Rethink Energy Accounting with Cooperative Game Theory

Mian Dong
Rice University
dongmian@gmail.com

Tian Lan
George Washington University
tlan@gwu.edu

Lin Zhong
Rice University
lzhong@rice.edu

ABSTRACT

Energy accounting determines how much a software principal contributes to the total system energy consumption. It is the foundation for evaluating software and for operating system based energy management. While various energy accounting policies have been tried, there is no known way to evaluate them directly simply because it is hard to track all hardware usage by software in a heterogeneous multicore system like modern smartphones and tablets.

In this work, we argue that energy accounting should be formulated as a cooperative game and that the Shapley value provides the ultimate ground truth for energy accounting policies. We reveal the important flaws of existing energy accounting policies based on the Shapley value theory and provide Shapley value-based energy accounting, a practical approximation of the Shapley value, for battery-powered mobile systems. We evaluate this approximation against existing energy accounting policies in two ways: (*i*) how well they identify the top energy consuming applications, and (*ii*) how effective they are in system energy management. Using a prototype based on Texas Instruments Pandaboard and smartphone workload, we experimentally demonstrate existing energy accounting policies can deviate by 400% in attributing energy consumption to running applications and can be up to 25% less effective in system energy management when compared to Shapley value-based energy accounting.

Categories and Subject Descriptors

D.4.1 [**Operating Systems**]: Process Management

Keywords

Energy accounting; Energy management; Mobile systems

1. INTRODUCTION

The ability to account system resource usage by software is key to the design and operation of computers. While modern computers can account CPU and memory usage by software easily, they can not yet do so for energy, an increasingly important system resource due to electricity and thermal concerns. We use *software principals* to denote the software entities to which energy consumption has to be attributed. Examples of software principal include applications, processes, threads, and tasks. Given the system energy consumption E and the set of software principals $\mathbb{N} = \{1, 2, \ldots, n\}$ that are active during time interval T, *energy accounting* determines the energy contribution of each software principal in $\mathbb{N}$. Energy accounting is the foundation for evaluating software by their energy use, e.g., [23, 19], and for operating system (OS) based energy management that schedules processes for tradeoffs between system energy consumption and process utility, e.g., [32, 25].

Energy accounting is a long-standing hard problem for multiprocessing systems in which multiple software principals can be active at the same time. In the past decade, many energy accounting policies have been tried for various applications as exemplified recently by [33, 23, 19, 31], including the "Battery" feature in Android. However, there is no known *ground truth* against which a policy could be evaluated. The fundamental reason is that it is practically infeasible to track how software uses each hardware component in a heterogeneous multicore system like modern mobile devices. Usually, only the total system energy consumption can be measured, and the usage of only a few hardware components, e.g., CPU and memory, can be attributed to a software principal.

In this work, we explore cooperative game theory for energy accounting, reveal the important flaws of existing policies, and suggest a novel policy that does not have these flaws. Our key insight is that *energy accounting can be formulated as a cooperative game theory problem*: when multiple players participate in a game and the game produces a surplus, how should that surplus be divided among the players? Note that "cooperative" implies that the players can influence each other via both collaboration and competition. In energy accounting, the game is the system during time interval T, players software principals, and energy consumption the surplus. We show that the Shapley value [27], a single value solution to this problem, provides the theoretical ground truth for energy accounting. We show how existing accounting policies violate the four self-evident properties that uniquely define the Shapley value.

We further provide a practical approximation of the Shapley value, called *Shapley value-based energy accounting*, for battery-powered mobile devices. It is infeasible to obtain the exact Shapley value in practice because calculating it requires the system energy consumption for all possible combinations of software principals under question, i.e., $E(\mathbb{S})\ \forall\ \mathbb{S} \subseteq \mathbb{N}$. Our Shapley value-based energy accounting acquires system energy consumption *in situ* for very short time intervals, down to 10 ms, uses heuristics to estimate $E(\mathbb{S})$ for unobserved $\mathbb{S}$, and extend the Shapley value theory to work with partially defined cooperative games and non-deterministic energy consumption $E(\mathbb{S})$. We report a prototype implementation

MobiCom'14, September 7-11, 2014, Maui, Hawaii, USA.
Copyright 2014 ACM 978-1-4503-2783-1/14/09 ...$15.00.
http://dx.doi.org/10.1145/2639108.2639128.

based on Google Android, Texas Instruments PandaBoard ES with OMAP4460, and MAXIM DS2756 battery fuel gauge.

Using this prototype, we experimentally demonstrate how erroneous existing energy accounting policies can be (up to 400%), how little difference in energy management can they make, and that Shapley value-based energy accounting outperforms other energy accounting policies in energy management by up to 25%. Our results also show that sophisticated energy accounting policies used in recent work are not better than very simple ones as long as they guarantee that energy contributions from all software principals add up to the total system energy consumption. Our results mandate a rethinking of energy accounting and its application in energy management. Not only can existing energy accounting policies be off-the-mark by a large degree (now we know why many Android users feel the “Battery” feature is inaccurate), but also existing budget-based energy management may be unnecessarily complicated yet underperforming. While our prototype and evaluation are based on battery-powered mobile systems, we believe most of our results are directly applicable to other computer systems as long as fairly high-frequency system energy measurements are available.

In summary, we make the following contributions to energy accounting in both theory and practice.

- We show that energy accounting should be formulated as a cooperative game and that the Shapley value provides the theoretical ground truth. We reveal how existing accounting policies fail the axiomatic properties required by this ground truth.
- We provide Shapley value-based energy accounting, a practical approximation of the Shapley value, for battery-powered mobile devices. Using a prototype realization, we demonstrate our approximation is superior to existing energy accounting policies experimentally.

The rest of this paper is organized as follows. Section 2 provides necessary background and introduces existing energy accounting policies. Section 3 presents the Shapley value as ground truth for energy accounting. Section 4 describes key techniques to acquire system energy consumption in short time intervals *in situ* in order to realize Shapley value-based energy accounting. Section 5 and Section 6 present system implementation and evaluation, respectively. Section 7 surveys related work. Section 8 discusses how this work can be extended and concludes the paper.

2. BACKGROUND

We first provide some background to the problem and describe existing energy accounting policies.

2.1 Problem Definition and Notations

Energy accounting determines how much a software principal contributes to the system energy consumption. Let T denote the time during which the system energy consumption E must be attributed; $\mathbb{N} = \{1, 2, \ldots, n\}$ denote the software principals that are active during this time. An energy accounting policy determines the contribution ϕ_i by process $i \in \mathbb{N}$ to the system energy consumption E. Mathematically a policy is a family of mapping $\{\phi_i(E(\mathbb{N})), \forall i \in \mathbb{N}\}$ from total energy consumption $E(\mathbb{N})$ to per-software principal energy consumptions ϕ_i.

A self-evident property that an energy accounting policy must have is: $\sum_i \phi_i = E(\mathbb{N})$. That is, the sum of energy contributions by all software principals must be the same as the total system energy consumption. We call this property *Efficiency* according to [27].

Table 1 summarizes the main notations used in this paper.

Table 1: Notations

Symbol	Description
T	Time in which system energy consumption is under question
$\mathbb{N}$	A set of n software principals that are active during T
m	Max number of concurrent software principals allowed
$\mathbb{S}$	A subset of $\mathbb{N}$; a *coalition* of software principals
$E(\mathbb{S})$	System energy consumption in T if only $\mathbb{S}$ are active
ϕ_i	Contribution by software principal i to $E(\mathbb{N})$
σ	System state that is out of the control of the processes
$E(\mathbb{S}, \sigma)$	System energy consumption when $\mathbb{S}$ are active with state σ

2.2 Existing Energy Accounting Policies

We next discuss the major types of energy accounting policies reported in the literature.

Policy I: $\phi_i = E(\mathbb{N})/|\mathbb{N}|$. This policy states that the energy contributed by software principal i is equal to the total energy consumption divided by the number of software principals. That is, each software principal gets an equal share of the total energy consumption.

Policy II: $\phi_i = E(\{i\})$. This policy states that the energy contributed by software principal i is the same as its stand-alone energy consumption $E(\{i\})$ when only software principal i is running in the system. WattsOn [19] utilizes such policy to obtain the energy consumption of each mobile application to enable third-party app developers to improve energy efficiency of their products.

Policy III: $\phi_i = E(\mathbb{N}) - E(\mathbb{N} \setminus \{i\})$. This policy states that the energy contributed by software principal i is equal to its marginal energy contribution, which is the difference between system energy consumption when software principal i is running and when it is not running, while all other conditions remain unchanged [34].

Policy IV: The energy contributed by software principal i is equal to the sum of system energy consumption over all scheduling intervals when software principal i is scheduled in CPU. PowerScope [12] used this policy to account energy consumption of each program running in a system. Obviously, this approach will work only if one software principal can be running at the same time.

