Into the Wild: Studying Real User Activity Patterns to Guide Power Optimizations for Mobile Architectures

Alex Shye Benjamin Scholbrock Gokhan Memik
Northwestern University
Electrical Engineering and Computer Science Department
{shye, scholbrock, g-memik}@northwestern.edu

ABSTRACT

As the market for mobile architectures continues its rapid growth, it has become increasingly important to understand and optimize the power consumption of these battery-driven devices. While energy consumption has been heavily explored, there is one critical factor that is often overlooked – the end user. Ultimately, the energy consumption of a mobile architecture is defined by user activity. In this paper, we study mobile architectures in their natural environment – in the hands of the end user. Specifically, we develop a logger application for Android G1 mobile phones and release the logger into the wild to collect traces of real user activity. We then show how the traces can be used to characterize power consumption, and guide the development of power optimizations.

We present a regression-based power estimation model that only relies on easily-accessible measurements collected by our logger. The model accurately estimates power consumption and provides insights about the power breakdown among hardware components. We show that energy consumption widely varies depending upon the user. In addition, our results show that the screen and the CPU are the two largest power consuming components. We also study patterns in user behavior to derive power optimizations. We observe that majority of the active screen time is dominated by long screen intervals. To reduce the energy consumption during these long intervals, we implement a scheme that slowly reduces the screen brightness over time. Our results reveal that the users are happier with a system that slowly reduces the screen brightness rather than abruptly doing so, even though the two schemes settle at the same brightness. Similarly, we experiment with a scheme that slowly reduces the CPU frequency over time. We evaluate these optimizations with a user study and demonstrate 10.6% total system energy savings with a minimal impact on user satisfaction.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-based Systems**]: Real-time and embedded systems; C.0 [**Computer Systems Organization**]: General—*Modeling of computer architecture*; H.1.2 [**Models and Principles**]: User/Machine Systems—*Human Factors*

General Terms

Design, Measurement, Human Factors

MICRO'09, December 12–16, 2009, New York, NY, USA.

1. INTRODUCTION

In recent years, there has been a tremendous shift in the market for personal computing. Users are in the midst of a mass transition from stationary desktop computers to a mix of mobile architectures, e.g., PDAs, cellular phones, media players, and netbooks. Portable music players are now ubiquitous with nearly four in ten Americans owning a portable MP3 player [2]. Mobile phones have been adopted faster than any technology in history [12]. Netbooks are projected to follow a similar trend in the coming years [11]. Although mobile architectures provide the convenience of portable computation, entertainment, and communication, their utility is severely constrained by their battery life. As the demand for mobile architectures continues to grow, it will become increasingly important for architects to focus on understanding and optimizing the power consumption of these energy-constrained architectures.

Power estimation and optimization has been a popular area of research in embedded and mobile architectures for many years. Researchers have studied power at many levels, including the circuit, architecture, and system levels. However, there is one critical factor that architects have largely ignored – the end user. On a mobile architecture, the end user is the workload: mobile architectures typically run applications which interact directly with them. Execution of batch jobs and long-running services are minimized, or even disallowed, as on the iPhone [1]. As a result, the usage behavior of the end user drives execution, which in turn, determines the power consumption. Architects should treat the end user as the workload, and study trends, properties, and patterns in user activity. Without understanding these patterns, it is not possible to clearly understand the impact of any optimization on user experience or the real device power consumption.

In this paper, we describe tools and methods for studying the power consumption of real mobile architectures with respect to user activity. We develop a logger application for Android G1 mobile phones that logs user activity and sends traces back to our servers. We release the logger into the wild to collect traces of real users on real mobile devices in real environments. We then demonstrate how the traces can be used to characterize power consumption, and guide optimizations on mobile architectures.

We present an regression-based power estimation model which uses high-level system measurements to estimate power consumption. The measurements used as inputs are chosen to be representative of underlying hardware components. Furthermore, the measurements are easily accessible and can be collected by our logger (which operates entirely in user space). We develop the model using in-house power measurements and show that the power estimation model can accurately predict the power consumption by validating the model using a separate device and random workloads.

We then use our power model to characterize the power consumption of the Android G1. We first analyze a set of

synthetic testing workloads, and find that the breakdown of power consumption among hardware components varies significantly based upon the workload. Our findings motivate the need for understanding real workloads in real environments. We then analyze the traces from our 20 real users. Again, we find a large variation in the power breakdown between users. Averaged across our users, our results also indicate that the CPU and the screen are the two most power-consuming components.

Finally, we demonstrate an example of studying user activity patterns to guide the development of novel power optimizations. We study active screen behavior and observe that the majority of active screen time is dominated by a relatively small number of long active screen intervals. Thus, optimizing for long screen intervals would be profitable for reducing power consumption. Targeting these long intervals enables us to develop a novel scheme that utilizes *change blindness*. Change blindness refers to the inability of humans to notice large changes in their environments, especially if the changes occur in small increments. We implement optimizations that slowly decrease CPU frequency and screen brightness during long active screen intervals. We conduct a user study testing these schemes and show that users are more satisfied with a system that slowly reduces the screen brightness rather than abruptly doing so, even though the two schemes reach the same brightness level. Overall, our schemes save 10.6% of the phone energy consumption on average with minimal impact on user satisfaction.

Overall, we make the following contributions:

- We develop an accurate linear-regression-based power estimation model which leverages easily-accessible measurements to accurately predict the system-wide power consumption of a mobile architecture;
- We use our power estimation model to characterize the power consumption of an Android G1 mobile architecture with respect to user activity patterns;
- We demonstrate an example of developing optimizations for CPU frequency scaling and screen brightness based upon user activity patterns; and
- We utilize change blindness for power optimization during active use.

The rest of the paper proceeds as follows. In Section 2, we discuss our experimental setup. Section 3 presents our linear-regression-based power estimation model. In Section 4, we study user activity traces to derive the power breakdown on real mobile phones and identify a potential optimization direction by studying screen activity. Section 5 presents optimizations for the CPU and screen that leverages change blindness. We evaluate the optimizations in Section 6. Section 7 discusses related work and we conclude in Section 8.

2. EXPERIMENTAL SETUP

Our target mobile architecture in this paper is the HTC Dream, a smartphone developed by HTC that supports the open source Google Android mobile device platform [14]. Although we focus on a specific mobile architecture for experimentation, our contributions and findings could easily extend to other mobile architectures.

We use the G1 Android Developer Phone 1 (ADP1), a rooted and SIM-unlocked version of the HTC Dream. We use the Android OS 1.0 stock system image for the ADP1 and develop with the Android 1.0 SDK. The Android platform consists of a slightly modified 2.6.25 Linux kernel, and

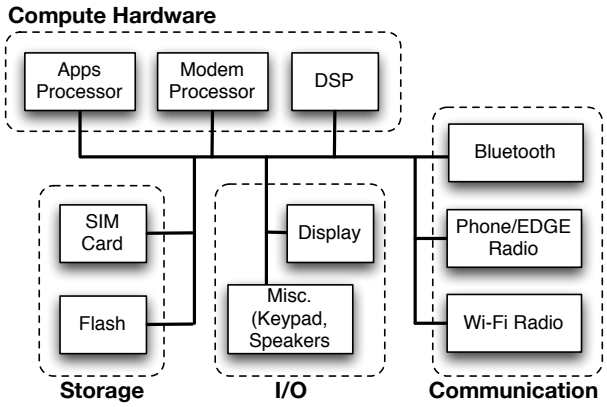


Figure 1: High-level overview of the target mobile architecture.

a general framework of C, C++, and Java code. The framework includes the Dalvik Virtual Machine (VM), a variant of Java implemented by Google. All userspace applications are Dalvik executables that run in an instance of the Dalvik VM.

