# A Software-based Approach to Extract Energy Profiles of Mobile Apps

Dario Di Nucci*, Fabio Palomba*, Annibale Panichella†, Andy Zaidman†, Andrea De Lucia*
*University of Salerno, Fisciano (SA), Italy — †Delft University of Technology, The Netherlands
ddinucci@unisa.it, fpalomba@unisa.it, a.panichella@tudelft.nl, a.e.zaidman@tudelft.nl, adelucia@unisa.it

## I. Background and Motivations

The attention toward energy efficiency issues of both smartphones and data centers have driven the research community in spending a lot of effort in the construction of new methods able to extract the energy profiles of devices, as well as providing guidelines to help developers in writing green source 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

Broadly speaking, the methodologies proposed to measure the energy consumption of devices can be categorized in (i) hardware-based, (ii) power models-based, and (iii) software-based.

The first class refers to approaches that require a specific hardware toolkit to perform the measurements. While such methodologies are quite popular in other research communities, such as high performance analysis [1] or large scale integration systems [2], in the context of software enginering these solutions have been only partially explored.

Flinn and Satyanarayanan [3] proposed a tool named PowerScope. It is based on the adoption of a digital multimeter connected to a computer, which is used to monitor the energy variations – recorded by the multimeter – of processes that are running on a laptop. Hindle *et al.* devised GreenMiner [4], an hardware mining testbed based on a Arduino board with an INA219 chip [5]. Besides the extraction of the energy consumption of mobile devices, GreenMiner also provides a web application[1] for (i) automating the testing of applications running on a device, and (ii) analyzing the results. Finally, other researchers exploited the Monsoon power monitor [6] to measure energy consumption of APIs of Android apps [7].

The need of meeting specialized hardware requirements encouraged researchers in finding alternative ways to approximate the energy consumption. A proxy measure can be computed by constructing power models, which are based on the definition of specific functions able to estimate the energy consumed by a device during its activities. Bourdon *et al.* [8] defined PowerAPI, an approach where the energy estimation is based on analytical models that characterize the consumption of various hardware components (*e.g.,* CPU). Noureddine *et al.* [9] introduced Jalen, a Java agent which uses statistical sampling for the energy estimations. The model proposed by Pathak *et al.* [10] [11] 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 [12] considers the battery voltage dynamics for generating a power model. It neither needs of external power meters nor relies on the battery current sensing capability. On the same line, Balasubramanian *et al.* [13] defined an energy consumption model, named TailEnder, to estimate to what extent moduls such as 3G and GSM contribute to the battery drain of mobile apps. Ding *et al.* [14] proposed SEMO, a monitoring tool powered by an energy model based on the usage of the battery and its temperature. Zhang *et al.* [15] 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 use the model provided by PowerBooter for generating online power estimation. Lastly, it is worth mentioning the Microsoft Joulemeter tool[2], which uses energy models specific for each hardware configuration.

Finally, software-based approaches exclusively use system's functionalities to estimate the power consumption, without constructing any specific model. In this category falls ptop, the approach proposed by Do *et al.* [16]. ptop takes into account CPU frequency, hard disk and memory consumption as sources of information to estimate the joules consumed by a process. A fine-grained estimation of energy consumption at code level is instead provided by eLens [17], which relies on a combination of program analysis and energy modeling. Furthermore it produces a visual feedback to the developer that can understand the application behavior with respect to energy consumption.

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

Looking at the differences between the approaches discussed above and the technique proposed in this paper, it is important to note that PETRA (i) does not require any additional hardware equipment and therefore any strong experience in the set up of the test bed, (ii) provides an estimation of the energy consumption at method-level rather than at process-level, and (iii) is based on reliable tools coming from the Android Toolkit and does not exploit energy models that need to be calibrated. Moreover, most of the approaches proposed in literature (including the software-based techniques) are not publicly available.

