Detecting Energy Patterns in Software Development

November 16, 2011
Technical Report
MSR-TR-2011-106

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

Detecting Energy Patterns in Software Development

Ashish Gupta[1], Thomas Zimmermann[2], Christian Bird[2], Nachiappan Nagappan[2], Thirumalesh Bhat[2], Syed Emran[2]

[1] Indian Institute of Technology
Kanpur, India
ashgupta@cse.iitk.ac.in

[2] Microsoft Corporation
Redmond, WA, USA
{tzimmer, cbird, nachin, thirub, semran}@microsoft.com

***Abstract*—With the advent of increased computing on mobile devices such as phones and tablets, it has become crucial to pay attention to the energy consumption of mobile applications. The software engineering field is now faced with a whole new spectrum of energy-related challenges, ranging from power budgeting to testing and debugging the energy consumption. To the best of our knowledge there has been little work on the analysis of energy patterns. In this paper, we present our work for the Windows Phone platform. We first describe the data that is collected for testing (power traces and execution logs). We then present several approaches for describing power consumption and detecting anomalous energy patterns and potential energy defects. Finally, we describe prediction models to estimate the overall energy consumption based on usage of individual modules. This allows assessing the individual impact of modules on the overall energy consumption and supports overall energy planning.**

***Keywords*—power, energy, testing, power traces, power spikes, anomalies, prediction**

I. Introduction

I have researched all the many ways to save battery life. I have apps that kill other apps. I turn off Wi-Fi and 4G and Bluetooth until I need them.
—Scott Adams [1] (Creator of Dilbert)

For several decades, power consumption has been a secondary concern (if a concern at all) in software engineering.[1] Most software has been developed for desktop computers, which have a continuous power supply. While industries like satellite sciences and healthcare have been traditionally more power-aware, the general software engineering community did not have to research power consumption. *This is about to change*—or depending on the viewpoint has changed now. With mobile phones and tablets gaining wide usage in everyday life, new challenges are brought to the software engineering community. There are many stakeholders that now care about power: *end-users* like Scott Adams [1] realize that certain applications can reduce battery life dramatically and consider energy consumption as an important quality attribute. As a consequence building energy-efficient applications will become important for *developers*. There are many ways that a developer can influence the power consumption of a mobile app, for example, the decision to use TCP vs. UDP, or keeping sockets and connections open longer than needed. Another example is making a lot of requests to a server instead of batching up requests so that they utilize wireless (a high energy component) effectively. Ultimately power consumption comes down to how the hardware components are used, but these are driven by software design decisions.

Other communities such as Graphics, Human Computer Interaction, Networking and Systems have already started to work on research related to energy consumption (see Section III). With this paper, we present to the best of our knowledge the first study from a software engineering perspective on energy awareness problems.

We introduce a methodology for collecting and analyzing power data on mobile devices running Windows Phone 7. Our methodology focusses on three parts: (1) describe and quantify power consumption, (2) detect anomalies in power consumption, and (3) predict power consumption. Anomalies identified by our approach have been confirmed as true defects by developers who used the anomalies to perform root-cause-analysis to detect defects in phone software. More specifically the questions that we answer are:

- What modules consume the most power?[2]
- Does co-occurrence of specific modules increase or decrease energy consumption?
- What are the characteristic energy shape patterns of certain modules?
- What are anomalies in the power traces?
- Can we predict overall power consumption?

These results also hold value the major stakeholders in mobile devices. The *OS platform developers* and *application developers* specifically need to be aware of individual energy consumption patterns and can use overall prediction models to determine the energy usage within in a particular scenario to decide on the need for energy optimizations or rethink the design aspects of the scenario. *End-users* need to be aware of the energy consumption levels to plan better for the battery life under different load conditions. These are two simple situations where knowing about energy patterns is of value. In the remainder of the paper we discuss in detail the under-

[1] Throughout the rest of this paper we use the term *energy* and *power* interchangeably.

[2] We use the term *modules* (or components) for executable files and shared libraries.

lying analysis and methodology. One goal of this paper is to also expose the need of more software engineering research on energy awareness, utilization, and optimization.

This paper is organized as follows. We discuss in Section II the relevance of this work to software engineering, in Section III the related work and in Section IV the collection and alignment of power traces and execution logs. In Section V we present the descriptive data analysis and Section VI the predictive data analysis. In Section VII we close the paper with conclusions and consequences.

II. Relevance to Software Engineering

With this section we wish to emphasize why software engineering researchers and practitioners should care about energy, as the importance might not be immediately clear. The events of the last few two years have significantly changed the face of personal computing and we present two different observations:

- Energy awareness is relevant now
- Energy awareness is relevant for the software engineering community

A. *Energy awareness is relevant now*

The main reason for the increased importance of energy analysis is because of the advent of smart phones and tablets. With the explosion of smartphones (for example, Windows Phone, Android, iPhone, and Blackberry), Nielsen Media Research expects more smartphones in the U.S. market than feature phones in 2011 [2]. The market analysis company IDC estimates the media tablet market in 2010 at nearly 17 million units and forecasts 44.6 million will ship in 2011, with the U.S. representing nearly 40% of the total. In 2012, IDC forecasts worldwide shipments of 70.8 million units [3]. According to IDC, smartphone started outselling PCs in the fourth quarter of 2010 with 100.9 million shipped devices vs. 92.1 million and there will be more mobile Internet users than wire line users in the U.S. by 2015 [4].

With the growth of tablets and smartphones the problems related to energy consumption are increasing. Both end-users and developers are sensitive to the energy consumed by individual components of the phone (such as Wi-Fi, 3G) as well as applications downloaded and running on the phone. While there is little formal research, several technology blogs analyze ways to improve battery consumption. A more reliable source, Computer World magazine discusses *"More tips for boosting Android battery life"* where they present ten simple options to increase battery life, including turning off Wi-Fi, turning off Bluetooth, dimming the background, and running an energy monitoring app [4].

B. *Energy awareness is relevant for the software engineering community*

The rapid growth of the market for mobile devices brings an urgent need for understanding various aspects of energy consumption. For example, the Networking community has started investigating the impact of energy consumption on bandwidth and wireless aspects [5,6,7] the CHI community has papers and tracks specifically focused on mobile computing devices and tablets (for example [8]). Given these recent developments it is imperative for the software engineering community to work on the energy aspects of mobile devices.