A high-level diagram of the mobile architecture is shown in Figure 1. The ADP1 has a 3.2 inch HVGA 65K color capacitive touch screen, uses a Qualcomm MSM7201A chipset, and a 1150 mAh lithium-ion battery [30]. The Qualcomm MSM7201A chipset contains a 528 MHz ARM 11 apps processor, an ARM 9 modem processor, a 528 MHz ARM 11 Jazelle Java hardware acceleration, QDSP4000 and QDSP5000 high-performance digital signal processors, quadband GPRS and EDGE network, integrated Bluetooth, and Wi-fi support.

To the best of our understanding, the ARM 11 apps processor runs the Android platform and executes the applications on the device. It is rated at 528 MHz and supports dynamic frequency scaling (DFS), but is scaled down in the platform to run at 124 MHz, 246 MHz, and 384 MHz. The highest frequency of 528 MHz is not used. The ARM 9 modem processor is a separate processor that runs a proprietary operating system and is in charge of the communications of the phone. The Jazelle Java hardware acceleration processor is not used as the Android platform runs Dalvik executables which are not fully compatible.

We build our power estimation model using real power measurements. We instrument the contact between the phone and the battery and measure the current with a Fluke i30 AC/DC current clamp. We use the OS reported battery voltage as the operating battery voltage. The linear regression model is created using the R Statistical Computing Environment [24].

We develop a logger application that logs system performance metrics and user activity. The logger runs as a Dalvik executable. It does not require any special hardware or OS support, and runs on consumer HTC Dream devices, such as the T-Mobile G1 phone. The logger periodically looks for a network connection and sends the logs back to our server. All data is anonymous by the time it reaches our server.

To obtain users for our study, we publicized our project for a month on multiple university campuses, as well as to the general public. Users install the logger through the Android Market. To minimize potential bias in our data, all volunteers remain anonymous. Volunteers are notified that we do not collect any data that could be used to identify them. We also provide a complete list of collected data to maintain transparency with the users. To avoid any change in user behavior, the logger application is designed to be

HW Unit	Parameter	Description	Range (of $\beta_{i,j}$)	Coefficient (c_j)	*units*
CPU	`hi_CPU_util`	Average CPU utilization while operating at 384 MHz	0–100	3.97	$^{mW}/_{\%}$
	`med_CPU_util`	Average CPU utilization while operating at 246 MHz	0–100	2.79	$^{mW}/_{\%}$
Screen	`screen_on`	Fraction of the time interval with the screen on	0–1	150.31	mW
	`brightness`	Screen brightness	0–255	2.07	$^{mW}/_{(step)}$
Call	`call_ringing`	Fraction of the time interval where the phone is ringing	0–1	761.70	mW
	`call_off_hook`	Fraction of time interval during a phone call	0–1	389.97	mW
EDGE	`edge_has_traffic`	Fraction of time inverval where there is EDGE traffic	0–1	522.67	mW
	`edge_traffic`	Number of bytes transferred with the EDGE network during time interval	≥ 0	3.47	$^{mW}/_{byte}$
Wifi	`wifi_on`	Fraction of time interval Wifi connection is on	0–1	1.77	mW
	`wifi_has_traffic`	Fraction of time inverval where there is Wifi traffic	0–1	658.93	mW
	`wifi_traffic`	Count of bytes transferred with Wifi during interval	≥ 0	0.518	$^{mW}/_{byte}$
SD Card	`sdcard_traffic`	Number of sectors transferred to/from Micro SD card	≥ 0	0.0324	$^{mW}/_{sector}$
DSP	`music_on`	Fraction of time interval music is on	0–1	275.65	mW
System	`system_on`	Fraction of time interval phone is not idle	0–1	169.08	mW

Table 1: Parameters used for linear regression in our power estimation model.

as unintrusive as possible. It automatically starts upon installation or after the boot process, and consumes minimal system resources.

For the data in this paper, we use the logs from the 20 users who have the largest logged activity. The cumulative log data represents approximately 250 days of real user activity. To explore usage patterns when the mobile device is battery-constrained, we focus on time intervals when the battery is not charging. From all of the logs, we extract 860 time intervals where the battery is not charging, which add up to a total of 145 days of user activity.

3. POWER MODEL

We now discuss our approach to system-level power estimation for mobile architectures. We model the architecture having two distinct power states:

Active : The apps processor is operational. This occurs during active usage when the screen is on, or if a system wake lock is held to ensure that the apps processor remains on while the screen is off.

Idle : The device is in a low-power sleep mode. The apps processor is not operational but the modem processor is still active (also called "Standby" mode).

The power consumed in the Idle state is significantly lower than the Active state, and is relatively invariant under typical circumstances (measured to be around 70 mW). In contrast, power consumed in the Active state is considerably higher (300∼2000+ mW), and varies significantly by workload. Our power estimation model focuses primarily on modeling Active state power consumption.

We build our power estimation model based on high-level measurements collected for a set of the hardware units. We choose a linear regression method to build the model. Linear regression fits an output variable to a set of independent input parameters by corresponding linear coefficients.

3.1 Choosing Parameters

Table 1 lists the parameters selected for the power estimation model, including the final coefficients used in our power estimation model. We model most of the hardware components on the ADP1, including the CPU, screen, calls, EDGE/Wi-fi network, SD card, and the DSP processor.

CPU : The CPU refers to the apps processor and supports DFS between three frequencies, as described in Section 2. The lowest frequency is never used on consumer versions of the phone, and is too slow to perform basic tasks. Thus, only the high (384 MHz) and medium (246 MHz) frequencies are considered in our model.

Screen : The screen parameters include a constant offset indicating whether the screen is on and a second parameter to model the effect of the screen brightness.

Call : We model the power during phone calls by measuring the time spent ringing and the duration of the phone calls.

EDGE : The EDGE network power consumption parameters consider whether there is any traffic and the number of bytes of traffic during a particular time interval.

Wi-fi : The Wi-fi power consumption is modeled similar to the EDGE network but also includes a parameter for whether Wi-fi connectivity exists.

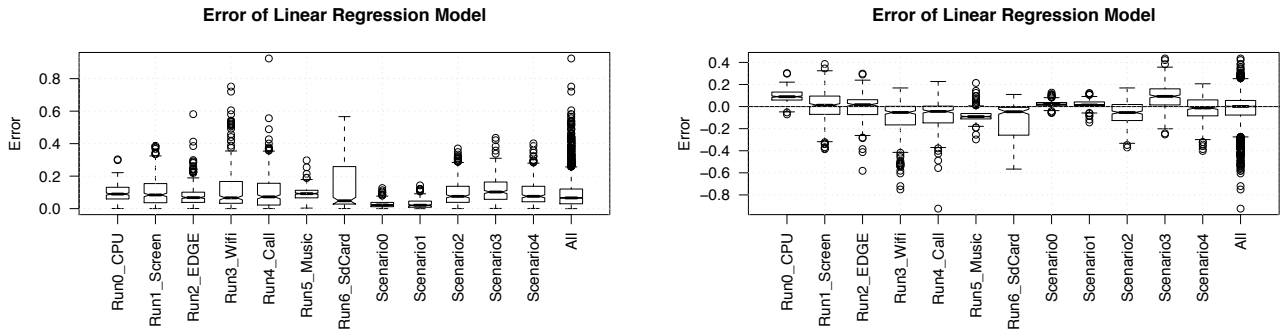
SD Card : We consider the number of sectors transferred per time interval.

DSP : We model the DSP by checking an internal variable within the Android SDK for whether there is a multimedia file playing. This variable is on during music and video playback.

System : The power that is not accounted for with the hardware components listed above are put into a catch-all variable that we simply refer to as the miscellaneous **System** power in Table 1. The System power corresponds to the constant y-intercept in a linear regression model, or k, as it will be described in the following section.

3.2 Building the Estimation Model

To develop our model, we use real-time measurements of our target phone. The overall idea is to find the relationship between the collected system statistics and the power consumption. Hence, the input to our model is the statistics collected from the phone. The output is the total power consumption. During training, we provide the measured power consumption and use the R-tool to build the linear regression model. Specifically, during training, we have performed a series of tasks to stress different components of the hardware. During these tests, we (1) measure the real-time power consumption of the phone and (2) collect statistics about the chosen parameters described in the previous section. The raw data samples are collected every second for the synchronous data (e.g., CPU utilization). We sample at 1 Hz to reduce perturbation on the system and minimize the execution and power consumption of the logger.