### *B. Green Mining: An Overview of the Empirical Studies*

In recent years an ever increasing number of empirical studies aimed at understanding the reasons behind energy leaks in the 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 [18], [19]. On the other hand, approaches for the detection of portion of source code affected by energy leaks have been proposed by Zhang *et al.* [20] and by Gupta *et al.* [21] who exploited the use of dynamic analysis, but also by Li *et al.* [22] that proposed a technique for detecting specific lines of code affected by an energy bug.

Hindle [23] conducted a case study aimed at investigating to what extent changes made by developers across software versions affect the energy consumption. The results of the study indicated that (i) software change can effect the power consumption and (ii) there seems to exist a relationship between software metrics and power consumption.

Other researchers focused their attention in studying the relationship between the practices adopted by programmers during the development of a project and the energy consumption. Sahin *et al.* [24] studied the impact of code obfuscation on energy consumption, finding that the magnitudes of such impacts are unlikely to impact end users. The same authors also reported an analysis of the role of design patterns [25], finding the existence of some patterns (*e.g.,* the *Decorator* pattern) which negatively influence the energy efficiency of a software project. Similar results have been found by Noureddine and Rajan [26].

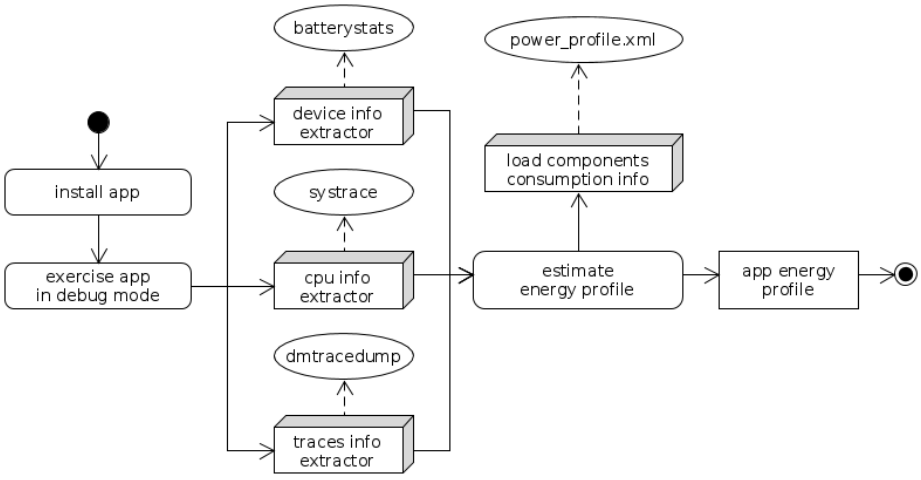
Hasan *et al.* [27] analyzed the impact of the data structures used by the developers, and specifically the influence of different Java Collections type. Results of their study showed that the application of the wrong type of data structure can increase the energy consumption by up to 300%. Other factors studied in the past and having a negative impact on energy efficiency are (i) the different sorting algorithms exploited [28], (ii) the use of lock-free data structures [29], (iii) the colors used in the GUI of software projects [30], (iv) the API usage of Android apps [7], and (v) the different refactorings applied to simplify the source code [31], [32].

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 to researchers and practitioners an easier way to approximate the energy consumption of the methods of a mobile app without hardware requirements: this can possibly help the community in conducting even more studies aimed at understanding and solving energy-related issues.

## II. PETRA: A POWER ESTIMATION TOOL FOR ANDROID APPLICATIONS

This section presents PETRA a (**P**ower **E**stimation **T**ool fo**r** **A**ndroid), a novel software-based approach to measure the power consumption of mobile apps. Figure 1 depicts the process followed by our approach to estimate the energy consumption at a method-level granularity.

Fig. 1: PETrA Architecture




The proposed approach is composed by three main blocks: (i) app preprocessing, (ii) energy profile computation, and (iii) output produced. In the following we detail each part independently.