From a more specific context, there are broad reaching implications (*which our paper does not address*) for several research communities in software engineering. We hope our paper serves as an introduction for the software engineering community to get excited in energy awareness. A few of the high level areas, which are by no means complete in terms of areas or technical depth:

Requirements Engineering. In addition to the design of the functionality of the system, one of the goals of requirements documents is on avoiding defects but currently a focus on energy is often missing. Going forward most software and systems will need to be designed with energy in mind. This leads to several open questions for the requirements engineering community, for example: How do engineers determine energy limits in the requirements phase? Is there a power budget requirements engineers can work with?

Programming Languages and Software Development. The aspect of energy aware computing is still very new. For example, a fundamental question to which we have no proper answer yet is the *relationship between energy consumption and coding patterns*. We know that there is a positive relationship between energy consumption and utilization of the CPU: the more CPU is utilized, the more energy is consumed. Similarly we know that complex code (like coupled objects, or depth of inheritance) consumes more CPU cycles compared to simple sequential code. But we don't exactly know the relationship between the type of coding and energy consumption. For example, is the relationship between code with high complexity and energy consumption linear, quadratic, or exponential? Furthermore, there can be coding patterns that consume more power than others. While some of these patterns are known, many more need to be identified. Going forward design for energy awareness is an important topic and compiler optimizations for energy awareness will be an emerging need in future years. These are just a few small examples to highlight a rather fundamental question.

Testing and Analysis. Simply put: how do we test for energy? We have an extensive body of knowledge on testing, test prioritization etc. but we need to start designing new methods for testing applications, features, and components for energy-awareness, both to determine the amount of energy they consume and to ensure they are not consuming more than their allotted energy from a power budget. Testing will also evolve into connecting devices into applications to monitor power spikes, outliers etc. The work in this paper is a first step into this direction in order to test energy consumption and determine outliers in terms of energy consumption.

The above areas are merely meant as examples to highlight the potential of research into energy awareness from the software engineering community. Other related areas related to software engineering and energy awareness are for example, *human computer interaction* (HCI), e.g., how to collect data from mobiles devices in a controlled setting or for user hosted beta testing programs; and *security*, e.g., how to prevent malware applications that target mobile devices with the goal of depleting the energy stored in batteries (for example [6]).

III. RELATED WORK

To the best of our knowledge there has been no research in the software engineering community on energy analysis, energy awareness, and energy debugging. However there has been some work in the Networking and Systems communities, which we highlight below. Though most of these studies have no implications for software engineering, they involve monitoring the energy consumption in real world situations which is related to our energy measurement work for identifying anomalies and predicting energy consumption.

Balasubramanian et al. [9] measured energy consumption of three mobile networking technologies: 3G, GSM, and Wi-Fi. They observed that 3G and GSM have a high tail energy consumption and developed a protocol to reduce the energy consumption of common mobile applications by modeling the network activity for each technology.

Kim et al [5] discussed the fact that the limited battery-lifetime in mobile devices is worsened by the presence of mobile malware targeting the depletion of the battery. They presented a framework with a power monitor for collection power samples and building a power consumption history and a data analyzer that generates a power signature from the constructed history. Their results on a HP iPAQ show a 99% true-positive rate in classifying mobile malware.

Pathak et al. [7] observed that capturing power consumption data based on utilization of a hardware component is insufficient due to power behavior not always directly related to smartphone component utilization (because low level power optimizations in device drivers are missed). The authors presented a power modeling scheme to accurately measure power consumption based on utilization and non-utilization on the Android and Windows Mobile platforms. Similarly Ge et al. [10] designed a framework called PowerPack to measure and isolate the power consumption in disks, memory, multi-core, and multiprocessor based nodes. PowerScope [11] is another energy profiling tool and combines hardware instrumentation with kernel software support to measure the system activity. For the measurements in this paper, we used the *actual power consumption* rather than relying on models based on utilization of components.

The closest in spirit to our work, is the work by Shye et al. [12] who developed and deployed with real users. They observed that the Screen and CPU are the two largest power consuming components. They also built a regression model to predict total energy consumed and identified patterns in user behavior to drive optimizations. The biggest advantage of our study over Shye et al. [12] is the level of granularity. The observation that Screen and CPU consume most power is at a very coarse level of granularity and not of much use to developers and users. Similarly, without any fine-grained level of information, the regression models do not help in optimizing usage patterns in an operational way. Instead of Hardware component level (CPU, screen), our work is on module level, which is more actionable for developers.

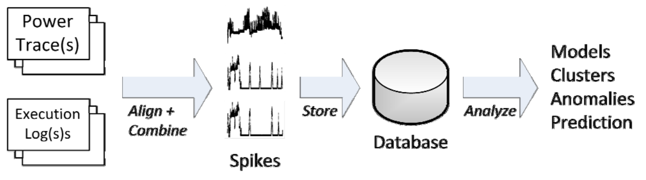


Figure 1. Overview of our approach. We use two data sources: power traces and execution logs. The data is then aligned and combined and we extract power spikes from, which are stored in database and serve as input for the analysis throughout this paper.

While not directly related to energy awareness on mobile devices, energy optimization is an increasing important topic in datacenter operations in the systems and networking research community. Recently the First Conference on Energy-Efficient Computing and Networking focused on these aspects. For example, Schröder et al. [14] presented a vision minimizing the energy costs, using geographical specific characteristics of servers and data centers and a potential savings of up to 40% using distributed load management. Das et al. [15] from IBM Research discussed the importance of utility functions in controlling the energy of datacenters. They also discussed energy savings using experimental data from a real data center collected by room air-conditioning. These research topics are still an emerging area of systems and networking research and are aligned with the advancement of energy awareness form a software engineering viewpoint in coming years.

IV. DATA COLLECTION

We use two sources of data for the power analysis in this paper (see Figure 1). We first collect logs of the executable files and shared libraries, hereafter referred to as *modules*, which are active at certain points in time (Section IV.B). Next we collect traces of power consumption over time (Section IV.C and IV.D). Finally we align and combine both sources into the data that we will use throughout this paper (Section IV.E).