(a) Absolute relative error.

(b) Relative error.

Figure 2: Error of logger when building the power estimation model on one ADP1 and validating with logs from another ADP1 device.

During training, the collected statistics and the measured power consumption levels are fed into the R-tool to find the regression coefficient c_j for each parameter. Once a model is generated, one can predict the power consumption by simply providing the statistics for the selected parameters.

As we will discuss in Section 3.3, this approach results in a highly accurate power model. In addition, it can be used to estimate the power consumption of each individual hardware component. If a single measurement (e.g., the value for screen brightness) in sample i is $\beta_{i,j}$, then the power $p_{i,j}$ contributed by the corresponding hardware component for the parameter coefficient c_j is:

$$p_{i,j} = \beta_{i,j} \cdot c_j \tag{1}$$

Power not attributable to any available measurement is aggregated into a constant offset k. The total system power P_i for sample i with n available measurements is then modeled with the sum of these n power values:

$$P_i = k + (p_{i,0} + p_{i,1} + \ldots + p_{i,n}) \tag{2}$$

$$= k + ((\beta_{i,0} \cdot c_0) + (\beta_{i,1} \cdot c_1) + \ldots + (\beta_{i,n} \cdot c_n)) \tag{3}$$

Allowing $x_i = (\beta_{i,0}, \beta_{i,1}, \cdots \beta_{i,n})$ for each sample i and $c = (c_0, c_1, \cdots c_n)$ reduces this to:

$$P_i = k + x_i \cdot c \tag{4}$$

Taken across m samples, this model takes the form:

$$\begin{pmatrix} P_0 \\ P_1 \\ \vdots \\ P_m \end{pmatrix} = k \cdot \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} + \begin{bmatrix} \beta_{0,0} & \cdots & \beta_{0,n} \\ \beta_{1,0} & \cdots & \beta_{1,n} \\ \vdots & \ddots & \vdots \\ \beta_{m,0} & \cdots & \beta_{m,n} \end{bmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix} \tag{5}$$

Letting $P = \begin{pmatrix} P_0 \\ \vdots \\ P_m \end{pmatrix}$, $X = \begin{bmatrix} \beta_{0,0} & \cdots & \beta_{0,n} \\ \beta_{1,0} & \cdots & \beta_{1,n} \\ \vdots & \ddots & \vdots \\ \beta_{m,0} & \cdots & \beta_{m,n} \end{bmatrix}$ and $e = \begin{pmatrix} 1 & \cdots & 1 \end{pmatrix}^T$ yields:

$$P = k \cdot e + Xc \tag{6}$$

Once values for k and c have been determined, any sample of system measurements x_i may be used to approximate the power consumed by the whole system P_i at the time of the sample with Equation 4, and the power contributed by each hardware component during the sample may be approximated using its corresponding measurement in Equation 1.

Additionally, the total energy E consumed by the system across a set of such samples X with sampling period t_s may be approximated by the sum:

$$E = t_s \cdot sum\,(P) = \sum_{i=0}^{m} t_s \cdot P_i = t_s \sum_{i=0}^{m} (k + x_i \cdot c) \tag{7}$$

When the phone is in the Idle state, a constant power value (p_{idle} ≈68.3 mW) is used to approximate power consumption. The `system_on` ratio from Table 1 indicates the portion of time the system is in the Active state as a ratio between 0 and 1. Thus, when a log contains both the Active and Idle states, power consumption for a single sample i is modeled as:

$$\texttt{Power}_i = \texttt{system\_on} \cdot (P_i) + (1 - \texttt{system\_on}) \cdot (p_{idle}) \tag{8}$$

When the system is in Active state, the power is approximated by the linear regression model P_i; in Idle state, p_{idle} is used as the approximation. In the linear regression model for Active power, k represents the coefficient for `system_on`.

3.3 Validating the Power Model

We approximate the values of offset k and the coefficient vector c using a set of sampled system measurements X with experimentally measured power consumption $\hat{P}$. Samples are taken from varying workloads to cover the spectrum of possible use scenarios. From Equation 6 in Section 3.2, this produces the linear equation $\hat{P} = k \cdot e + Xc$, solving for an approximation of k and c. The approximations are shown in Table 1 (k is represented by the coefficient for `system_on`; other values in the column jointly form the vector c).

To demonstrate the accuracy of the model, we collect additional logs of system measurements and power consumption. In addition, we collect this set of logs on a separate ADP1 device to ensure that our power estimation model generalizes beyond the device used for training the model. We collect two types of logs. The first type targets specific hardware components of the phone. We name these logs `Run`i`_Unit`. For example, `Run1_CPU` corresponds to a log with varying CPU utilization. These logs are used for training our power model. The second type of log corresponds to a scenario, or a mix of usage behavior, that stresses multiple hardware components. We name these logs `Scenario`i. As an example, `Scenario2` simulates a user listening to music while browsing the web, and then answering a phone call. These logs are not used during training and used to analyze the accuracy of our model for workloads that are not part of the training set. Each log is approximately 5 minutes long.

We use this set of logs from a separate mobile device to approximate the error of our power estimation model. Equa-

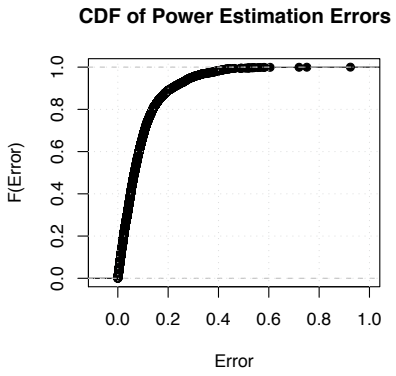


Figure 3: Cumulative distribution of power estimation error.

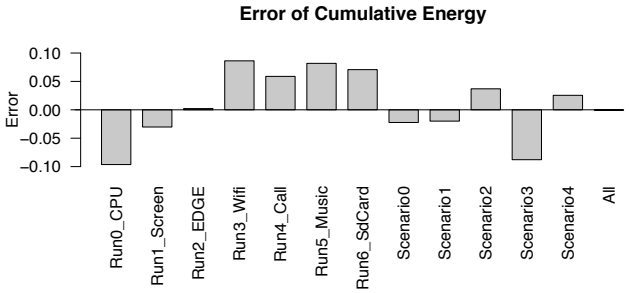


Figure 4: Cumulative total energy error.

tion (4) in Section 3.2 provides a power estimation for each sample i. The error considered is the percent absolute relative error ($error_i$) and the percent relative error ($error_j$):

$$error_i = \left|\frac{actual - estimated}{actual}\right| = 100 \cdot \left|\frac{\hat{P}_i - P_i}{\hat{P}_i}\right| \% \quad (9)$$

$$error_j = \frac{actual - estimated}{actual} = 100 \cdot \frac{\hat{P}_i - P_i}{\hat{P}_i} \% \quad (10)$$

Figure 2 presents the range of errors for each of the logs collected, including the logs used in training and the scenario-based logs used for validation. In the figures, the median error for each set is a bold line, the boxes extend to 25% and 75% quartiles, the whiskers extend to the most extreme sample point within 1.5× the interquartile range, and outliers are independent points. Figure 2(a) shows the absolute relative error and Figure 2(b) shows the relative error.

