Software-Based Energy Profiling of Android Apps: Simple, Efficient and Reliable?

Dario Di Nucci*, Fabio Palomba*, Antonio Prota*, Annibale Panichella†, Andy Zaidman‡, Andrea De Lucia*
*University of Salerno, Fisciano (SA), Italy
‡SnT Centre, University of Luxembourg, Luxembourg
†Delft University of Technology, The Netherlands

***Abstract*—Modeling the power profile of mobile applications is a crucial activity to identify the causes behind energy leaks. To this aim, researchers have proposed hardware-based tools as well as model-based and software-based techniques to approximate the actual energy profile. However, all these solutions present their own advantages and disadvantages. Hardware-based tools are highly precise, but at the same time their use is bound to the acquisition of costly hardware components. Model-based tools require the calibration of parameters needed to correctly create a model on a specific hardware device. Software-based approaches are cheaper and easier to use than hardware-based tools, but they are believed to be less precise. In this paper, we take a deeper look at the pros and cons of software-based solutions investigating to what extent their measurements depart from hardware-based solutions. To this aim, we propose a software-based tool named PETRA that we compare with the hardware-based MONSOON toolkit on 54 Android apps. The results show that PETRA performs similarly to MONSOON despite not using any sophisticated hardware components. In fact, the mean relative error with respect to MONSOON is always lower than 0.05. Moreover, 95% of the estimation errors are within 5% of the actual values measured using the hardware-based toolkit.**

***Index Terms*—Energy Consumption; Mobile Apps; Estimation**

I. INTRODUCTION

Nowadays, over 2 billions users rely on smartphones and tablets to perform their daily activities as reported by the Statistics Portal association [1]. Not only do users play games or send messages, they use mobile applications (*a.k.a.*, apps) for every type of need, including social and emergency connectivity [2]. Due to this ever-increasing number of mobile devices and apps, energy consumption is becoming a critical factor in user satisfaction for both paid and free apps [3].

Energy related issues mainly involve the efficiency of hardware components such as the CPU and other electronic elements. However, Flinn and Satyanarayanan [4] pointed out that *"there is growing consensus that advances in battery technology and low-power circuit design cannot, by themselves, meet the energy needs of future mobile computers"* [4]. This observation has been confirmed by recent advances in *green* software engineering, which demonstrated how the source of energy leaks can be software as well [5], [6], [7], [8].

For instance, Sahin *et al.* [5] have shown that good design principles, and design patterns in particular, have a *negative* impact on the energy efficiency of mobile apps. Along the same line, previous studies have investigated the power efficiency consequences of refactoring, demonstrating the flip side of operations that are supposed to improve non-functional attributes of source code [7], [9].

Despite the aforementioned research efforts, Harman *et al.* [10] highlight that there is a lack of tools that quickly and efficiently measure the energy consumption of mobile applications. Existing tools fall into three main categories: (i) hardware-based, (ii) model-based, and (iii) software-based approaches. Together with their own advantages, such solutions present various limitations that adversely affect their practical applicability. While hardware-based tools are able to delineate the exact energy profile of a mobile app, they require *ad-hoc* hardware components that are expensive and difficult to set up. Model-based approaches try to define mathematical functions able to estimate the energy consumption of mobile apps on a given hardware device. However, such tools require careful calibration of the parameters to correctly estimate power consumption. Finally, software-based approaches estimate the power profile of a mobile application solely relying on the system's functionalities of a device, as an example the CPU frequency. As a drawback, such techniques are *supposed* to be less precise than hardware-based approaches [11].

In this paper, we aim at investigating to what extent software-based tools are less precise than hardware-based tools for energy consumption. In other words, *is the higher cost of hardware-based solutions justified by a sensibly more accurate energy profiling? Can software-based solutions lead to close measurements without any cost overhead?* To answer these questions, we built a novel tool for extracting the energy profile of mobile applications, which we coined PETRA (**P**ower **E**stimation **T**ool fo**r** **A**ndroid), specific for Android OS. PETRA relies on the publicly available `Project Volta` Android tools[1] and has the following characteristics:

- **Efficiency and Granularity.** PETRA is able to quickly estimate the energy consumed by an app at the method level. It is worth noting that this level of granularity allows the developer to calculate the energy estimations per test case. The tool does not require any human effort.
- **Hawthorne Effect and Impact of Sampling Frequency.** The technologies on which PETRA relies are an integral part of the core Android OS and the instrumentation has

[1]https://developer.android.com/about/versions/android-5.0.html

little influence on the estimation process. In this way is possible to minimize the *Hawthorne Effect* in which the measurements are affected by the measurement process [10]. Moreover, since the tool does not rely on hardware components, it does not suffer of the sampling frequency highlighted by Saborido *et al.* [12].
- **Specialized Hardware Requirements.** PETRA does not need any particular hardware and provides simple output that can be easily analyzed.

For the evaluation, we run PETRA on 54 mobile applications belonging to the publicly available dataset provided by Linares-Vasquez *et al.* [6]. Thus, we compare the energy measurements provided by PETRA against the actual energy consumption computed using the MONSOON hardware toolkit [13] for the same apps and using the same hardware/software setting (i.e., smartphone model, operating systems, etc.). The collected results showed that the energy estimations produced by PETRA are very close to the measurements obtained with the MONSOON tool, with an error which is within 5% in 95% of the source code units under analysis. A direct implication of our study is that we recommend both the research community and industry to further investigate software-based tools for energy profiling of mobile applications as a viable alternative to hardware-based ones.

Structure of the Paper. Section II discusses the related literature in the context of power consumption measurement and empirical studies aimed at investigating the causes of energy leaks. Section III presents our software-based approach PETRA. In Section IV the design of the empirical study is described, while Section V reports the results achieved when comparing the performance of our tool with the ones achieved by a hardware-based approach. Section VI discusses the threats that could affect the validity of our study. Finally, Section VII concludes the paper.

II. BACKGROUND AND MOTIVATION

The attention toward energy efficiency issues have driven the research community in spending a lot of effort on the construction of new methods to extract the energy profiles of devices, as well as providing guidelines to help developers in writing green code. This section describes on the one hand the tools proposed in recent years to measure energy consumption, and on the other hand the empirical studies conducted in the context of software maintenance and evolution.

A. Measuring the Energy Profile of Hardware Devices

A first category of strategies to measure the energy consumption of devices is hardware-based since specific hardware toolkits are required to perform measurements. While such methodologies are quite popular in other research communities, such as high performance analysis [14] or large scale integration systems [15], they have been only partially explored in the context of software engineering.

Flinn and Satyanarayanan [16] proposed a tool named POWERSCOPE. It is based on the adoption of a digital multi-meter connected to a computer, which is used to monitor the energy variations – recorded by the multi-meter – of processes that are running on a laptop. Hindle *et al.* devised GREENMINER [2], a hardware mining testbed based on an Arduino board with an INA219 chip [17]. Besides the extraction of the energy consumption of mobile devices, GREENMINER also provides a web application[2] for (i) automating the testing of applications running on a device, and (ii) analyzing the results. Finally, other researchers exploited the MONSOON power monitor [13] to measure energy consumption of APIs of Android apps [6].