Policy V: This sophisticated policy relies on a system energy model $E = f(x_1, x_2, ..., x_p)$ where $x_k, k = 1, 2, ...p$, are predictors that can be obtained in software such as CPU and memory usage. The policy figures out how much software principal i contributes to each of the predictors, $x_{k,i}$ and determines the energy contribution by i by plugging $x_{k,i}$ back into f, i.e., $\phi_i = f(x_{1,i}, x_{2,i}, \ldots, x_{k,i}, \ldots, x_{p,i})$. To have the Efficiency property described above, however, f must be a linear function, i.e., $E = \beta_0 + \sum_{k=1}^{p} \beta_k \cdot x_k$ and there must be a heuristic to split the constant β_0 among the software principals. Variants of Policy V have been used by Android [1], PowerTutor [2] and AppScope [31] for smartphones, by ECOSysem [32] for laptops, by Joulemeter [14] and Power Container [28] for servers, and by Quanto [13] for sensor networks. Policy V, however, is fundamentally limited because how it determines the energy contribution by a software principal depends on how it estimates the system energy consumption. In particular, the estimation can only consider resource usage that can be attributed to individual software principals; and the estimation must employ a linear model. These two limitations can be very problematic for modern mobile systems where many hardware components including some major energy consumers are invisible to the OS [20], e.g., system bus and graphics, and others whose usage cannot be easily attributed to a software principal, e.g., GPS and display. Moreover, it has been shown that linear models are inadequate for estimating system energy consumption [17, 8]. As a result, Policy V can not only significantly underestimate the energy contribution by a software principal but also fail to attribute a significant portion of system energy consumption to running software principals. Due to the lack of ground truth for energy accounting, the problems of Policy V have never been quantified.

In the rest of the paper, we will reveal the fundamental flaws of these energy accounting policies analytically (Section 3.3) and quantitatively (Section 6.3).

3. SHAPLEY VALUE AS GROUND TRUTH

We now introduce the Shapley value as the ground truth for energy accounting. We will discuss how the theory can be applied to energy accounting and demonstrate how existing energy accounting policies lack one or more of the four properties that uniquely define the Shapley value. In Section 4 and Section 5, we will discuss how to address system challenges of realizing Shapley value-based energy accounting and provide a practical, approximate implementation, respectively.

3.1 Shapley Value

We observe a problem similar to energy accounting has been extensively studied in cooperative game theory: how to determine the contribution of each individual player in a game with multiple players? Consider a cooperative game where a set of n players, $\mathbb{N} = \{1, \ldots, n\}$, collectively generate a grand surplus $v(\mathbb{N})$. The set of players, $\mathbb{N}$, is also called the *grand coalition*. Plainly speaking, "cooperative" here refers to the fact the players are not playing in isolation: they can influence each other via both collaboration and competition. Let $v : 2^n \rightarrow \mathbb{R}$ be a *characteristic function* that describes the payoff $v(\mathbb{S})$ a subset of players $\mathbb{S} \subseteq \mathbb{N}$ can gain by playing the same game. A powerful result proven by Shapley in 1953 [27] defines the Shapley value $\phi_i(v)$ for $i = 1, \ldots, n$ as the *unique* way to distribute the grand surplus $v(\mathbb{N})$ among the n players that satisfies four simple axioms:

Efficiency: The sum of share of all players is the grand surplus, i.e. $\sum_i \phi_i(v) = v(\mathbb{N})$. This is the Efficiency property mentioned in Section 2.2.

Symmetry: Symmetric players receive equal shares, i.e., if $v(\mathbb{S} \cup \{i\}) = v(\mathbb{S} \cup \{j\})\ \forall \mathbb{S} \subseteq (\mathbb{N} \setminus \{i, j\})$, then $\phi_i(v) = \phi_j(v)$. That is, if replacing player i with player j in any coalition does not change the game surplus and vice versa, the shares of the two players should be identical.

Null Player: Player i receives zero share if he does not change the payoff in any coalition $\mathbb{S}$, i.e., if $v(\mathbb{S} \cup \{i\}) = v(\mathbb{S})\ \forall \mathbb{S} \subseteq \mathbb{N}$, then $\phi_i(v) = 0$. That is, if adding a player to any coalition does not change the game surplus, the player should receive zero.

Additivity: The share to player i in a sum of two games is the sum of the allocations to the player in each individual game, i.e., $\phi_i(v) + \phi_i(w) = \phi_i(v + w),\ \forall i$. That is, if we view two games as a single game, the share a player receives from the combined game should be the same as the sum of what he would receive from each of the two original games.

The Shapley value is the unique distribution function that satisfies the above four axioms:

$$\phi_i(v) = \sum_{\mathbb{S} \subseteq \mathbb{N} \setminus \{i\}} \frac{v(\mathbb{S} \cup \{i\}) - v(\mathbb{S})}{(|\mathbb{N}| - |\mathbb{S}|)\binom{|\mathbb{N}|}{|\mathbb{S}|}}, \quad (1)$$

where $|\mathbb{S}|$ is the cardinality of $\mathbb{S}$, i.e., the number of members of $\mathbb{S}$; similarly, $|\mathbb{N}|$ is the number of members of $\mathbb{N}$, or n; $\binom{|\mathbb{N}|}{|\mathbb{S}|}$ is $|\mathbb{N}|$-choose-$|\mathbb{S}|$. The Shapley value captures the *average marginal contribution* of player i, averaging over all the different sequences according to which the grand coalition ($\mathbb{N}$) could be built up from the empty coalition. It has been widely applied for solving benefit/cost sharing problems in diverse fields, including computer science, e.g., [18, 11]. To calculate the Shapley value $\phi_i(v)$, one need not only the grand surplus when all n players play, $v(\mathbb{N})$, but also the surplus when any subset of the n players play, or $v(\mathbb{S})$ for all $\mathbb{S} \subseteq \mathbb{N}$.

3.2 Ground Truth for Energy Accounting

We argue the Shapley value provides a natural and intuitive ground truth for energy accounting. We treat the time interval T in which the system under question consumes energy as the cooperative game; the software principals active in the same time interval $\mathbb{N}$ the players; and total system energy consumption, $E(\mathbb{N})$, the grand surplus to distribute. Thus, the energy contribution ϕ_i by software principal i is given by the Shapley value $\phi_i(v)$ of the game with $v(\mathbb{S}) = E(\mathbb{S})$ $\forall \mathbb{S} \subseteq \mathbb{N}$ in Equation (1). Note that to calculate the Shapley value, one must know the system energy consumption in the same time interval T if only a subset of $\mathbb{N}$ are active during T for all possible subsets, i.e., $E(\mathbb{S})\ \forall \mathbb{S} \subseteq \mathbb{N}$. This poses a significant challenge to the practical use of the Shapley value as the ground truth for energy accounting. In Section 4, we will present techniques that partially overcome this challenge to provide a practical approximation of the ground truth.

Since the Shapley value is uniquely defined by the four axioms, we show that these four axioms are self-evident and logical for energy accounting. (*i*) *Efficiency* requires the sum of the energy contributions by all software principals be equal to the total system energy consumption. In other words, an energy accounting policy must be able to apportion energy consumption among software principals without any residual left. This is the same Efficiency property discussed in Section 2.2. (*ii*) *Symmetry* requires two software principals to be assigned the same energy contribution if replacing one with the other under any circumstances does not change the system energy consumption. (*iii*) *Null Player* requires that if adding a software principal under any circumstances does not change the system energy consumption, the energy contribution by this software principal should be zero. Finally, (*iv*) *Additivity* says that if we break T into multiple time intervals and apply the same accounting policy to them, the energy attributed to a software principal in T should be equal to the sum of energy attributed to it in these shorter time intervals under the same energy accounting policy.

It is important to note that these four axioms are the properties for energy accounting policies; they are not assumptions made for the computer or its software. The Shapley value is the only possible policy that has all four properties and should be considered as the ground truth for energy accounting.

3.3 Existing Policies Against the Shapley Value

While it is infeasible to obtain the Shapley value in practice, we can reveal the flaws of the five types of existing policies against the four axioms that uniquely define the Shapley value. Table 2 summarizes how they violate the four axioms. In Section 6, we will offer quantitative evidence regarding their flaws.

Policy I violates *Null Player* because a software principal always gets positive energy contribution according to it.

Policy II violates *Efficiency*. For example, consider a system with two software principals. According to Policy II, energy contributed by software principal 1, or software principal 2, is equal to its stand-alone energy consumption $E(\{1\})$, or $E(\{2\})$, respectively. However, to satisfy *Efficiency* requires $\phi_1 + \phi_2 = E(\{1\}) +$

Table 2: How existing energy accounting policies violate the axioms that define the Shapley value

Policies	Efficiency	Symmetry	Null Player	Additivity
I			×	
II	×		×	
III	×			×
IV		×	×	
V	×	×	×	

$E(\{2\}) = E(\{1,2\})$. Unfortunately, the last equality does not hold in many cases, e.g., when software principals 1 and 2 share some system resources, one usually observes $E(\{1\})+E(\{2\}) > E(\{1,2\})$.

Policy III violates *Efficiency*. Using the same example above, Policy III requires $\phi_1 = E(\{1,2\}) - E(\{2\})$ and $\phi_2 = E(\{1,2\}) - E(\{1\})$. Thus, the sum of shares is $\phi_1 + \phi_2 = 2E(\{1,2\}) - E(\{1\}) - E(\{2\})$. Again, $E(\{1\}) + E(\{2\}) = E(\{1,2\})$ is necessary for *Efficiency* to hold. The LEA^{2}P platform [26] utilizes an improved Policy III that normalizes all the $\phi_i = E(\mathbb{N}) - E(\mathbb{N} \setminus \{i\})$ to enforce *Efficiency*. However, the normalization will lead to violation of *Additivity* because different normalization factors may be used in different time intervals.