Our results indicate that the power estimation model accurately predicts the system-level power consumption of the logs, even though a separate mobile device is used. The median absolute relative error across all of the samples is 6.6%. The median relative error rate is $< 0.1\%$. The hardware-specific logs demonstrate the accuracy of predicting the power consumption of specific hardware components. In general, the model predicts the CPU, EDGE, and music with a low median error rate, and a low variance in the error rates. The error rates in the screen, Wi-fi, phone call, and SD card show a higher amount of variance; but their power consumption can still be predicted accurately with a low median error rate. The scenario-based logs demonstrate that our power estimation model also extends to workloads that combine multiple hardware components, with median errors similar to the hardware-specific logs. Figure 3 shows the cumulative distribution of the sample errors. Each (x, y) point represents the ratio of samples (y) at or below a particular absolute relative error (x). 65% of the individual samples are approximated by the model to within 10% absolute relative error, and 90% of the samples are within 20%.

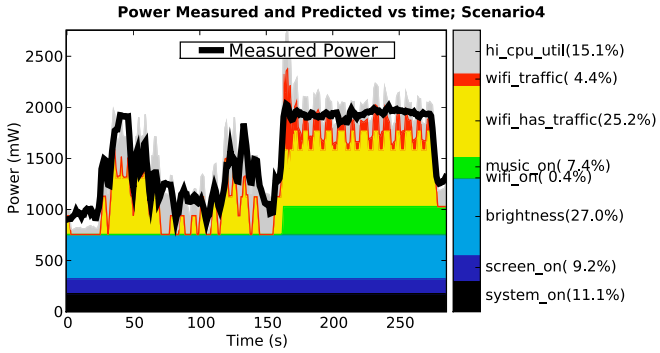


Figure 5: Power consumption timeline.

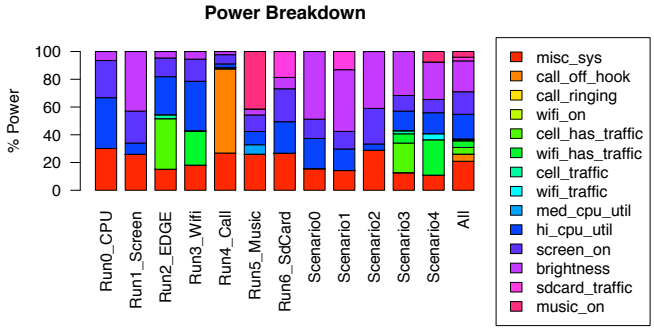


Figure 6: Power consumption breakdown for traces that stress specific hardware units.

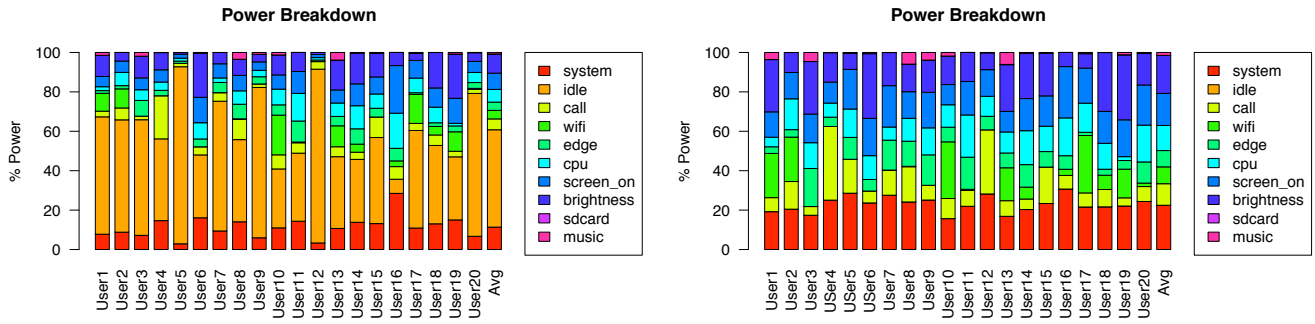
The errors shown in Figure 2 are for estimating the average power consumption of individual 4 second time windows. However, it does not provide any indication of total energy estimation across an entire trace. In Figure 4, we present the error when comparing the total energy between the power readings, and the results of our power estimation model. Over all of the logs, we achieve $< 0.1\%$ mean error.

3.4 Per-Component Power Consumption

With an accurate power estimation model, the combined system power P_i may easily be approximated for any sample i. Furthermore, since this approximation is a sum of smaller power components $p_{i,j}$ each corresponding to individual hardware units, the power contribution of an individual unit in the model may also be approximated, using Equation 1 in Section 3.2.

Figure 5 shows an example of the estimated and actual power over time for `Scenario4`. Each colored region represents the power $p_{i,j}$ attributed to a particular measurement, as labeled on the right of the plot. The total system power approximation at any point in time is represented by the top of the shaded regions. Measured system power is overlaid as a bold line. In this scenario, the user is surfing the Internet, then activates streaming media at 160 seconds. Our model estimates a total system power that closely tracks the actual measured power.

The same approach for per-component power approximation is applied to each of the logs used for the validation of our power estimation model. The estimated power from each product term in the linear regression model is accumulated and shown in Figure 6. The x-axis represents each of the logs, and each of the stacked bars corresponds to the power contribution from a specific parameter in our linear-



(a) Power breakdown including idle time.

(b) Power breakdown excluding idle time.

Figure 7: Power consumption breakdown from real user traces.

regression-based power estimation model. The y-axis represents the percent of total power for the particular log. Test logs stressing particular components, such as EDGE or Wi-fi communications, have larger portions of power attributed to corresponding measurements.

The power breakdown for each of the test logs indicates that the relative power contribution per-component may vary drastically based upon the workload. For example, during a phone call (shown in `Run4_Call`), over 50% of the total system power is consumed by the call. If only music is playing, and the screen is off (shown in `Run5_Music`), the DSP consumes significant power. The power breakdown is also dependent upon the system settings. For example, in `Scenario0`, the screen is at the highest brightness the entire time and dominates the power consumption of the system.

To better improve power consumption of any mobile platform, optimizations must target components with significant relative power consumption. However, as Figure 6 demonstrates, the per-component power breakdown widely varies with respect to the workload. Thus, it is important for architects to use representative workloads to characterize power consumption on mobile architectures. Such workloads should reflect the real user activity to correctly estimate the effect of any optimization.

Overall, our results show that (1) our high-level power estimation model can accurately predict the power consumption of the total system, (2) the power model can be used to derive a power breakdown of the total system, and (3) the power breakdown of a system is highly dependent upon the workload running on the mobile architecture.

4. STUDYING THE USER FOR GUIDING OPTIMIZATIONS

In this section, we explore the real user activity logs uploaded onto our server. We apply the power estimation model developed in Section 3 to characterize the power breakdown of mobile phones in the wild. We then present a study of active screen intervals, which suggest a potential power optimization for long screen intervals.

4.1 Characterizing Real User Workloads

As described in Section 3.4 (and shown in Figure 6), the workload of a mobile architecture has a large effect on its power consumption; the hardware components that dominate power consumption vary drastically depending upon the workload. Since the user determines the workload for a mobile architecture, we must study real user behavior to understand the actual power consumption of mobile architectures in real environments. To this end, we have collected logs from users who have downloaded our logging application from Android Market, as described in Section 2. The logs contain the activity of 20 users, each for a duration exceeding a week, and accounting for approximately 250 days of user activity.

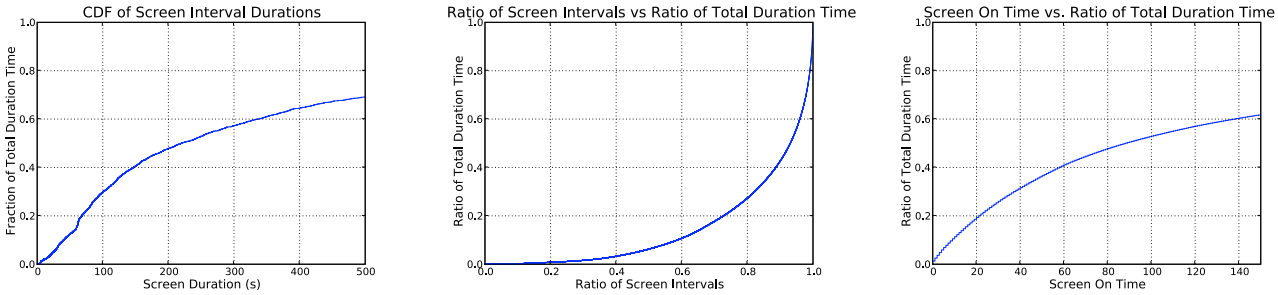
The power breakdown from each of the user logs is shown in Figure 7. Figure 7(a) shows the power breakdown including the estimated power contribution of the Idle state. To provide a clear breakdown of the power in the Active state, Figure 7(b) shows the same power breakdown excluding the samples from the Idle state. The x-axis represents each of the users. The product terms for each of the hardware components are combined for readability. The only exception is the screen, which is still shown separately as `screen_on` and `brightness`. We show both because the screen contributes heavily to the power breakdown of the system, and also because one of our optimizations (described later in Section 5) specifically targets reducing the screen brightness.