The costly hardware requirements needed for hardware-based energy profiling encouraged researchers to find alternative ways to approximate the energy consumption. A proxy measure can be computed by constructing models, which are based on the definition of specific functions to estimate the energy consumed by a device during its usage. Bourdon *et al.* [18] defined POWERAPI, an approach that leverages on analytical models characterizing the consumption of various hardware components (*e.g.,* CPU). Noureddine *et al.* [19] introduced JALEN, a Java agent which uses statistical sampling for the energy estimations. The model proposed by Pathak *et al.* [20] [21] is based on system calls, and it was implemented in EPROF, an energy counterpart of `gprof`, the `gnu` profiler tool, for profiling application energy drain. V-EDGE [22] considers the battery voltage dynamics for generating a power model. It neither needs external power meters nor relies on the battery current sensing capability. Along the same line, Balasubramanian *et al.* [23] defined an energy consumption model, named TAILENDER, to estimate to what extent modules such as 3G and GSM contribute to the battery drain by mobile apps. Ding *et al.* [24] proposed SEMO, a monitoring tool powered by an energy model based on the usage of the battery and its temperature. Zhang *et al.* [25] proposed a model-based solution with POWERBOOTER and POWERTUTOR. POWERBOOTER is a technique for automated power model construction that relies on battery voltage sensors and knowledge of battery discharge behavior. It does not require external power meters. POWERTUTOR uses the model provided by POWERBOOTER for generating online power estimation. Lastly, it is worth mentioning Microsoft's JOULEMETER tool[3], which uses energy models specific for each hardware configuration.

Finally, software-based approaches exclusively use the system functionalities to estimate the power consumption, without constructing any specific model. In this category, Do *et al.* [26] developed PTOP, an approach proposed that takes into account CPU frequency, hard disk and memory consumption as sources of information to estimate the joules consumed by a process. ELENS [11] provides a more fine-grained estimation of energy consumption at method, path or line-of-source level. It relies on a combination of program analysis and energy modeling and it produces visual feedback to help developers in better understanding the application behavior.

[2]http://softwareprocess.es/static/GreenMining.html
[3]http://tinyurl.com/jkvo9qa

Differently from the techniques/tools discussed above, PETRA does not require any additional hardware equipment and therefore any strong experience in the setup of the test bed. It uses reliable tools coming from the Android Toolkit and does not exploit energy models that need to be calibrated. Finally, unlike the tools measuring energy consumption at process level, PETRA works at method-level granularity. In addition to that, most of the approaches proposed in literature (including the software-based techniques) are not publicly available.[4]

B. Empirical Studies in Green Software Engineering

In recent years an ever increasing number of empirical studies aimed at understanding the reasons behind energy leaks in source code have been carried out. On the one hand, researchers have investigated the possibility to predict the energy consumption of mobile devices relying on empirical data, paving the way for new prediction models able to alert developers of the presence of energy bugs [27], [28]. On the other hand, Zhang *et al.* [29] and Gupta *et al.* [30] have proposed dynamic analysis based approaches for detecting portions of source code affected by energy leaks, while Li *et al.* [31] have put forward a technique for detecting specific lines of code affected by an energy bug.

Hindle [32] has investigated to what extent changes made by developers across software versions affect the energy consumption. He found that (i) software change can affect the power consumption and (ii) there seems to exist a relationship between software metrics and power consumption.

Other researchers focused their attention on the relationship between the development practices adopted by programmers and the energy consumption. Sahin *et al.* [33] showed that code obfuscation can negatively affect energy consumption, but the observed difference with non-obfuscated code unlikely impacts end-users. The same authors have also reported on an analysis of the role of design patterns [5]. In particular, they found that some patterns (*e.g.,* the *Decorator* pattern) negatively influence the energy efficiency. Similar results have been found by Noureddine and Rajan [34].

Hasan *et al.* [8] analyzed the impact of the data structures used by the developers, and specifically the influence of different Java Collections types. Results of their study showed that the use of the wrong type of data structure can increase the energy consumption by up to 300%. Other factors that have been studied that have a negative impact on energy efficiency are (i) the different sorting algorithms exploited [35], (ii) the use of lock-free data structures [36], (iii) the colors used in the GUI of software projects [37], (iv) the API usage of Android apps [6], and (v) the different refactorings applied to simplify the source code [7], [9].

Most of the studies mentioned above relied on hardware-based tools (*e.g.,* MONSOON). The final goal of the approach proposed in this paper is to provide researchers and practitioners an easier way to approximate the energy consumption of the methods of a mobile app without specialized hardware requirements: this can possibly help (i) the research community in conducting more studies aimed at understanding and solving energy-related issues and (ii) practitioners to take energy efficiency into account when developing mobile apps.

[4]PETrA will be released after acceptance of the paper.

III. PETRA: A POWER ESTIMATION TOOL FOR ANDROID APPLICATIONS

This section presents our novel software-based approach, coined PETRA (**P**ower **E**stimation **T**ool for**r** **A**ndroid), suitably developed to measure the power consumption of mobile apps at a method-level granularity. As depicted in Listing 1, its main process is composed of three main blocks: (i) app preprocessing, (ii) energy profile computation, and (iii) output generation. In the following we detail each part independently.

App Preprocessing. In the first step, PETRA needs to set the software environment before measuring the energy consumed when executing a mobile app. To this aim, it uses as input an executable version of the app under analysis in the form of an apk file. The app is identified by the apk location and the name of the app to profile, which correspond to `apkLocation`, and `appName` in Listing 1 respectively. Then, PETRA installs the apk on a mobile phone able to run it (*e.g.,* a smartphone having an arbitrary version of the Android operating system) and enables the `debuggable` option. Enabling debugging is mandatory, because otherwise the instrumentation of the app, needed to profile it, would not be possible.

Energy Profile Computation. Once the app is properly set, PETRA exercises the app under consideration using a test case given as input, i.e., `testCase` in Listing 1. This test case can be created with automated tools (*e.g.,* MONKEYRUNNER or MONKEY) or with manual operations performed by the software engineer. Once the test case is ran, the *core* process behind PETRA starts.

Listing 1: PETrA workflow

```
1  computeEnergyConsumption(apkLocation,
    appName, testCase, nRuns){
2    installApp(apk);
3    for (run=0; run<nRuns; run++) {
4      clearAppCache(appName);
5      resetBatteryStats();
6      startProfiler();
7      exerciseApp(appName, testCase);
8      stopProfiler();
9      collectBatteryStatsData();
10     collectSysTraceData();
11     collectDMTraceDumpData();
12     loadPowerProfile();
13     for each method call in trace file {
14       computeCallEnergyConsumption();
15     }
16     saveResults();
17     stopApp(appName);
18   }
19   uninstallApp(apk)
20 }
```

For the profiling phase, we leverage the `Project Volta` Android tools, such as `dmtracedump`[5], `Batterystats`[6], and `Systrace`[7]. Specifically:

- `dmtracedump` provides an alternate way to show trace log files. The files generated by `dmtracedump` are easy to parse and allow the developers to establish precisely, at microseconds granularity, when a method call has been invoked and when it returned. PETRA relies on this component in order to store the execution traces of the app under analysis. For each method call `dmtracedump` provides the entry and the exit time. The final output is a list of the executed method calls during the run.
- `BatteryStats` is an open source tool of the Android framework able to collect battery data from the device under evaluation. In particular, it is able to show which processes are consuming battery energy and which task should be modified in order to improve battery life. It is executable via the command line. The data collected can be analyzed as log file or can be converted to an HTML visualization that can be viewed in a browser using `Battery Historian`[6]. `PETrA` uses the `Batterystats` log in order to retrieve the active smartphone components and their status in a specific time window. Furthermore, it can provide the information about the device voltage. Given this information it is then possible to calculate the energy consumed by the smartphone during a time window.
- Systrace is a tool that can be used to analyze application performance. It captures and displays the execution times of the active processes of a smartphone, combining data from the Android kernel, *i.e.,* the CPU scheduler, disk activity, and application threads. The data can be viewed as an HTML report that shows the overview of the processes in a given time window. In PETRA, the information provided by `Systrace` is used to capture the frequency of the CPU in a given time window. Considering that CPUs have different consumptions as their frequency varies, this information completes the one provided by `Batterystats` improving the estimations.

