Pervasive and Mobile Computing 20 (2015) 47–63

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

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

A middleware framework for application-aware and user-specific energy optimization in smart mobile devices

Sudeep Pasricha [a,*], Brad K. Donohoo [b], Chris Ohlsen [c]

[a] Colorado State University, Fort Collins, CO 80523, USA
[b] U.S. Department of the Air Force, Roy, UT 84067, UT 84067, USA
[c] Woodward, Inc., Fort Collins, CO 80525, USA

ARTICLE INFO

Article history:
Received 5 August 2014
Received in revised form 22 December 2014
Accepted 7 January 2015
Available online 14 January 2015

Keywords:
Energy optimization
Smart mobile systems
Pervasive computing
Machine learning
Middleware

ABSTRACT

Mobile battery-operated devices are becoming an essential instrument for business, communication, and social interaction. In addition to the demand for an acceptable level of performance and a comprehensive set of features, users often desire extended battery lifetime. In fact, limited battery lifetime is one of the biggest obstacles facing the current utility and future growth of increasingly sophisticated "smart" mobile devices. This paper proposes a novel application-aware and user-interaction aware energy optimization middleware framework (*AURA*) for pervasive mobile devices. *AURA* optimizes CPU and screen backlight energy consumption while maintaining a minimum acceptable level of performance. The proposed framework employs a novel Bayesian application classifier and management strategies based on Markov Decision Processes and Q-Learning to achieve energy savings. Real-world user evaluation studies on Google Android based HTC Dream and Google Nexus One smartphones running the *AURA* framework demonstrate promising results, with up to 29% energy savings compared to the baseline device manager, and up to 5×savings over prior work on CPU and backlight energy co-optimization.

1. Introduction

Mobile smartphones and other portable battery operated systems (PDAs, tablets) are pervasive computing devices that have emerged in recent years as essential instruments for communication, business, and social interactions. At the end of 2013, there were more than 6.8 billion mobile subscribers worldwide, with smartphone sales showing the strongest growth. Over 1,600,000 apps have been developed across various mobile platforms, with the Apple App Store and Google Play being the largest app stores. Popular mobile activities include web browsing, multimedia, games, e-mail, and social networking [1]. Overall, these trends suggest that mobile devices are now the new development and computing platform for the 21st century.

Performance, capabilities, and design are all primary considerations when purchasing a smart mobile device; however, battery lifetime is also a highly desirable attribute. Most portable devices today make use of lithium-ion polymer batteries, which have been used in electronics since the mid 1990s [2]. Although lithium-ion battery technology and capacity has improved over the years, it still cannot keep pace with the power consumption demands of today's mobile devices. Until a new battery technology is discovered, this key limiter has led to a strong research emphasis on battery lifetime extension, primarily using software optimizations [3–14].

* Corresponding author.
E-mail addresses: sudeep@colostate.edu (S. Pasricha), brad.donohoo@gmail.com (B.K. Donohoo), chris.ohlsen@gmail.com (C. Ohlsen).

http://dx.doi.org/10.1016/j.pmcj.2015.01.004
1574-1192/

It is important to note that outside of the obvious differences between mobile devices and a general PC – weight and size, form factor, computational capabilities, and robustness – a key difference can be found in the user interaction patterns and interfaces. Unlike a desktop or notebook PC in which a user typically interacts with applications using a pointer device or keyboard, applications on mobile devices most often receive user input through a touch screen or keypad events. Many times applications are interacted with for short durations throughout the day (e.g., few seconds or minutes instead of hours) and these patterns are often unique to each individual user. Significant differences in user interaction patterns make a general-purpose power management strategy unsuitable for mobile devices.

In this work, we present a novel application and user interaction aware energy management framework (*AURA*) for pervasive mobile devices, which takes advantage of the user-specific patterns of interactions with applications running on mobile devices to optimize CPU and backlight energy consumption. In order to balance energy consumption and quality of service (QoS) requirements that are unique to each individual user, *AURA* makes use of a Bayesian application classifier to dynamically classify applications based on user interaction. Once an application is classified, *AURA* utilizes Markov Decision Process (MDP) or Q-Learning based power management algorithms to adjust processor frequency and screen backlight levels to reduce system energy consumption between user interaction events. Overall, we make the following novel contributions:

- We conduct usage studies with real users and develop a Bayesian application classifier tool to categorize mobile applications based on user interaction activity;
- We develop an integrated MDP/Q-Learning based application and user interaction-aware energy management framework that adapts CPU and backlight levels in a mobile device to balance energy consumption and user QoS;
- We characterize backlight and CPU power dissipation on Android OS based HTC Dream and Google Nexus One smartphone architectures;
- We implement our framework as middleware running on the HTC Dream and Google Nexus One smartphones and demonstrate real energy savings on commercial apps running on the devices.
- We perform real-world user evaluation studies with the Google Android based HTC Dream and Google Nexus One mobile devices running the *AURA* framework and demonstrate promising results, with up to 29% energy savings compared to the baseline device manager; and up to 5× savings over the best known prior work on CPU and backlight energy co-optimization, with negligible impact on user quality of service.

This work is a significantly extended version of our previously published conference paper [15] with the following major additions: (i) additional power models, algorithm analysis, and results for a new phone architecture—the Google Nexus One; (ii) analysis and results for a new power management algorithm based on Q-Learning; (iii) a finer-grained app classification that allows the algorithms to be tuned to each application more precisely; (iv) a more extensive user–device interaction field study in which we examine the variability of user interaction patterns in Section 2; (v) further details in Section 3.2 about the activation of DVFS during idle periods; (vi) a study of the effect of successful prediction rates on actual user satisfaction and definition of a minimum acceptable performance level, in Section 5; (vii) a study on the performance of the power management algorithms when more mispredictions are allowed, in Section 5; and (viii) more comprehensive related work in Section 6 explaining how our work is different and novel in comparison with previously published work.

2. AURA energy management framework

In this section, we present details of the *AURA* framework. In Section 2.1 we first describe our fundamental observations that lay the foundation for energy savings in mobile devices. Section 2.2 presents results of field studies involving users interacting with apps on mobile devices. Section 2.3 gives a high level overview of the *AURA* middleware framework. Subsequent sections elaborate on the major components of the framework.

2.1. Fundamental user–device interaction mechanisms

Here we explain the underlying concepts stemming from the psychology of user–device interactions that drive the CPU and backlight energy optimizations in the *AURA* framework.

There are three basic processes involved in a user's response to any interactive system [16], such as a smartphone or a personal computer. During the *perceptual* process, the user senses input from the physical world. During the *cognitive* process, the user decides on a course of action based on the input. Finally, during the *motor* process, the user executes the action with mechanical movements. These three processes are consecutive in time and can be characterized by an *interaction slack*. For instance, when interacting with an app on a mobile device, the time between when a user interacts with an app (e.g., touching a button) and when a response to that interaction is perceptible to the user (e.g., the button changes color) is the perceptual slack; the period after the system response during which the user comprehends the response represents the cognitive slack; and finally the manual process of the next interaction (e.g., moving finger to touch the screen) involves a motor slack. Fig. 1 illustrates this process. Before the user physically touches the input peripherals, the system is idle during the cumulative slack period, for hundreds to possibly many thousands of milliseconds. During this time, if the CPU frequency can be reduced, energy may be saved without affecting user QoS. **CPU energy** optimizations in *AURA* exploit this inherent slack that arises whenever a user interacts with a mobile device.

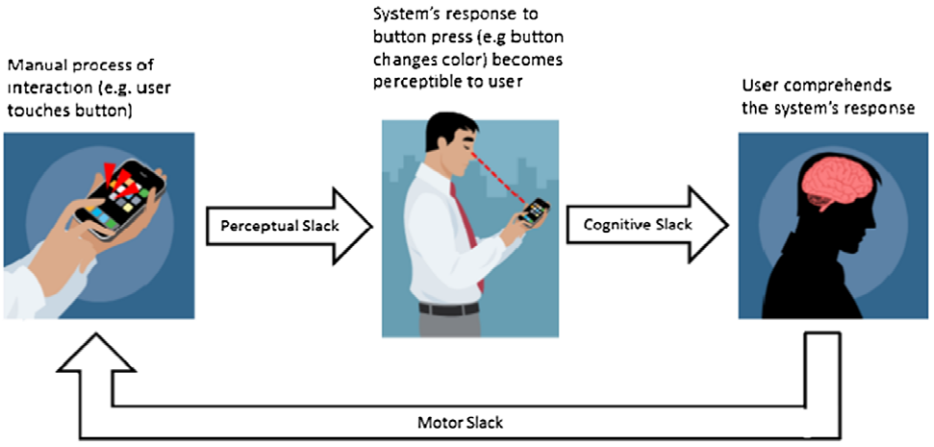

Fig. 1. Interaction slack process.

Another interesting phenomenon that can open up new possibilities for energy optimizations is the notion of *change blindness* [17] as revealed by researchers in human psychology and perception. Change blindness refers to the inability of humans to notice large changes in their environments, especially if changes occur in small increments. Many studies have confirmed this limitation of human perception [18]—a majority of observers in one study failed to observe when a building in a photograph gradually disappeared over the course of 12 s; in another study, gradual color changes over time to an oil painting went undetected by a majority of subjects but disruptive changes such as the sudden addition of an object were easily detected. By gradually reducing screen brightness over time in certain scenarios, energy consumption in mobile devices may thus be reduced without causing user dissatisfaction. This principle is exploited by *AURA* to optimize screen **backlight energy**.

2.2. User–device interaction field study

To understand how users interact with mobile applications in the real world, we conducted a field study with users running a variety of apps on a popular smartphone. It is well-known that apps running on smart mobile devices can invoke vastly different interaction behaviors from users—whereas some applications require constant interaction with the user, others may experience bursts of user interaction followed by long idle periods. These unique interaction patterns make conventional general-purpose energy management strategies ineffective on mobile devices. We hoped to uncover trends from our field study that would potentially guide scenario-specific energy optimizations.