When examining the power consumption of the Idle state in Figure 7(a), two points are apparent. First, the power consumed during the Idle state can contribute to a significant fraction of the total system power consumption. The power consumed in the Idle state accounts for 49.3% of the total system power when averaging across all of the users. Second, the fraction of total power consumed during the Idle state varies significantly across the users. At the extremes, the power consumption of Idle states contribute to 89.9% of the total power for User 5, but only 7.17% for User 16. This indicates that there is considerable variation in the usage patterns of mobile architectures across individual users.

When isolating the power consumption during the Active state (shown in Figure 7(b)), we again notice a large variation in the activity among all 20 users. For example, the power breakdown for User 4 and User 12 is dominated by the phone calls. User 6 and User 19 have their screen brightness set high, and thus, the brightness dominates their power breakdown. In addition, there is varying activity with regard to EDGE network usage versus Wi-fi network usage.

Overall, during Active usage time, two hardware components dominate the power consumption when averaging across all users: the screen and the CPU. The screen largely dominates the Active power breakdown and consumes 35.5% of the Active power; 19.2% due to the screen brightness and 16.3% due to the screen being on. The CPU accounts for 12.7% of the total Active power.

Although the Idle state may sometimes dominate the total system power, in this paper, we primarily focus on the power during the Active state. There are three reason to be concerned with the Active state. First, the power consumed during the Idle state ($\approx$ 68 mW) is significantly lower than the power that can be consumed in the Active state (up to



(a) CDF of screen interval durations. (b) Percentage of screen time relative to percentage of screen durations. (c) Ratio of Total Screen Duration Constituted by Screen Interval

Figure 8: Screen durations based upon user activity.

2000 mW when listening to music and using Wi-fi as shown in Figure 5). Second, the Active state contributes highly to the user experience since the user is actively engaged during the Active state. Any application that requires the apps processor would require the device to wake up and exit Idle mode. Finally, the Active state still accounts for large fraction of the power consumed, accounting for 50.7% of the total system power.

4.2 Screen Usage of Real Users

Because the screen is the primary output device for interacting with the end user, the screen is a good indicator of user activity patterns. In addition, as we have shown in the previous section, it is the highest power consuming component on the device. We parse all of the user activity logs to extract *screen intervals*. A screen interval is a continuous block of time where the screen is on. The *duration* of a screen interval refers to the length of time that corresponds to the screen interval. The *total duration time* refers to the sum of durations for all screen intervals. From these intervals, we extract 9678 screen durations from our database, which accounts for 8.8 days worth of cumulative active "screen on" time.

Figure 8(a) shows the cumulative fraction of the total duration for screen intervals up to 500 seconds. We see that screen intervals of 100 seconds or more constitute roughly 70% of the total screen duration (equivalently, as shown in Figure 8(a), 30% of the total screen duration is contributed by intervals shorter than 100 seconds). Figure 8(b) shows the cumulative distribution function (CDF) relating the total duration time to percentage of screen intervals. In other words, it shows the percentage of total duration time (shown on the y-axis) accounted for by the fraction of screen intervals with the shortest durations (x-axis). Figure 8(c) shows the ratio of *total* duration time accounted for by intervals shorter than a specific screen duration time, up to 150 seconds. This means that it shows the fraction of total duration if the first X seconds of each screen interval is considered.

Studying the screen intervals indicates that the total duration time is dominated by a relatively small percentage of long screen intervals. If we observe the 40 second mark on the x-axis of Figure 8(b) and Figure 8(c), we see that the first 40 seconds of all screen intervals only accounts for 31% of the total duration time, but accounts for over 80% of all screen intervals. This means that if we have an optimization that saves power after 40 seconds of screen time, it would affect 69% of the total screen duration time, and only take effect in about 17% of the screen intervals. Based upon these observations, we conclude that it would be profitable to optimize for power during the long screen intervals. In the next section, we describe such an optimization.

5. USER-AWARE OPTIMIZATIONS

As described in the Section 4, surprisingly, a few long screen intervals dominate the overall screen duration time. In addition, the power consumption during Active time is dominated by the screen and the CPU. To reduce the power consumption during these long intervals, we devise a scheme that reduces the brightness. Instead of simply dropping the brightness abruptly, we utilize *change blindness*, which is described in the next section. We also devise a similar scheme to control the CPU frequency.

5.1 Change Blindness

Researchers in human psychology and perception have revealed an inability for humans to detect large changes in their surrounding environment. One commonly-known study involves a video that prompts the viewer to count the number of times a basketball is passed. Halfway through the video, a man in a gorilla suit walks into the middle of the group of basketball players, thumps its chest, and then walks away. Surprisingly, the majority of viewers do not remember seeing a man in a gorilla suit, even though the concept is absurd and is in clear view in the middle of the video [27]. *Change blindness* refers to this inability for humans to detect large changes in their environment. The gorilla-suit study refers to change blindness of dynamic events, and occurs because although a human will view the entire video, their attention dictates the visual data that is processed. There have also been studies exploring change blindness in the case of gradual changes. Change blindness in the presence of gradual changes is more surprising as humans will miss significant changes without being distracted or disrupted. One study demonstrates change blindness as objects within images are removed from a picture, or as the color of objects are slowly changed [28]. Another demonstrates change blindness as facial expressions are slowly changed in a picture.

We aim to utilize change blindness to reduce the power consumption of the device without causing any dissatisfaction to the users. Specifically, we devise schemes that reduce the screen brightness and CPU frequency slowly to save power. We compare these schemes to alternatives where the brightness and frequency are abruptly reduced and show that change blindness can indeed be utilized to save power consumption while minimizing the user dissatisfaction. We describe these schemes in the following section. To the best of our knowledge, this is the first study analyzing change blindness in the context of computer performance.

5.2 CPU Optimization

Existing DFS. The default system image used on the HTC Dream platform supports dynamic frequency scaling (DFS) on the ARM 11 apps processor, but uses a naïve DFS al-

Algorithm 1: `ondemand` DFS algorithm.

```
1  procedure ONDEMAND-DFS(cpu_util)
2    if cpu_util ≥ UP_THRESHOLD then
3      SET-FREQUENCY(highest frequency)
4    else if cpu_util ≤ DOWN_THRESHOLD then
5      requested_freq ← frequency that maintains a
       utilization of at least UP_THRESHOLD−10%
6      SET-FREQUENCY(requested_freq)
7    return
8
9  procedure SET-FREQUENCY(freq)
10   if powersave_bias = 0.0 then
11     Set CPU frequency to freq
12   else
13     Alter CPU frequency dynamically to maintain an
       effective frequency of:
14     freq × ((1000 - powersave_bias) * 0.001)
15   return
16
```

gorithm based upon the screen[1]. If the apps processor is active and the screen is on, the processor is set to the highest frequency (384 MHz). If the apps processor is active and the screen is off, the processor is set to the middle frequency (246 MHz).