Policy IV does not work for multiprocessing systems. To extend Policy IV for multiprocessing systems, one can attribute energy consumption to software principals based on their CPU usage, as in [3]. Such extension, however, violates *Null Player* because a software principal always has positive CPU usage and thus positive energy contribution according to the policy. It further violates *Symmetry* due to its heavy dependency on timing of energy consumption. For instance, the same software principal may be charged different energy bills when it consumes energy through asynchronous I/O operations and suffers random delays, as mentioned by [32].

Policy V violates the *Efficiency*, *Symmetry*, and *Null Player*. First, the inaccuracy in its energy model as highlighted in Section 2.2 may lead to violation of *Efficiency*. Second, there may exist software principals that make use of different hardware components, but make the same energy contributions in all coalitions. These software principals are indistinguishable with respect to their energy behaviors and therefore should be charged equal energy bills according to *Symmetry*. However, if they make different contributions to the predictors in the used energy model, Policy V would dictate that these software principals make different energy contributions and thus would violate *Symmetry*. Finally, a software principal always gets positive energy contribution because it requires CPU time and system resources so that it always makes positive contributions to the predictors of the energy model.

4. SHAPLEY VALUE-BASED ENERGY ACCOUNTING

While we argue that the Shapley value should be considered as the ground truth for energy accounting, this ground truth is practically impossible to obtain via measurement. The key problem is to obtain $E(\mathbb{S})$ for all $\mathbb{S} \subseteq \mathbb{N}$, which poses three system challenges. (*i*) First, there are 2^n subsets for n software principals. One must obtain $E(\mathbb{S})$ for all the 2^n different coalitions, which can be practically infeasible. (*ii*) Second, $E(\mathbb{S})$ depends on not only which software principals are running, but also the dynamics of software execution such as the CPU time of each software principal and the execution order. Therefore, $E(\mathbb{S})$ is not a fixed number but a random variable. The variance of $E(\mathbb{S})$ introduces uncertainty in energy accounting by the Shapley value. (*iii*) Finally, $E(\mathbb{S})$ is further affected by the hardware state in a mobile system such as CPU frequency, LCD brightness, and WiFi active/idle mode. This will introduce even more variance in $E(\mathbb{S})$.

In this section, we present techniques to partially overcome the above challenges so that we can implement an energy accounting policy that is based on the Shapley value yet practical. We call this policy *Shapley value-based energy accounting*. This policy is not the same as the Shapley value but rather an approximate, in the sense that the input for the Shapley value calculations, $E(\mathbb{S})$, are estimated using practical methods.

4.1 Key Ideas

We tackle the first two challenges by using very short time intervals, i.e.,10 ms, as the game. In a shorter time interval, fewer software principals can run and the software execution dynamics has less impact on energy consumption. On the other hand, measuring or estimating the system energy consumption E for them will be less accurate and less efficient. Our experience with our prototype reported later has led us to believe 10 ms strives a good balance between these two aspects. By monitoring the system energy consumption *in situ* for an extended period of time, one can potentially acquire $E(\mathbb{S})$ for many different $\mathbb{S}$. Modern mobile systems already have a battery interface that measures system energy consumption at a modest rate [8]. Using this interface, it has been shown that system energy consumption for 10 ms time intervals can be estimated *in situ* with about 80% accuracy [8]. In Section 5.1, we present an enhanced battery interface that is able to measure system energy consumption of 10 ms time intervals with an accuracy over 95%.

We address the third challenge by incorporating hardware states as a condition into $E(\mathbb{S})$. Let σ denote current hardware state. We can estimate system energy consumption for a coalition $\mathbb{S}$ given the hardware state σ, denoted by $E(\mathbb{S},\sigma)$. As a result, one can apply the Shapley value separately to time intervals with a given hardware state. However, such an approach will further increase the number of $E(\mathbb{S},\sigma)$. As a result, there will be $E(\mathbb{S},\sigma)$ for numerous $\mathbb{S}$ and σ that are not observed by measurement. Therefore, we employ two solutions to estimate unknown $E(\mathbb{S},\sigma)$ from measured $E(\mathbb{S},\sigma)$ in historical data.This is the focus of Section 4.2. Finally, we extend Shapley value-based energy accounting to deal with non-deterministic factors in system energy consumption in Section 4.3.

4.2 Partially Defined Shapley Value

When $E(\mathbb{S},\sigma)$ is not observed for some $\mathbb{S}$ or σ, the classic Shapley value cannot be applied directly to determine per-software principal contribution. In such a cooperative game with unknown coalition values, the Shapley value is "partially" defined [30]. We now describe our techniques to determine the partially-defined Shapley value for per-software principal energy accounting. All the $E(\mathbb{S},\sigma)$ can be represented in a two-dimension matrix, with only a part of the all the elements can be obtained by measurement. As shown in Figure 1, we use *recursive definition* to fill all the columns, and *linear estimation* to fill all the rows. Finally, we use *M-games* to reduce the number of rows needed.

4.2.1 *Recursive Definition*

We first present a method to recursively define the Shapley value for all partially defined games. The key idea is to approximate $E(\mathbb{S})$ for unobserved $\mathbb{S}$ by the energy costs allocated to members of $\mathbb{S}$, which are presumably solvable by the Shapley value. We introduce the following extension of a partially defined game E to a fully defined one:

$$\hat{E}(\mathbb{S}) = \begin{cases} E(\mathbb{S}) & \text{if } \mathbb{S} \text{ is known} \\ \sum_{i\in\mathbb{S}} \phi_i(\hat{E}) & \text{otherwise} \end{cases} \quad (2)$$

Here $\phi_i(\hat{E})$ is the Shapley value of the fully defined game $\hat{E}$, corresponding to energy cost attributed to software principal i. By extending E to $\hat{E}$, we are assuming that the energy cost of an unknown coalition $\mathbb{S}$ is approximately the sum of the energy cost of members of the coalition $\mathbb{S}$. To complete the definition, $\phi_i(\hat{E})$ is the standard Shapley value for $\hat{E}$ as defined by Equation (1).

We note that the definition above is recursive since $\hat{E}$ and $\phi_i(\hat{E})$ are defined by each other. Therefore, we use an iterative algorithm to compute the Shapley value $\phi_i(\hat{E})$. At each iteration, we compute $\phi_i(\hat{E})$ for the fully defined game $\hat{E}$, and whenever the computation needs to probe an unknown coalition $\mathbb{S}$, then $\hat{E}(\mathbb{S})$ is calculated using the Shapley value $\phi_i(\hat{E})$ obtained in previous iteration.

4.2.2 *Linear Estimation*

We further utilize a linear model to estimate energy costs of unobserved coalitions for different $E(\mathbb{S},\sigma_y)$ with the same coalition $\mathbb{S}$ but different power states σ_y, $y = 1,2,\ldots,K$. Suppose that an unknown energy cost $E(\mathbb{S}_x,\sigma_y)$ of coalition $\mathbb{S}_x$ needs to be estimated for the set of resulting feasible coalitions to form a desired structure. We can employ the least-square method to find a linear substitute for σ_y based on the energy costs of other coalition in the same states, e.g., $E(\mathbb{S}_l,\sigma_y)$, $l \in 1,2,\ldots,L$ and $E(\mathbb{S}_l,\sigma_k)$, $k = 1,2,\ldots,K$, $k \neq y$. More precisely, we derive $K+1$ parameters $\theta_0,\theta_1,\ldots,\theta_K$ by solving the following optimization problem:

$$\text{minimize} \qquad \sum_{l:l\neq x} \|E(\mathbb{S}_l,\sigma_y) - \sum_{k=1;k\neq y}^{K} \theta_k \times E(S_l,\sigma_k)\|$$

Then we can calculate $E(\mathbb{S}_x,\sigma_y)$ by

$$E(\mathbb{S}_x,\sigma_y) = \sum_{k=1;k\neq y}^{K} \theta_k \times E(\mathbb{S}_x,\sigma_k). \tag{3}$$

4.2.3 *M-games*

We consider a symmetric, partially-defined game where the number of software principals in the coalition, or $|\mathbb{S}|$, solely determines whether $E(\mathbb{S})$ is known. For example, if any set of up to m concurrent software principals have been measured for their energy consumption, we can use the measurements to find energy consumption for all coalitions $\mathbb{S}$ satisfying $|\mathbb{S}| \leq m$, while coalition of size greater than m are treated as unknown. Let M be a set of known coalition sizes. An M-game is defined through energy costs $E(\mathbb{S})$ for coalitions whose sizes are in M, i.e., $|\mathbb{S}| \in M$. Notice that we assume that the energy cost of the grand coalition $\mathbb{N}$ is always known. It is easy to see that an M-game becomes a fully-defined cooperative game if M contains all possible coalition sizes.

A reduced Shapley value is defined on M-games [30] and it satisfies the axioms of additivity, efficiency, dummy-player and symmetry. For an M-game, the energy contribution of software principal i is given by

$$\phi_i = \frac{1}{n}\sum_{m\in M}\left[\frac{\sum_{|\mathbb{S}|=m,i\in\mathbb{S}} E(\mathbb{S})}{\binom{n-1}{m-1}} - \frac{\sum_{|\mathbb{R}|=m,i\notin\mathbb{R}} E(\mathbb{R})}{\binom{n-1}{m}}\right] \tag{4}$$

This formula can be interpreted as the average, over all known coalition sizes, of the differences between the coalitions $\mathbb{S}$ containing software principal i and that coalitions $\mathbb{R}$ not containing software principal i. If all coalitions are known, then the average is taken among all coalition sizes and reduces to the classic Shapley value defined in (1).