After gathering the information related to the active components with their status, the CPU frequencies and the method call invocations, the `power profile` file is loaded. The `power profile` values define the current consumption for a component along with an approximation of the battery drain caused by each component over time. For instance, it specifies how many milliAmperes of current are required to run the CPU at a certain frequency. Every smartphone has its own power profile. It is worth noting that each device manufacturer must provide this information and that this info can be found in a defined location in the device[8].

[5]https://developer.android.com/studio/profile/traceview.html
[6] https://developer.android.com/studio/profile/battery-historian.html
[7]https://developer.android.com/studio/profile/systrace-commandline.html
[8]https://source.android.com/devices/tech/power/values.html

Given the previous data it is possible to compute the energy consumed for every method call invocation. First of all, given a method call invocation and its termination we can calculate the overall time window T_w as the arithmetic difference between the two time instants when these two events occurred. However, the energy consumed within one single time window is not constant but may change because of a CPU frequency variation or a component state change happened. Therefore, we divided the time windows in smaller time units, i.e., data frames T_Δ. When the entry to a method is registered, a new time window T_w and a new time frame T_Δ start. Whenever a component changes its state, the existing time frame T_Δ is terminated and a new one (for the new state) is started. When the exit point to a method is registered, then the corresponding time window T_w is terminated as well as the latest time frame T_Δ. In this way, each data frame T_Δ is characterized by coherent components' states (e.g., CPU frequency) and by a coherent (constant) energy drain. For example, if the CPU is working at the maximum frequency and none of the components change its state, the time windows T_w will be composed by only one time frame T_Δ of the same duration, i.e., be the difference between the method entry and exit. Therefore, we can calculate the current power intensity at each time frame T_Δ as follows:

$$I_\Delta = \sum_{\forall c \in C} I_{\Delta,c,s} \tag{1}$$

where C is the set of smartphone hardware components, $I_{\Delta,c,s}$ is the current intensity of the component c with the state s within the current time frame T_Δ. For example 92.6 is the number of milliAmpere consumed by a Nexus 4 when the CPU frequency is fixed to 384Mhz in a second.

After calculating the current intensity, it is possible to calculate the energy consumed in a time frame, as follow:

$$J_\Delta = I_\Delta \times V_\Delta \times T_\Delta \tag{2}$$

where J_Δ is the consumed energy in Joule, I_Δ is the current intensity in Ampere, V_Δ is the device voltage in Volt and T_Δ is the length of the time frame in seconds.

Finally, the energy consumed by a method call can be calculated by summing up the energy consumed in each time frame in which the method call was active:

$$J = \sum_{T_\Delta \in T_w} (I_\Delta \times V_\Delta \times T_\Delta) \tag{3}$$

Output Generation. The final output provided by PETRA is a csv file, containing the energy estimation for each method call. More precisely, it provides the signature of each executed method call, along with the consumption in Joule and the execution time in seconds.

By default PETRA uses `Monkey` to generate test cases, which are used to exercise the mobile app under analysis and to store the energy drain information for every exercised method call. It is important to highlight that the usage of `Monkey` is not mandatory, as other tools can be also used with PETRA. However, in order to have a fair comparison

with the estimations provided by Linares-Vasquez *et al.* [6] we used `Monkeyrunner` (i.e., we used the same test cases).

Finally, `PETrA` relies on the `Android Activity Manager`[9], so the apk must be enabled for debugging. Furthermore, in order to provide a better estimation, `PETrA` exercises the app multiple times (`nRuns` in Listing 1). Note that in our experiments `nRuns` is fixed to 10 and that in order to avoid any bias due to multiple runs, at the start of each run the app cache is cleaned and `Batterystats` is resetted (lines 4 and 5 in Listing 1).

IV. Evaluating the estimations provided by PETrA

The *goal* of the study is to analyze the accuracy of `PETrA` in providing energy consumption estimations of mobile apps at method-level granularity with the *purpose* of investigating whether the proposed approach can be used as a valid alternative to hardware-based solutions. More specifically, the study aims at addressing the following research question:

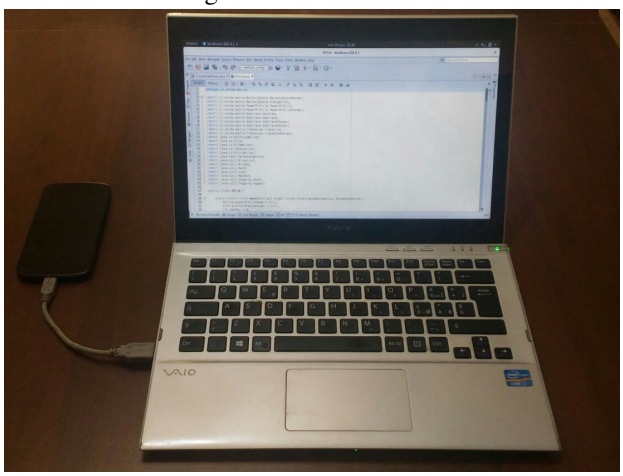
> **RQ$_1$:** *How close are the estimations from `PETrA` to a hardware-based tool?*

A. Context Selection and Oracle Extraction

The *context* of the study consisted of a set of 54 Android apps from the `Google Play Store` having different categories and scope. Table I reports for each app (i) an identifier we assigned to simplify referencing to the app, (ii) its name, (iii) its `Google Play Store` identifier, (iv) the specific version taken into account, and (v) the number of APIs used by the app. The choice of using these apps is not random, but rather guided by the need of having a set of applications for which an oracle reporting the consumption measured at the method level with hardware-based tools is publicly available. Indeed, since we had no hardware-based tool available to perform measurements we had to look for alternative solutions.

Some available datasets provide data about the energy consumption of software changes [38] or system calls [39]. However, these datasets are not suitable for our purpose because they (i) do not provide detailed measures for source code at the method level and (ii) contain data from desktop and web applications (*e.g.,* Firefox) rather than mobile apps. For this reason, we relied on the dataset provided by Linares-Vasquez *et al.* [6], that reports the actual power consumption of the methods belonging to the APIs used by the 54 mobile apps considered in the study. The authors computed the measurements relying on the Monsoon toolkit [13]. Note that the dataset also contains the test data needed to exercise the app in the same manner as done by Linares-Vasquez *et al.* [6] (more details on the measurement process come later in this section). A direct consequence of our choice to rely on the Linares-Vasquez *et al.* dataset is that we had to limit the focus of our analysis to methods belonging to APIs. However, we still took into account the energy consumption of 414.899 API calls belonging to the 321 APIs used by the considered apps. Moreover, according to recent findings achieved by Li and Gallagher [40], method invocations represent the more influencing energy consuming operation and, therefore, it is particularly interesting the analysis of the context of API calls.

Fig. 1: Test environment



B. Test Environment Setup and Energy Profiles Extraction

Figure 1 shows our test environment. As PETrA is a software-based approach, it requires a simple test environment composed only of a smartphone and a PC. While seemingly simple, we need to ensure a well-isolated test environment in order to avoid biases. To this aim, we carefully followed the guidelines from previous work in the field [2], [6], [11], [41]. The subsequent subsections detail each setup choice.