`ondemand` governor. A commonly used DFS scheme on desktop/server environments is the Linux `ondemand` DFS governor. The general algorithm is shown in Algorithm 1. At a high-level, the `ondemand` makes decisions based upon the CPU utilization. If the utilization is above a `UP_THRESHOLD`, it raises the CPU to the highest frequency. If the utilization is below a `DOWN_THRESHOLD`, it calculates the frequency that would maintain the utilization below `UP_THRESHOLD`, and sets the frequency to that level. By setting the CPU frequency based upon CPU utilization, the `ondemand` governor saves power by reducing the frequency during times of low CPU utilization. We tune a knob within the `ondemand` governor called the `powersave_bias`, which is typically set to 0. The `powersave_bias` is a value between 0 and 1000 that specifies percentage with which to decrease the effective frequency of the CPU. `powersave_bias` increases in increments of 0.1%. A value of 0 indicates that the frequency should not be reduced at all. A value of 1000 indicates that the frequency should always be reduced by 100%, effectively reducing the CPU to its lowest frequency. If the `powersave_bias` indicates that the CPU frequency should be set to a frequency between two processor-supported frequencies, the `ondemand` governor will dynamically switch between the frequencies to simulate the frequency required.

Our DFS scheme: We use the `ondemand` governor and tune the `powersave_bias` knob leveraging change blindness for long screen intervals. Our DFS scheme hooks into the screen events. Every four seconds, we increase the `powersave_bias` in increments of 30 (decrease effective frequency by 3%), until a maximum limit of 300 is reached. If the screen is turned off, the `powersave_bias` is reset back to 0. Thus, it reaches 70% of the frequency requested by `ondemand` within 40 seconds.

5.3 Screen Optimization

We implement a screen optimization to leverage change blindness that is similar to our CPU optimization. Again, we hook into the screen on and off events. We keep track of the user-set screen brightness. When the screen turns on, we set a timer for 3 seconds. Every 3 seconds, we decrease the brightness of the screen by 7 units (out of a maximum brightness of 255). We continue until the brightness reaches 60% of the user-set screen brightness and then stop. When the screen is turned off, we set the brightness back to the regular user-set screen brightness.

The idea in this scheme is to utilize two previous observations. First, since we slowly reduce the screen brightness, we will not reduce the power consumption on small screen intervals. However, as we have shown in the previous section, long screen intervals dominate the total screen duration, hence our optimization should still be able to save considerable fraction of the overall screen power consumption. Second, since our scheme reduces the screen brightness slowly, we expect that the users will be less likely to distinguish the change when compared to a sudden decrease in the screen brightness. Our experiments, described in Section 6, confirm that both of these goals are achieved.

6. EXPERIMENTAL RESULTS

We now evaluate our optimizations described in Section 5. We refer to the two optimizations as *Screen Ramp* and *CPU Ramp*, for the screen and CPU optimizations, respectively. To test the change blindness hypothesis, we also introduce two more control schemes: *Screen Drop* and *CPU Drop*. Both of the *Drop* schemes wait 30 seconds before dropping to the respective minimum threshold levels of each of the change-blindness-inspired optimizations. In other words, the *Drop* and *Ramp* schemes eventually settle at the same brightness/frequency. However, the *Ramp* schemes slowly reach this destination, whereas the *Drop* schemes wait at the high brightness/frequency for the initial 30 seconds, before adjusting sharply to the final level.

We first evaluate the potential power savings of the optimization schemes by emulating the optimizations on the user logs. We then conduct a user study to assess user acceptance of our optimization schemes and to test our hypothesis on whether change blindness can be leveraged to optimize long screen intervals.

6.1 Power Savings

We approximate the power savings for each of our schemes by emulating the optimizations on user activity logs. To estimate the power savings of the screen optimizations, we adjust the brightness measurements in the logs to reflect the *Screen Drop* and *Screen Ramp* optimizations. Then, these new values are fed into the power model to generate the power consumption of the alternatives. To estimate the power savings of the CPU optimization, we first perform an estimation of the `ondemand` governor. If the CPU utilization is below 20%, we assume that `ondemand` would set the frequency to the lower level. We then simulate our ramp-down mechanism on top of the `ondemand` scheme by multiplying the CPU product terms by a fraction that decreases in similar to *CPU Ramp* or *CPU Drop*. The newly-generated logs are processed with our power model to find the power consumption of *CPU Ramp*, *CPU Drop*, and `ondemand`.

Figure 9 shows the total system power savings when compared to the base scheme for each of the studied optimizations. On average, *CPU Ramp* saves 4.9% of the total system power. This corresponds to a 22.8% power savings when considering only the total CPU power. Of this, we estimate that 10.5% of the savings can be attributed to the base `ondemand` DFS governor, and the other 12.3% power savings is due to ramping the CPU frequency with the `pow-`

[1]We have not found a confirmed description of this DFS scheme in any documentation on the HTC Dream, but have discovered this DFS behavior through our own experience with the device.

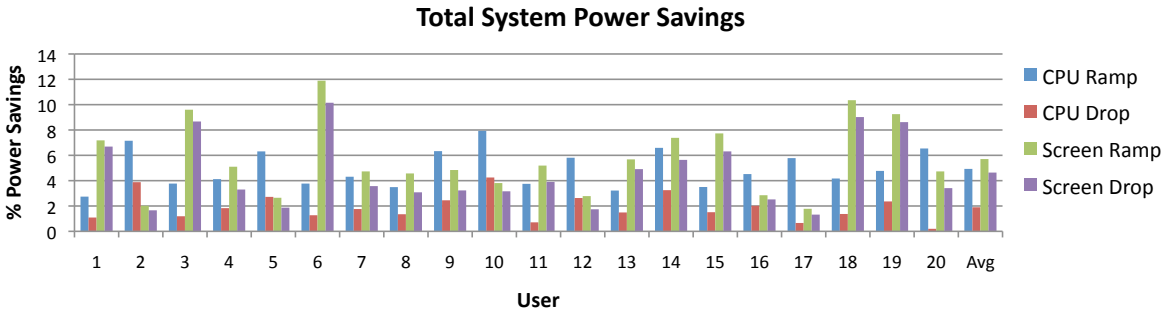


Figure 9: Total system power savings for each of the optimizations as estimated by our power model.

`ersave_bias`. The *CPU Drop*, on the other hand, achieves a total system power savings of 1.9%. When we compare the *CPU Ramp* to *CPU Drop*, we see that our *CPU Ramp* scheme achieves higher power savings. The reason for this result is the 30 second wait period of the *CPU Drop* scheme; while the *CPU Ramp* almost immediately starts reducing the CPU frequency, the *CPU Drop* scheme remains at the high frequency for 30 seconds, which causes a higher power consumption level. *Screen Ramp* saves 5.7% of total system power over all of the 20 users, and saves 19.1% of the total screen brightness power. With *Screen Drop*, 4.6% of the total system power is conserved. Similar to the CPU schemes, when we compare the *Screen Ramp* to *Screen Drop*, we see that *Screen Ramp* achieves a higher power reduction level.

6.2 Impact on User Satisfaction

To evaluate the impact of our power saving techniques on the individual user satisfaction, we conduct another user study with 20 users. The user study involves three applications:

- **Web browsing:** Surfing Wikipedia with the web browser on the phone.
- **Game:** The BreakTheBricks game where the user moves a paddle on the bottom of the screen to bounce a ball and break a pattern of bricks.
- **Video:** The user watches a video with the PlayVideo application.

For each application, we perform six runs consisting of (1) *CPU Ramp*, (2) *CPU Drop*, (3) *Screen Ramp*, (4) *Screen Drop*, (5) *Ondemand*, and (6) the *Control*. The *Control* scheme is the default CPU and screen manager of the commercial phone. The order of runs are randomized so that the particular order is blind to the user as well as the proctor of the user study. After each run, we ask the user for a verbal user satisfaction rating ranging from 1 (Not Satisfied) to 5 (Satisfied).

