The effect of mobility during Wi-Fi communication is not yet covered in our model.

*2) Analysis:* Fig. 6 displays the results from the inbound and outbound transmissions performed. As the Wi-Fi transmission rate increases by 100 KB/s, the rate of power discharge increases by an average of 0.11 mAh/m. The slope of the two curves can be averaged to form (7).

$$current\ discharge\ rate\ (cdr) = 0.0011x * 2.739. \quad (7)$$

To calculate the Wi-Fi subsystem usage $U_{WiFi}$ for each time window, we calculate the *cdr* for average transmission rate from (7), and get the time that the Wi-Fi subsystem was active in the given interval.

$$U_{WiFi}(t) = cdr * WiFi\ uptime. \quad (8)$$

The subsystem uptime is found using the wakelock *wlanxwake* as mentioned in Table I. This lock records the cumulative amount of time that the Wi-Fi device was active.

### D. Audio subsystem

The audio subsystem is associated with the wakelock *AudioHardwareQSDOut*. The audio subsystem wakelock records the amount of time that the device was active.

We created an experiment in which we set the volume of the phone to 13%, 60% and 87% for 10 minutes each. We found the recorded wakelock uptime to be the same value at all three volume levels.

However, the power consumption depends not only on the time that the audio driver was used for, but also on the volume of the sound being output from the speakers. Therefore, just as we did in the case of transmission rate for Wi-Fi, we can derive an equation that calculates the power consumption at different audio volume levels, and use that in conjunction with wakelock uptime.

### E. Other subsystems

Other subsystems can also be characterised using the same method as described for display, Wi-Fi and audio. For a few of the subsystems, like graphics and touchscreen, it is adequate to solely use the amount of uptime the subsystem was active as the subsystem usage *U*.

## VI. Performance of our logger

The experiment discussed in this section determined the efficiency of our logger application when using different sampling windows.

### A. Experimental setup

All tests in this experiment were performed on a Nexus One phone running Android 2.3.3, Cyogenmod build. Our logger application captures battery charge readings and wakelock driver statistics at regular intervals and writes them to two log files. The efficiency of the logger was tested in its 5 second, 30 second, and 300 second sampling window modes. The following steps were performed:

1) A baseline rate of current discharge was calculated manually by noting the current charge before performing the test by reading the $charge\_counter$ file provided by the $sys$ filesystem, then letting the phone go into a low power state with the screen off for 90 minutes, and then noting the current charge after the 90 minute test. The rate of current discharge for the baseline was determined to be 0.0003 mAh/m.
2) Then, the logger application was started and set to the 5 second, 30 second, and 300 second sampling windows. The same 90 minutes interval measurements,

Table III
OUR LOGGER APPLICATION'S POWER CONSUMPTION, WHEN OPERATED AT DIFFERENT MODES

| Logger's Sampling Window | mAh/m |
|---|---|
| 5 s | 0.47 mAh/m |
| 30 s | 0.18 mAh/m |
| 300 s (5 min) | 0.11 mAh/m |

like in step 1, were performed. The logger application has an alarm function which kept the phone from going into low power state during measurements. The increase in the rate of current discharge (i.e. the amount of power $P_{logger}$, used by the logger) was calculated by using the following formula in (9).

$$P_{logger} = P_{logger+baseline} - P_{baseline} \quad (9)$$

### B. Results

Table III shows the performance of the logger at the various chosen sampling windows. The application uses less current charge per minute if the sampling window to parse system data was set to a larger value. What is to be noted here is that although a 5 second interval produces 0.47 mAh/m, a 30 second interval brings this down to 0.18 mAh/m. As the sampling window is increased further, the gain in power efficiency tends towards 1.

## VII. APPLICATION ANALYSIS

We used the generated power model to meter a few chosen applications on the Nexus One device. Each test case consisted of running an application on the phone for 30 minutes, and having our logger application record statistics that were used to perform off-line calculations using our power model. This allowed us to demonstrate that the calculated subsystem constants $C_i$ can be used in predicting the rate of current discharge attributed to an application.

### A. Test Setup

The applications were evaluated by running multiple tests as described below, and the collected data was operated upon using the Octave scripts described in Section III. Each test ran for 30 minutes.