Choice of the Smartphone. Table II reports the characteristics of the phone used in the experiment. Specifically, we selected a factory resetted `LG Nexus 4` having Android 5.1.1 Lollipop as operating system, and equipped with a 1.5 GHz quad-core Snapdragon S4 Pro processor with 2 GB of RAM, and having a 2100 mAh, 3.8V battery. The choice is guided by the need to have the same smartphone used in the paper by Linares-Vasquez *et al.* [6] in order to conduct a fair evaluation. Moreover, it is worth noting that this particular hardware allows to be connected via a data cable, namely a cable where the USB charging can be disabled[10]. Thus, during the experiment no energy is transferred over the cable, allowing more stable measurements.

Isolating the execution of an app. To isolate the behavior of an application being executed on the smartphone, we adopted a number of precautions. In particular, we firstly disabled all the unnecessary apps and processes (*e.g.,* Google Services) running on the phone to avoid race conditions. Then, we avoided asynchronous events, such as incoming messages or calls by removing the sim card from the phone. Finally, we held the phone steady to avoid energy measurements by sensors and WiFi signal changes.

Extraction of the Energy Profiles of APIs. To extract the energy profiles of the apps in our dataset,

[9]https://developer.android.com/studio/command-line/shell.html

[10]http://android.stackexchange.com/questions/54902/disable-usb-charging

TABLE I: The mobile apps considered in our evaluation

#	Name	ID	Version	# of APIs
1	Battery HD	ch.smalltech.battery.free	1.16	3
2	Textgram	codeadore.textgram	2.3.15	3
3	Write Now Notepad	com.aerodroid.writenow	1.1.5	5
4	AndRecorder Free	com.andrconstruction.andrecord	3	2
5	Antivirus Free	com.antivirus	-	2
6	Botanica	com.app.botanica.layout	1	3
7	Sleep Sound Aid	com.arcdroid.sleep	20121007	2
8	Battery Drainer	com.batterydrainer	1.4.0	3
9	Better Browser	com.browser.sogood.ui	2.3	2
10	AudioPlayer	com.bytemystery.audioplayer	1.2	3
11	gReminders	com.diegoyarza.greminders	0.9.7	3
12	Dr.Web Antivirus Light	com.drweb	-	2
13	Sniper shooter	com.fungamesforfree.snipershooter.free	1.6.0	2
14	Despicable me (minion rush)	com.gameloft.android.ANMP.GloftDMHM	1.1.0	2
15	10,000 Quotes DB (FREE!)	com.hmobile.quotesmegacollection	3.0.4	3
16	Icey Slot	com.jbiz.iceyslotz	2.9	1
17	Android Music Player	com.jrtstudio.music	4.0.4b3	3
18	Android Antivirus	com.lab4apps.antivirus	2.0.1	1
19	Bubble blast 2	com.magmamobile.game.BubbleBlast2	1.0.34	2
20	Map quest	com.mapquest.android.ace	1.8.1	3
21	Oxford AZ of English Usage	com.mobisystems.msdict.embedded.wireless.oxford.azenglishusage	4.3.059	2
22	Livo Recorder Lite	com.mp1.livolite	3.7.0.a	1
23	Simple Weather	com.netthreads.android.weather	1.1.3	2
24	Opera Mini web browser	com.opera.mini.android	7.5.3	2
25	MasterCard ATM Hunter	com.orbiscom.ATMHunter	1.4	2
26	SimpleNews	com.prss.simplenews	1.4	1
27	Punjab Radio	com.prstudio.radio.punjabi	1.0.4	5
28	25000 Best Quotes	com.puissantapps.quotesapp.free	1.0.7	2
29	aTimer	com.r4ph4.timer	1.3	3
30	Star Wars Angry birds	com.rovio.angrybirdsstarwars.ads.iap	1.3.0	2
31	Classical Music Radio Lite	com.rslclasslite	1.0.3	3
32	Activity Express Task Manager	com.sayhello2theworld.te	1.22	1
33	news|swipe	com.segvic.news	1.0.0	2
34	Video Poker	com.sg.js.VidPoker	1.2.1	2
35	Galaxy Torch	com.swijaya.galaxytorch	1.4	3
36	TED	com.ted.android	2.0.1	3
37	Easy Birthday Reminders	com.tencentapps.bdays	1.2.1	2
38	World Travel Guide by Triposo	com.triposo.droidguide.world	2.1	2
39	8,500+ Drink Recipes	com.webworks.drinkscocktails	1.0.6	3
40	Droid Notepad	com.williamkingdom.droidnotepad	1.11	2
41	5001 Amazing Facts Free	com.ximad.wff	3.2.0	3
42	Inspiring Quotes	com.xstudio.inspiringquotes	1.2	3
43	Battery Info	com.zgame.batteryinfo	1.6	2
44	Anti Mosquito Sonic Repellent	com.zodinplex.antimosquito	-	2
45	AnEq Equalizer Free	de.ebbert.audioeq	1.0.9	4
46	Anime Radio Online	free.animeradioonline.gutisoft	1.06	3
47	Wifi Radar	girsas.wifiradar	1.06	2
48	Rome	hu.pocketguide.bundle.Rome	10-Jul-13	3
49	Battery Info Always	jp.dip.sys1.android.battery	1.2.0	2
50	Advanced Task Manager	mobi.infolife.taskmanager	2.1.2	3
51	Anti dog mosquito whistle	mz.anti.dog.cat.mosquito.insect.repellent.whistle	1.3	2
52	Meridian Media Player Revolute	org.iii.romulus.meridian	2.4.5	2
53	Better Notepad	org.strive.notes	0.0.5	2
54	Arcane legends	sts.al	1.0.7.0	1

we have to enable the debug. This entails manually adding `android:debuggable="true"` in the file `AndroidManifest.xml` and regenerating the `apk` file for each app (i.e. the executable), using ANDROID STUDIO [42].

Once we created these debuggable and executable versions of the apps, we applied PETRA to them. As explained in Section III, our approach receives as input a set of test cases for exercising the app under consideration and measuring the energy consumption at method-level granularity. In the context of this experiment, we exercised the apps in our dataset by using exactly the same `Monkeyrunner`[11] test cases used by Linares-Vasquez *et al.* [6]. This allows a fair comparison between the energy profiles extracted using our approach and the oracle provided using the MONSOON toolkit [13].

The output of this step consisted of a set of files reporting the execution traces of each app, accompanied by the information on the energy consumed by each method during that execution. It is important to note that in this stage we collected the information for all the methods belonging to an application. However, to compare the energy profiles extracted by PETRA with the ones extracted using MONSOON [13], we needed to select only the methods belonging to an API. To this aim, we selected from the final output produced by PETRA only the Android public methods, removing also the

[11]https://developer.android.com/studio/test/monkeyrunner/

calls to other Java APIs. This means that those methods were considered when estimating the energy consumption of the public methods, but not included in our study.

Moreover, to be more confident about the energy profiles built by PETRA, we repeated the measurements 10 times. Similarly to Linares-Vasques *et al.* [6], we aggregated the results of the 10 runs (*i.e.,* the joules consumed by the methods in each run) using the mean operator. Therefore, the final output consisted of a unique value representing the average energy consumed by the method belonging to an API exercised during the test execution.