We selected 18 Android apps from the following categories: *games, communication, multimedia, shopping, personal/business, travel/local*, and *social networking*. A custom background interaction logger app was developed to record interaction traces for users running each of these apps. The study involved five different users interacting with the apps over the course of a day on an Android OS based Google Nexus One smartphone. The users selected for the study spanned the proficiency spectrum: user 3 for instance mainly used his smartphone for phone calls, and was not familiar with many of the selected applications, whereas user 2 relied on her smartphone for a variety of tasks, and was quite comfortable while interacting with the chosen applications.

Table 1 shows a summary of the interactions logged for the users for each of the selected apps. The patterns in the table are classified based on interaction frequency (*very-low, low, low-medium, medium, medium-high, high, very-high*). An obvious observation that we can immediately make from these results is that although some apps belong to the same category (e.g., *Sudoku* and *Jewels* are both games) they possess vastly different interaction characteristics. It is also interesting to see that user–device interactions for some apps are very **application-specific**. For example, *Jewels*, a fast-paced gaming application, is always classified as very-high-interaction, and *Music* is almost always classified as very-low-interaction because users usually select a song and then cease interaction while the song plays. On the other hand, some classifications are noticeably **user-specific**, meaning that they may be classified quite differently based on the type of user interacting with them. Interaction results for the *Minesweeper*, *Gallery*, and *News and Weather* apps illustrate this idea.

Next, we conducted a study to examine the variability in interaction patterns for different applications. By averaging the times between touch events for each user's interaction sessions from the previous study, we obtained interaction distributions for eight common mobile applications, shown in Fig. 2. In the figure, the circle sizes represent the average (taking into account all users) of how frequently the between-touch times occurred for each application. For example: *WordFeud*'s large circles at 1 and 7+ indicate that most of the touch events in its interaction pattern had either 1 s or greater than 7 s separating them. Note that the times between touch events are rounded to the nearest second. Looking at the figure, it is clear that the interaction patterns for the *Market* and *Gallery* applications demonstrate notable variations. Alternatively, the interaction patterns for the *SMS* and *Jewels* applications are fairly invariant, exhibiting a large percentage of shortly spaced touch events.

Table 1
App interaction classification.

APPLICATION	CLASSIFICATION				
	User 1	User 2	User 3	User 4	User 5
1 Market (com.android.vending)	Med High	Med	Med	Low Med	Med
2 Gallery (com.cooliris.media)	Med High	Low	Med High	Low Med	Med
3 Jewels (org.mhgames.jewels)	Very High	Very High	Very High	Very High	Very High
4 Wordfeud FREE (com.hbwares.wordfeud.free)	Very Low	Very Low	Low	Low Med	Very Low
5 Facebook (com.facebook.katana)	High	Med High	High	Med High	Med
6 Gmail (com.google.android.gm)	Med	Low	Low Med	Low Med	Low Med
7 SMS (com.android.mms)	Very High	Very High	Very High	Very High	Very High
8 Browser (com.android.browser)	High	Very High	Med	Med High	High
9 News & Weather (com.google.android.apps.genie.geniewidget)	Low Med	Low	Low Med	Med High	Med
10 YouTube (com.google.android.youtube)	Very Low	Very Low	Very Low	Very Low	Very Low
11 Calculator (com.android.calculator2)	Very High	Very High	Very High	Very High	Very High
12 Twitter (com.android.twitter)	Med High	Med	Med High	Med High	Med
13 Calendar (com.android.calendar)	Med	Med High	Low Med	Low Med	Med
14 Google Maps (com.google.android.apps.maps)	Med	Very High	High	High	High
15 Sudoku (cz.romario.opensudoku)	High	Low	Very High	High	Low Med
16 Music (com.android.music)	Very Low	Very Low	Very Low	Low	Very Low
17 Solitaire (com.softick.android.solitaire.klondike)	Very High	High	Very High	High	Very High
18 Minesweeper (artfulbits.aiMinesweeper)	Med High	Med High	High	Very High	Med

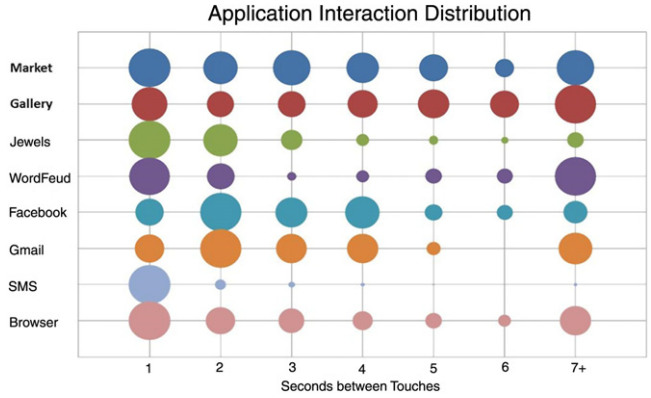

Fig. 2. Application interaction distributions.

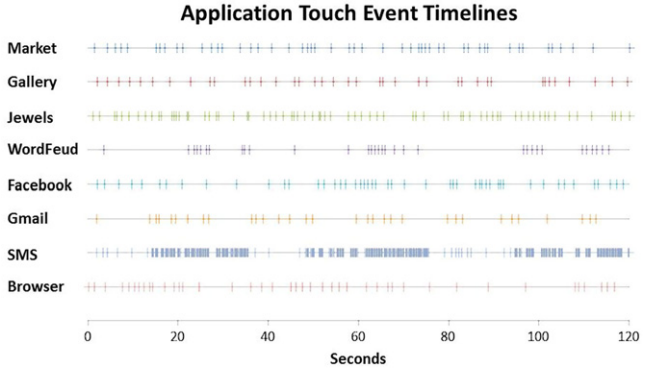

Fig. 3. Application touch event timelines.

To further our analysis of the variability of the interaction patterns for different applications, we plotted occurrences of actual touch events over two minute intervals for the eight previously selected applications. Fig. 3 shows examples of the touch events that occurred in a single two minute session with each application. The figure allows us to see the "bursty"

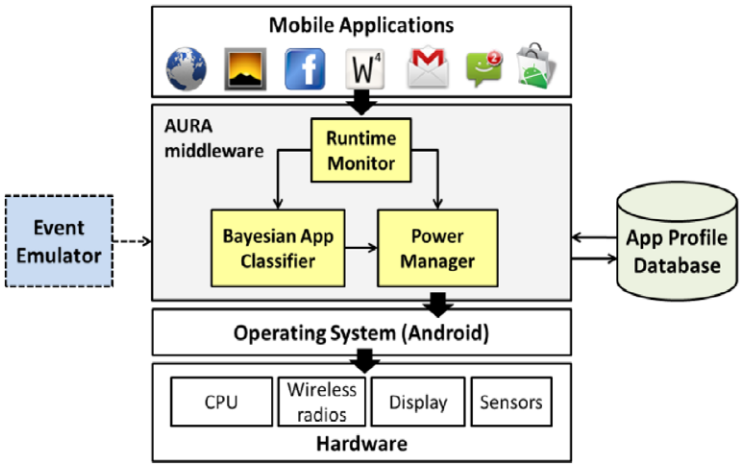

Fig. 4. AURA energy optimization middleware framework.

interaction that some applications display, e.g., *WordFeud*, *SMS*, and *Gmail*. In addition, Fig. 3 also shows the variability of *Market*, *Gallery*, *Facebook*, and *Browser*, and strengthens the claims that *Jewels* and *SMS* are very-high-interaction applications, while *WordFeud* is a very-low-interaction application.

We can thus conclude that users interact with devices in a very user-specific and application-specific manner. Any framework that would attempt to exploit interaction slack and change blindness to save energy (as discussed in the previous section) must tailor its behavior uniquely across users and applications.

2.3. AURA middleware framework overview

We now present a high level overview of the *AURA* energy optimization framework for mobile devices. Fig. 4 shows the *AURA* framework that is implemented at the middleware level in the software stack of the Android OS [19], and runs in the background as an Android Service. As the user switches between applications, the *AURA* **runtime monitor** will receive a "Global Focus Event", and update any relevant session information of the previous foreground application (the application that was previously displayed on the screen) to an **app profile database**, stored in memory. The runtime monitor will then search the database for the current foreground application. If the application exists in the database and is classified, the **power manager** will make use of the application's learned classification and user interaction pattern statistics (e.g., mean and standard deviation of touch events) to invoke a user-specific power management strategy that is optimized to the application. The power manager directly accesses the mobile device's hardware components to change screen backlight and CPU frequency. If the application does not exist in the database, or the application exists in the database but is not classified, *AURA* will make use of the user interaction events ("Global Touch" and "Global Key" events) to *dynamically classify* the application using a **Bayesian application classifier**. Following classification, the power manager will run an appropriate power management strategy on the next invocation of the app. An **event emulator** was also integrated into *AURA* to simulate user interaction events for validation purposes.

The following sections elaborate on the major components of the AURA framework that were highlighted in the above overview.

2.4. Runtime monitor

The runtime monitor is an event-driven module responsible for monitoring user interactions with the mobile device. The Android OS makes use of public callback methods, known as event listeners that allow applications to receive and handle user interactions (e.g., touch and key events) with items displayed on the user interface (UI) [19]. Unfortunately, there is no API to allow applications to receive "global" UI events, which are often consumed by application code or the current foreground application. In order for the runtime monitor to receive global UI events, a set of custom event broadcasts were created that are sent when any touch or key event occurs, when an application gains focus, and when the phone configuration changes (e.g., hard keyboard is utilized). The runtime monitor is only invoked whenever such custom global UI events are broadcast. The module keeps tracking variables for each current foreground application, the class of user interaction events, and the mobile device state, which are all updated on receiving global UI events. When a new application gains foreground focus, the runtime monitor will save session information from the previous foreground application (e.g., classification, user interaction stats) to the **app profile database** and retrieve any stored application information from the database for the new foreground application. If the new foreground application has already been classified, then the runtime monitor will invoke the power manager, otherwise, it will invoke the Bayesian classifier to dynamically classify the application as it receives user interaction events.

Table 2
Application classes.