The tests run were in the following combinations:

1) Only the logger application.
2) The logger + Music.
   Music is an application that comes standard with Android Nexus One phones. The Music application was set to play a 3 minute music file repeatedly for the duration of the test.
3) The logger + Sound Recorder.
   This Sound Recorder application is also one of the standard applications on the device, and uses the microphone to record ambient sound.
4) The logger + GPSTracker.
   GPSTracker is an application from the market which logs the GPS coordinates into a trace file.

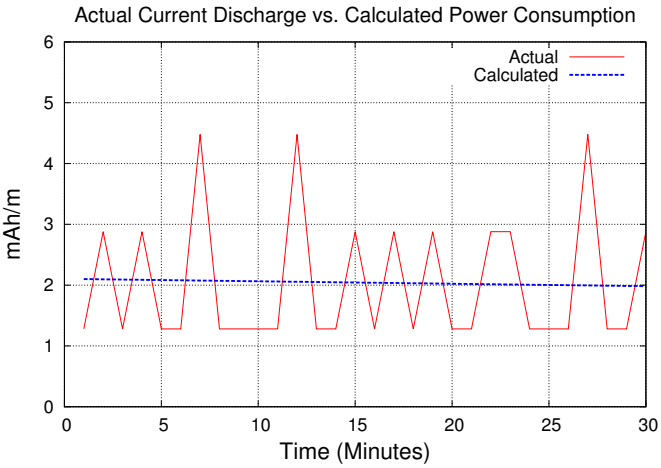


Figure 7. Current discharge for Sound Recorder : measured vs. calculated

### B. Analysis

This analysis compares the manually calculated power consumption to the consumption calculated by our power model.

Table IV shows a list of applications along with their 'actual' power usage. This value was calculated as follows:

1) The application was started and set to a state close to a normal use-case.
2) The state was maintained for 30 minutes.
3) The beginning and ending current discharge values were recorded and subtracted by-hand.

In the above procedure, the phone was left on battery mode, so that the discharge values were available. The mAh value obtained was subtracted from the baseline, in order to account for system noise.

We then used the logger application to monitor the chosen application using the tests listed in Section VII-A. We have the subsystem constants $C_i$ from a previously run regression analysis. We collected the subsystem usage data $U_i^j$ for the given application *j* and calculated the power consumed by the chosen application for every time interval.

This calculation is a product of the constants vector *C* of length *i* and the subsystem usage matrix of size $n * i$, for *n* recorded time intervals. Equation 3, discussed in Section III-B, explains the basic calculation that is represented by this matrix multiplication. Since we are running this for 30 mins, on a 1 min sampling interval, we have 30 readings. By computing the matrix multiplication, we can determine a power consumption vector *P* of size *n*, which is the power consumed for every time interval. These values are averaged and tabulated in Table IV. Further, the error between the calculated and actual $P^j$ is also tabulated.

We can further analyse the per-interval results calculated using the matrix multiplication. Fig. 7 shows the actual readings for Sound Recorder application's current discharge,

Table IV
ERROR IN CALCULATING POWER DISCHARGE FOR EACH APPLICATION

| Application | Actual $P^j$ | Calculated $P^j$ | Error |
|---|---|---|---|
| Logger | 0.32 mAh/m | 0.3082 mAh/m | 3.69 % |
| Music | 1.17333 mAh/m | 1.1882 mAh/m | 1.27 % |
| Sound Recorder | 2.0267 mAh/m | 2.0416 mAh/m | 0.74 % |
| GPSTracker | 2.0267 mAh/m | 0.3318 mAh/m | 3.68 % |

Table V
AREA UNDER THE CURVES FROM FIG. 7 FOR SOUND RECORDER

| Range | Actual Area | Calculated Area | Error |
|---|---|---|---|
| 1–30 | 58.72 | 59.21 | 0.83 % |
| 5–15 | 20.0 | 20.64 | 3.19 % |
| 20–30 | 20.0 | 20.03 | 0.17 % |
| 1–5 | 8.32 | 8.37 | 0.58 % |
| 15–20 | 10.4 | 10.17 | 2.23 % |

as collected from the operating system, plotted against those calculated using our power model. The readings for Sound Recorder in Table IV present the averaged values of the readings which were plotted.