C. Data Analysis and Metrics

Once extracted the energy profiles using `PETrA`, we answered $\mathbf{RQ}_1$ by comparing the energy profiles computed using `PETrA` with the oracle provided by Linares-Vasquez *et al.* in their empirical study [6]. To evaluate to what extent the energy consumption provided by our approach is close to the values measured with a hardware based tool, we employed a set of metrics widely used in the area of cost estimation [43] [44]. Specifically, we used the *Mean Magnitude Relative Error* (MMRE) [43] defined as follow:

$$MMRE = \frac{1}{N}\sum_{i=1}^{n} MRE_i \tag{4}$$

where n is the number of energy estimations computed by PETRA on each app (i.e., number of methods), and MRE indicates the *Magnitude Relative Error* [43] and has values in the range defined by the following formula:

$$MRE_i = \frac{|J_i - \hat{J}_i|}{J_i} \tag{5}$$

TABLE II: Characteristics of the smartphone used in our study

Component	Specification
Name	LG Nexus 4
Screen	4.7" diagonal
	1280x768 pixel resolution (320 ppi)
	WXGA IPS
	Corning Gorilla Glass 2
Size	133.9 x 68.7 x 9.1mm
Weight	139g
Cameras	8 MP (main)
	1.3 MP (front)
Memory	16GB
	2GB RAM
CPU	Qualcomm Snapdragon S4 Pro 1.5GHz
Sensors	Microphone
	Accelerometer
	Compass
	Ambient light
	Barometer
	Gyroscope
	GPS
Network	Unlocked GSM/UMTS/HSPA+
	GSM/EDGE/GPRS (850, 900, 1800, 1900 MHz)
	3G (850, 900, 1700, 1900, 2100 MHz)
	HSPA+ 42
Wireless	Wi-Fi (802.11 a/b/g/n)
	NFC (Android Beam)
	Bluetooth
Battery	2,100 mAh non-removable battery
OS	Android 5.1.1 (Lollipop)

where J_i and $\hat{J}_i$ are the the energy estimations produced by MONSOON and `PETrA` respectively for the method i.

Besides determining the mean error in the estimations provided by our approach, we also computed the $PRED(x)$ metric, namely the *Relative Error Deviation Within x%* [45]. This measure gives an indication of how many estimation errors of our approach are within x% of the estimation values provided by MONSOON. In particular, $PRED(x)$ is defined as the average fraction of the $MREs$ off by no more than x as defined by Jorgensen [46].

$$PRED(x) = \frac{1}{n}\sum_{i=1}^{n}\begin{cases} 1 \; if \; MRE_i \leq x \\ 0 \; otherwise \end{cases} \tag{6}$$

In the field of cost estimation, the parameter x is usually set to 25, *i.e.,* the estimated cost is within 25% of the actual cost of a project [44]. However, in our context an estimation error of 25% could be very large. For instance, a variation of 25% is very large when estimating the energy consumed by a data structure used in the source code [8]. An analysis done in this way would be too coarse-grained. Thus, we verified whether `PETrA` can achieve a lower estimation error, by setting x=$2; 5; 7; 10; 20$. In this way, we were able to control how the estimation errors of our approach are distributed.

Obviously, the estimation errors in $PRED(5)$ are also included in $PRED(10)$, the estimation errors in $PRED(10)$ in $PRED(20)$ and so on. This means that the distribution over the different $PRED(x)$ measures is cumulative. For instance, if $PRED(2)$ is equal to 0.91 and $PRED(5)$ is 0.96, then 5% of the estimation errors are between 2% and 5%.

Finally, we performed a *fine-grained* analysis aimed at understanding the types of errors achieved by PETRA during the energy profile estimations. To this aim, we (i) measured the ratio of over/under estimations provided by our approach, and (ii) provided practical explanations of the motivations behind the estimation errors.

V. ANALYSIS OF THE RESULTS

For each app considered in the empirical study, Table III shows the $MMRE$ and the distribution of the $PRED(x)$ achieved when comparing the estimations provided by PETRA with those reported by Linares Vasquez *et al.* [6]. Table III also reports the percentage of over/under estimations given by PETRA. Finally, the row *Overall* shows the results obtained when considering all the estimations as a unique dataset.

A first point to discuss is the mean estimation error provided by our tool (column $MMRE$). From Table III we observe that for all 54 mobile apps the *mean relative error* ($MMRE$) is always lower than 0.05, being 0.01 on overall. To have a practical idea of the magnitude of the error, an $MMRE = 0.01$ corresponds to a percentage of battery discharge of $3.1 \times 10^{-6}\%$, and thus we can claim that PETRA misses the correct energy consumption by a small factor. For instance, let us consider the case of the SIMPLENEWS app (row #26 in Table III): during its execution, the app makes 15 calls to the API method `writeCommStatusAndClose` of the class `android.os.ParcelFileDescriptor`. This

TABLE III: MMRE, PRED(x), over estimations, and under estimations computed for the apps under evaluation

#	MMRE	PRED(2)	PRED(5)	PRED(7)	PRED(10)	PRED(20)	Over Estimations	Under Estimations
1	0.04	0.71	1.00	1.00	1.00	1.00	1.00	0.00
2	0.01	0.92	0.94	0.94	0.95	0.95	0.91	0.09
3	0.01	0.89	0.99	0.99	0.99	0.99	0.99	0.01
4	<0.01	0.82	0.92	1.54	0.92	0.92	0.92	0.08
5	<0.01	0.91	1.00	1.00	1.00	1.00	1.00	0.00
6	0.01	0.78	1.00	1.00	1.00	1.00	0.89	0.11
7	0.02	0.75	0.99	0.99	1.00	1.00	0.89	0.11
8	0.01	0.83	1.00	1.00	1.00	1.00	0.84	0.16
9	0.01	0.89	1.00	1.00	1.00	1.00	1.00	0.00
10	<0.01	0.69	0.89	0.90	0.91	0.94	0.87	0.13
11	0.01	0.93	1.00	1.00	1.00	1.00	0.94	0.06
12	<0.01	0.71	0.98	0.98	0.99	0.99	0.94	0.06
13	0.02	0.91	0.95	0.95	0.96	0.97	0.85	0.15
14	<0.01	0.84	0.99	0.99	1.00	1.00	0.88	0.12
15	<0.01	0.89	0.99	0.99	0.99	1.00	0.99	0.01
16	0.02	0.91	0.99	0.99	1.00	1.00	1.00	0.00
17	<0.01	0.87	0.99	0.99	0.99	0.99	0.99	0.01
18	<0.01	1.00	1.00	1.00	1.00	1.00	1.00	0.00
19	0.02	0.96	0.97	0.97	1.00	1.00	0.95	0.05
20	0.02	0.87	1.00	1.00	1.00	1.00	0.82	0.18
21	<0.01	0.78	0.81	0.87	1.00	1.00	0.92	0.08
22	<0.01	1.00	1.00	1.00	1.00	1.00	1.00	0.00
23	<0.01	0.88	0.98	0.98	0.98	0.99	0.98	0.02
24	0.01	0.77	0.99	0.99	0.99	0.99	0.93	0.07
25	0.01	0.91	0.96	0.97	0.97	0.98	0.96	0.04
26	<0.01	0.30	0.32	0.32	0.32	0.38	0.28	0.72
27	0.02	0.95	0.99	1.00	1.00	1.00	0.76	0.24
28	<0.01	1.00	1.00	1.00	1.00	1.00	1.00	0.00
29	0.01	0.96	0.98	0.98	0.98	0.99	0.79	0.21
30	0.02	0.96	0.97	0.97	1.00	1.00	0.89	0.11
31	0.01	0.96	1.00	1.00	1.00	1.00	1.00	0.00
32	0.01	0.99	1.00	1.00	1.00	1.00	0.91	0.09
33	<0.01	0.10	0.11	0.12	0.13	0.17	0.11	0.89
34	0.01	0.88	0.98	0.99	1.00	1.00	0.78	0.22
35	0.01	0.91	0.94	0.95	0.96	0.97	0.82	0.18
36	0.01	0.63	0.94	0.94	1.00	1.00	0.91	0.09
37	0.02	0.93	1.00	1.00	1.00	1.00	0.97	0.03
38	<0.01	0.95	1.00	1.00	1.00	1.00	0.81	0.19
39	0.01	0.92	1.00	1.00	1.00	1.00	0.78	0.22
40	<0.01	0.92	0.99	0.99	0.99	0.99	0.99	0.01
41	0.02	0.94	0.99	0.99	0.99	0.99	0.85	0.15
42	<0.01	0.83	0.92	0.92	0.92	0.92	0.92	0.08
43	0.01	0.99	1.00	1.00	1.00	1.00	1.00	0.00
44	0.01	0.95	1.00	1.00	1.00	1.00	0.91	0.09
45	0.02	0.97	0.99	0.99	0.99	0.99	0.94	0.06
46	0.01	0.97	0.98	0.98	0.98	0.99	0.98	0.02
47	0.03	0.09	0.95	0.95	0.95	0.95	0.95	0.05
48	0.02	0.68	1.00	1.00	1.00	1.00	0.82	0.18
49	<0.01	0.91	0.97	0.97	0.97	0.97	0.96	0.04
50	0.01	0.91	1.00	1.00	1.00	1.00	0.82	0.18
51	0.02	0.77	1.00	1.00	1.00	1.00	0.89	0.11
52	0.01	0.92	1.00	1.00	1.00	1.00	1.00	0.00
53	0.02	0.95	1.00	1.00	1.00	1.00	1.00	0.00
54	<0.01	0.97	1.00	1.00	1.00	1.00	1.00	0.00
Overall	**0.01**	**0.85**	**0.95**	**0.96**	**0.96**	**0.96**	**0.89**	**0.11**