4.3 Dealing with Non-deterministic $E(\mathbb{S})$

So far we have attributed a single value of energy consumption to a software principal given the hardware states. However, the energy contribution by a software principal may vary due to many factors other beyond the hardware states considered. For example, the transmission power of the 3G/4G radio interface highly depends on volatile cellular signal strength and thus results in non-deterministic energy consumption. Uncertainty also arises from CPU scheduling and memory management. As a result, $E(\mathbb{S})$ should be treated as a non-deterministic variable, instead of single value. To take non-deterministic $E(\mathbb{S})$ into account, we extend our Shapley value-based energy accounting framework by treating $E(\mathbb{S})$ as a random variable and leverage all the theory work on random Shapley value that require the distribution. The only thing that requires modification in our system is the Shapley value calculation, while the statistics of $E(\mathbb{S})$, e.g., mean and variance can be obtained from the history.

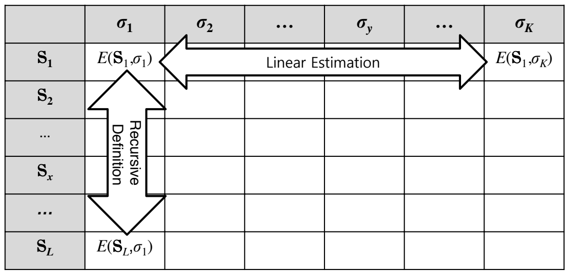

Figure 1: Estimation from partial information. Columns are filled by recursive definition and rows are filled by linear estimation.

For each coalition $\mathbb{S}$, we regard its energy consumption $E(\mathbb{S})$ as a random variable, whose distribution can be measured from the history. For random $E(\mathbb{S})$, we still use the Shapley value defined in (1) in Section 3 to calculate per-software principal energy $\vec{\phi}_i(E)$ for $i = 1,\ldots,N$, which now becomes a random variable. It is easy to see that the distribution of $\vec{\phi}_i(E)$ can be obtained by a convolution of the PDFs of energy consumption variables $E(\mathbb{S})$. In this section, we are mostly interested in the mean and variance of $\vec{\phi}_i(E)$, which indicate the average per-software principal energy consumption and how far it is spread out. Let $\mu[X]$ be the expectation of a random variable X and Var the variance. Using (1), we can derive the mean of per-software principal energy

$$\mu[\phi_i(v)] = \sum_{\mathbb{S}\subseteq\mathbb{N}\backslash\{i\}} \frac{\mu[E(\mathbb{S}\cup\{i\})] - \mu[E(\mathbb{S})]}{(|\mathbb{N}|-|\mathbb{S}|)\binom{|\mathbb{N}|}{|\mathbb{S}|}}, \tag{5}$$

and the variance of per-software principal energy

$$\text{Var}[\phi_i(v)] = \sum_{\mathbb{S}\subseteq\mathbb{N}\backslash\{i\}} \frac{\text{Var}[E(\mathbb{S}\cup\{i\})] + \text{Var}[E(\mathbb{S})]}{\left\{(|\mathbb{N}|-|\mathbb{S}|)\binom{|\mathbb{N}|}{|\mathbb{S}|}\right\}^2}, \tag{6}$$

A small variance indicates that the per-software principal energy tends to be very close to the mean and hence to be less volatile, while a high variance indicates that the per-software principal energy is very spread out from the mean and has large volatility.

4.4 What is Approximated?

It is important to note that when the measurement and estimation of $E(\mathbb{S})$ and $E(\mathbb{S},\sigma)$ are accurate, the above heuristics will lead to the exact Shapley value. Unfortunately there is inaccuracy in both the *in situ* measurement and the heuristics-based estimation of unknown $E(\mathbb{S},\sigma)$. As a result, our design approximates the payoffs of the game, essentially approximating the game itself. The approximation of provided by our Shapley value-based energy accounting is indeed the actual Shapley value of the approximate game.

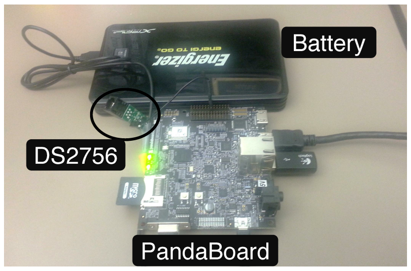

Figure 2: Prototype: a TI PandaBoard ES and a MAXIM DS2756 battery fuel gauge evaluation module.

5. IMPLEMENTATION

We have implemented a prototype of Shapley value-based energy accounting described in Section 4. The prototype includes a Texas Instruments (TI) PandaBoard ES [4] and a MAXIM DS2756 battery fuel gauge evaluation module (EVM) [5], as shown in Figure 2. The PandaBoard ES serves as a mobile system and is powered by an external battery pack. It comes with Android 4.04 and a TI OMAP4460 application processor, similar to that used by the Galaxy Nexus smartphone and RIM Playbook. The DS2756 EVM serves as an enhanced battery interface and measures the energy consumption by the Pandaboard ES from the external battery pack at 200 Hz. The measurement data is transferred from DS2756 EVM to the PandaBoard ES through GPIO using the 1-wire communication protocol.

The prototype uses *application* as software principal. We chose application instead of OS process because end users often care about the energy contribution by application, not by process, and because there are much fewer applications than processes. According to a recent study [29], the top ten most frequently used applications account for the total usage time by as much as 90%. Therefore, the prototype only considers the top ten applications for each user and treat other applications as a single software principal "other". It also combines all the Android system activities as a single software principal to be consistent with Android's Battery feature. As a result, the prototype accounts energy for up to 12 software principals.

We next elaborate the implementations of the enhanced battery interface and the software.

5.1 *In Situ* Measurement of $E(\mathbb{S})$

As explained in Section 4, the key to Shapley value-based energy accounting is to efficiently obtain the system energy consumption for very short time intervals, i.e., 10 ms. While prior work has demonstrated reasonably accurate estimation for 10 ms intervals using battery interfaces in commodity mobile devices [8], our prototype employ an enhanced battery interface that achieves even higher accuracy, i.e., over 95%. It does so with an ultra low-power system-on-a-chip consisting of a 16-bit microcontroller (MCU), 32 KB on-chip memory and two 14-bit analog-to-digital converters (ADCs) that comes with the DS2756 EVM, overall with a few hundred μW of active power consumption. Prior to prototyping, we performed extensive analysis to determine the measurement requirement: 8-bit ADC at 200 Hz as summarized in Section 5.1.1.

Our implementation includes both software in the mobile system and an enhanced battery interface as illustrated by Figure 3. The MCU in the enhanced battery interface periodically measures the voltage of and the discharge current from the battery using two 14-bit ADCs, calculates the energy consumption, and writes it to the local memory along with a timestamp. The battery interface driver

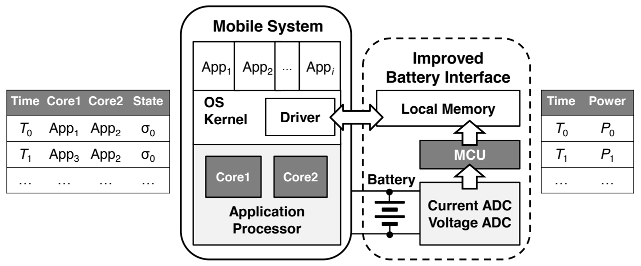

Figure 3: Architecture of the *in situ* high-speed, high-accuracy energy measurement solution: The battery interface measures energy periodically and stores the measurement data with timestamps in its local memory; the mobile system also maintains a timestamped data structure of application combination executed in the OS kernel, utilizes a driver to read the energy measurement data structure from the local memory in the battery interface, and combines the two data structures to get all the $E(\mathbb{S})$.

running in the mobile system reads the local memory in the battery interface via a serial bus in the standard smart battery interface upon a request from the mobile OS. The driver also keeps a timestamped record for the Android UID of applications that are running in each OS scheduling period as well as the hardware state of the system. Then the mobile OS combines the timestamped energy data from the battery interface and the local timestamped UID data to obtain the system energy consumption for each scheduling period.

It is critical to synchronize the UID data collected by the battery interface driver in the mobile system and the energy data collected by the battery interface. The UID data collected by the driver is timestamped with a timer based on a high accuracy clock. The energy data, however, is timestamped by the timer based on a local low-accuracy clock of the MCU, usually implemented using an LC oscillator to achieve power efficiency. Inaccuracy in the energy data timestamp will lead to that the energy data misalign with the UID data. To tackle this challenge, the prototype employs two techniques. First, the driver periodically writes the OS timestamp to a specific memory location in the battery interface and the latter utilizes this timestamp to correct its own timer. Then, the data transfer delay between the mobile system and the battery interface is further accounted with a simple calibration procedure.

5.1.1 Measurement Requirement

The key component of the enhanced battery interface in Figure 3 is the ADC. The most important properties of an ADC are *sampling rate* and *resolution*. Insufficient sampling rate or resolution will cause error in energy measurement while excessive high sampling rate or resolution will lead to energy waste.

In order to determine the sufficient sampling rate and resolution, we characterize the power consumption traces of three commercial smartphones (Samsung Galaxy S, S II, and S III), and ten popular applications as shown in Table 3. We employ one external power supply for constant 4.2 V input and measure the current drawn. We employ a high-speed Oscilloscope to collect the ground truth power traces, using 500 KHz sampling rate, for each of the device and application.