It can be observed that the actual current discharge readings were quite bursty. This is primarily due to the fact that the battery driver in Nexus One records current discharge in multiples of 1.6 mAh only. The baseline noise of 0.32 mAh has been subtracted from all the points of the 'actual' curve plot in Fig. 7. In contrast, the 'calculated' plot purely depends on the subsystem usage measurements *U* and the subsystem constants *C*. When the power consumption *P* is calculated using our regression model, the resulting per-minute consumption is very close to a curve-fit of the data plot for the raw current charge data. This view is demonstrated by finding the area under the curves in the Fig. 7. The areas were found to be nearly identical (See Table V). The overall error between the two curves was 0.83%, and at any given time, the worst case error was 3.19%. It is to be noted however, that our model is rather more accurate than the actual current discharge value collected from the operating system, because the actual readings have a lowest precision level of 1.6 mAh.

## VIII. USE CASES

### A. Energy performance ratings

Since we are able to characterize power consumption for individual subsystems as well as applications, we can use the calculated energy consumption rates as a benchmark for rating applications. Applications can be classified in the market depending upon their amount of power consumption. Even more interesting is that we can assign power consumption ranges to different types of applications. Thereby, we can give an 'Energy Star'-like rating which is relative to the average power consumption of an application in its own class of applications.

### B. Power usage based automatic policy enforcement

The results and analysis from our model can be fed into a policy enforcement subsystem which limits the use of individual components based on a set policy. Traditional Linux systems have existing tools in order to limit processes based on CPU and memory consumption [16], [17]. On smartphones, energy can be used as the metric for such rationing. Pering et al. demonstrated, using their tool CoolSpots [11], that automatic policy enforcement based on expected power consumption is effective. Their study was focused only on wireless interfaces of mobile devices.

### C. Malware detection

If we record the calculated power consumption statistics for each application as well as the entire system, with enough data from multiple users, we will be able to perform granular anomaly detection. There is existing preliminary research for creating such a tool. Buennemeyer et al. proposed a technique to correlate Snort alerts with abnormality alerts from their battery-sensing Intrusion Detection System in order to justify the use of energy patterns as a method to identify abnormalities [18].

### D. Prediction of power consumption

We can perform power-usage and battery-life predictions using the data analyzed in this model. Research conducted by Trung et al. suggests that users were able to save energy consumption by just being aware of the amount of energy a task on the phone consumes [19]. Their tool predicts expected energy consumption for individual tasks on the device, based on surveying the usage patterns of multiple users.

By automating this process, we can also create a real-time hard deadline type of framework, where a user can check the lifetime before starting a task, for example: playing Angry Birds in the train when returning home from work. Ravi et al. presented a system to project the current state of the smartphone in order to predict the expected uptime and warn the user if the expected next battery-charging opportunity was outside this timeline [20].

## IX. CONCLUSION

Our study indicates that our measuring and power modelling approach is both accurate and has a small power footprint. This allows our approach to avoid interference with the actual measuring process. From our performance analysis of our logging facilities, we conclude that due to the use of native hooks and simplicity of the model, the power consumption of the logger is as low as 0.32 mAh/m. The computation of the subsystem constants ($C_i$) using OLS is instantaneous on an external computer and we argue that this regression is simple enough to be performed on the phone in real-time. We expect the calculations to be efficient enough to be used frequently in order to keep

the subsystem constants current and adapt according to environmental changes. We also conclude that, using the regression model, we are able to attribute power usage to individual applications with an error rate of under 4%, as presented in Table IV.

The individual subsystem results presented in Section V have their use even outside our power model. Knowing that the variation in power consumption is a function of pixel strength of the display and the exact behaviour of the Wi-Fi subsystem under various modes can help developers make better decisions for power efficiency.

## ACKNOWLEDGMENTS

We would like to thank Dr. Nelson Nazzicari for his formal explanation of the least squares regression technique with respect to our dataset.

## REFERENCES

[1] T. Mogg and B. B. Corporation, "Free mobile apps drain battery faster," 2012, [accessed 3-April-2012]. [Online]. Available: http://www.bbc.co.uk/news/technology-17431109

[2] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the energy spent inside my app? fine grained energy accounting on smartphones with eprof," in *Proceedings of the sixth conference on Computer systems*, ser. EuroSys '12. New York, NY, USA: ACM, 2012.