A. *Overview*

When a mobile device is tested for power usage, a recent build is loaded onto the phone. Based on the operations that are being tested a number of tests are run repeatedly on the phone for a 12 hour period. The record of power usage from measured by a power meter is called a *power trace*.

As mobile devices optimize for power consumption, power traces show periods of inactivity (low power use),

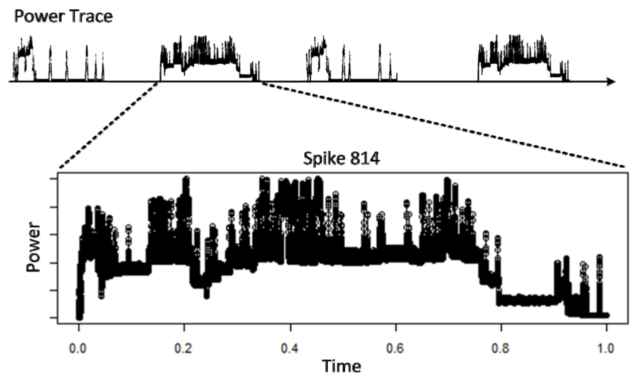


Figure 2. Examples of a power trace (usually taken from of a 12 hour test session on a mobile device) and a power spike. Spikes are isolated based on longer periods of inactivity in the power trace. Each such spike is then normalized in time to range from 0 to 1.

punctuated by brief periods of high activity (high power use). We term each of these high power intervals a *power spike* or just *spike*. Note that within a spike, there may also be fluctuations in power consumption. The duration of spikes ranges from one tenth of a second to several seconds. In order to isolate individual power spikes in a power trace, we use the periods of inactivity as shown in Figure 2. In some cases a manual approach may also be used depending on the nature of the test cases, for example, when there are not enough idle periods in the data. For the analyses presented in this paper, we used so-called idle tests, which on purpose leave enough idle time between test activities.

After identifying spikes and aligning power traces with execution logs, we have the following information available for each spike (example is depicted in Figure 3):

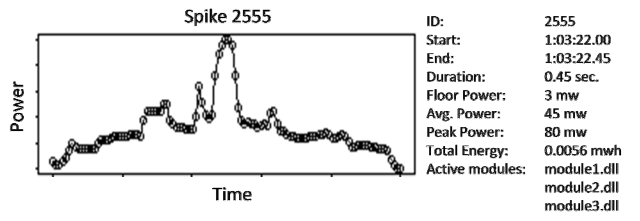
- Spike ID, a unique identifier for the spike
- Start time and end time (hours : minutes : seconds)
- Duration (seconds)
- Floor power (milliwatt)
- Average power (milliwatt)
- Peak power (milliwatt)
- Total energy consumed (milliwatt hours)
- Active modules

This data allow us to perform a number of analyses on energy use as discussed in Section IV and V. To facilitate access, we store the spike information in a database.

In addition to the above information, each spike is linked to the power trace where it originated from. For power traces we record in the database:

- Build ID, which allows to find the associated state of the source code
- Model of the mobile device that was used to collect the trace.

Note that for the analysis presented in this paper we do not aggregate spikes across different mobiles devices models because they may have slightly different power utilization characteristics due to varying specifications (such as display type, processor speed, existence of specific sensors).



ID:	2555
Start:	1:03:22.00
End:	1:03:22.45
Duration:	0.45 sec.
Floor Power:	3 mw
Avg. Power:	45 mw
Peak Power:	80 mw
Total Energy:	0.0056 mwh
Active modules:	module1.dll
	module2.dll
	module3.dll

Figure 3. A spike from a trace log and the associated data that is computed and stored in a database for further analysis.

B. *Collecting execution data*

The first source of data comprises information regarding what code is being executed on the mobile device at a specific period of time. Unfortunately, measuring code execution at fine granularity (recording method calls, statement execution, etc.) is highly intrusive and introduces an overhead in terms of storage and CPU utilization that creates a non-trivial level power usage on the device itself. Simply writing the record of execution to flash uses significantly more power than would actually occur during normal use by a consumer. Thus, we use a more coarse grained level of data. At each context switch and whenever a thread is started or dies, we record the time for the active thread. The modules active at a point in time are indicative of the type of operation occurring on the mobile device. For instance, there may be active modules for Wi-Fi operations or HTML rendering. We call the sequence of context switches with the accompanying time and lists of active modules, the *execution log*.

C. *Measuring energy consumption*

In addition, we measure the actual energy consumed by the mobile device during operation. We use a special power meter that connects to the battery contacts of the mobile device in place of the battery itself and measures power draw at a rate of 5,000 samples per second. The power meter reports power draw and time of each sample to a standard desktop machine via USB. While this sampling rate may seem low by modern standards, mobile devices context do switch less often than normal computers (one reason is minimizing power use) and only have one active user level application running. Thus, we actually record a large number of power samples (on the order of thousands) for each context switch.

D. *Controlling for the many sources of energy consumption*

There are three main components that are directly linked to power consumption on mobile devices: the CPU, the cellular chipset, and the graphics processing unit (GPU). For the analysis in this paper, we focus on CPU activity and control for activity on the chipset and GPU as discussed below.

While there are many other components such as modem, speakers, and GPS that consume power on mobile devices, they are typically controlled by code that is executed on the CPU and thus show up on our execution logs. For instance, driver code may transmit or receive information via wireless modem. Code executed on the CPU may update the display, activate the GPS sensor, or play music on the speakers. While each of these operations leads to power usage, they are ultimately dictated by the code loaded and executing on the CPU. This allows us to analyze the relation between operations and power usage strictly by examining modules that are active in our execution logs.

There are two sources of power usage, which are largely independent of the CPU:

- The *cellular chipset*, which is responsible for maintaining contact with cellular towers as well as making and receiving calls, runs on its own firmware. We do not monitor its execution because it is not a general purpose CPU and is proprietary.
- In addition mobile devices have *graphics processing units (GPUs)* that provide hardware acceleration capabilities. We currently do not monitor the code executing on this chip due to data gathering limitations.

Because we measure the aggregated power usage on the device, these cellular chipset and GPU may introduce a small amount of noise to our analysis of power usage by the CPU. We mitigate these problems in three ways.