Figure 10 shows the results of our user study for three applications. The three graphs show the user satisfaction ratings for each of the runs for the 20 users. Each of the graphs shows a set of clustered bars, each bar corresponding to the user satisfaction rating for a single run. At first glance, the average user satisfaction ratings look very similar for both Web Browsing and the Video, but they differ for the Game. For an in-depth analysis, we perform a paired t-test analysis for each application, comparing the set of user satisfaction ratings for each of the optimizations, to the set of user satisfaction ratings for the *Control* run. The paired t-test shows that there are five comparisons against the *Control* scheme where there is a statistically significant difference. In all other cases, there were no statistically significant changes between the *Control* and the studied schemes. Among the five cases that show difference, the four are in the Game application; all the four studied techniques exhibit reduced user satisfaction when compared the *Control*. The fifth comparison that differs is the *Screen Drop* for Web Browsing. Overall, when comparing the different schemes, *Screen Drop* was statistically different from the *Control* run for the Game and Web applications, and the other schemes (barring the *Ondemand* scheme) differed from the *Control* run only on the Game application.

6.3 User Feedback and Acceptance

At the end the study, we debriefed each user by informing them of the purpose of the experiment. We discussed the power characterization study, that the CPU and screen tended to dominate the power consumption, and introduced our different power saving schemes to them. Afterwards, we questioned each user about their opinions on our various power saving schemes for the screen and CPU.

When compiling notes on the discussions with the users, we recognized two general trends:

- Most users determine their user satisfaction based upon how smoothly the computer responds to their input. 9 of the 20 users let us know they rated runs poorly when there were pauses, or the screen became jumpy/jittery. Our schemes performed the worst on the Game because any glitch would affect the smoothness of the bouncing ball and would be immediately noticeable. A result of this is that the rate of change for the CPU frequency does not matter – once the application becomes jittery, user satisfaction decreases. This trend is another reason why we did not observe a difference between the *CPU Ramp* and *CPU Drop* schemes; both of these schemes cause jitters after a certain CPU level is reached and regardless of how slowly we reduce the frequency, the glitches are noticeable. Hence the users provided the same level of satisfaction for these two alternatives. However, we must note that the *CPU_Ramp* achieves a higher power saving when compared to *CPU_Drop*.
- Change blindness can be leveraged for the screen. 8 out of the 20 users noticed the drop in screen brightness during *Screen Drop* experiments. Only one of the users noticed the screen slowly dimming during *Screen Ramp*. In fact, almost all of the users were surprised when we told them that the screen was being slowly dimmed. As a result of this, we also observe that the users were less satisfied with the *Screen Drop* on the Web application, whereas they showed the same satisfaction level with the *Control* and *Screen Ramp* schemes.

At the end of the user study, we also asked the users whether they would turn a combination of these schemes on

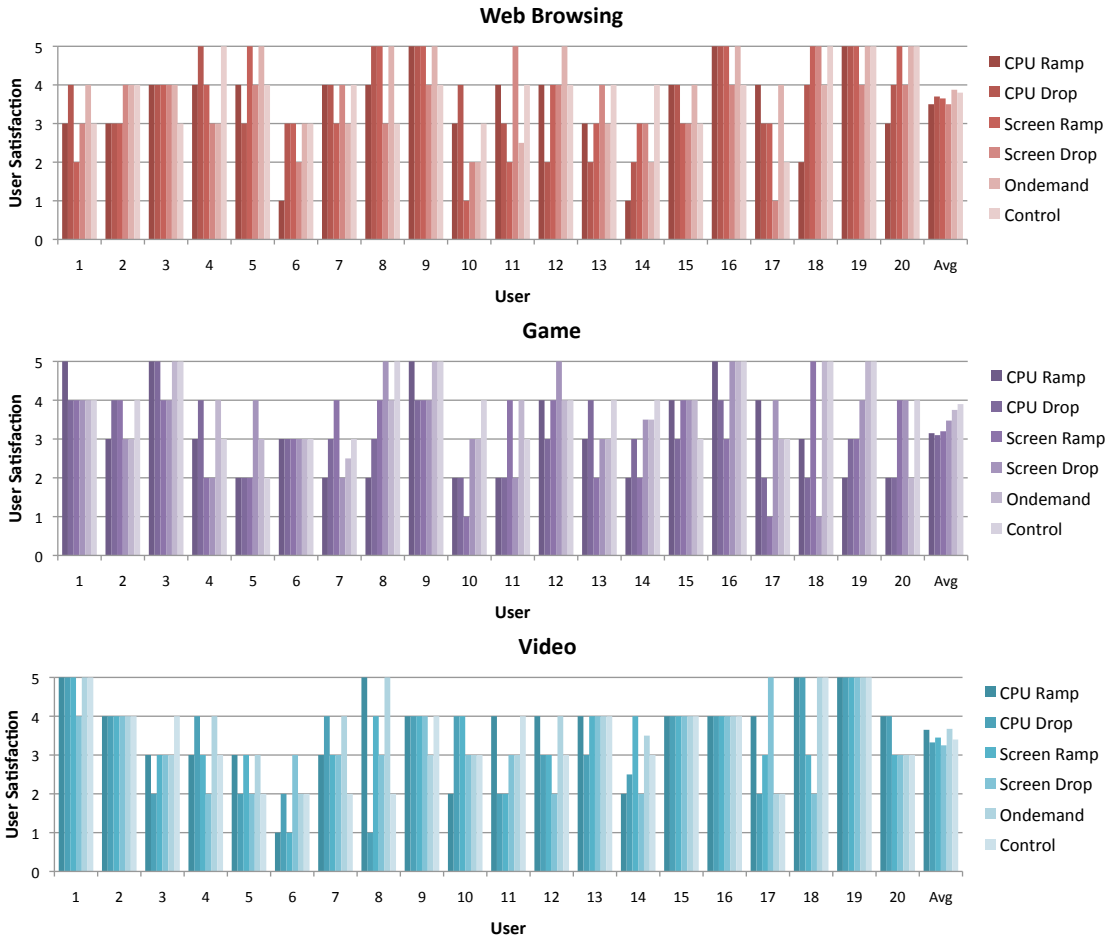


Figure 10: Reported user satisfaction for the (a) Web Browsing, (b) Game, and (c) Video applications.

if they had a tool to control them and knew they would save about 10% of their battery life. Out of the 20 users, 15 said that they would use these optimizations, 1 was apathetic, and 4 of the users would not use the optimizations. Out of the 15 that responded with a yes, 5 of them expressed a desire for application-dependent optimization. For example, while they were fine with the dimming for the web application, they preferred a brighter screen for the video.

In summary, our results show that we can achieve significant reduction in power consumption by considering user activity patterns. Our *Screen_Ramp* and *CPU_Ramp* schemes reduce the power consumption by 5.7% and 4.9%, respectively, achieving a combined power saving of 10.6%. We also show successfully demonstrate power optimizations based upon indiscernible changes on the system parameters.

7. RELATED WORK

Power modeling and estimation has been heavily studied from various angles. Wattch estimates microprocessor power consumption using low-level architectural features [7]. Several researchers use performance counters to estimate the power consumption of both high-performance and embedded microprocessors [4, 5, 9, 18, 19]. Gurun uses performance counters and communication measurements to estimate power consumption on an iPaq [16]. Cignetti uses power measurements to derive a power breakdown for Palm devices [8]. Tan uses function-level power models for software-implemented power estimation [29]. The power consumption of the operating system has been explored for high-performance [20] and embedded platforms [3, 10]. We differ from prior art by developing a software-implemented, system-level power model that uses easily-accessible measurements and does not require specialized hardware (e.g., hardware performance counters) or software (e.g., hooks into the operating system).

SoftWatt uses simulation to understand the power consumption of the processor, memory, and disk on a high-performance architecture [15]. Mahesri measures the power breakdown on a laptop and discovers that the hardware components which dominate power consumption change depending upon the workload [21]. We use our power estimation model and traces to estimate the power breakdown of mobile architectures used by real users in real environments.