Application class	Mean value (in ms)	Std. dev. value (in ms)
Very-low-interaction	$x_1 \leq 1000$	$x_2 \leq 1000$
Low-interaction	$1000 < x_1 \leq 2000$	$1000 < x_2 \leq 2000$
Low-medium-interaction	$2000 < x_1 \leq 3000$	$2000 < x_2 \leq 3000$
Medium-interaction	$3000 < x_1 \leq 4000$	$3000 < x_2 \leq 4000$
Medium-high-interaction	$4000 < x_1 \leq 5000$	$4000 < x_2 \leq 5000$
High-interaction	$5000 < x_1 \leq 6000$	$5000 < x_2 \leq 6000$
Very-high-interaction	$x_1 > 6000$	$x_2 > 6000$

2.5. Bayesian application classifier

Bayesian learning [20] is a form of supervised learning that involves using evidence or observations along with prior outcome probabilities to calculate the probability of an outcome. More specifically, the probability that a particular hypothesis is true, given some observed evidence (the *posterior probability* of the hypothesis), comes from a combination of the *prior probability* of the hypothesis and the compatibility of the observed evidence with the hypothesis (the *likelihood* of the evidence). *AURA* uses a form of Bayesian learning based on the classification method executed by Jung et al. [21]. This method of classification involves mapping a set of *input features*, $X = \{x_1, x_2, \ldots, x_n\}$, to a set of output measures, $Y = \{y_1, y_2, \ldots, y_n\}$. In our work, the input features are the mean (x_1) and standard deviation (x_2) of the times between user interaction (e.g., touch) events. We propose a 7-part categorization of input features into *very-low*, *low*, *low-medium*, *medium*, *medium-high*, *high* and *very-high* interaction classes based on corresponding values of mean and standard deviation. Such a fine-grained classification allows us to categorize applications so that the power management algorithms (explained in the following section) can be adjusted to each application more effectively. We define a single output measure: the resulting application classification (y). The pre-defined application classes, along with their threshold values for mean and standard deviation of time between touch events (in ms), are shown in Table 2.

Thus the classifier ultimately allows us to get a characterization of user and application specific interaction intensity. Learning for our classifier is accomplished through creation of a training set of size γ_{TS} for each application. Finding a consistent mapping from input features to an output measure using this training set enables the classifier to accurately predict the class of an application. The classification task consists of calculating posterior probabilities for each class, based on the *prior probability*, *class likelihood*, and *evidence*, then choosing the class with the highest posterior probability. The posterior probabilities are obtained using the following equation:

$$\textit{posterior probability} = \frac{\textit{prior probability} \times \textit{likelihood}}{\textit{evidence}} \tag{1}$$

$$\downarrow$$

$$Prob\,(y = c|x_1, x_2) = \frac{Prob\,(x_1, x_2|y = c) \times Prob\,(y = c)}{Prob\,(x_1, x_2)}. \tag{2}$$

The denominator $Prob\,(x_1, x_2)$, or the evidence, is the marginal probability that an observation X is seen under all possible hypotheses, and is irrelevant for decision making as it is the same for every class assignment [20]. $Prob\,(y = c)$, the class likelihood, is the prior probability of the hypothesis that the application class y is c. This is easily calculated from the training set. $Prob\,(x_1, x_2|y = c)$ is the per-input-feature conditional probability of seeing the input feature vector X given that the application class y is c. From Jung et al. [21] we have:

$$Prob\,(x_1, x_2|y = c) = Prob(x_1|y = c) \times Prob(x_2|y = c). \tag{3}$$

Thus, the posterior probability of an application class may be computed as follows:

$$Prob(y = c) = Prob(x_1, x_2\,|y = c) \times Prob(y = c). \tag{4}$$

There may be some cases in which input features do not occur together with an output measure due to an insufficient number of data points in the training set, resulting in a zero conditional probability. To deal with this, we eliminate them by smoothing:

$$Prob(x_i|y = c) = \frac{freq\,(x_i, y = c) + \lambda}{freq\,(y = c) + \lambda n_x} \tag{5}$$

where λ is a smoothing constant ($\lambda > 0$), and n_x is the number of different attributes of x_i that have been observed.

To illustrate this process, consider an example of dynamic classification based on the training set shown in Table 3. When looking at this table, it should be noted that $x_1 = high$ denotes a *high-interaction* mean value, thus a smaller mean gap between events. Also, for simplicity, in this example we only consider three application classes: *low-interaction*, *medium-interaction*, and *high-interaction*. Let the bottom row be a new input feature ($x_1 = med$, $x_2 = high$) which was not in the

Table 3
Example training set.

Input features		Output measure
Mean (x_1)	Std. dev. (x_2)	Classification
High	Low	High-interaction
High	High	Med-interaction
Med	Med	Med-interaction
Med	Low	Med-interaction
Low	High	Low-interaction
Med	High	Med-interaction

training set, that is presented to the classifier. Assuming that the smoothing constant $\lambda = 1$, this new input is classified as follows. For the hypothesis $y = high\text{-}interaction$, the posterior probability is:

$$Prob(x_1 = med, x_2 = high|y = high) = \frac{1}{3} \times \frac{1}{3} \times \frac{1}{5} = \frac{1}{45}$$

because

$$Prob(x_1 = med|y = high) = \frac{1}{3}$$

and

$$Prob(x_2 = high|y = high) = \frac{1}{3}$$

and

$$Prob(y = high) = \frac{1}{5}.$$

Note that $Prob(x_1 = med|y = high)$ and $Prob(x_2 = high|y = high)$ would be 0 if not for the smoothing constant. Similarly, for the hypothesis $y = med\text{-}interaction$, the posterior probability is:

$$Prob(x_1 = med, x_2 = high|y = med) = \frac{3}{5} \times \frac{2}{5} \times \frac{3}{5} = \frac{18}{125}$$

and for hypothesis $y = low\text{-}interaction$, the posterior probability is:

$$Prob(x_1 = med, x_2 = high|y = low) = \frac{1}{3} \times \frac{2}{3} \times \frac{1}{5} = \frac{2}{45}.$$

The output measure of the new input feature is thus *med-interaction*. An application is classified in this manner on each interaction event. Upon being assigned the same classification m (a user-defined parameter) number of times, the application is marked as classified, and an appropriate power management strategy (discussed in the next section) will be run on its next invocation.

2.6. Power manager

2.6.1. Markov decision process (MDP) based manager

As part of the *AURA* framework, we developed three MDP power management algorithms to transition between a nominal state and below nominal state at discrete time intervals based on probabilities of user interaction, processor demands, and application classification. Markov Decision Processes (MDP) [22] are discrete time stochastic control processes that are widely used as decision-making models for systems in which outcomes are partly random and partly controlled. MDPs consist of a four-tuple $(S, A, P_A, R_{S,A})$, where S is a finite number of states, A is a finite set of actions available from each state, P_A is the probability that action a in state s at time t will lead to state s^* at time $t + 1$, and $R_{S,A}$ is a reward from transitioning to state s^* from state s.

All three of our MDP based algorithms make use of the state flow diagram illustrated in Fig. 5. When an application is classified and gains foreground focus, the classification and learned user event statistics are used to set algorithm control parameters (e.g., evaluation interval τ, screen backlight adjustment levels Δ_S, probability thresholds P_T) and speculate on the probability of a user event within the next evaluation interval. When the probability of an event is below a given high-to-low threshold (P_{HLT} in Fig. 5) and processor utilization is below a given utilization threshold (U_{HLT} in Fig. 5), the state transitions to *Below Nominal* and the processor frequency level is adjusted to the lowest supported frequency. Note that the frequency levels shown in the figure are the values corresponding to the Google Nexus One mobile device. While the state is *Below Nominal*, the screen brightness is gradually adjusted down at each evaluation interval, taking advantage of the concept of change blindness. (Note: the actual values used for backlight adjustment levels, evaluation interval duration, etc., are shown in Table 6 in Section 4: Experimental Methodology.) A transition to the mid-level frequency while in the *Below*

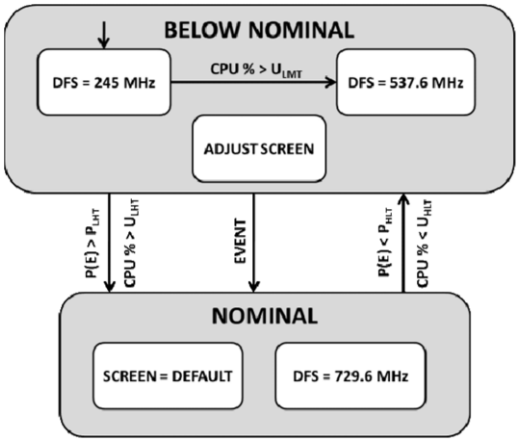

Fig. 5. MDP control algorithms state flow diagram.

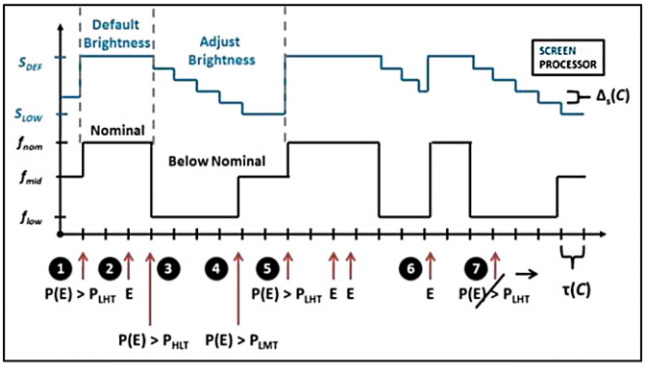

Fig. 6. Power management control algorithm timeline.

Nominal state will also occur if processor utilization reaches a low-med utilization threshold (U_{LMT} in Fig. 5) or an immediate transition back to *Nominal* state will occur if processor utilization exceeds an even higher utilization threshold (U_{LHT} in Fig. 5). If processor utilization is not a dominating factor then a transition back to *Nominal* will occur as the probability of a user event approaches a given low-to-high threshold (P_{LHT} in Fig. 5) or any user event occurs, in which the nominal processor frequency and screen backlight level are set back to user-specified defaults.