is done to update the status of the events requested when the application is running. The average energy consumption of the method invocation is 1.602×10^{-5} Joules when computed by the MONSOON toolkit, while PETRA estimates a value of 1.605×10^{-5} Joules. Thus, our tool overestimates the actual consumption by 3.0×10^{-8} Joules ($MRE = 0.18\%$).

If we consider the other mobile apps in the study, we observe that the results are mostly consistent. Specifically, for 39 out of the 54 apps that we considered, the mean estimation error (MMRE) is equal to or less than 1%. For the remaining 15 apps, we experienced an average error of 2.20%. One of the worst estimations regards the app BATTERY HD (row #1 in Table III). This app monitors the battery discharge of mobile phones and tables, by proposing the user a set of statistics about the energy consumption of the apps installed on the device. The app also provides real-time monitoring of the apps that are active on the device, which is implemented through the invocation of the API method `ArrayAdapter.notifyDataSetChanged`. This method simply implements an `Observer` design pattern [47] that monitors the changes in the status of the applications currently opened on the device. As reported by Linares-Vasquez *et al.* [6], this is a well-known energy bottleneck,

object of several discussions among the Android developers[12]. Indeed, on average the method consumes 2.43×10^{-4} Joules, *i.e.,* 35% more energy compared to the average consumption of the other methods. This confirms previous findings on the high energy consumption of such a design pattern [5]. In this particular case, PETRA estimates that 2.48×10^{-4} Joules of energy are consumed by the API, thus leading to an error of 5.00×10^{-6} Joules ($MRE = 2.06\%$). We further investigated the behavior of our tool in this case in order to understand the reason behind this error. Specifically, we found that every time a change status notification is sent (*i.e.,* each call to the method `notifyDataSetChanged`), PETRA accumulates small errors, which lead to a total of 5.00×10^{-6} Joules because of the 24 times the method has been called. Putting things into perspective, we see that although the worst case estimation error is still small, the particular example that we highlighted does indicate a potential drawback of using software-based approaches for energy profiling. Indeed, a high frequency of method calls in a method under observation can result in an accumulation of estimation errors.

Observation 1. In 72% of mobile apps the *mean estimation error* provided by PETRA is at most 1%. Although in the other cases the difference is slightly bigger, it still only reaches at most 4%. Thus, the proposed software-based solution provides energy estimations that are quite close to the actual values.

Observation 1 only provides a partial view on the performance of our tool as we also need a better understanding of the relative difference between the estimation of PETRA and the hardware based solution. This is why we use the $PRED(x)$ metric, which indicates the percentage of methods in each app with an error (MRE) lower than $x\%$. Table III shows that in 95% of the methods our tool is able to provide an estimation error within 5% of the actual values measured using the MONSOON toolkit. Moreover in 85% of the methods our tool provides estimation within 2% of error. These data confirm what we found when analyzing the $MMRE$ metric: a software-based solution built using public Android APIs performs similarly to a hardware-based solution that computes the energy consumption of Android apps. The result is particularly evident on 23 apps of our dataset (43%), where we observed that all the estimations of PETRA falls back into 5% of the actual energy consumption (*i.e.,* $PRED(5) = 1.00$). For example, the app BETTER NOTEPAD (row 53 in Table III) implements a notepad which allows the import/export of the notes and their sharing on the main social networks, besides common features such as writing/editing/deleting of notes. The app relies on a single external API, named `android.os.Message`, and in particular it called 5 times the method `sendToTarget` during its executions. The method is in charge to exchange messages with the `Handler` of the app, thus providing a way to monitor the correct execution of the application. In this case, PETRA perfectly estimated the actual energy consumption of the method twice (*i.e.,* 1.27×10^{-5} Joules), while on the remaining three calls the errors (MRE values) are 0.20% and 0.10%, respectively.

While the results for $PRED(5)$ already revealed the effectiveness of PETRA, it is also important to mention that only 2% of methods have estimation errors outside the 50% of the actual energy consumption. To understand the characteristics of the outliers, let us discuss the cases of SIMPLENEWS (row 26 in Table III) and NEWS—SWIPE (row 33 in Table III) apps, where we observed an unusual behavior of our tool. Indeed, in the former app only 32% of the estimations are within 5% of the actual values, while a still lower percentage of them (11%) is within the 5% of the oracle values in the NEWS—SWIPE app. Both the apps are newspaper readers that function as integrators of news items published on online websites. The need of retrieving news over the network implies a non-deterministic waiting time due to delay that querying the external websites induces. As a consequence, such communication overhead makes it difficult to compare different approaches as the experiments where done under different network conditions.

While PETRA generally works well and provides estimations very close to the actual values, there are few methods where such errors are higher than the ones discussed above. In particular, for 5% of methods the corresponding estimations are not within 5% of the oracle values. Further analyzing the factors behind such deviations, we observed that these are due to the usage of sensors (*i.e.,* motion, environmental, and position sensors). Unfortunately, the measurement of the power consumption of sensors is still an open issue in software-based energy measurement [48]. In the case of PETRA these components are not analyzed by Android tools and as such, we inherit such weakness. We plan to analyze this aspect as a future work, in order to improve the performances of our tool.

Observation 2. In 95% of the methods PETRA provides an *estimation error* within 5% of the actual values measured with the MONSOON toolkit, confirming the good performances of our tool. Errors in measurement mainly occur in cases where there is significant use of network capabilities or when the sensors are used.

Finally, regarding the types of estimations provided by PETRA, we observed that our tool rarely underestimates the energy consumption (overall, in 11% of the cases), while 89% of the estimations are slightly higher than the actual values. This is mainly due to the fact that in each energy estimation some noise is summed up during the measurement (see, for instance, the case of the app SIMPLENEWS previously discussed). Thus, the estimation results are typically higher than the actual values. However, as previously shown, our tool is able to achieve a good compromise between the errors committed and the accuracy of the evaluations.