Recent studies incorporate the user into the architecture evaluation and optimization process [25, 26]. PICSEL [22] is the most related to our work and takes user perception into account for controlling CPU frequency. Their work primarily focuses on CPU frequency scaling on desktop and laptop machines. In contrast, our work targets mobile architectures and leverages user activity patterns for optimization.

Researchers have studied screen optimizations that would be enabled with OLED display technology. They explore altering the user interface to dim certain parts of screen to save considerable power, and do a user acceptance study [6, 17]. Our work operates on existing screen technology and leverages change blindness with gradual changes to save power.

Phillips studies user activity for predicting when to sleep for wireless mobile devices [23]. MyExperience [13] gathers traces from user phones in the wild, similar to our work, but uses the traces to study high-level user actions. We study user activity patterns to understand system performance and for saving power on mobile architectures.

8. CONCLUSION

In this paper, we have studied mobile architectures in their natural environment – in the hands of the end user. We present tools and methods for collecting and analyzing logs of real activity patterns for characterizing the power consumption and guiding optimization of mobile architectures accordingly. We build a logger application for the Android G1 phone and release it to the general public to collect logs from real users on real devices. We then develop a linear-regression-based power estimation model, which uses high-level measurements of each hardware component, to estimate the system power consumption. We show that the power estimation model is highly accurate and that it can provide insights about the power breakdown of the hardware components in a mobile architecture. By analyzing the user logs, we find that the power breakdown of a device is highly dependent upon the individual user, but that the screen and the CPU tend to dominate the active power consumption. We then demonstrate an example of leveraging user behavior to identify new optimizations. Specifically, we study active screen intervals and discover that a relatively small number of long screen intervals dominate the active screen time. Based upon this observation, we develop an optimization for the screen and the CPU that advantage of change blindness. We demonstrate that our optimizations can save up to 10% total system power while minimally impacting user satisfaction.

9. REFERENCES

[1] Apple Inc. iPhone OS Technology Overview: About iPhone OS Development, October 2008.

[2] Arbitron and Edison Research Media. The Infinite Dial 2008: Radio's Digital Platforms.

[3] K. Baynes, C. Collins, E. Fiterman, B. Ganesh, P. Kohout, C. Smit, T. Zhang, and B. Jacob. The performance and energy consumption of three embedded real-time operating systems. In *Proceedings of the Intl. Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 203–210, November 2001.

[4] F. Bellosa. The benefits of event-driven energy accounting in power-sensitive systems. In *Proceedings of the SIGOPS European Workshop*, September 2000.

[5] W. L. Bircher, M. Valluri, J. Law, and L. K. John. Runtime identification of microprocessor energy saving opportunities. In *Proceedings of the Intl. Symposium on Low Power Electronics and Design*, pages 275–280, 2005.

[6] L. Bloom, R. Eardley, E. Geelhoed, M. Manahan, and P. Ranganathan. Investigating the relationship between battery life and user acceptance of dynamic, energy-aware interfaces on handhelds. In *Proceedings of the Intl. Conference on Human-Computer Interaction with Mobile Devices and Services*, pages 13–24, September 2004.

[7] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the Intl. Symposium on Computer Architecture*, pages 83–94, 2000.

[8] T. L. Cignetti, K. Komarov, and C. S. Ellis. Energy estimation tools for the Palm^{TM}. In *Proceedings of the Intl. Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, August 2000.

[9] G. Contreras and M. Martonosi. Power Prediction for Intel XScale® Processors Using Performance Monitoring Unit Events. In *Proceedings of the Intl. Symposium on Low Power Electronics and Design*, pages 221–226, August 2005.

[10] R. P. Dick, G. Lakshminarayana, A. Raghunathan, and N. K. Jha. Power analysis of embedded operating systems. In *Design Automation Conference*, pages 312–315, 2000.

[11] Display Search; NPD Group. Strong Mini-Note PC Demand Expected to Buoy Notebook Market in 2009, April 2009. http://www.displaysearch.com/.

[12] N. Eagle and A. Pentland. Social serendipity: Mobilizing social software. *IEEE Pervasive Computing*, 4(2):28–34, January–March 2005.

[13] J. Froehlich, M. Y. Chen, S. Consolvo, B. Harrison, and J. A. Landay. Myexperience: A system for in situ tracing and capturing of user feedback on mobile phones. In *Proceedings of the Intl. Conference on Mobile Systems, Aapplications and Services*, pages 57–70, 2007.

[14] Google, Inc. Android - An Open Handset Alliance Project. http://developer.android.com.

[15] S. Gurumurthi, A. Sivasubramaniam, M. J. Irwin, N. Vijaykrishnan, M. Kandemir, T. Li, and L. K. John. Using Complete Machine Simulation for Software Power Estimation: The SoftWatt Approach. In *Proceedings of the Intl. Symposium on High Performance Computer Architecture*, pages 141–150, February 2002.

[16] S. Gurun and C. Krintz. A run-time feedback-based energy estimation model for embedded devices. In *Proceedings of the Intl. Conference on Hardware/Software Codesign and System Synthesis*, pages 28–33, October 2006.

[17] T. Harter, S. Vroegindeweij, E. Geelhoed, M. Manahan, and P. Ranganathan. Energy-aware user interfaces: An evaluation of user acceptance. In *Proceedings of the Conference on Human Factors in Computing Systems*, pages 199–206, April 2004.

[18] R. Joseph and M. Martonosi. Run-time power estimation in high performance microprocessors. In *Proceedings of the Intl. Symposium on Low Power Electronics and Design*, August 2001.

[19] I. Kadayif, T. Chinoda, M. T. Kandemir, N. Vijaykrishnan, M. J. Irwin, and A. Sivasubramaniam. vec: virtual energy counters. In *Proceedings of the Workshop on Program Analysis For Software Tools and Engineering*, June 2001.

[20] T. Li and L. K. John. Run-time modeling and estimation of operating system power consumption. In *Proceedings of the Intl. Conf. on Measurements and Modeling of Computer Systems*, 2003.

[21] A. Mahesri and V. Vardhan. Power consumption breakdown on a modern laptop, workshop on power aware computing systems. In *Proceedings of the Workshop on Power-Aware Computer Systems*, December 2004.

[22] A. Mallik, J. Cosgrove, R. Dick, G. Memik, and P. Dinda. PICSEL: Measuring user-percieved performance to control dynamic frequency scaling. In *Proceedings of the Intl. Conference on Architectural Support for Programming Languages and Operating Systems*, March 2008.

[23] C. Phillips, S. Singh, D. Sicker, and D. Grunwald. Applying models of user activity for dynamic power management in wireless devices. In *Proceedings of the Intl. Conference on Human-Computer Interaction with Mobile Devices and Services*, pages 315–318, September 2008.

[24] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009. ISBN 3-900051-07-0.

[25] A. Shye, B. Ozisikyilmaz, A. Mallik, G. Memik, P. A. Dinda, R. P. Dick, and A. N. Choudhary. Learning and leveraging the relationship between architecture-level measurements and individual user satisfaction. In *Proceedings of the Intl. Symposium on Computer Architecture*, June 2008.

[26] A. Shye, Y. Pan, B. Scholbrock, J. S. Miller, G. Memik, P. A. Dinda, and R. P. Dick. Power to the people: Leveraging human physiological traits to control microprocessor frequency. In *Proceedings of the Intl. Symposium on Microarchitecture*, December 2008.

[27] D. J. Simons and C. F. Chabris. Gorillas in our midst: sustained inattentional blindness for dynamic events. *Perception*, 28:1059–1074, 1999.

[28] D. J. Simons, S. L. Franconeri, and R. L. Reimer. Change blindness in the absence of a visual disruption. *Perception*, 29:1143–1154, 2000.

[29] T. K. Tan, A. Raghunathan, G. Lakshiminarayana, and N. K. Jha. High-level softwrae energy macro-modeling. In *Proceedings of Design Automation Conference*, pages 605–610, June 2001.

[30] Wikipedia: The Free Encyclopedia. HTC Dream. http://en.wikipedia.org/wiki/Gphone.