Our baseline MDP algorithm, *NORM*, makes use of a Gaussian distribution to predict user events based on learned classification and stored user-interaction statistics. More specifically, the mean (μ) and standard deviation (σ) of times between touch event are used as parameters in the following probability density function to calculate the probability of a touch event occurring within the next evaluation interval:

$$f(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\pi)^2}{2\sigma^2}}. \tag{6}$$

Two derivatives of *NORM* were created to *dynamically adapt to real-time user-interaction* during each classified application invocation. The *E-ADAPT* variant is event-driven and uses the most recent W_E user events to predict future interaction events. The *T-ADAPT* variant makes use of a moving average window of size W_T, to dynamically track recent user interaction within a temporal context.

Fig. 6 illustrates the basic working of the MDP based power management algorithms. At Point 1, the probability of a user event reaches the low-to-high probability threshold and the state transitions back to *Nominal*. After an event occurs at Point 2, the probability of a user event falls below the high-to-low threshold at Point 3, and the state transitions to the *Below Nominal* state. While in the below nominal state, the screen backlight gradually changes during this predicted user idle time. At Point 4, the probability of a user event rises above the low-to-med threshold and the processor frequency is adjusted accordingly. At Point 5, the probability of a user-event reaches the low-to-high threshold and the state transitions back to *Nominal*. Point 6 shows a misprediction, in which a user event occurs while in the *Below Nominal* state. This causes the state to transition back to *Nominal* again. Mispredictions are used as a coarse QoS metric to evaluate the effectiveness of the algorithms and ensure that energy savings are not diminishing overall QoS.

Table 4
Q-Learning algorithm reinforcements.

State	Touch received?	Action	Reinforcement
Nominal	Yes	Do NOT change to *Below Nominal*	2
Nominal	No	Do NOT change to *Below Nominal*	−1
Nominal	Yes	Change to *Below Nominal*	−2
Nominal	No	Change to *Below Nominal*	1
Below Nominal	Yes	Do NOT change to *Nominal*	−2
Below Nominal	No	Do NOT change to *Nominal*	1
Below Nominal	Yes	Change to *Nominal*	2
Below Nominal	No	Change to *Nominal*	−2

2.6.2. Q-Learning based manager

We also developed a Q-Learning based power management algorithm to include in the *AURA* framework. Q-Learning is a reinforcement learning technique that does not require a model of the environment. In other words, the next state probability distributions that are used in the MDP algorithms are not required. The Q-Learning algorithm creates a model by exploring the environment. For the exploration phase, one option is to use the epsilon-greedy (ε-*greedy*) search, in which we choose an action randomly out of all possible actions with probability ε, and choose the best action with probability $1 - \varepsilon$ [23]. The cost of a single action taken from a state is the *reinforcement* for that action from that state. The value of that state-action is the expected value of the full *return*, or the sum of the reinforcements that will follow when that action is taken [20]. This state-action value function is called the Q function because it calculates the *quality* of a state-action combination. Given the current state, s_t, an action, a_t, the reinforcement for taking action a_t in state s_t, r_{t+1}, the next state, s_{t+1}, and the next action, a_{t+1}, new Q function values are calculated for each possible state-action pair using the following equation:

$$Q\,(s_t, a_t) \leftarrow Q\,(s_t, a_t) + \alpha\,(r_{t+1} + \gamma + \max_{\alpha_{t+1}} Q\,(s_{t+1}, a_{t+1}) - Q\,(s_t, a_t)) \tag{7}$$

where $0 < \alpha \leq 1$ is the *learning rate* and $0 \leq \gamma < 1$ is the *discount factor*. The learning rate determines to what extent the old Q values are overridden. A learning rate of 0 will not update the old Q values at all, while a learning rate of 1 will only consider the most recent information. The discount factor determines the importance of future rewards. A factor value of 0 will make the algorithm opportunistic, considering current rewards more importantly, and a factor value of 1 will make the algorithm strive for a higher long-term reward.

In the *AURA* power manager, the Q-Learning algorithm **state** is made up of two parts—(1) whether the device is *Nominal* (nominal processor frequency and default screen brightness) or *Below Nominal* (processor frequency is set to the lowest level and the screen brightness is gradually adjusted down), and (2) the evaluation interval number. The evaluation intervals in our power manager are discrete time intervals in which to re-evaluate the system state and decide appropriate actions, and the evaluation interval number keeps track of how many evaluation intervals have passed since the last touch event. It can be one of 11 values: 1–10 or >10. The two possible actions are whether the state should change or not. The reinforcements for each state action pair are shown in Table 4. The values for the rewards bias the algorithm towards QoS—higher rewards are given for when touch events are predicted correctly, and higher penalties are given when they are predicted incorrectly. Each application has its own table of Q values, which is stored in the app profile database when the app loses foreground focus.

2.6.3. Power manager control parameters

Table 5 lists several key algorithm control parameters that can be used to customize the algorithms discussed above to each individual user and application. For example, the evaluation interval parameter is determined based on the application classification. High interaction applications will require more frequent evaluation, while low interaction applications can use a larger evaluation interval. Event probability thresholds are also set based on application classification and coefficient of variance, which gives a measure of the variability of the learned user interaction pattern. For example, low interaction applications with low variability will use high event probability thresholds to bias towards energy savings. For applications with high variability in user-interaction patterns, event probability thresholds will be slightly biased towards QoS. Processor utilization thresholds are statically set at design time; however, could also be set dynamically if finer-grained control is required. Screen backlight levels are set based on application classification, where high interaction applications are again biased towards QoS and low interaction applications are biased towards energy savings. It is important to note that screen backlight levels are also limited by a QoS-driven lower bound of 40%. For the *E-ADAPT* variant, a larger W_E value can be used to get a global view of the user-interaction pattern or a smaller W_E value can be used for a more local view of the user-interaction pattern. A similar argument can be made for the *T-ADAPT* variant which dynamically adapts the user-event statistics based on a moving average window of size W_T. Again, the actual algorithm control parameter values that we used are shown in Table 6 in Section 4: Experimental Methodology. It is important to note that the values we chose for our experiments are not necessarily optimal for every circumstance—they can be tuned by the user to achieve the desired energy savings and performance.

Table 5
Power management algorithm control parameters.

Parameter	Dependencies	Description
Evaluation interval (τ)	Application classification	Discrete time interval in which to re-evaluate the system state and decide appropriate action(s)
Event probability threshold (P_T)	Application classification; Not used in *Q-LEARNING*	Threshold to determine state transitions based on user-event prediction
Processor utilization threshold (U_T)	Static; Not used in *Q-LEARNING*	Conditional threshold to determine state transitions based on CPU demands
Backlight adjustment levels (Δ_S)	COV[a]	Screen backlight adjustment levels while in state *Below Nominal*
Event window (W_E)	*E-ADAPT* only	Number of most recent user events to use for dynamic adaptation
Time window (W_T)	*T-ADAPT* only	Size of moving average window to use for dynamic adaptation
Epsilon (ε)	*Q-LEARNING* only	Probability that an action will be chosen randomly, as opposed to choosing the best action
Learning rate (α)	*Q-LEARNING* only	Determines to what extend old Q values are overridden
Discount factor (γ)	*Q-LEARNING* only	Determines the importance of future rewards

[a] COV ≡ Coefficient of Variance.

Table 6
Algorithm control parameter values.

Module	Parameter	Value(s)
Bayesian classifier	Smoothing constant (λ)	1
	Training set size (γ_{TS})	50
All algorithms	Evaluation interval (τ)	1000/850/700/550/400/250/100 (VH/H/MH/M/LM/L classification[a])
MDP algorithms	High-to-low event probability threshold (P_{HLT})	0.55/0.5/0.45/0.4/0.35/0.3/0.25 (VH/H/MH/M/LM/L classification[a])
	Low-to-high event probability threshold (P_{LHT})	0.75/0.7/0.65/0.6/0.55/0.55/0.45 (VH/H/MH/M/LM/L classification[a])
	High-to-low processor utilization threshold (U_{HLT})	0.5
	Low-to-med processor utilization threshold (U_{LMT})	0.6
	Low-to-high processor utilization threshold (U_{LHT})	0.8
	Backlight adjustment levels (Δ_S)	0.05/0.1/0.15 (H/M/L COV)
	Event window size (W_E)	6
	Time window size (W_T)	6
Q-Learning algorithm	Epsilon (ε)	0.1
	Learning rate (α)	1
	Discount factor (γ)	0.1

[a] VH ≡ very-high-interaction, H ≡ high-interaction, MH ≡ medium-high-interaction, M ≡ medium-interaction, LM ≡ low-medium-interaction, L ≡ low-interaction, VL ≡ very-low-interaction.

3. Device power modeling

3.1. Overview

In order to quantify the energy-effectiveness of our proposed framework, power analysis was performed on two Android based smartphones: HTC Dream and Google Nexus One, with the goal of creating power models for the CPU and backlight. We use a variant of Android OS 2.2 (Froyo), on the HTC Dream and a variant of Android OS 2.3.3 (Gingerbread) on the Google Nexus One. We use Android SDK Revision 11 on both devices. The Google Android platform is comprised of a slightly modified 2.6.35.9 Linux kernel, and incorporates the Dalvik Virtual Machine (VM), a variant of Java implemented by Google. All user-space applications are Dalvik executables that run in an instance of the Dalvik VM.

We built our power estimation models using real power measurements on the HTC Dream and Google Nexus One devices. We instrumented the contact between the smartphone and the battery, and measured current using the Monsoon Solutions power monitor [24]. The monitor connects to a PC running the Monsoon Solutions power tool software that allows real-time current and power measurements over time. We developed a custom test app that was capable of switching dynamic frequency scaling (DFS) levels and screen backlight levels over time. The corresponding power traces from the Monsoon Power Tool were then used to obtain power measurements for each DFS frequency and screen brightness level.