Sampling Rate: By examining the spectrum of the power traces, we are able to identify 200 Hz as sufficient to include 99% of power density in spectrum for all the applications. As an example, Figure 4 (left) shows the power spectrum of Galaxy S III running the Phone application. We can see most of energy locates in the low-frequency range, while the high-frequency spectrum is flat. As shown in the black curve in the figure, 200 Hz sampling rate includes over 99% of the signal energy. The power traces of other

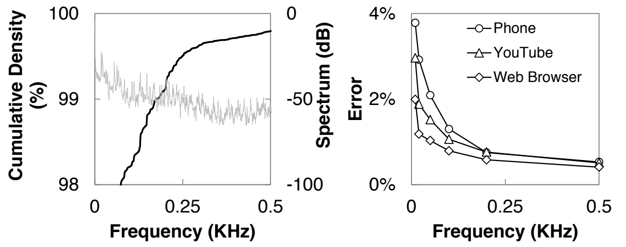

Figure 4: Smartphone workload power characterization for sampling rate: 200 Hz sampling rate is able to cover 99% of the signal information of the power trace (left); 200 Hz sampling rate introduces 1% error in energy measurement of each 10 ms (right).

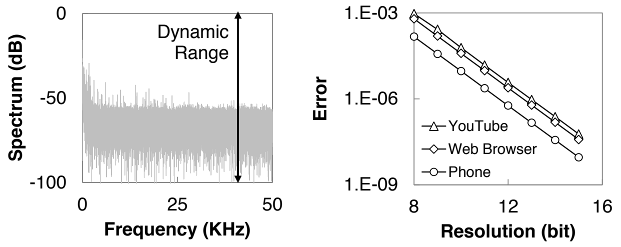

Figure 5: Smartphone workload power characterization for resolution: dynamic range of power trace is around 100 dB and requires a 16-bit ADC for faithful measurement (left); using a 8-bit ADC introduces 0.1% error in energy measurement of each 10 ms (right).

Table 3: Sufficient sampling rate for applications to have 99% of the signal energy of the power consumption trace (Unit: Hz)

Application	Galaxy S	Galaxy S II	Galaxy S III
Phone	197	185	200
SMS	97.1	88.5	93.3
Web Browser	102	105	95.4
Facebook	107	95.5	102
Google Hangout	87.2	85.3	80.8
YouTube	97.4	91.6	102
CamCorder	18.9	24.1	15.1
GPS Navigation	32.5	46.1	30.7
Quake III	67.2	85.6	90.8
Angry Bird	28.4	45.5	57.2

applications show similar characteristics (See Table 3). We next use the 500 KHz data traces as the ground truth and investigate the average error in each 10 ms introduced by using a sampling rate less than 500 KHz. Figure 4 (right) shows the three applications with the highest errors. As shown in the figure, a 200 Hz sampling rate introduces less than 1% error for 10 ms intervals. Therefore, we conclude that 200 Hz sampling rate is sufficient for all the characterized applications on all the smartphones in our experiment.

Resolution: Resolution of an ADC is determined by the dynamic range of the input signal. We characterize the dynamic range by reducing quantization noise below the background noise. As shown in Figure 5 (left), background noise is around -100 dB in normalized spectrum; this implies the effective number of bit (*ENOB*) should be 16 or more because $SNR = 1.76 + 6.02 \times ENOB$. Thus, a 16-bit ADC is sufficient to record transient power trace. We also study how much error will be introduced by using an ADC of lower resolution. As shown in Figure 5 (right), the quantization error introduced by using a 8-bit ADC only introduces 0.1% error in the system energy measurement for each 10 ms.

To summarize, an 8-bit ADC with 200 Hz sampling rate is sufficient for the system energy measurement for 10 ms time intervals needed by Shapley valued-based energy accounting. Thanks to fast development in low-power CMOS technology, a 15-bit audio ADC, with sampling rate higher than 20 KHz, only consumes less than 200 μW [24]. The power consumption of a 8-bit ADC with 200 Hz sampling rate should be negligible.

5.2 Software

Our prototype includes two software modules: one energy measurement module running in the enhanced battery interface and one energy accounting module running in the mobile system. The energy measurement module provides timestamped 200 Hz energy measurement data. The energy accounting module collects $E(\mathbb{S},\sigma)$, estimates the missing $E(\mathbb{S},\sigma)$, and calculates energy contribution by each application using Shapley value-based energy accounting.

The energy measurement module is a stand-alone process running in the battery interface's MCU, developed in C language with about 830 lines of code. We use software interrupts to realize the 200 Hz energy measurement. After synchronization described above, the MCU acquires the Android OS timestamp from the mobile system, updates its local timer, and uses this local timer to trigger interrupts to get ADC readings of voltage and current measurement and calculate the energy consumption accordingly.

The energy accounting module in the mobile system includes two sub-modules: $E(\mathbb{S},\sigma)$ collection and Shapley value calculation. The $E(\mathbb{S},\sigma)$ collection sub-module includes two parts, i.e., a kernel modification to align timestamp and energy measurement and a user-space thread to collect data. We modify the Android kernel to include the timestamped data structure (See Figure 3) in the kernel space. Upon process context switch, the OS updates the data structure by appending a new entry to it. This data structure is exposed to the user-space thread in the /sys virtual file system. The $E(\mathbb{S},\sigma)$ collection sub-module is developed on the top of the Android 4.04 code base with 17 files change and totally 2700 lines of code modification.

The Shapley value calculation sub-module is a user-space application. It periodically reads the timestamped data structure from the virtual file system before the data structure overflows and stores the measurement data in the SD card. When the measurement data in the SD card accumulates to a threshold, the user-space thread launches a routine to estimate unknown $E(\mathbb{S},\sigma)$ from historical measurement data, as illustrated in Figure 1. In our current implementation, we set the threshold as 100 MB, approximately the data of one day.

6. EVALUATION

Using the prototype described in Section 5, we next report a two-part evaluation of energy accounting policies. First, we evaluate how well our Shapley value-based energy accounting approximates the Shapley value and outperforms existing accounting policies in three aspects: (*i*) the accuracy of $E(\mathbb{S})$ estimation; (*ii*) the capability to identify "top energy consumer"; and (*iii*) the application in energy management. Second, we evaluate energy accounting results from existing policies and those from Shapley value-based energy accounting and demonstrate how different they can be.

6.1 Experimental Setup

In our experiment, we employ four benchmarks: (*i*) *Download*: Network intensive process that downloads a large file using HTTP protocol via WiFi. (*ii*) *Web*: Android built-in web browser application to visit web pages as recorded in a URL list via WiFi. Each

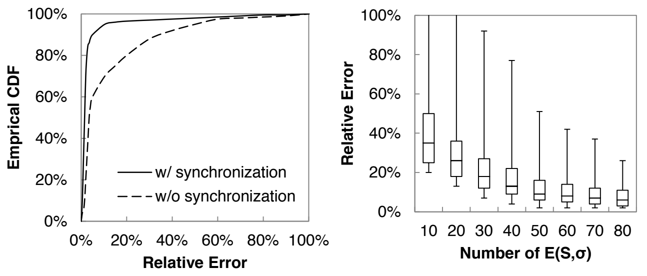

Figure 6: **Error in obtaining $E(\mathbb{S},\sigma)$: Over 90% of energy measurement by the battery interface are with error less than 5% and synchronization does improve accuracy (left); estimation has a median error of 10% using one third of total $E(\mathbb{S},\sigma)$ (right).**

webpage is loaded and shown for thirty seconds before next one is loaded. (*iii*) *Video*: Video player based on ffmpeg. (*iv*) *Game*: Quake 3, an open source 3D video game, played with the included demo scripts.

Out of the four benchmarks, Video, Web and Game cannot run at the same time because they must be executed in the foreground to present content on the screen. Therefore, we further define three scenarios that represent three typical use cases with multiple applications concurrently running:

- Scenario 1: Web + Download + Android.
- Scenario 2: Video + Download + Android.
- Scenario 3: Game + Download + Android.

We leverage the LiveLab mobile usage traces [29] to complement the above benchmarks. The LiveLab usage traces record daily mobile app and web usage for 34 iPhone 3GS users from six to twelve months and are available from [6]. The URL list used in Web includes top twenty most visited websites from the LiveLab traces. For each website, we randomly selected ten pages from the database for the URL list.

6.2 Shapley Value-based Energy Accounting

As described in Section 3, the Shapley value should be the ground truth for energy accounting in theory. In practice, however, it is infeasible for one to obtain the "true" Shapley value. Therefore, we choose to use three following *indirect* ways to evaluate how our Shapley value-based energy accounting approximates the Shapley value and outperforms other accounting policies. First, we evaluate how accurate the $E(\mathbb{S})$ can be estimated by our implementation. Second, we evaluate how much better our implementation, compared to existing policies, can identify the "top energy consumer" or the app that contributes most to the total system energy consumption. Third, we evaluate how much better our implementation can help energy management in enhancing utility value compared to existing policies.

6.2.1 Accuracy of $E(\mathbb{S})$ estimation

We first evaluate the accuracy of energy measurement by the prototype by comparing the measurement results against readings by a high-end oscilloscope. As shown in Figure 6 (left), over 90% of measurement are with error less than 5%. This is significantly better than the accuracy achieved using existing battery interface as reported in [8]. Figure 6 (left) also illustrates the importance of synchronization. As shown in the figure, 90% of measurement are with error less than 30% without synchronization.