On the other hand, the few underestimations are gener-

[12]http://stackoverflow.com/questions/15990849/

ally due to the usage of sensors. For instance, the class `SensorManager` used by the app ANDROID ANTIVIRUS (row #18 in Table III) is responsible for the management of the environmental sensors needed to check the status of external factors possibly threatening the security of the device. In this case, PETRA is not able to take into account the energy consumption of such sensors, therefore providing a lower estimation (*i.e.,* 1.35×10^{-5} Joules) with respect to the actual one (*i.e.,* 1.37×10^{-5} Joules).

Observation 3. In 89% of the methods PETRA overestimates the energy consumed. This is mainly due to the noise accumulated over the different APIs estimations. In the remaining 11% of the methods, our tool underestimates the actual energy consumption because of the presence of sensors.

Once evaluated the three different aspects described above (*i.e.,* $MMRE$, $PRED(x)$, and over/under estimations), we can provide an answer to our **RQ**. First of all, we can conclude that PETRA *provides estimations close to the actual ones*: indeed, the *mean relative error* ($MMRE$) produced by our tool is always lower than 0.05 (0.01 on overall). At the same time, *95% of the estimation errors are within 5% of the actual values computed using a hardware-based tool*: this confirms that a software-based approach can actually perform similarly to the alternative hardware solution. Finally, our analyses highlighted some potential drawbacks of our tool. Indeed, the significant use of network capabilities as well as the usage of sensors can lead to higher estimation errors.

VI. THREATS TO VALIDITY

This section discusses the threats to the validity of our empirical evaluation, classifying them into construct, internal, external, and conclusion validity.

The main threats related to the relationships between theory and observation (*construct validity*) are due to imprecisions in the measurements we performed. PETRA relies on different sources of information, so imprecisions in those sources could affect the quality of the estimations. For example, the values contained in the `power profile` file, provided by the device manufacturer, define just an approximation of the battery drain caused by a component in a second. Moreover PETRA does not consider the consumption due to the usage of sensors and GPU. For this reason PETRA could not be the best solution for energy estimations of apps that strongly stress this kind of hardware.

In order to limit the factors that can affect our results (*internal validity*) we disabled the mobile connections (not WiFi), did a factory reset of the device before starting the experiments, and avoided any not needed processes on the device. Lastly, we repeated the measurements 10 times and we aggregated the results using the mean operator, as was previously done by Linares-Vasquez *et al.* [6].

Regarding the generalization of our findings (*external validity*) we considered 54 apps of different categories from the Linares-Vasquez *et al.* dataset [6]. Relying on this dataset, we had to limit our analysis to methods belonging to APIs that could not represent the full variety of methods. For this reasons, further studies aiming at replicating our work on larger datasets are desirable and part of our future agenda. Another threat could be related to the smartphone that we used in our study. It is worth to note that we used a LG Nexus 4 in order to compare our estimations with those provided by Linares-Vasquez *et al.* [6] in their study.

Finally the threats related to the relationship between the treatment and the outcome (*conclusion validity*) are represented by the analysis methods exploited in our study. We discuss our results applying widely used metrics (*i.e.,* $MMRE$ and $PRED$) able to compare our numerical outcomes with numerical oracles.

VII. CONCLUSION AND FUTURE WORK

This paper investigates the wide-spread assumption that energy consumption profiles calculated using software-based tools are less precise than the actual values as measured using hardware-based tools. To this end, we propose PETRA, a new Android-specific tool for estimating the energy consumption of mobile applications. Our tool relies on public available tools, developed by Google in the context of `Project Volta`.

We evaluated PETRA on 54 mobile applications from the dataset provided by Linares-Vasquez *et al.* [6]. For each application, this dataset contains the energy consumption of methods belonging to APIs using the MONSOON hardware toolkit [13]. Our results showed that the estimations produced by PETRA are very close to the actual values, more precisely:

- *Error Magnitude.* The average estimation error achieved using PETRA is 4% with respect to actual value calculated using MONSOON.
- *Error Reasons.* The measurement errors are mainly due to a significant use of network capabilities or sensors.
- *Error Type.* 89% of the estimations are overestimations, mainly due to the accumulated noise achieved during the estimations. In the remaining cases, the use of sensors and network produces underestimations.

These observations represent the main input for our research agenda. We will focus on designing and developing new techniques that better estimate the energy consumption of components such as sensors and network. As such, i) we will investigate Android tools able to give more precise information regarding the state of components (*i.e., GPU Monitor*[13] and *Network Monitor*[14]), ii) we will design new services able to capture those components not analyzed by `Android tools`, e.g. sensors. We should carefully design these services in order to avoid the *Hawthorne Effect* [10], that currently does not impact our tool. Moreover, in order to asses these new techniques and to validate the currently achieved results, we plan to replicate our study on a larger set of applications, considering also other methods than API methods.

[13]https://developer.android.com/studio/profile/am-gpu.html
[14]https://developer.android.com/studio/profile/am-network.html

REFERENCES

[1] The statistics portal association. [Online]. Available: http://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[2] A. Hindle, A. Wilson, K. Rasmussen, E. J. Barlow, J. C. Campbell, and S. Romansky, "Greenminer: A hardware based mining software repositories software energy consumption framework," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 12–21. [Online]. Available: http://doi.acm.org/10.1145/2597073.2597097

[3] C. Wilke, S. Richly, S. Götz, C. Piechnick, and U. Aßmann, "Energy consumption and efficiency in mobile applications: A user feedback study," in *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*. IEEE, 2013, pp. 134–141.

[4] J. Flinn and M. Satyanarayanan, "Energy-aware adaptation for mobile applications," in *Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles*, ser. SOSP '99. New York, NY, USA: ACM, 1999, pp. 48–63. [Online]. Available: http://doi.acm.org/10.1145/319151.319155

[5] C. Sahin, F. Cayci, I. L. M. Gutiérrez, J. Clause, F. Kiamilev, L. Pollock, and K. Winbladh, "Initial explorations on design pattern energy usage," in *Proceedings of the First International Workshop on Green and Sustainable Software*, ser. GREENS '12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 55–61. [Online]. Available: http://dl.acm.org/citation.cfm?id=2663779.2663789

[6] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk, "Mining energy-greedy api usage patterns in android apps: An empirical study," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 2–11. [Online]. Available: http://doi.acm.org/10.1145/2597073.2597085

[7] C. Sahin, L. Pollock, and J. Clause, "How do code refactorings affect energy usage?" in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '14. New York, NY, USA: ACM, 2014, pp. 36:1–36:10. [Online]. Available: http://doi.acm.org/10.1145/2652524.2652538

[8] S. Hasan, Z. King, M. Hafiz, M. Sayagh, B. Adams, and A. Hindle, "Energy profiles of java collections classes," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16. New York, NY, USA: ACM, 2016, pp. 225–236. [Online]. Available: http://doi.acm.org/10.1145/2884781.2884869

[9] J.-J. Park, J.-E. Hong, and S.-H. Lee, "Investigation for software power consumption of code refactoring techniques," in *Proceedings of the Twenty-Sixth International Conference on Software Engineering and Knowledge Engineering*, ser. SEKE '14, 2014.

[10] M. Harman, Y. Jia, and Y. Zhang, "Achievements, open problems and challenges for search based software testing," in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2015, pp. 1–12.