3.2. CPU DFS power model

Dynamic frequency scaling (DFS) is a standard technique in computing systems to conserve power by dynamically changing clock frequency of a digital component (e.g., processor) at runtime when the system is idle or under-utilized. Linux-based systems, such as Android, provide a baseline DFS kernel subsystem (*cpufreq*) that is a generic framework for managing the processor operation. The *cpufreq* subsystem supports a variety of DFS drivers, many of which are configurable [25]. The

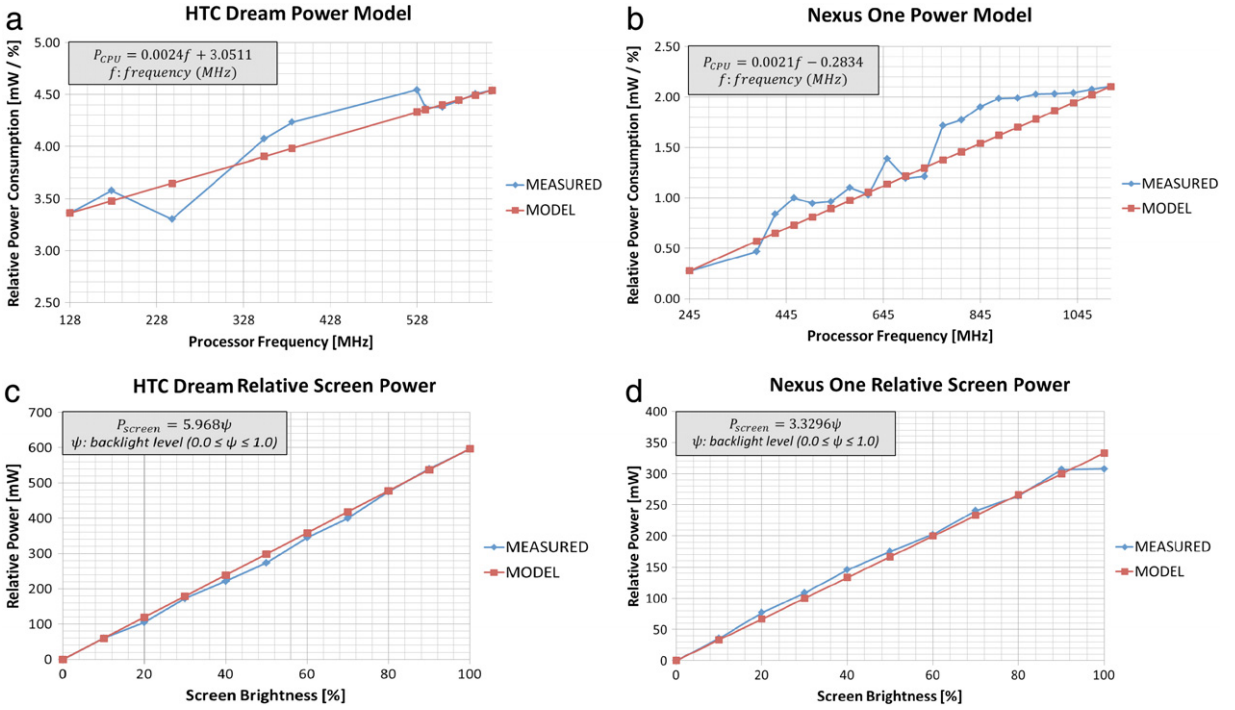

Fig. 7. DFS-based CPU and screen backlight power models for HTC Dream and Nexus One mobile devices.

userspace driver, which is used in this work, allows a user-space program to manually control DFS levels based on a user-defined set of rules.

The HTC Dream smartphone consists of a Qualcomm® MSM7201A processor. This processor is nominally clocked at 528 MHz; however, it supports 11 different frequency levels ranging from 128 MHz up to 614 MHz. The Google Nexus One consists of a Qualcomm QSD8250 Snapdragon processor. It supports 21 frequency levels ranging from 245 MHz to 1.1 GHz, and is nominally clocked at 729.6 MHz. In order to get power models for the DFS levels, the test app was used to manually switch frequency domains. Although these devices support numerous frequency levels, for simplicity, the *AURA* framework makes use of only three of them—the nominal frequency, the lowest frequency, and a mid-level frequency between those two values. Given the difficulty of controlling CPU utilization explicitly, an intensive computation loop, followed by a wait function was used to toggle processor utilization between 0% and 100% for each frequency level, and the power consumption was recorded respectively. Based on the observed measurements, linearity was assumed for CPU utilization between 0 and 100% for each frequency. Linear regression analysis [26] was used to determine appropriate coefficients and develop our CPU power estimation models with units of mW/% *utilization*. The DFS power models for the two mobile devices are shown in Figs. 7(a) and (b).

3.3. Screen backlight power model

Screen power consumption was also measured using the Monsoon Power Tool with the help of our custom test app which allowed static and dynamic control of screen backlight levels. Screen backlight levels were incremented in steps of 10% and relative average power measurements were recorded with the Monsoon Power Tool. Fig. 8 shows the instantaneous power measured using the tool as the backlight level is ramped up. Linear regression analysis was again used to obtain relative power models with units of *mW*. The screen backlight power models for the two devices are shown in Figs. 7(c) and (d).

4. Experimental methodology

We implemented our *AURA* framework on the actual HTC Dream and Google Nexus One mobile devices. The CPU and backlight power models described above were integrated into the *AURA* framework, to guide our MDP and Q-Learning based power management strategies and also to quantify any resulting energy savings. To test the effectiveness of the *AURA* framework, we obtained stimuli from two sources:

- **Event emulator:** This module emulates various key and touch based user interaction event patterns, including: (i) *Constant*—sends events at a user-defined constant rate; (ii) *Stochastic*—sends events using a Gaussian distribution probability with a user-defined mean and standard deviation; (iii) *Random*—sends events using a pseudo-random uniform distribution; and (iv) *High Burst Long Pause (HBLP)*—sends a quick burst of 20 events, followed by a long 10 s pause.

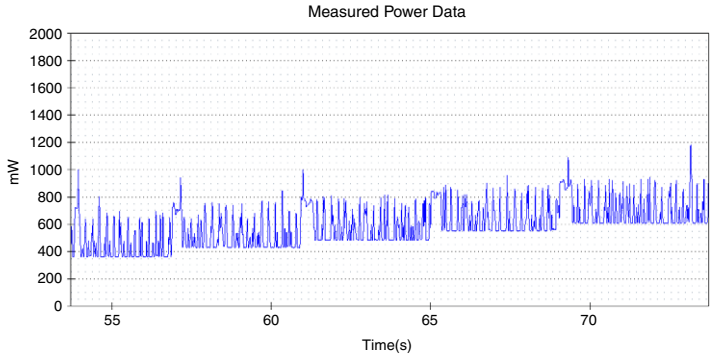

Fig. 8. Monsoon power monitor screen power.

- **Real users:** We involved five users who interacted with applications on their Android mobile devices as they would normally, generating user-specific interaction events in the process.

The algorithm control parameter values assumed in our implementation are shown in Table 6.

5. Experimental results

5.1. Event emulation results

In our first experimental study, the event emulator was used to imitate four different user-interaction patterns, to contrast the effectiveness of our four power management algorithms. As a first step, we were primarily interested in the energy savings from our algorithms compared to the default DFS driver (*on-demand*) and default screen backlight settings on the HTC Dream and Google Nexus One devices, in which the backlight was not controlled by an ambient light sensor. Ambient light sensors are not available on many devices, and many users choose not to enable adaptive backlight control even if it is available. Moreover, our user-interaction-aware framework could be implemented in conjunction with adaptive backlight control to potentially offer even greater energy savings. For example, the chosen default brightness could be a function of ambient light, and then *AURA*'s power management algorithms could ramp down the screen brightness from there depending on user interaction.

To evaluate the level of QoS being offered, a successful prediction rate metric was defined. The *successful prediction rate* metric keeps track of the number of events that occurred while in *Below Nominal* state. A low successful prediction rate is indicative of diminished QoS. QoS is subjective and difficult to quantify at a user level, and this is well known—two users may perceive a common event very differently. Therefore, the successful prediction rate is merely an attempt at measuring user satisfaction.

To discover how our prediction rate metric was correlated with actual user satisfaction, we modified the control parameters of the *NORM* algorithm to offer different successful prediction rates. We then asked users to interact with eight common Android apps with the algorithm running and recorded their level of satisfaction on a scale of 1 to 5 according to the following criteria:

1. *Very satisfied:* app functions normally and screen brightness changes are not intrusive.
2. *Somewhat satisfied:* very little lag noticeable; screen brightness changes noticeable but not overly distracting.
3. *Mediocre:* app functions slower than usual; screen brightness is intrusive at times; app is still enjoyable to use.
4. *Somewhat dissatisfied:* lag and screen brightness changes very noticeable, but not to the point of ruining app functionality.
5. *Very dissatisfied:* app runs frustratingly slow; screen brightness changes make viewing difficult; app is rendered useless.

We define the *minimum acceptable prediction rate* for an application to be the prediction rate that does not cause **noticeable degradation** in the performance of an app, and that offers a user satisfaction level of 1 or 2 on the above scale. Fig. 9 shows the minimum acceptable prediction rates for different applications. Evidently, different applications can tolerate different minimum acceptable prediction rates. Thus, in an attempt to offer acceptable QoS across all applications, we define a *performance threshold* of 80%, which is a conservative average of the eight apps' minimum acceptable prediction rates. The goal is to tune our power management algorithms so that they never offer prediction rates below the performance threshold.

Fig. 10(a) and (b) shows the energy savings achieved by the four power management algorithms we propose in this work, for the four emulated interaction patterns on the HTC Dream and Google Nexus One devices. It can be seen that *E-ADAPT* is capable of offering the best energy savings out of the MDP algorithms (up to 20%) on the HTC Dream due to the responsive event-triggered nature of its dynamic adaptation. The algorithm only adapts on instances of user events and does not require constant adaptation during each evaluation interval. This also reveals why *E-ADAPT* performs poorly in conditions like *HBLP*—the quick events during the burst are stored in the *E-ADAPT* window as they occur and used to calculate the mean

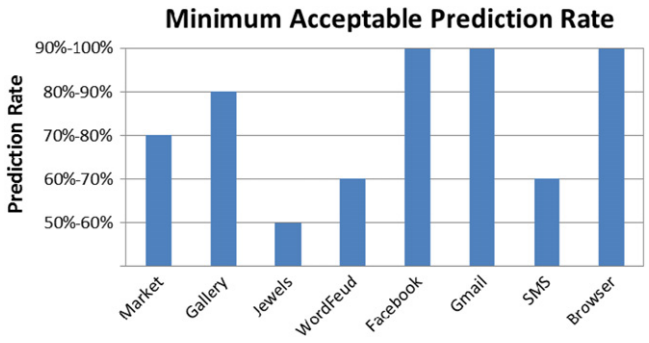

Fig. 9. Minimum acceptable prediction rates for the chosen applications.