We then evaluate the accuracy of energy estimation from partial data. The prototype support 12 states in total, i.e., four states for

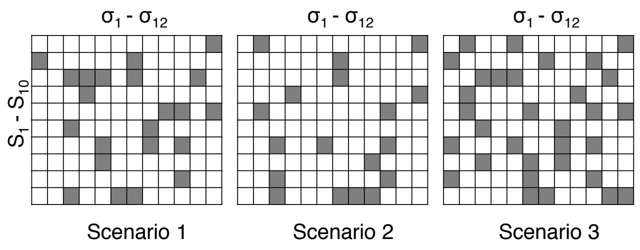

Figure 7: **Footprint of all the missing $E(\mathbb{S},\sigma)$ (grey cells) and obtained $E(\mathbb{S},\sigma)$ from measurement (white cells) in all three scenarios.**

CPU and three states for WiFi. There are 10 application combinations for each of the three scenarios included in our experiment. Therefore, thee are 120 $E(\mathbb{S},\sigma)$ for each scenarios. In our experiment, we run each scenarios for a sufficient long period and finally obtain 97, 102, and 86 $E(\mathbb{S},\sigma)$, respectively. The footprint of the missing $E(\mathbb{S},\sigma)$ and obtained $E(\mathbb{S},\sigma)$ from measurement are illustrated in Figure 7. Each of the 10 by 12 matrix represents a scenario where each row represents each application combination $\mathbb{S}$ and each column represents each states σ. A grey cell means the corresponding $E(\mathbb{S},\sigma)$ is missing while a white cell means the corresponding $E(\mathbb{S},\sigma)$ is obtained from measurement. We treat these obtained $E(\mathbb{S},\sigma)$ (white cells) as ground truth and randomly choose part of these $E(\mathbb{S},\sigma)$ to estimate the remaining part. Figure 6 (right) shows the results. As shown in the figure, our estimation algorithms can achieve a median error of 10% only using one third, or 40 out of 120, of the total number of $E(\mathbb{S},\sigma)$.

6.2.2 Identify Top Energy Consumer

We next evaluate how well energy accounting policies can identify the mobile application that contributes most to the system energy consumption in a day of use, or *top energy consumer*. In particular, we compare our implementation of Shapley value-based energy accounting against Policy V, the most widely used policy by many systems including Android. Our methodology is to replay real-world mobile application usage traces on our prototype with reduced fidelity.

We use the LiveLab mobile usage traces for replay. Because the LiveLab traces are collected from iPhone and have up to hundreds of iOS foreground applications, all without user input, we have to reduce them in the following ways in order to replay them on our Android prototype. First, we reduce the foreground applications to only five software principals: Web, Video, Game, Mail, and Music. We categorize all the apps from the LiveLab traces into one of these five software principals. For example, we treat all game apps as Game; all email and instant messenger apps as Mail; and all web browsing and social networking apps as Web. We ignore applications that do not belong to any of the five types, such as Map, Alarm and Calculator. To replay Web, Video, and Game, we invoke the Web, Video, and Game Android benchmarks described in Section 6.1, respectively. We invoke the Android native email client to send a precomposed email and receive in order to replay Mail and invoke ffmpeg to play an MP3 file to replay Music. Second, we use the Download benchmark described in Section 6.1 to mimic background services in Android such as data sync of social networking apps. In our experiment, we make Download start every fifteen minutes and run for one minute. These reductions lead to replay traces with only six software principals, i.e., Web, Video, Game, Mail, Music and Download.

We also make the following changes to our prototype described in Section 5 for replay. First, we change the power supply from the

Table 4: Top energy consumers identified by Shapley value-based accounting (*Shapley*) and Policy V when they disagree for 19 out of the 50 days of LiveLab data tested

#	User	Shapley	Policy V	#	User	Shapley	Policy V
1	A	Game	Download	11	B	Video	Mail
2	A	Game	Download	12	B	Music	Download
3	A	Game	Download	13	C	Web	Download
4	A	Game	Download	14	C	Web	Download
5	A	Game	Mail	15	C	Music	Download
6	A	Game	Mail	16	C	Music	Mail
7	A	Game	Mail	17	D	Game	Mail
8	B	Game	Download	18	D	Music	Download
9	B	Game	Download	19	E	Mail	Web
10	B	Video	Download				

external battery pack to a USB cable connected to a PC to support sufficient long time for our experiment. As a result, the enhanced battery interface measures the voltage (around 5V) and the current in the USB cable and calculates the power consumption. Second, we deploy an Android ADB based testing framework in the PC to automatically launch and terminate applications in the PandaBaord through the USB cable. By feeding the usage trace to the testing framework, we are able to replay the reduced LiveLab usage traces on our prototype.

We randomly select five out of the 34 users in the LiveLab trace database and label them as User A to User E. We then randomly choose ten days of usage traces for each user to produce 50 days of usage traces. We evaluate the energy accounting policies by how well they identify the top energy consumer for each of the 50 days of usage when the traces are replaced in the prototype as described above. Out of the 50 days, Policy V and Shapley value-based energy accounting disagree on the top energy consumer for 19 days as summarized by Table 4.

We then replay the traces of the 19 days two more times but in each time we exclude the top energy consumer reported by Policy V and Shapley value-based accounting, respectively. We find that, for all 19 cases, the energy consumption excluding top energy consumers identified by Shapley value-based energy accounting is lower than its Policy V counterpart. This finding indicates Shapley value-based energy accounting identifies an application that contributes more energy than that identified by Policy V does. As shown in Table 4, many disagreements are due to Policy V underestimating the energy contribution of GPU, e.g., in the case of a heavy mobile gamer like User A. We also observe that Policy V tends to overestimate network related applications such as Download and Mail. The reason is that the network interface wakeup and timeout to enter the sleep state consumes a significant amount of energy; when the network interface is already active, the additional energy consumed by the a short period of data traffic is relatively small. Policy V, however, only estimates energy consumption base on the amount of data traffic, which easily leads to false accounting results. This also confirms the observations by [22].

6.2.3 Application in Energy Management

Energy accounting is the foundation to OS-based energy management that seeks to control the energy use by processes and schedule their execution to maximize aggregate system utility under energy constraints. Various energy accounting policies have been tried [12, 32, 14, 28, 25, 15]. We now evaluate energy accounting policies in the context of energy management, in particular, the popular budget-driven energy management (BEM) [32, 25, 15]. We show that existing accounting policies achieve 5% to 25% less utility compared to our Shapley value-based energy accounting.

In BEM, each process receives an independent energy budget and disburses it according to the energy accounting policy. The use of per-process budget not only enables a single energy source to be

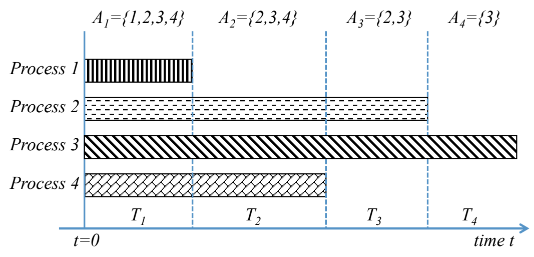

Figure 8: Illustration of four processes and their executions under budget-based energy management (BEM).

shared by applications in a fair and simple fashion, but also offers composability among processes. For instance, in a smartphone, an energy budget can be reserved for phone calls to guarantee certain talk time. Each process is assigned a utility function $U_i(t_i)$, which measures the utility of process i receiving run time t_i. $U_i(t_i)$ is assumed to be monotonically increasing and concave [7]. The goal of BEM is to allocate per-process budgets $B_1,\ldots,B_n$ among n processes, in order to maximize aggregate utility $\sum_{i=1}^n U_i(t_i)$ under total energy available. We next present the problem formulation.

Process Run Time: We consider n independent processes, denoted by $\mathbb{N}=\{1,2,\ldots,n\}$. Under BEM, each process is scheduled to run until its energy budget reaches zero. To determine process run time, we suppose that processes deplete their energy budgets in the following order: $\pi(1),\pi(2),\ \ldots,\pi(n)$, where $\pi(k)$ be the kth process that runs out of energy budget (If multiple processes reach zero budget simultaneously, then their ordering can be arbitrary). Then, the execution of all n processes can be divided into n epochs, $T_1,T_2,\ldots,T_n$, where each epoch T_k represents the time interval between successive energy depletion of processes $\pi(k-1)$ and $\pi(k)$. It is easy to verify that the sequence of active processes $\mathbb{A}_k=\{\pi(k),\pi(k+1),\ldots,\pi(n)\}$ within each epoch T_k form a contraction with $\mathbb{A}_{k+1}\subseteq\mathbb{A}_k$ for $k=1,\ldots,n$.

Figure. 8 illustrates our model for $n=4$ processes. Given process $\pi(1)=\{1\}$ is the first to run out of budget, epoch T_1 consists of 4 active processes $\mathbb{A}_1=\{1,2,3,4\}$, while epoch T_2 consists of 3 active processes $\mathbb{A}_2=\{2,3,4\}$. Similarly, $\pi(2)=\{4\}$ and $\pi(3)=\{2\}$ means that epoch T_3 contains 2 active processes $\mathbb{A}_3=\{2,3\}$ and epoch T_4 contains a single process $\mathbb{A}_4=\{3\}$. Using these notations, we can describe process execution under BEM by n distinct epochs. The total run time received by process i is given by $t_i=\sum_{k=1}^{\pi^{-1}(i)}T_k$ where process i depletes its budget at the end of $\pi^{-1}(i)$-th epoch, and the summation is over all epochs where i is active. We refer to t_i as the run time assigned to process i.