First, we do not make or receive any phone calls while recording power usage and code execution for our analysis. While the cellular chipset contacts the nearest towers every few seconds, this operation is consistent both in the power draw and timing. Furthermore the location of the phone is static during testing, so that the distance to cell towers and thus the power used to make contact, remains constant. This allows us to remove the power spikes that occur as a result of the cellular "heartbeat".

Second, although we do not record the code executing on the GPU, the content of the phone display is almost always indicative of the operations being performed (and the code executing on the CPU). Therefore we expect that power usage resulting from GPU activity does not introduce any bias to our analysis.

Third, we collect the power traces from 12 hours sessions with different mobile device usage scenarios, each executed a large number of times. Therefore we expect that the large number of samples will hide any noise introduced by other sources of power use on the mobile device.

E. *Aligning execution log and power sample*

Before we combine power traces and execution logs to extract spikes, we need to ensure that the time for the power usage (recorded via the power meter on a desktop machine) is in sync with the time for the executed code (recorded on the mobile device).

We use a process that we term *temporal alignment* to sync both data sources. The process consists of two steps:

1. First, we identify the spikes corresponding to the cellular antenna contacting cell towers by locating a set of spikes that (a) occur at regular intervals, (b) last the same amount of time, and (c) have the same power usage.
2. After removing these spikes, we align based on the time intervals between spikes in the power trace and between context switches in the execution log.

As a concrete example, if there is one period of 3 minutes with no context switches or code execution in the execution log and one period of 3 minutes of low power usage in the power trace, then these corresponding points in time can be used as anchors to align the rest of the two data sources.

While our approach for alignment is automated, a manual approach may also be used depending on the context of the data collection and the regularity of the data itself.

V. Descriptive Analysis

The first part of energy analytics is of descriptive nature: for one or more given power traces, engineers want to understand the energy consumed by different modules and if there are any patterns (anomalies) that they should pay special attention to. They typically ask questions such as:

- What modules consume the most power?
- Does co-occurrence of specific modules increase or decrease energy consumption?
- What are the characteristic energy shape patterns of certain modules?
- What are anomalies in the traces?

To answer these questions, we implemented a tool based on supervised learning techniques (for example, regression models, decision trees) and unsupervised learning techniques (frequent pattern mining, clustering). All our analysis is done in an automatic fashion and requires little involvement and statistical knowledge on the user side.

A. *What modules consume the most power?*

Because in the data, energy consumption is only linked to a *set of modules* and *not individual* modules, it is not trivial to identify the energy consumption of a single module. One of the reasons is that besides the granularity limitations (as discussed in the previous section), spikes often have tail-end energy which cannot be attributed to any single module in the spike. To isolate the energy consumption for different modules we use therefore two supervised techniques: regression analysis and decision trees. For this analysis, we used the following input data:

Spike ID	Adrion.dll	Allen.dll	Bachman.dll	Backus.exe	……	Zweben.exe	Avg. Power (mW)
123338	1	0	0	1		1	254.76
123563	0	0	1	0		1	680.23
123789	0	1	1	0		0	110.56

Each observation has a unique spike identifier, followed by flags to indicate the presence of modules in the spike (0 for absence, 1 for present), and the average power consumption. For legal reasons, we anonymized module names throughout the paper with the last names of laureates of ACM Turing Awards as well as ACM SIGSOFT Distinguished Service Awards, Outstanding Research Awards, and Influential Educator Awards.

Factor	Estimate β_i
(Intercept)	297.053
Lampson.dll	608.625
Mills.dll	421.397
Weyuker.dll	264.602
Valiant.dll	252.904
Hamming.dll	192.448
Adrion.dll	118.889
Rabin.dll	–33.156
Feigenbaum.dll	–42.860
Ritchie.dll	–88.667
Holzmann.dll	–137.778
Harel.dll	–139.671
Leveson.dll	–183.723
Sommerville.dll	–292.122
Johnson.dll	–329.861

Figure 4. Example of a regression model for the relationship between average energy consumption and the presence of individual modules. Modules with high values of β_i increase the average energy consumption substantially. Only statistically significant coefficents are included.

Regression analysis. With linear regression [13] we model the relationship between the average energy consumption y of a spike and the presence of individual modules as denoted by the dummy variables x_m (which take values 0 or 1) for module m.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_M x_M$$

Informally, if a module m is present ($x_m=1$), it contributes β_M to the estimated energy consumption.

Figure 4 shows an example of a regression model. The model has an intercept β_0 of 297mW, which is an estimate of the energy consumption when no modules are present in a spike. (Note that this number does not correspond to the energy consumption in idle state.) The remaining coefficients show that the modules Lampson.dll (estimate 608mW) and Mills.dll (estimate 421mW) are associated with high energy consumption in spikes. The modules that are associated with power spikes with a low average energy consumption have negative coefficients, for example Sommerville.dll (–292mW) and Johnson.dll (–329 mW). In Figure 4, we only included modules for which the coefficients are different from 0 at a significance level of 0.001 (t-test).

The estimates in Figure 4 allow engineers to better understand which modules are likely correlated with high or low power consumption. It is important to understand that the coefficients are estimates and do not necessarily correspond to the actual power consumption. We unfortunately do not have the actual energy contribution of the modules and could not assess the correctness of the coefficients. However, we did validate the coefficients for several traces with the engineers who confirmed the correctness of rankings based on their previous experience.

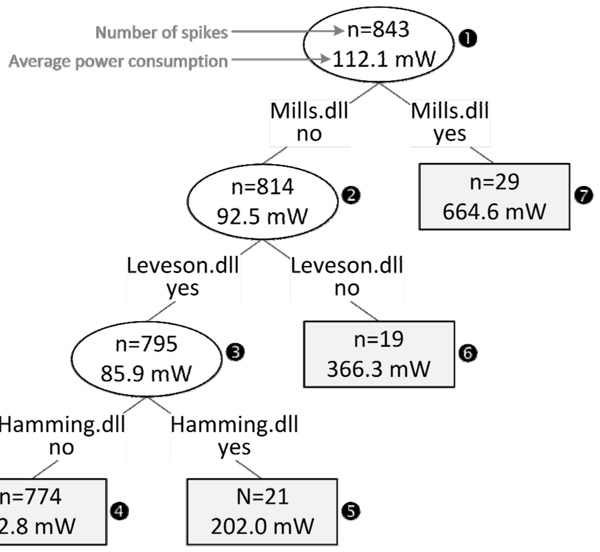


Figure 5. Example of a decision tree learned from energy consumption data. On the first level the spikes are split based on the presence of module Mills.dll—spikes that contain Mills.dll ❼ consume on average *six times* the power than spikes that do not contain the module ❷.