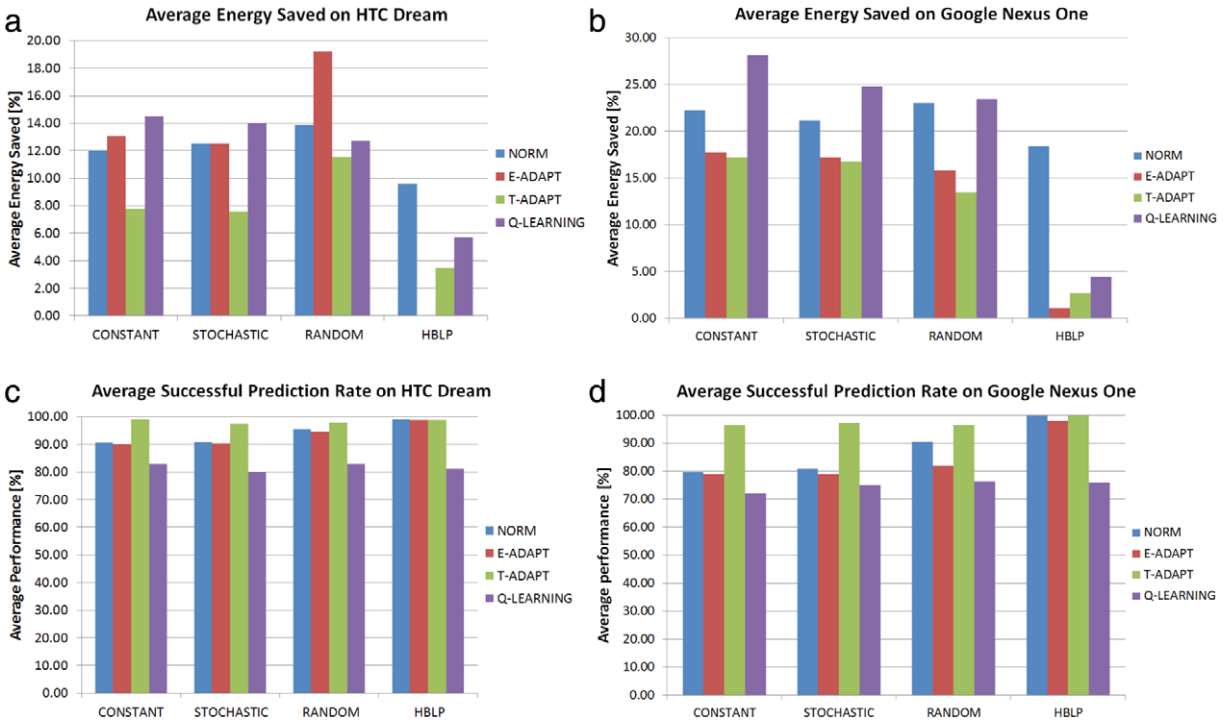

Fig. 10. Event emulation results.

for prediction, but the long pause is not taken into account until the event after the long pause occurs. On the other hand, *T-ADAPT* offers lower energy savings due to its time-based nature. Whereas *E-ADAPT* must wait for an event occurrence in order to adjust its prediction, *T-ADAPT* adjusts on each evaluation interval. This form of adaptation is slightly more computationally intensive. Additionally, if the state is *Nominal*, it will not drop to *Below Nominal* under *T-ADAPT* unless CPU utilization is below U_{HLT} (even if the probability of an event is very low). Because *T-ADAPT* shifts its window and does computations on a regular interval instead of waiting for an event to occur, the CPU utilization is high more often, thus the state transitions to *Below Nominal* less often. *NORM* offers the best overall energy savings.

NORM performs well in certain conditions such as those found in *HBLP* because it takes the average time between all touch events into account, giving the algorithm a more global view than *E-ADAPT* or *T-ADAPT*. When considering energy savings across all four emulated patterns, *Q-LEARNING* is a close second to *NORM*. However, one reason for this is that the algorithm was tested as if the emulated patterns were seen on the initial invocation of an application (i.e., good Q values have not been learned yet), Thus, the higher energy savings may have a negative impact on user QoS.

Fig. 10(c) and (d) shows average successful prediction rates of each algorithm for the event emulation patterns. *Q-LEARNING*'s successful prediction rate is the lowest of the four algorithms. This can be attributed to the fact that the algorithm has not had a chance to learn a good model for the emulated patterns. If the algorithm had more of a chance to learn good models for each pattern, better prediction rates could have been achieved (possibly at the cost of lower energy savings). It is clear that all three MDP algorithms have high successful prediction rates, validating the robustness of our chosen classification and power management algorithms. Interestingly, a higher successful prediction rate does not necessarily

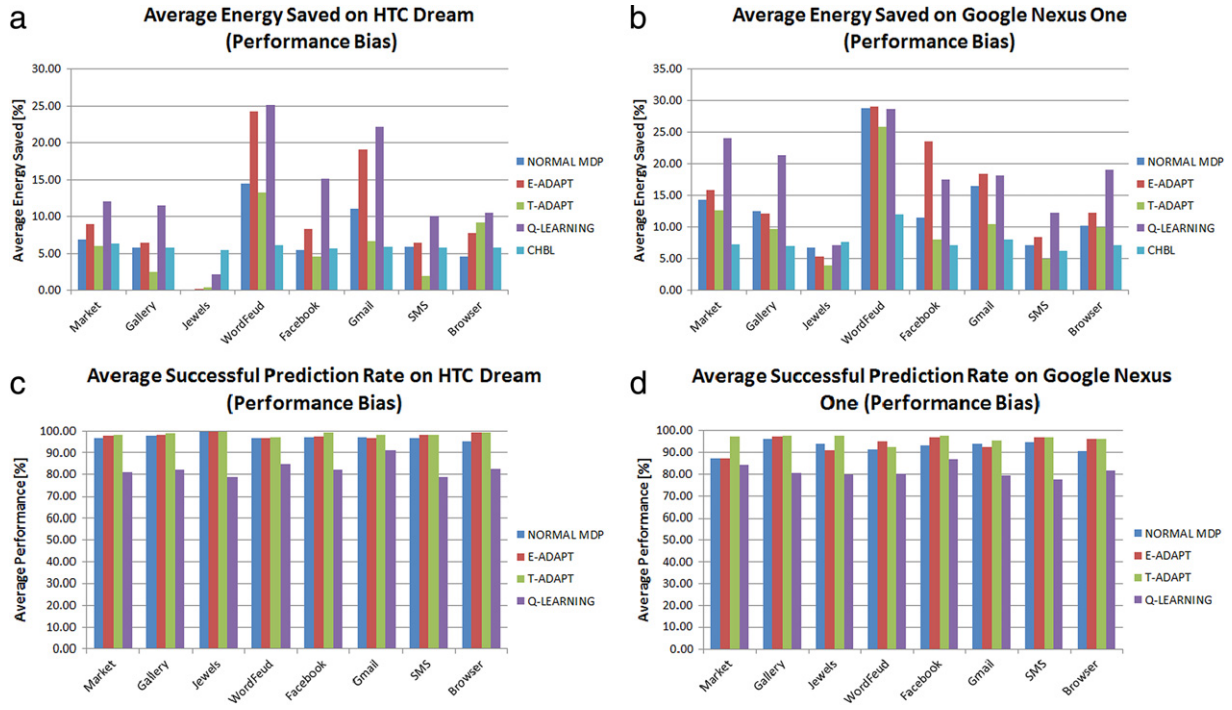

Fig. 11. Real user study results (performance bias).

imply higher energy savings for the following reason: an event may be predicted to occur much earlier than the actual time of its occurrence and the prediction will still be considered successful. For example, if the state is *Below Nominal* and the probability of an event occurrence exceeds P_{LHT}, the state will switch to *Nominal* and remain there until an event occurs and the probability of an event occurrence is low again. Therefore, if an event does not occur quickly, less energy will be saved because the *Nominal* state will be prolonged (but the prediction will still be considered successful). This is the reason why although *T-ADAPT* offers the best successful prediction rates due to its superior temporal locality characteristics, it offers lower energy savings than *NORM* and *E-ADAPT*. The fact is also applicable only to the MDP algorithms, and not to *Q-LEARNING* as it does not wait for an event to occur before switching to *Below Nominal*. It should be noted that in our experiments, the MDP algorithm control parameters were biased towards high successful prediction rates, thus maintaining a high minimum level of user QoS for all three algorithms. The control parameters could potentially be adjusted to accept lower successful prediction rates in order to offer even higher energy savings.

5.2. User study results

In our second experimental study we focused on eight common Android apps and five real users using these apps on the HTC Dream and Google Nexus One mobile devices. The users were asked to interact with the different applications over the course of several sessions during a day, with different energy saving algorithms enabled in each session. In addition to our MDP based and Q-Learning based power management algorithms running as part of the *AURA* middleware framework, we implemented the framework proposed by Shye et al. [3] that also aims to improve CPU and backlight energy consumption for mobile devices, by using the phenomenon of change blindness to gradually ramp down both CPU frequency and screen backlight levels over time to save energy.

Fig. 11(a) and (b) compares the average energy savings of the technique proposed by Shye et al. (*CHBL*) with our four power management techniques for the eight different applications, across all users. It can be seen that *CHBL* offers fairly consistent (but low) energy savings across all applications. The consistency can be explained by noting that the change blindness optimization employed in *CHBL* is independent of user interaction patterns and application type, instead using constant time triggered scaling of the CPU frequency and backlight levels. In contrast, our user-aware and application-aware algorithms (specifically *E-ADAPT* and *Q-LEARNING*) offer higher energy savings because they can dynamically adapt to the user interaction patterns and take full advantage of user idle time. For instance, the *E-ADAPT* and *Q-LEARNING* algorithms save up to 4× and 5× more energy compared to *CHBL* for the *Gmail* and *WordFeud* applications on the HTC Dream, respectively. On the Nexus One, *E-ADAPT* and *Q-LEARNING* together save an average of approximately 2.5× more energy than *CHBL* across all applications. In some cases, for instance the high interaction *Jewels* application, interaction slack is too small to be exploited by our algorithms. While *CHBL* offers higher energy savings for *Jewels* due to its lack of user-awareness and application-awareness during scaling, it comes at a cost: noticeable QoS degradation issues for the user.