Energy Constraints: BEM relies on energy accounting to split total energy consumption among individual processes. Let $\phi_i(E(\mathbb{S}))$ denote the energy consumption attributed to process i according to the energy accounting policy during a scheduling unit in which $\mathbb{S}$ are scheduled to run and $i\in\mathbb{S}$. After this scheduling period, $\phi_i(E(\mathbb{S}))$ will be deducted from process i's energy budget B_i. When a process runs out of budget, it will not be scheduled any more. Using our notations above, total energy attributed to each process i is the sum of individual energy bill $\phi_i(E(\mathbb{A}_k))$ over all epochs $k=1,\ldots,\pi^{-1}(i)$ where process i is active. It implies that

$$\sum_{k=1}^{\pi^{-1}(i)}\phi_i(E(\mathbb{A}_k))\cdot T_k=B_i. \tag{7}$$

Further, the total system energy consumption must be bounded by the battery capacity, C, i.e., $\sum_{i=1}^n B_i\le C$.

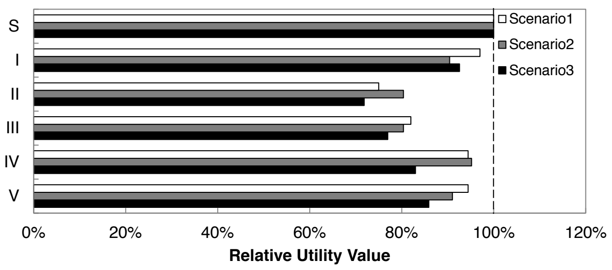

Figure 9: Optimal utility values achieved by BEM with various energy accounting policies. All numbers are normalized by that achieved by Shapley value-based accounting (S).

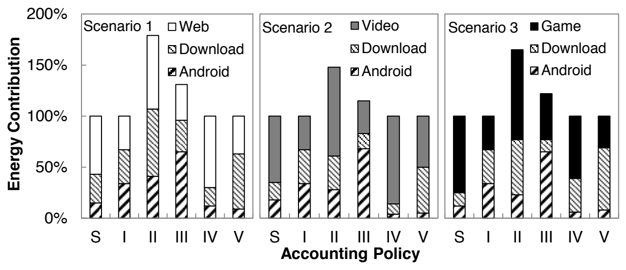

Figure 10: Energy accounting results for Shapley value-based accounting (S) and Policy I-V. The energy consumption (Y axis) is normalized by the total system energy consumption.

Utility Optimization: Each process is assigned a utility function $U_i(t_i)$, which measures the utility of process i receiving run time t_i. We make the common assumption that $U_i(t_i)$ is assumed to be monotonically increasing and concave [7].

The goal of BEM is to allocate per-process budgets $B_1, \ldots, B_n$ among n processes, in order to maximize aggregate utility $\sum_{i=1}^{n} U_i(t_i)$ under energy capacity C. More precisely, we formulate it as the following utility optimization:

Problem BEM :

$$\text{maximize} \quad \sum_{i=1}^{n} U_i(t_i) \tag{8}$$

$$\text{subject to} \quad t_i = \sum_{k=1}^{\pi^{-1}(i)} T_k, \ \forall i, \tag{9}$$

$$\sum_{k=1}^{\pi^{-1}(i)} \phi_i(E(\mathbb{A}_k)) \cdot T_k = B_i \ \forall i \tag{10}$$

$$\sum_{i=1}^{n} B_i \leq C \tag{11}$$

$$\text{variables} \quad \pi, B_i, T_k. \tag{12}$$

Notice that the optimization is a joint maximization over per-process budgets B_i, epoch length T_k, and process ordering π, which are not independent. It is easy to see that Problem BEM is non-convex due to arbitrary process ordering π.

We implement BEM with various energy accounting policies on top of Android's own kernel scheduler. Our implementations uses *application* instead of process as the schedule entity for the same reason described in Section 5.1 To solve the BEM optimization problem, we uses the GNU Scientific Library to implement the standard Newton Method [7]. The BEM optimization is triggered when a new application is launched or every 100 seconds, depending on which comes first. The BEM optimization can take as long as several milliseconds in our experiment, which is comparable to the length of process scheduling period itself.

Our results show that Shapley value-based energy accounting is superior compared to other accounting policies, achieving the highest optimal utility value in BEM. In our experiment, we combine BEM with energy accounting based on each of Policy I to V and Shapley value-based accounting, as shown in Figure 9. To guarantee fairness, we set the utility function as the total execution time of the two applications in three scenarios using the proportional fairness utility function, the weights for the foreground application and background application is 80% and 20%, respectively. Figure 9 shows the corresponding optimal utility values of the optimization for the six combinations. Among all the combinations, Shapley value-based energy accounting (S) has the highest utility value and we make it as the baseline to normalize the utility values of other combinations. As shown in the figure, all other energy accounting policies achieve 5% to 25% less utility value than Shapley value-based energy accounting. In particular, Policy II and Policy III have the least utility values because they violate Axiom of Efficiency. This is because, when Efficiency does not hold, the energy budget of an application can either be depleted earlier than expected or be left unfinished in the end. In both cases, the total energy will not be used efficiently, eventually leading to reduced utility value.

Moreover, we observe that Policy V, the most sophisticated policy used in recent work, is no better than much simpler Policy I and Policy IV in energy management. This suggests that existing budget-based energy management solutions may be unnecessarily complicated. This result is unsurprising because Policy I, IV, and V are all applying a linear function to divide the total energy consumption to multiple components, each component for an application. The only difference is the coefficients in the linear function. By solving the BEM optimization, one can always find the optimal budget for the linear function as long as the Efficiency property holds. In Chapter 3 of [9], a formal proof of this property is provided.

6.3 Evaluating Existing Policies

The previous evaluation experiments demonstrate our implementation performs well as an approximation of the Shapley value. We next evaluate the five types of energy accounting policies described in Section 2.2 by comparing their results against those from our approximation of the Shapley value. The implementations of Policy I to IV are straightforward. We implement Policy V using the system energy model from the open source project PowerTutor [2], which is very similar to Android's own. We update the coefficients of the model following the procedures described in [33] based on offline power benchmarking and linear regression using the benchmark data. The accuracy of the updated model is higher than 90% for one reading per second.

Figure 10 presents the accounting results for the three scenarios. The Y axis is normalized by the total system energy consumption. If a policy meets the Axiom of Efficiency, the stacked energy contribution should be exactly 100%.

All the five types of energy accounting policies can deviate from our approximation of the Shapley value significantly. Policy II and Policy III obviously violate the Axiom of Efficiency, corroborating our analysis summarized in Table 2. Policy IV is inaccurate because it only considers the usage of CPU. Therefore, Policy IV tends to overestimate for applications that require high CPU activities such as Video in Scenario 2. Policy V is inaccurate because its energy model does not account the usage of GPU. This is obvious in Scenario 3 where Game extensively uses GPU and Policy V significantly underestimate its energy use. As a result, Policy V overestimates the energy contribution of Download by four times.

7. OTHER RELATED WORK

In Section 2.2, we already discussed representative work in energy accounting and evaluated their policies throughout the paper. We next discuss related work in the applications of energy accounting, i.e., energy management and software evaluation, where Shapley value-based energy accounting can be readily applied. Due to the space limit, we mainly discuss related work in mobile systems.

Energy Management: ECOSystem [32] presents "currentcy", an abstraction of energy and incorporates it into energy management. Energy consumption by each hardware component is modeled as charge in "currentcy" that attributed to each running process. This is exactly the BEM approach as described in 6.2.3. Like ECOSystem, Cinder [25] also utilize BEM, but advances ECOSystem by providing more sophisticated mechanisms for processes to delegate. Both energy management solutions employ a version of Policy V that uses a software-based energy model. Therefore, their performance is limited by the drawbacks of Policy V as discussed in Section 3.3 and experimentally demonstrated in Section 6.3.

Evaluating Software for Energy Efficiency: Energy accounting is also popularly used for evaluating software. Existing work often employs various forms of Policy V for this purpose. Pathak et al [23] developed a framework based on their FSM model [22]. Such framework logs the traces of system calls by each application and performs an offline analysis that assigns energy contribution to each application based on the FSM model and the system call traces. To solve the same problem, Carat [21] resorts to crowdsouring. Carat samples the battery level of an iPhone periodically and what applications are actively running in each sample period. After collecting sufficient data from many users, Carat can identify energy-hungry applications. There is also work that addresses energy issues of smartphones without direct information of energy consumption. For example, eDoctor [16] is able to identify abnormal battery drain issues on smartphones by capturing the execution-phase time-varying behavior of each application.

8. CONCLUDING REMARKS

In this paper we answer an important and long-standing question about energy accounting with theory, practical approximation, and experimental evaluation: how to evaluate an energy accounting policy? Our answer is: the Shapley value should serve as the ground truth for energy accounting. We analytically show how existing energy accounting policies violate the four self-evident axioms that define the Shapley value. More importantly, we also show that an approximation of the Shapley value can be practically realized with *in situ* high-rate energy measurement. Using a mobile system development board, we realize this approximation, called Shapley value-based energy accounting, and show existing energy accounting policies deviate significantly from it. Our results are fundamental to evaluating energy accounting policies that have been widely used in software evaluation and energy management.