Decision trees. In addition we use decision trees [14] to model the influence of modules on average energy consumption. In our case, the inner nodes indicate the presence of certain modules (yes/no). Each node holds the average energy consumption for several spikes (as described by the path to the root note). For example in Figure 5, node ❸ lists an average energy consumption of 85.9 mW for the 795 power spikes for which Mills.dll is absent and Leveson.dll is present.

The decision trees help developers to better understand how power consumption and modules are related in one or more traces. Figure 5 shows a decision tree that describes 843 spikes within a trace. The average energy consumption for all spikes is 112.1mW as indicated in the root node ❶ of the tree. On the first level the spikes are split based on the presence of module Mills.dll: for the 814 spikes that do not contain Mills.dll, the average energy consumption is 92.5mW ❷; however, for the 29 spikes that contain Mills.dll the average *increases by six times* to 664.6mW ❼. On the second level, the absence of Leveson.dll increases energy consumption by a factor of four (compare nodes ❸ and ❻), and on the third level the presence of Hamming.dll increases energy consumption by a factor of three (nodes ❹ and ❺).

We informally validated the decision trees with engineers who confirmed their usefulness for understanding trace data.

A limitation of our current data (not the approach) is that we only have information about the presence of modules (0 or 1), but not the actual usage (numerical). Both regression and decision trees do support numerical input data and we are currently exploring other lightweight tracing techniques for collecting more fine-grained data without altering power usage.

B. *Does co-occurrence influence energy consumption?*

To identify pairs of modules that significantly increase energy consumption *in combination* but not individually, we use frequent pattern mining [14] and statistical significance testing [13]. First we split spikes into two groups based on the average energy consumption:

- *High*. This group contains the 10% of spikes with the highest average power consumption.
- *Medium/Low:* This group contains the remaining 90% of spikes.

We then count for each pair p of modules the frequency in the High group and the Medium/Low group. The result is a contingency table:

	Does contain p	Does not contain p	
High	A	C	10%
Medium/Low	B	D	90%

For each pair $p=(m_1, m_2)$ we compute the likelihood that spikes which contain p are part of the High group

$$P(\text{High} \mid \text{contains } p) = A/(A+B)$$

We then use Fisher Exact value tests [13] to compare with the likelihoods for the complement (does not contain p) and the individual modules m_1 and m_2:

- $P(\text{High} \mid \text{does not contain } p) = C/(C+D)$
- $P(\text{High} \mid \text{contains } m_1)$
- $P(\text{High} \mid \text{contains } m_2)$

In several power traces, we found pairs of modules that increased power when used in combination but not individually. Fortunately, not all traces have "power-hungry" pairs of modules. This analysis helps developers to focus on combinations of modules and not just individual modules. In future work, we plan to extend our approach to look arbitrary sets of modules and not just pairs.

C. *What are the characteristic energy shape patterns?*

Power traces consist of hundreds, often thousands of spikes, which all can have very similar shapes. By clustering spikes based on their shapes, we can identify characteristic shape patterns and reduce the number of spikes that need to be investigated by developers for a power trace. Rather than looking at all spikes, developers instead can focus on a small number of clusters (typically 10-20), each corresponding to a characteristic shape pattern with a list of associated spikes. Developers can also rollup the meta-information for each spike (such as length, modules, etc.) to the cluster level.

Developers can choose different input data for clustering. They can cluster all spikes in a power trace or only subsets, for example all spikes related to a module. Figure 6 shows a small example with 11 spikes for module Sommerville.dll. While the human eye can easily spot two clusters, detecting the clusters in an automated fashion is slightly more complicated. We need a distance function to compare spikes and a clustering technique:

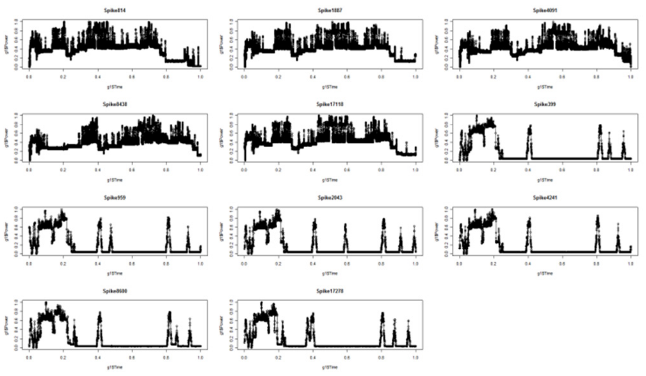


Figure 6. Example with 11 spikes for module Sommerville.dll.

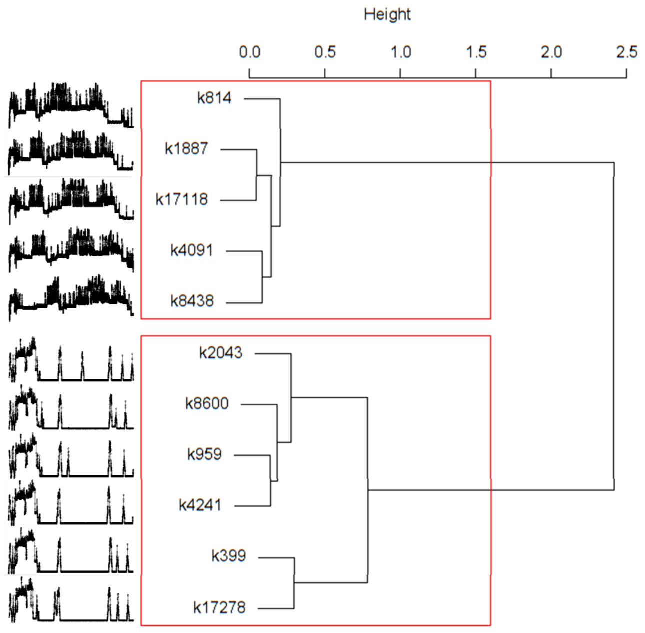


Figure 7. Dendrogram of the hierarchical clustering of the 11 spikes for module Sommerville.dll. Initially each spike is assigned to its own cluster and then iteratively at each stage the two most similar clusters are joined.