[11] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan, "Estimating mobile application energy consumption using program analysis," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 92–101. [Online]. Available: http://dl.acm.org/citation.cfm?id=2486788.2486801

[12] R. Saborido, V. V. Arnaoudova, G. Beltrame, F. Khomh, and G. Antoniol, "On the impact of sampling frequency on software energy measurements," PeerJ PrePrints, Tech. Rep., 2015.

[13] Moonsoon-solutions. power monitor. [Online]. Available: http://www.msoon.com/LabEquipment/PowerMonitor/

[14] I. Hur and C. Lin, "A comprehensive approach to dram power management," in *2008 IEEE 14th International Symposium on High Performance Computer Architecture*, Feb 2008, pp. 305–316.

[15] P. Choudhary and D. Marculescu, "Power management of voltage/frequency island-based systems using hardware-based methods," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 3, pp. 427–438, March 2009.

[16] J. Flinn and M. Satyanarayanan, "Powerscope: A tool for profiling the energy usage of mobile applications," in *WMCSA'99*, 1999, pp. 1–9.

[17] Arduino. [Online]. Available: https://www.arduino.cc

[18] A. Bourdon, A. Noureddine, R. Rouvoy, and L. Seinturier, "Powerapi: A software library to monitor the energy consumed at the process-level," in *PoweERCIM News*, 2013.

[19] A. Noureddine, A. Bourdon, R. Rouvoy, and L. Seinturier, "Runtime monitoring of software energy hotspots," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2012. New York, NY, USA: ACM, 2012, pp. 160–169. [Online]. Available: http://doi.acm.org/10.1145/2351676.2351699

[20] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang, "Fine-grained power modeling for smartphones using system call tracing," in *Proceedings of the sixth conference on Computer systems*. ACM, 2011, pp. 153–168.

[21] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof," in *Proceedings of the 7th ACM european conference on Computer Systems*. ACM, 2012, pp. 29–42.

[22] F. Xu, Y. Liu, Q. Li, and Y. Zhang, "V-edge: Fast self-constructive power modeling of smartphones based on battery voltage dynamics," in *NSDI'13*, 2013, pp. 43–56.

[23] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: A measurement study and implications for network applications," in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '09. New York, NY, USA: ACM, 2009, pp. 280–293. [Online]. Available: http://doi.acm.org/10.1145/1644893.1644927

[24] F. Ding, F. Xia, W. Zhang, X. Zhao, and C. Ma, "Monitoring energy consumption of smartphones," in *Internet of Things (iThings/CPSCom), 2011 International Conference on and 4th International Conference on Cyber, Physical and Social Computing*. IEEE, 2011, pp. 610–613.

[25] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, 2010, pp. 105–114.

[26] T. Do, S. Rawshdeh, and W. Shi, "ptop: A process-level power profiling tool," in *in Proceedings of the 2nd Workshop on Power Aware Computing and Systems (HotPower09*, 2009.

[27] N. Amsel and B. Tomlinson, "Green tracker: A tool for estimating the energy consumption of software," in *CHI '10 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '10. New York, NY, USA: ACM, 2010, pp. 3337–3342. [Online]. Available: http://doi.acm.org/10.1145/1753846.1753981

[28] K. Aggarwal, C. Zhang, J. C. Campbell, A. Hindle, and E. Stroulia, "The power of system call traces: Predicting the software energy consumption impact of changes," in *Proceedings of 24th Annual International Conference on Computer Science and Software Engineering*, ser. CASCON '14. Riverton, NJ, USA: IBM Corp., 2014, pp. 219–233. [Online]. Available: http://dl.acm.org/citation.cfm?id=2735522.2735546

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

[30] A. Gupta, T. Zimmermann, C. Bird, N. Nagappan, T. Bhat, and S. Emran, "Mining energy traces to aid in software development: An empirical case study," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '14. New York, NY, USA: ACM, 2014, pp. 40:1–40:8. [Online]. Available: http://doi.acm.org/10.1145/2652524.2652578

[31] D. Li, S. Hao, W. G. J. Halfond, and R. Govindan, "Calculating source line level energy information for android applications," in *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, ser. ISSTA 2013. New York, NY, USA: ACM, 2013, pp. 78–89. [Online]. Available: http://doi.acm.org/10.1145/2483760.2483780

[32] A. Hindle, "Green mining: A methodology of relating software change and configuration to power consumption," *Empirical Softw. Engg.*, vol. 20, no. 2, pp. 374–409, Apr. 2015. [Online]. Available: http://dx.doi.org/10.1007/s10664-013-9276-6

[33] C. Sahin, P. Tornquist, R. Mckenna, Z. Pearson, and J. Clause, "How does code obfuscation impact energy usage?" in *Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution*, ser. ICSME '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 131–140. [Online]. Available: http://dx.doi.org/10.1109/ICSME.2014.35

[34] A. Noureddine and A. Rajan, "Optimising energy consumption of design patterns," in *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, ser. ICSE '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 623–626. [Online]. Available: http://dl.acm.org/citation.cfm?id=2819009.2819120

[35] C. Bunse, H. Hpfner, E. Mansour, and S. Roychoudhury, "Exploring the energy consumption of data sorting algorithms in embedded and mobile environments," in *2009 Tenth International Conference on Mobile Data Management: Systems, Services and Middleware*, May 2009, pp. 600–607.

[36] N. Hunt, P. S. Sandhu, and L. Ceze, "Characterizing the performance and energy efficiency of lock-free data structures," in *2011 15th Workshop on Interaction between Compilers and Computer Architectures*, Feb 2011, pp. 63–70.

[37] M. Linares-Vásquez, G. Bavota, C. E. B. Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk, "Optimizing energy consumption of guis in android apps: A multi-objective approach," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: ACM, 2015, pp. 143–154. [Online]. Available: http://doi.acm.org/10.1145/2786805.2786847

[38] C. Zhang and A. Hindle, "A green miner's dataset: Mining the impact of software change on energy consumption," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 400–403. [Online]. Available: http://doi.acm.org/10.1145/2597073.2597130

[39] C. Zhang, J. Campbell, and A. Hindle, "Green trace: The impact of software change on system calls and energy consumption," in *in submission to Mining Software Repositories (MSR), 2014 11th IEEE Working Conference on*. IEEE, 2014.

[40] X. Li and J. P. Gallagher, "Fine-grained energy modeling for the source code of a mobile application," PeerJ PrePrints, Tech. Rep., 2016.

[41] D. Li, S. Hao, J. Gui, and W. G. J. Halfond, "An empirical study of the energy consumption of android applications," in *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, Sept 2014, pp. 121–130.

[42] Android studio. [Online]. Available: https://developer.android.com/studio/index.html

[43] L. C. Briand and I. Wieczorek, "Resource estimation in software engineering," *Encyclopedia of software engineering*, 2002.

[44] S. D. Conte, H. E. Dunsmore, and V. Y. Shen, *Software engineering metrics and models*. Benjamin-Cummings Publishing Co., Inc., 1986.

[45] D. P. U. V. Nguyen and T. M. WVU, "Studies of confidence in software cost estimation research based on the criterions mmre and pred," 2009.

[46] M. Jorgensen, "Experience with the accuracy of software maintenance task effort prediction models," *IEEE Transactions on software engineering*, vol. 21, no. 8, pp. 674–681, 1995.

[47] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[48] P. Hurni, B. Nyffenegger, T. Braun, and A. Hergenroeder, "On the accuracy of software-based energy estimation techniques," in *Proceedings of the 8th European Conference on Wireless Sensor Networks*, ser. EWSN'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 49–64. [Online]. Available: http://dl.acm.org/citation.cfm?id=1966251.1966257