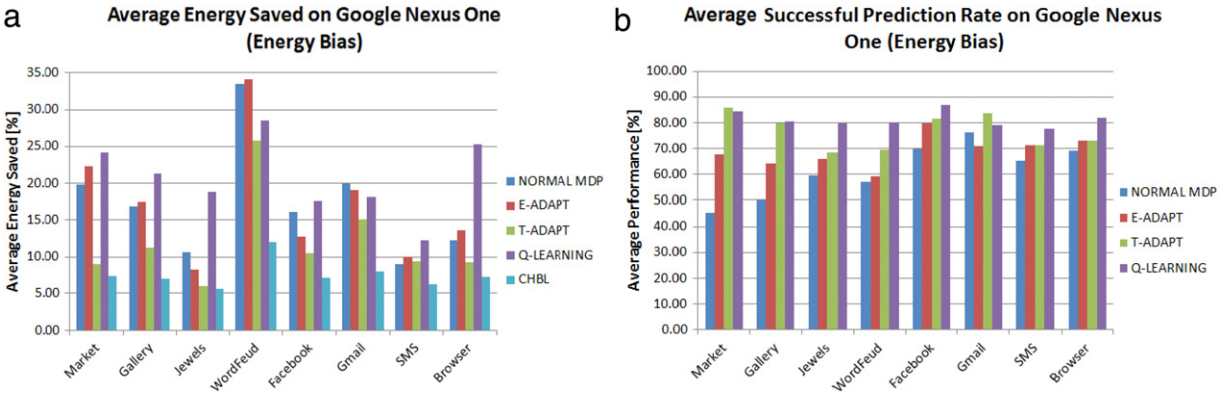

Fig. 12. Real user study results (energy bias).

Fig. 11(c) and (d) shows the average successful prediction rates for the real user interaction patterns. Shye et al.'s algorithm, *CHBL*, was not included because it does not contain defined states or prediction mechanisms, making determining mispredictions impossible. The figures show high successful prediction rates similar to those that were seen in the event emulation results. The reason for this is that the algorithm control parameters used for the results in Fig. 11 were biased towards maintaining fewer mispredictions.

In a complementary analysis study, we adjusted the algorithm control parameters for the MDP algorithms to allow even higher energy savings to see how the number of mispredictions was affected. The event probability thresholds P_{HLT} and P_{LHT} were increased by 20%, and the processor utilization thresholds U_{HLT}, U_{LMT}, and U_{LHT} were increased by 10% from their original values in Table 6. Fig. 12 shows the results of our four algorithms with the adjusted algorithm control parameters on the Google Nexus One compared with *CHBL*, which is unchanged. Fig. 12(a) shows that energy savings are clearly higher with the adjusted parameters, with *E-ADAPT* and *Q-LEARNING* offering energy savings of up to 34% over default device settings. However, the successful prediction rates for the MDP algorithms in Fig. 12(b) have dropped significantly. In fact, most of the prediction rates are below the acceptable performance threshold of 80%, making this configuration unacceptable. Overall, our proposed *AURA* framework with the *E-ADAPT* or *Q-LEARNING* schemes enables as much as 29% energy savings over the baseline device settings for frequency and screen backlight without noticeably degrading user QoS, and improves upon prior work (*CHBL* [3]) by as much as 5× for some applications and devices.

5.3. Implementation overhead

When considering the feasibility of an energy optimization framework such as *AURA*, it is important to consider the overhead costs associated with its implementation. The energy savings results presented in the previous sections included the energy consumption overhead of our framework. In other words, the energy saved by our framework significantly outweighs the energy consumed by it. Because our framework runs in the background as an Android service, it does not require any extraneous run-time. Also, the framework does not become more complex, even when hundreds of new applications are installed, because the power management algorithms focus on optimizing only the foreground application. As the user invokes new applications while running the framework, the size of the application database will increase, but the amount of information stored is insignificant (well under 1 kB per app, depending on the number of collected samples) when considering the storage capabilities of modern mobile devices.

6. Related work

A significant amount of work has been done in the area of energy optimization for mobile devices. Much of this work focuses on optimizing energy consumed by the device's wireless interfaces (e.g., WiFi, 3G/EDGE networks). In [5] an energy-aware handoff algorithm is proposed based on the energy consumption of UMTS (3G) and WiFi. The energy costs for transmitting data over different wireless interfaces are explored by Ra et al. [6], with the goal of balancing energy and delay during data transfers. In [27], the energy consumption of various wireless interfaces (WiFi, 2G, 3G) is measured, with the goal of finding the most energy-efficient interface for opportunistic offloading of tasks to the cloud. In [28], a similar idea is used to offload computations from mobile devices using wireless links to save energy. In [29], an optimized design of a radio frequency energy harvesting antenna is presented, with potential applications to mobile devices. Other work [7–9,30–36] focuses on energy-efficient location-sensing schemes aiming to reduce high battery drain caused by location interfaces (e.g., WiFi, GPS). Our work in [37] uses machine learning techniques to exploit the spatiotemporal and device contexts of users to predict energy-efficient mobile device interface configurations. Lee et al. [38] propose a Variable Rate Logging (VRL) mechanism that disables location logging or reduces the GPS logging rate by detecting if the user is standing still or indoors. Nishihara et al. [39] propose a context-aware method to determine the minimum set of resources

(processors and peripherals) that result in meeting a given level of performance. A framework for mobile sensing that efficiently recognizes contextual user states is proposed by Wang et al. [10]. Our work on optimizing the backlight and CPU energy is complementary to these works. While we do not optimize energy consumed by the location sensor and wireless interface, *AURA* can potentially be implemented alongside other strategies that do.

There have been some efforts to optimize CPU and backlight energy consumption in recent years. Shye et al. [3] in particular make several important observations that are relevant to this work: (1) the screen and the CPU are the two largest power consuming components, and (2) users are generally more satisfied with a scheme that gradually reduces screen brightness rather than one that changes abruptly. Based on the second observation, the authors implement a scheme that utilizes *change blindness* [17,18] by slowly reducing screen brightness and CPU frequency over time. However, their approach does not consider application-aware and user-aware scaling as *AURA* does, because of which their approach tends to be overly conservative at times, and at other times detrimentally impacts user QoS. Other researchers have experimented with screen optimizations that would be enabled with OLED display technology by altering the user interface to dim certain parts of the screen to save considerable power, then conducting user acceptance studies [40,41]. Dong et al. [11] present power models for OLED displays at pixel, image, and code level, then propose techniques that optimize GUI power consumption. Bi et al. propose a user interaction-aware DFS approach in [4] much like *AURA*. However, their approach is directed towards stationary desktop systems, and not towards the touch-based mobile devices that are becoming more prevalent today. Their proposed approach is also different in that it does not consider backlight optimization, and uses a simple user-aware but application-unaware history predictor to set CPU frequency levels based on demand. PICSEL [42] is another user-aware CPU DFS scheme that is directed towards desktop and laptop machines. A few methods [12,14] involve allowing a mobile device to dynamically enter low-power sleep states by predicting idle periods, much like *AURA*. But these techniques are conservative in the savings they achieve as they do not exploit unique user characteristics as AURA does. Still other DFS schemes are application-aware and observe application behavior with respect to CPU utilization and memory references to construct statistical models and apply voltage/frequency schedules based on these models [13,43,44], or detect regions of low CPU utilization and insert instructions to control the switching of voltage/frequency levels [45–48]. Tan et al. [49] present a power management technique based on reinforcement learning that attempts to optimize power consumption for a given performance constraint.

A few efforts perform software application optimization for energy efficiency. In [50], strategies for appropriate *wakelock* placement in apps are explored, to improve energy-efficiency of component utilization in mobile devices. In [51], a system is proposed for energy-efficient production in small and medium enterprise environments by fusing data from sensors and enabling energy-efficient decision making via mobile devices. A great deal of research has also been dedicated to mobile usage studies to understand how users interact with their phones in the real world. In [52] the authors demonstrated from a two month smartphone usage study that all users have unique device usage patterns. Many other works [3,53–59] utilize data gathered from groups of real smartphone users to emphasize this same conclusion. The conclusions of these studies are in line with those of our user studies that we performed as part of this work, as well as in our prior work [37].

7. Conclusions and future work

In this work we proposed *AURA*, an application-aware and user-interaction-aware energy optimization framework for pervasive mobile devices. As part of *AURA*, we introduced a novel method for classifying applications based on user interaction patterns using Bayesian classification. We also introduced novel MDP and Q-Learning based power management algorithms that can be specifically tailored to the needs of each classified application. We evaluated the framework on both emulated and real user interaction patterns, on real mobile devices, and found that *AURA* can achieve average energy savings of up to 29% depending on the mobile device and application, without perceptible impact on user QoS. When compared with prior work on CPU and backlight power savings in mobile devices, *AURA* can achieve up to 5× of their energy savings for certain applications. Lastly, because we showed that our proposed framework is able to take the high level of diversity exhibited in our user studies into account, we conclude that it is applicable beyond the users we studied, to a vast user base with varying usage patterns.

A possible extension to our work is to conduct large scale studies that recruit larger sample groups than that considered in this work, and monitor them for an extended period of time. Such large scale studies could lead to a better understanding of unique user interaction patterns, which in turn could allow for more aggressive optimization of our framework.

Acknowledgments

This research is sponsored in part by grants from NSF (CCF-1252500) and the Semiconductor Research Corporation (SRC).

References

[1] MobiThinking, Global mobile statistics 2013 Part A: Mobile subscribers; handset market share; mobile operators, August 2013. http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats/a#subscribers.

[2] Micro Power White Paper, Using Lithium Polymer Batteries in Mobile Devices. http://www.micro-power.com/userfiles/file/wp_polymer_final-1274743697.pdf [Online].