- *Distance function.* To compute the distance between two spikes, we use the Kullback-Leibler divergence [15]. To reduce the computational cost of comparing spikes, we divide each spike into 100 buckets, calculate the *average* energy consumption for each bucket, and compute Kullback-Leibler across these 100 buckets for each pair of spikes. The result of this step is a distance matrix D, where a cell value d_{xy} corresponds to the distance between spike x and y.

- *Clustering.* For clustering spikes we use the *Ward* hierarchical clustering method [16]. Initially, each spike is assigned to its own cluster; for an example see the spikes k814 to k17278 for module Sommerville.dll in Figure 7. Then iteratively at each stage the

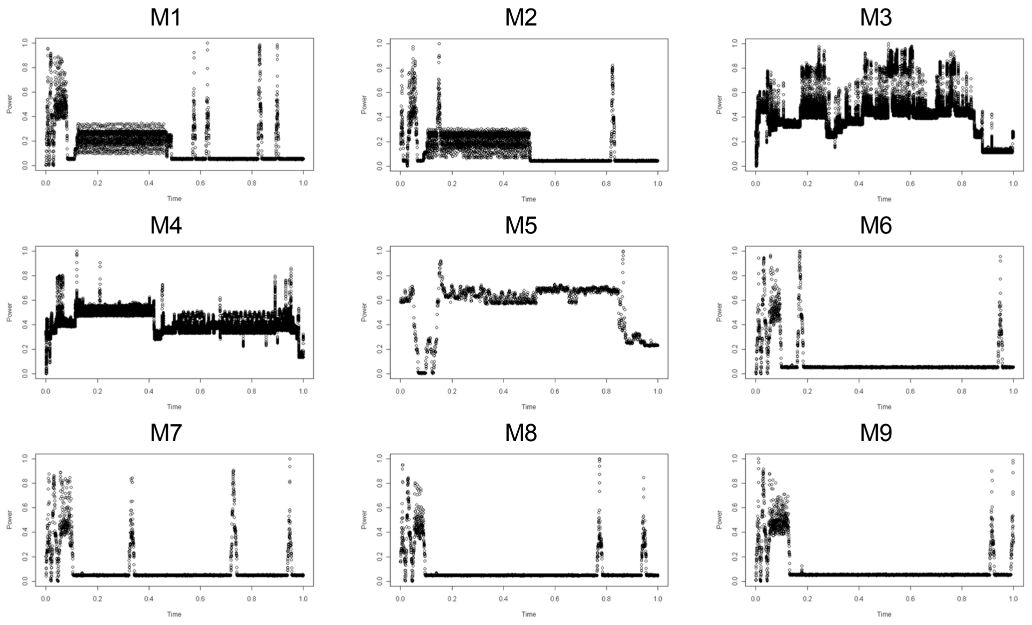


Figure 8. The nine clusters M1 – M9 of a manual card sort of 588 spikes. For each cluster we show one characteris spike.

two most similar clusters are joined until there is just a single cluster. For example k1887 and kk17118 are joined first and later combined with the cluster of k4091 and k8438. The result of hierarchical clustering is a tree-diagram of clusters (called *dendrogram*) that indicates the join order. The tree can then be cut into a certain number of clusters; in Figure 7 we cut the tree in two clusters as indicated by the red boxes.

To assess the quality of the clustering approach we built a gold set by manually clustering 588 spikes using a card sort [17] based on similarity of the shapes. The first four authors sorted 147 spikes each, resulting in four separate clusterings. Then clusteres were discussed and the four authors agreed on one clustering for all 588 cards. The resulting nine clusters of the manual card sort are displayed in Figure 8. Next we automatically clustered the 588 spikes with Ward and Kullback-Leibler and compared the two clusterings with the Meiler variation of information (VI) index. The values of the VI index range from 0 for identical clusterings to $2 \log_2 k$ for completely different clusterings where k is the number of clusters; in our case k=9, thus the best value for VI is 0 and the worst value is 6.34. The VI index for the automated clustering and the manual clustering is 1.44. While not perfect, our current clustering technique still has a high agreement with the gold set of roughly 75%.

In order to inspect the cluster differences we counted for each pair of a manual cluster M_i and an automated cluster A_j, the number of spikes that M_i and A_j share. The results are shown in Figure 9 and allow us drawing conclusions about the overlap between clusters. Most of the cluster pairs do not have any spikes in common as indicated by the many zeros, which is a good result. However, the automated clustering is not perfect: for example, M5 is split across two automated clusters A6 (with 98 spikes) and A7 (with 103 spikes). Also, the automated clustering does not distinguish between M6 to M9 as shown by the non-zero values with clusters A1 to A5 (dark gray area). In the card sort we considered the number and position of the peaks after the first initial peak as important; however, our current distance measure does not consider such properties. We can make a similar observation for M1 and M2 (light gray area), which differ in the numbers of peaks, but are combined into cluster A8 by the clustering. In our future work, we will evaluate other distance measures and clustering algorithms to improve the good performance of Ward clustering with Kullback-Leibler.

Rather than treating spikes as a simple time series, we can describe them with feature-vectors and include features such as number and position of peaks in the spikes. Another example is *dynamic time warping* [18], an algorithm frequently used in speech recognition for measuring the similarity between two sequences which may vary in time or speed.

	M1	M2	M3	M4	M5	M6	M7	M8	M9	**All**
A1	0	0	0	0	0	4	19	20	0	43
A2	0	0	0	0	0	16	3	10	5	34
A3	0	0	0	0	0	50	2	2	0	54
A4	0	0	0	0	0	67	4	8	0	79
A5	0	0	0	0	0	2	0	14	5	21
A6	0	0	0	0	**98**	0	0	0	0	98
A7	0	0	19	9	**103**	0	0	0	0	131
A8	59	24	4	0	0	0	0	0	0	87
A9	0	1	34	6	0	0	0	0	0	41
All	59	25	57	15	201	139	28	54	10	588

Figure 9. Comparison between the automated clustering and the manual card sort. Overall the agreement is fairly high.