Shaley value-based accounting vs. Policy V

It is important to reflect on why our Shapley valued-based energy accounting actually works better than other energy accounting policies. While the limitations of Policies I to IV are obvious, Policy V warrants more discussion. Policy V takes a *spatial* view of energy consumption. That is, it views energy consumption as resulting from a collection of energy-consuming hardware resources. Therefore, it reduces energy accounting to (*i*) resource usage accounting that relates resource usage to software principals and (*ii*) energy modeling that relates energy consumption to resource usage. To improve the accuracy of this approach, one has to improve resource usage accounting, especially the "spatial" resolution of resource usage accounting. While the usage of many hardware resources can be easily attributed to software principals in modern computers, e.g., CPU, memory and network interface, that of many others cannot. This is particularly true for the system-on-chips used in mobile systems that often integrate a large number of specialized hardware modules not visible to the OS. To account the usage of these modules requires per-module hardware support like that exploited by iCount [10]. That is, hardware modification to improve the "spatial" resolution of resource usage accounting is likely to be distributed, making it costly. More importantly, without better resource usage accounting, one cannot improve Policy V with more energy measurement data.

In contrast, Shapley value-based energy accounting takes a *temporal* view of energy consumption. That is, it views energy consumption as resulting from a collection of time intervals, or games. It reduces energy accounting to (*i*) measuring the system energy consumption and (*ii*) knowing which software principals are active in each time interval. The challenge to this approach, as highlighted at the beginning of Section 4, lies in that one has to obtain the energy consumption of time intervals in which all possible coalitions of software principals under question are active. That is, we need $E(\mathbb{S})\ \forall\ \mathbb{S} \subseteq \mathbb{N}$. Shapley value-based energy accounting addresses this by *in situ* energy measurement of very short time intervals and by using heuristics to estimate the energy consumption ($E(\mathbb{S})$) for unobserved coalitions ($\mathbb{S}$). To improve it, one needs (*i*) more accurate, higher rate energy measurement and (*ii*) more measurements so that more coalitions ($\mathbb{S}$) are observed. In contrast to how Policy V has to be improved, the required hardware modification is localized, i.e., only the battery interface, and more measurements will help. Therefore, we believe our Shapley value-based energy accounting not only is better than versions of Policy V in use today but also has a brighter prospect in practice moving forward.

Shortcomings aside, Policy V does have practical values, especially in evaluating and optimizing software for energy efficiency. It can be very good at identifying top energy consumers or energy-draining misbehaviors that consume energy via the hardware resources accounted by the policy, e.g., CPU, memory and data traffic. Because it relates system energy consumption to resource usage, it can offer insight into how software is draining energy and provide guideline for software optimization. In contrast, Shapley value-based energy accounting does not provide such insight at all.

Potential improvements

Again, the key to Shapley value-based energy accounting is to obtain $E(\mathbb{S})\ \forall\ \mathbb{S} \subseteq \mathbb{N}$. While we provide a suite of techniques to acquire and estimate $E(\mathbb{S})$ in Section 4, our solutions can be further improved in three important ways.

Using better resource usage accounting: Policy V can be improved with better resource usage accounting as discussed above. Improved resource usage accounting can equally improve Shapley value-based energy accounting: instead of using Shapley value-based energy accounting for the whole system energy consumption, one can now use it only for the portion of system energy consumption that cannot be attributed to accounted resource usage.

Using the crowd: In Section 4, we presented a few heuristics to obtain $E(\mathbb{S})$ for $\mathbb{S} \subseteq \mathbb{N}$ that are not observed in the system under question. As a model of smartphone is usually used by hundreds of thousands of users, the users of the same model can help each other by contributing their own observations of $E(\mathbb{S})$. Each observation includes the energy consumption, unique identities of the running software principals ($\mathbb{S}$), device model, hardware state (σ). The observations made by one device can be compacted and uploaded to a central server occasionally, e.g., when connected via WiFi and wall-powered. Similarly, a device can look up the same server for observations for the same model that are unavailable locally.

Our work can also be extended to further attribute the energy use by the OS to processes. Our current implementation treats the OS as a separate principal and allocates its own share of energy use. The OS, however, can perform a service on behalf of a process, e.g., sending a packet out through TCP/IP. Therefore, it is reasonable to attribute the energy use by such services to the process being served. Our implementation can be easily modified for this purpose because a modern OS like Linux keeps track of the processes that it is serving. Our implementation will be able to calculate the energy use by the OS when serving a process; the only modification is to attribute this energy use to the process, instead of the OS itself.

Acknowledgements

This work was supported in part by NSF Grants CNS #1065506 and CNS #1218041. The authors thank the anonymous reviewers, especially the paper shepherd, for their useful feedbacks that help improve the final version of the paper. .

9. REFERENCES

[1] `http://source.android.com/`.

[2] `http://powertutor.org/`.

[3] `http://www.dianxinos.com/`.

[4] `http://pandaboard.org/content/pandaboard-es`.

[5] `http://www.maximintegrated.com/en/products/power/DS2756EVKIT.html`.

[6] `http://livelab.recg.rice.edu/`.

[7] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

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

[9] Mian Dong. *Energy accounting and optimization for mobile systems*. PhD thesis, Rice University, 2013.

[10] P. Dutta, M. Feldmeier, J. Paradiso, and D. Culler. Energy metering for free: Augmenting switching regulators for real-time monitoring. In *Proc. IEEE. IPSN*, 2008.

[11] J. Feigenbaum, C. H. Papadimitriou, and S. Shenker. Sharing the cost of multicast transmissions. *Journal of Computer and System Sciences*, 63(1), 2001.

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

[13] R. Fonseca, P. Dutta, P. Levis, and I. Stoica. Quanto: Tracking energy in networked embedded systems. In *Proc. USENIX OSDI*, 2008.

[14] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A.A. Bhattacharya. Virtual machine power metering and provisioning. In *Proc. ACM Symp. Cloud Computing*, 2010.

[15] H. Lim, A. Kansal, and J. Liu. Power budgeting for virtualized data centers. In *2011 USENIX Annual Technical Conf.*, 2011.

[16] X. Ma, P. Huang, X. Jin, P. Wang, S. Park, D. Shen, Y. Zhou, L. K. Saul, and G. M. Voelker. eDoctor: Automatically diagnosing abnormal battery drain issues on smartphones. In *Proc. USENIX NSDI*, 2013.

[17] J.C. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A.C. Snoeren, and R.K. Gupta. Evaluating the effectiveness of model-based power characterization. In *USENIX Annual Technical Conf.*, 2011.

[18] V. Misra, S. Ioannidis, A. Chaintreau, and L. Massoulié. Incentivizing peer-assisted services: a fluid Shapley value approach. In *ACM SIGMETRICS Performance Evaluation Review*, 2010.

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

[20] R. Muralidhar, H. Seshadri, K. Paul, and S. Karumuri. Linux-based ultra mobile PCs. In *Proc. Linux Symposium*, 2007.

[21] A. J. Oliner, A. P. Iyer, I. Stoica, E. Lagerspetz, and S. Tarkoma. Carat: Collaborative energy diagnosis for mobile devices. In *Proc. ACM SenSys*, 2013.

[22] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y. M. Wang. Fine-grained power modeling for smartphones using system call tracing. In *Proc. ACM EuroSys*, 2011.

[23] A. Pathak, Y.C. Hu, and M. Zhang. Where is the energy spent inside my app?: fine grained energy accounting on smartphones with Eprof. In *Proc. ACM EuroSys*, 2012.

[24] A. Pena-Perez, E. Bonizzoni, and F. Maloberti. A 84dB SNDR 100kHz bandwidth low-power single op-amp third-order $\Delta\Sigma$ modulator consuming 140μW. In *Proc. IEEE ISSCC*, 2011.

[25] A. Roy, S.M. Rumble, R. Stutsman, P. Levis, D. Mazieres, and N. Zeldovich. Energy management in mobile devices with the Cinder operating system. In *Proc. ACM EuroSys*, 2011.

[26] S. Ryffel. LEA^{2}P: the linux energy attribution and accounting platform. Master's thesis, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, 2009.

[27] Lloyd Shapley. A value for n-person games. In H.W. Kuhn and A.W. Tucker, editors, *Contributions to the Theory of Games, volume II*, pages 307–317. Princeton University Press, 1953.

[28] K. Shen, A. Shriraman, S. Dwarkadas, X. Zhang, and Z. Chen. Power containers: An OS facility for fine-grained power and energy management on multicore servers. In *Proc. ACM ASPLOS*, 2013.

[29] C. Shepard, A. Rahmati, C. Tossell, L. Zhong, and P. Kortum. Livelab: measuring wireless networks and smartphone users in the field. *ACM SIGMETRICS Performance Evaluation Review*, 38(3), 2010.

[30] S.J. Willson. A Value for Partially Defined Cooperative Games. *International Journal on Game Theory*, 21:371–384, 1993.

[31] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha. AppScope: Application energy metering framework for android smartphone using kernel activity monitoring. In *USENIX Annual Technical Conf.*, 2012.

[32] H. Zeng, C.S. Ellis, A.R. Lebeck, and A. Vahdat. ECOSystem: Managing energy as a first class operating system resource. In *Proc. ACM ASPLOS*, 2002.

[33] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R.P. Dick, Z.M. Mao, and L. Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proc. IEEE/ACM/IFIP CODES+ISSS*, 2010.

[34] L. Zhong and N.K. Jha. Graphical user interface energy characterization for handheld computers. In *Proc. ACM CASES*, 2003.