[3] A. Shye, B. Scholbrock, G. Memik, Into the wild: Studying real user activity patterns to guide power optimizations for mobile architectures, in: MICRO-42'09, January 2009, pp. 168–178.
[4] M. Bi, I. Crk, C. Gniady, IADVS: On-demand performance for interactive applications, in: HPCA'10, April 2010, pp. 1–10.
[5] H. Petander, Energy-aware network selection using traffic estimation, in: MICNET'09, September 2009, pp. 55–60.
[6] M. Ra, J. Paek, A.B. Sharma, R. Govindan, M.H. Krieger, M.J. Neely, Energy-delay tradeoffs in smartphone applications, in: MOBISYS'10, June 2010, pp. 255–270.
[7] I. Constandache, S. Gaonkar, M. Sayler, R.R. Choudhury, L. Cox, EnLoc: Energy-efficient localization for mobile phones, in: INFOCOM'09, June 2009, pp. 19–25.
[8] K. Lin, A. Kansal, D. Lymberopoulos, F. Zhao, Energy-accuracy trade-off for continuous mobile device location, in: MOBISYS, June 2010, pp. 285–298.
[9] Z. Zhuang, K. Kim, J.P. Singh, Improving energy efficiency of location sensing on smartphones, in: MOBISYS, 2010, pp. 315–330.
[10] Y. Wang, J. Lin, M. Annavaram, Q.A. Jacobson, J. Hong, B. Krishnamachari, N. Sadeh, A framework of energy efficient mobile sensing for automatic user state recognition, in: MOBISYS'09, 2009, pp. 179–192.
[11] M. Dong, L. Zhong, Power modeling and optimization for OLED displays, in: TMC 2012, September 2012, pp. 1587–1599.
[12] A.W. Min, R. Wang, J. Tsai, M.A. Ergin, T.C. Tai, Improving energy efficiency for mobile platforms by exploiting low-power sleep states, in: CF'12, 2012, pp. 133–142.
[13] M. Yang, Y. Wen, J. Cai, C.H. Foh, Energy minimization via dynamic voltage scaling for real-time video encoding on mobile devices, in: ICC'12, June 2012, pp. 2026–2031.
[14] L. Huang, Optimal sleep-wake scheduling for energy harvesting smart mobile devices, in: WiOpt'13, May 2013, pp. 484–491.
[15] B.K. Donohoo, C. Ohlsen, S. Pasricha, AURA: An application and user interaction aware middleware framework for energy optimization in mobile devices, in: IEEE International Conference on Computer Design, ICCD, October 2011.
[16] S.K. Card, T.P. Moran, A. Newell, The Psychology of Human–Computer Interaction, Lawrence Erlbaum Assoc., Hillsdale, NJ, 1983.
[17] D.J. Simons, S.L. Franconeri, R.L. Reimer, Change blindness in the absence of a visual disruption, Perception 29 (2000) 1143–1154.
[18] D.J. Simons, R.A. Rensink, Change blindness: Past, present, and future, Trends Cogn. Sci. 9 (1) (2005).
[19] Google Android, official website, 2013. http://www.android.com.
[20] E. Alpaydin, Introduction to Machine Learning, second ed., The MIT Press, Massachusetts, 2010.
[21] H. Jung, M. Pedram, Improving the efficiency of power management techniques by using Bayesian classification, in: ISQED'08, March 2008, pp. 178–183.
[22] T.L. Cheung, K. Okomoto, F. Maker, X. Liu, V. Akella, Markov decision process (MDP) framework for optimizing software on mobile phones, in: EMSOFT'09, October 2009, pp. 11–20.
[23] C. Anderson, Introduction to Reinforcement Learning. Website, August 2013. http://www.cs.colostate.edu/~anderson/cs545/index.html/doku.php?id=notes:notes10b.
[24] Monsoon Solutions Inc., official website, 2008. http://www.msoon.com/LabEquipment/PowerMonitor.
[25] D. Brodowski, Linux CPUFreq—CPUFreq Governors. Linux Documentation, March 2011. http://mjmwired.net/kernel/Documentation/cpu-freq.
[26] J.J. Faraway, Linear Models with R, CRC Press, 2004.
[27] M. Segata, B. Bloessl, C. Sommer, F. Dressler, Towards energy efficient smart phone applications: Energy models for offloading tasks into the cloud, in: IEEE International Conference on Communications, ICC, 2014.
[28] A. Aucinas, J. Crowcroft, Demo: PhoneLets: Offloading the phone off your phone for energy, cost and network load optimization, in: Proceedings of the 20th Annual ACM International Conference on Mobile Computing and Networking, Mobicom, 2014.
[29] T. Ajmal, V. Dyo, B. Allen, D. Jazani, I. Ivanov, Design and optimisation of compact RF energy harvesting device for smart applications, Electron. Lett. 50 (2) (2014) 111–113.
[30] F.B. Abdesslem, A. Phillips, T. Henderson, Less is more: Energy-efficient mobile sensing with SenseLess, in: MOBIHELD'09, August 2009, pp. 61–62.
[31] J. Paek, J. Kim, R. Govindan, Energy-efficient rate-adaptive GPS-based positioning for smartphones, in: MOBISYS'10, June 2010, pp. 299–314.
[32] I. Shafer, M.L. Chang, Movement detection for power-efficient smartphone WLAN localization, in: MSWIM'10, October 2010, pp. 81–90.
[33] M. Youssef, M.A. Yosef, M. El-Derini, GAC: Energy-efficient hybrid GPS-accelerometer-compass GSM localization, in: GLOBECOM'10, December 2010, pp. 1–5.
[34] R. Jurdak, P. Corke, D. Dharman, G. Salagnac, Adaptive GPS duty cycling and radio ranging for energy-efficient localization, in: SENSYS'10, November 2010, pp. 57–70.
[35] H. Lu, J. Yang, Z. Liu, N.D. Lane, T. Choudhury, A.T. Campbell, The Jigsaw continuous sensing engine for mobile phone applications, in: SENSYS'10, November 2010, pp. 71–84.
[36] Y. Liu, S. Lu, Y. Liu, COAL: Context aware localization for high energy efficiency in wireless networks, in: WCNC'11, May 2011, pp. 2030–2035.
[37] B.K. Donohoo, C. Ohlsen, S. Pasricha, C. Anderson, Exploiting spatiotemporal and device contexts for energy-efficient mobile embedded systems, in: DAC'12, June 2012, pp. 1274–1279.
[38] C. Lee, M. Lee, D. Han, Energy efficient location logging for mobile device, in: SAINT'11, October 2010, p. 84.
[39] K. Nishihara, K. Ishizaka, J. Sakai, Power saving in mobile devices using context-aware resource control, in: ICNC'10, 2010, pp. 220–226.
[40] L. Bloom, R. Eardley, E. Geelhoed, M. Manahan, P. Ranganathan, Investigating the relationship between battery life and user acceptance of dynamic, energy-aware interfaces on handhelds, in: HCI'09, pp. 13–24, September 2004.
[41] T. Harter, S. Vroegindeweij, E. Geelhoed, M. Manahan, P. Ranganathan, Energy-aware user interfaces: An evaluation of user acceptance, in: CHI'04, April 2004, pp. 199–206.
[42] A. Mallik, J. Cosgrove, R. Dick, G. Memik, P. Dinda, PICSEL: Measuring user-perceived performance to control dynamic frequency scaling, in: ASPLOS'08, March 2008.
[43] K. Choi, R. Soma, M. Pedram, Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to on-chip computation times, in: TCAD'04, 2004.
[44] A. Weissel, F. Bellosa, Process cruise control: Event-driven clock scaling for dynamic power management, in: CASES'02, 2002.
[45] C.H. Hsu, U. Kremer, Single region vs. multiple regions: A comparison of different compiler-directed dynamic voltage scheduling approaches, in: Workshop on PACS, 2002.
[46] S. Lee, T. Sakurai, Run-time voltage hopping for low-power real-time systems, in: DAC'00, 2000.
[47] D. Shin, J. Kim, S. Lee, Intra-task voltage scheduling for low-energy, hard real-time applications, IEEE Des. Test. (2001).
[48] F. Xie, M. Martonosi, S. Malik, Compile-time dynamic voltage scaling settings: Opportunities and limits, in: PLDI'03, 2003.
[49] Y. Tan, W. Liu, Q. Qiu, Adaptive power management using reinforcement learning, in: ICCAD'09, November 2009, pp. 461–467.
[50] F. Alam, P.R. Panda, N. Tripathi, N. Sharma, Energy optimization in Android applications through wakelock placement, in: IEEE Design, Automation and Test in Europe Conference and Exhibition, DATE, 2014.
[51] U. Laufs, C. Ruff, J. Zibuschka, Mobile support for energy-saving production, in: Proc. Mobility in a Globalised World 2012, pp. 175–179.
[52] J. Kang, S. Seo, J.W. Hong, Usage pattern analysis of smartphones, in: APNOMS'11, November 2011, pp. 1–8.
[53] J. Froehlich, M.Y. Chen, S. Consolvo, B. Harrison, J.A. Landay, MyExperience: A system for in situ tracing and capturing of user feedback on mobile phones, in: MOBISYS, 2007, pp. 57–70.
[54] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, D. Estrin, Diversity in smartphone usage, in: MobiSys'07, 2010, pp. 179–194.
[55] C. Shepard, A. Rahmati, C. Tossell, L. Zhong, P. Kortum, LiveLab: Measuring wireless networks and smartphone users in the field, in: HotMetrics, June 2010, pp. 1–6.
[56] C. Tossell, P. Kortum, A. Rahmati, C. Shepard, L. Zhong, Characterizing Web use on smartphones, in: CHI, May 2012, pp. 2769–2778.
[57] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, O. Spatscheck, Profiling resource usage for mobile applications: A cross-layer approach, in: MobiSys, June 2011, pp. 321–334.
[58] J. Eberle, G.P. Perrucci, Energy measurements campaign for positioning methods on state-of-the-art smartphones, in: CCNC'11, May 2011, pp. 937–941.
[59] N. Balasubramanian, A. Balasubramanian, A. Venkataramani, Energy consumption in mobile phones: A measurement study and implications for network applications, in: IMC'09, November 2009, pp. 280–293.