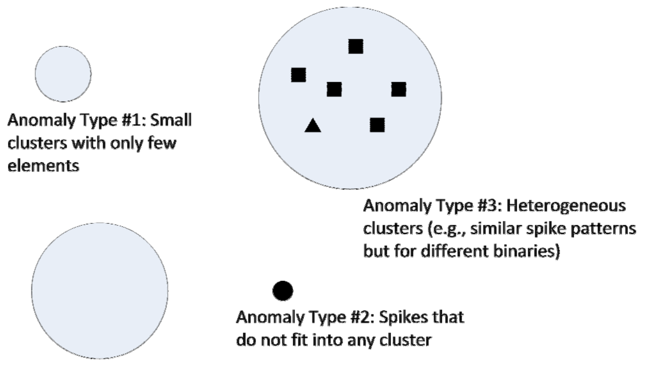

Anomaly Type #1: Small clusters with only few elements
Anomaly Type #3: Heterogeneous clusters (e.g., similar spike patterns but for different binaries)
Anomaly Type #2: Spikes that do not fit into any cluster

Figure 10. Three types of anomalies that can be identfied via clustering.

Alternatives for clustering are for example K-means and model-based approaches [14].

D. What are anomalies in the traces?

Currently developers are faced with the strenuous work of going through the power logs and reporting any inconsistency manually. One way to *automatically* identify anomalies in power traces is to look for extreme coefficients in the regression models or for extreme splits. Another way is to use the clustering of spikes. The underlying assumption is that spikes with similar shapes also share similar behavior. There are several types of anomalies based on clustering approaches (see Figure 10):

- *Small clusters.* Clusters that have only few spikes are candidate for anomalies. Here the cluster and all its spikes are treated as anomalies. In contrast, large clusters are likely not to be anomalous, because of the wisdom of the masses.
- *Spikes that do not fit.* Individual spikes that do not fit any cluster well are also candidates for anomalies.
- *Heterogeneous cluster*s. Within a (large) cluster we check for the homogeneity of the spikes. Recall that we for clustering we use the shape of spikes, which means that within a cluster the shapes are expected to be similar. To find anomalies within clusters, we therefore compare additional meta-information such as modules, average and peak power consumption, and duration of each spike. For example, if all spikes except for one spike contain the module Williams.dll (within a cluster), we consider the spike that does not contain Williams.dll to be anomalous.

We confirmed the identified anomalies with developers and were able to find bugs associated with modules Mills.dll, Ritchie.dll, Holzmann.dll, and Wirth.dll. The reason for the bug was a communication client (Wirth.dll), which was waking up every 30 minutes, but was not closing the network socket (Mills.dll) properly. The anomaly showed up as several spikes that were either running longer (several minutes) than other spikes within their clusters or as spikes that did not fit any cluster. Several anomalous spikes are displayed in Figure 11.

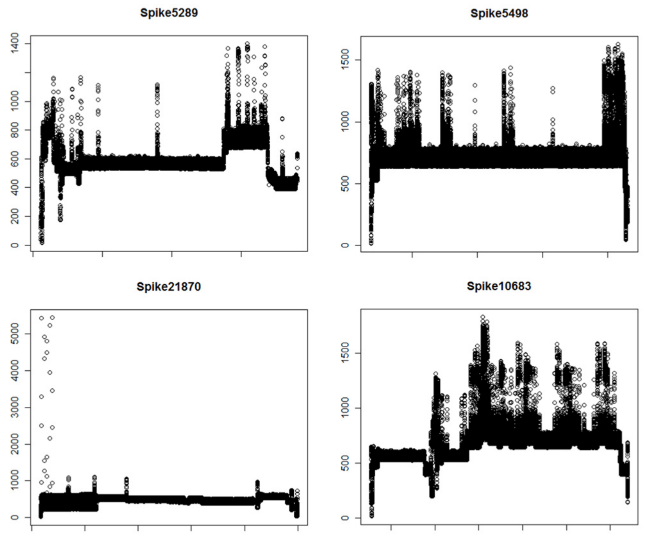


Figure 11. Some of the anomalous spikes identified by our approach.

VI. Predictive Analysis

Mobile applications are often developed within a power budget, that is, on average the application is only allowed to consume a certain amount of energy. Models that estimate power usage prior to development help developers in planning and allow them to stay within a power budget.

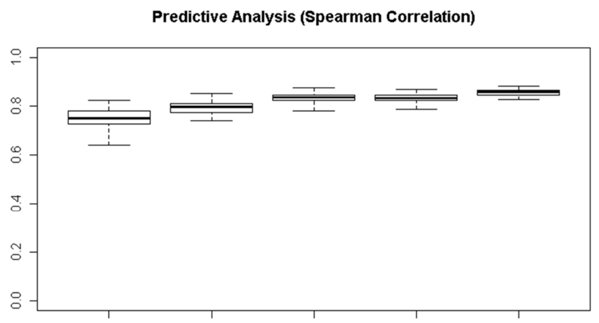
To test the hypothesis that power consumption can be predicted with modules used by an app, we built and tested prediction models for five different datasets:

- T1, T2, T3, and T4 are power traces (with 843, 912, 828, and 634 spikes respectively) and
- T1234, which is the combined dataset of T1–T4 (with 3215 spikes)

Each dataset contains a list of spikes with associated modules (input variables) and the average power consumption for each spike (output variable). For the predictions, we used linear regression models and focused on the *ranking* problem, i.e., predict the spikes that consume most power on average based on the modules used.

To assess the models we used a standard evaluation technique for prediction experiments called *data splitting* [19]: for each dataset, we randomly selected two thirds as training and one third as testing set and repeated this step 50 times. As evaluation criterion, we selected Spearman rank correlation, which is a commonly-used robust correlation technique because it can be applied even when the association between elements is non-linear [13]. Positive correlations result in a value of 1 and negative correlations in -1. For no correlation between the elements, the value is 0. In particular, a high positive value for Spearman means that two rankings are similar (or identical for a value of 1).