[3] ——, "Bootstrapping energy debugging on smartphones: a first look at energy bugs in mobile devices," in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, ser. HotNets '11. New York, NY, USA: ACM, 2011, pp. 5:1–5:6. [Online]. Available: http://doi.acm.org/10.1145/2070562.2070567

[4] J. Flinn and M. Satyanarayanan, "Powerscope: A tool for profiling the energy usage of mobile applications," in *Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*, ser. WMCSA '99. Washington, DC, USA: IEEE Computer Society, 1999, pp. 2–. [Online]. Available: http://dl.acm.org/citation.cfm?id=520551.837522

[5] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang, "Fine-grained power modeling for smartphones using system call tracing," in *Proceedings of the sixth conference on Computer systems*, ser. EuroSys '11. New York, NY, USA: ACM, 2011, pp. 153–168. [Online]. Available: http://doi.acm.org/10.1145/1966445.1966460

[6] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, ser. CODES/ISSS '10. New York, NY, USA: ACM, 2010, pp. 105–114. [Online]. Available: http://doi.acm.org/10.1145/1878961.1878982

[7] A. Shye, B. Scholbrock, and G. Memik, "Into the wild: Studying real user activity patterns to guide power optimizations for mobile architectures."

[8] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin, "Diversity in smartphone usage," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, ser. MobiSys '10. New York, NY, USA: ACM, 2010, pp. 179–194. [Online]. Available: http://doi.acm.org/10.1145/1814433.1814453

[9] M. Dong and L. Zhong, "Self-constructive high-rate system energy modeling for battery-powered mobile systems," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*, ser. MobiSys '11. New York, NY, USA: ACM, 2011, pp. 335–348. [Online]. Available: http://doi.acm.org/10.1145/1999995.2000027

[10] S. Iyer, L. Luo, R. Mayo, and P. Ranganathan, "Energy-adaptive display system designs for future mobile environments," in *Proceedings of the 1st international conference on Mobile systems, applications and services*, ser. MobiSys '03. New York, NY, USA: ACM, 2003, pp. 245–258. [Online]. Available: http://doi.acm.org/10.1145/1066116.1189045

[11] T. Pering, Y. Agarwal, R. Gupta, and R. Want, "Coolspots: reducing the power consumption of wireless mobile devices with multiple radio interfaces," in *Proceedings of the 4th international conference on Mobile systems, applications and services*, ser. MobiSys '06. New York, NY, USA: ACM, 2006, pp. 220–232. [Online]. Available: http://doi.acm.org/10.1145/1134680.1134704

[12] G. Castignani, N. Montavont, and A. Lampropulos, "Energy considerations for a wireless multi-homed environment," in *Proceedings of the 17th international conference on Energy-aware communications*, ser. EUNICE'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 181–192. [Online]. Available: http://dl.acm.org/citation.cfm?id=2040416.2040443

[13] R. Myers, *Classical and Modern Regression With Applications*, ser. Duxbury classic series. Duxbury/Thompson Learning, 1990. [Online]. Available: http://books.google.com/books?id=LOHHKQAACAAJ

[14] J. W. Eaton, "Gnu octave," 1994, [accessed 3-April-2012]. [Online]. Available: http://www.gnu.org/software/octave/

[15] G. D. Israel, "Determining sample size," *Program Evaluation and Organizational Development, IFAS, University of Florida*, 1992. [Online]. Available: http://edis.ifas.ufl.edu/pd006

[16] A. Marletta, "cpulimit," 2006, [accessed 3-April-2012]. [Online]. Available: http://cpulimit.sourceforge.net/

[17] P. Menage, R. Seth, P. Jackson, and C. Lameter, "Linux control groups," 2008, [accessed 3-April-2012]. [Online]. Available: http://www.mjmwired.net/kernel/Documentation/cgroups.txt

[18] T. K. Buennemeyer, T. M. Nelson, L. M. Clagett, J. P. Dunning, R. C. Marchany, and J. G. Tront, "Mobile device profiling and intrusion detection using smart batteries," *Hawaii International Conference on System Sciences*, vol. 0, p. 296, 2008.

[19] K. N. Truong, J. A. Kientz, T. Sohn, A. Rosenzweig, A. Fonville, and T. Smith, "The design and evaluation of a task-centered battery interface," in *Proceedings of the 12th ACM international conference on Ubiquitous computing*, ser. Ubicomp '10. New York, NY, USA: ACM, 2010, pp. 341–350. [Online]. Available: http://doi.acm.org/10.1145/1864349.1864400

[20] N. Ravi, J. Scott, L. Han, and L. Iftode, "Context-aware battery management for mobile phones," in *Proceedings of the 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications*, ser. PERCOM '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 224–233. [Online]. Available: http://dx.doi.org/10.1109/PERCOM.2008.108