The prediction results are displayed in Figure 12 as box plots, which show the smallest value, lower quartile, median, upper quartile, and largest value of the Spearman correlation. The results show that our models reliably identify high-



	T1	T2	T3	T4	T1234
Min	0.6416	0.7388	0.7814	0.7856	0.8273
Median	**0.7495**	**0.7968**	**0.8356**	**0.8352**	**0.8596**
Max	0.8251	0.8535	0.8762	0.8691	0.8835

Figure 12. The Spearman correlation values for the prediction experiments. The median Spearman correlation in the experiments ranges from 0.7495 to 0.8596, which is considered to be a strong correlation

power consuming spikes. The lowest correlation for all 250 runs is 0.6416. The median Spearman correlation in the experiments ranges from 0.7495 (T1) to 0.8596 (T1234), which is considered to be a strong correlation [20]. It is noteworthy that the Spearman correlations are the highest for the T1234 dataset, which is the composition of T1–T4. This suggests that traces from different applications can lead to better predictive performance. In summary, the presence of modules can produce a good ranking of power-consuming spikes, as demonstrated by very high correlation values between the predicted and observed values. Such predictions can help developers to optimize the power budget.

VII. Conclusions and Consequences

With the increasing popularity of mobile devices such as smartphones and tablets, energy awareness has become an important issue that all software engineers should care about. In this paper, we have presented a data analysis on Windows Phone 7 usage data. We addressed several questions related to identifying modules with most power-consumption, finding characteristic energy shape patterns, detecting anomalies, and predicting power consumption based on module usage.

Understanding which of modules consume more energy is useful information to both application and platform developers and helps them to drive better design, test efforts and influence new user scenarios. These results also enable users to understand how to conserve battery power energy; for example there are several public discussions on how to conserve energy for the phone by using various combination of Hardware components [4]). While this paper does not name specific modules/applications for legal reasons, the described methodology could be applied by end-users to understand how to improve battery life by using certain combination of components and applications.

Another important goal of this paper is to call the attention of the software engineering community for working on problems related to energy awareness from several different perspectives such as requirements engineering, programming languages, as well as testing and analysis. We plan to work with researchers in the testing community to leverage our techniques for optimizing testing for energy awareness. We have merely scratched the surface of this area and plan to expand our research in this area spanning user testing and reliability. Finally, we hope that many others in the software engineering community will follow us to work on problems related to energy awareness.

Acknowledgements: Thanks to the Microsoft Windows Phone team for their help in understanding the data. Ashish Gupta performed this work during a summer internship at Microsoft Research.

References

[1] Adams, S. *Uncommunication Devices*. http://dilbert.com/blog/entry/uncommunication_devices. 2011.

[2] Entner, R. *Smartphones to Overtake Feature Phones in U.S. by 2011*. http://blog.nielsen.com/nielsenwire/consumer/smartphones-to-overtake-feature-phones-in-u-s-by-2011/. 2010.

[3] IDC. *IDC - Press Release*. http://www.idc.com/about/viewpressrelease.jsp?containerId=prUS22660011. 2011.

[4] Raphael, J. *Android battery life: 10 ways to make your phone last longer*. http://blogs.computerworld.com/16965/improve_android_battery_life. 2010.

[5] Kim, H., Smith, J., and Shin, K.G. Detecting Energy-Greedy Anomalies and Mobile Malware Variants. In *MobiSys '08: Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services* (2008), 239-252.

[6] Cheng, J., Wong, S., Yang, H., and Lu, S. Smartsiren: Virus detection and alert for Smartphones. In *MobiSys '07: Proceedings of the 5th International Conference on Mobile Systems, Applications, and Services* (2007), 258-271.

[7] Pathak, A., Hu, Y.C., Zhang, M., Bahl, P., and Wang, Y.-M. Fine-Grained Power Modeling for Smartphones Using System Call Tracing. In *EuroSys '11: Proceedings of the Sixth European Conference on Computer Systems European Conference on Computer Systems* (2011), 153-168.

[8] Iqbal, S.T., Horvitz, E., Ju, Y.-C., and Mathews, E. Hang on a sec! Effects of Proactive Mediation of Phone Conversations while Driving. In *CHI '11: ACM CHI Conference on Human Factors in Computing Systems* (2011), 463-472.

[9] Balasubramanian, N., Balasubramanian, A., and Venkataramani, A. Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications. In *Internet Measurement Conference* (2009), 280-293.

[10] Ge, R., Feng, X., Song, S., Chang, H.-C., Li, D., and Cameron, K.W. PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications. *IEEE Transactions on Parallel and Distributed Systems*, 21, 5 (2010), 658-671.

[11] Flinn, J. and Satyanarayanan, M. PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications. In *WMCSA '99: Workshop on Mobile Computing systems and Applications* (1999), 2-10.

[12] Shye, A., Scholbrock, B., and Memik, G. Into the Wild: Studying Real User Activity Patterns to Guide Power Optimizations for Mobile Architectures. In *MICRO '09: 42st Annual IEEE/ACM International Symposium on Microarchitecture* (2009), 168-178.

[13] Waserman, L. *All of Statistics: A Concise Course in Statistical Inference*. Springer, 2010.

[14] Han, J., Kamber, M., and Pei, J. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2011.

[15] Kullback, S. and Leibler, R.A. On Information and Sufficiency. *Annals of Mathematical Statistics*, 22, 1 (1951), 79–86.

[16] Hastie, T., Tibshirani, R., and Friedman, J. *The Elements of Statistical Learning*. Springer, 2009.

[17] Wright, G. and Ayton, P. Eliciting and modelling expert knowledge. *Decision Support Systems*, 3, 1 (March 1987), 13-26.

[18] Myers, C.S. and Rabiner, L.R. A comparative study of several dynamic time-warping algorithms for connected word recognition. *The Bell System Technical Journal*, 60, 7 (September 1981), 1389-1409.

[19] Munson, J. and Khoshgoftaar, T. The Detection of Fault-Prone Programs. *IEEE Transactions on Software Engineering*, 18 (1992), 423-433.

[20] Cohen, J. *Statistical power analysis for the behavioral sciences*. Routledge Academic, 1988.