**App Preprocessing.** The first input needed by our approach is the app to profile. Specifically, the input is characterized by an executable version of the app in the form of an `apk` file. In the preprocessing phase, 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). It is worth noting that the app to profile must enables the `debuggable` option, otherwise the instrumentation of the app, needed to profile it, would have not possible.

**Energy Profile Computation.** The second input of our approach consists of a set of test cases able to exercise the app under consideration. These test cases can be automated using existing tools (*e.g.,* MONKEYRUNNER or MONKEY) either manual operations performed by the software engineer. Once the test cases are ran, the *core* process behind PETRA starts.

Our approach is built on the top of the `Project Volta` Android tools, such as `dmtracedump`[3], `Batterystats`[4], and `Systrace`[5].

Specifically:

- `dmtracedump` provides an alternate way to show trace log files. The files generated by `dmtracedump` are easy to parse and allows 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 command line. The data collected can be analyzed as log file or can be converted in an HTML visualization that can be viewed in a browser using `Battery Historian`[4]. `PETrA` uses the `Batterystats` log in order to retrieve the active smartphone components and their status in a specific time window. Furthermore, it can provides the information about the device voltage. Given this information 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 performances. It captures and display 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 informations provided by `Systrace` are used for capturing the frequency of the CPU in a given time window. 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. Every smartphone has its own power profile. Often power profile information are publicly available or can be mined directly from the device, considering that each device manufacturer must provide this information[6].

Given the previous data it is possible to compute for every method call invocation the energy consumed. First of all, given a method call invocation and its termination we can calculate the time frame in which a frequency variation happened. For each time frame, we know for each smartphone components its state and related consumption. Indeed we can calculate the current intensity.

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

Let $C$ be the set of smartphone components, $I_{c,s}$ is the current intensity of the component $c$ with the state $s$. For example 92.6 is the number of milliAmpere consumed by a Nexus 4 when the cpu frequency is fixed to 384Mhz.

After calculating the current intensity, it is possible to calculate the energy consumed in a time frame, as indicated in eq. 2.

$$J = I * V * T \tag{2}$$

where $J$ is the consumed energy in Joule, $I$ is the current intensity in Ampere, $V$ is the device voltage in Volt and $T$ is the length of the time frame in seconds.

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

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

[3]https://developer.android.com/studio/profile/traceview.html

[4] https://developer.android.com/studio/profile/battery-historian.html

[5]https://developer.android.com/studio/profile/systrace-commandline.html

[6]https://source.android.com/devices/tech/power/

Listing 1: PETrA workflow

```
computeEnergyConsumption(apk, appName, nRuns){
  installApp(apk);
  for (run=0; run<nRuns; run++) {
    clearAppCache(appName);
    resetBatteryStats();
    startProfiler();
    exerciseApp(appName);
    stopProfiler();
    collectData();
    loadPowerProfile();
    for each method call in trace file {
      computeCallEnergyConsumption();
    }
    saveResults();
    stopApp(appName);
  }
  uninstallApp(apk)
}
```

By default PETrA uses `Monkey` for exercising the apps, but other exercisers could be used. For example for evaluating PETrA we used `Monkeyrunner`.

It is worth to note that `PETrA` relies on the `Android Activity Manager`[7], so the apk must be enabled for debugging. Furthermore, in order to provide better estimation, `PETrA` exercises the app multiple times (in our experiments `nRuns` is fixed to 10). It is worth to note that to avoid the bias due to multiple runs, at each run the app cache is cleaned and `Batterystats` is resetted.

## III. Evaluating the estimations provided by PETrA

TABLE I: The mobile apps considered in our evaluation

| # | Name | ID | Version | # of APIs |
|---|---|---|---|---|
| 1 | Battery HD | ch.smalltech.battery.free | 1.16 | 4.504 |
| 2 | Write Now Notepad | com.aerodroid.writenow | 1.1.5 | 16.755 |
| 3 | AndRecorder Free | com.andrconstruction.andrecord | 3 | 129 |
| 4 | Antivirus Free | com.antivirus | varies | 422 |
| 5 | Better Browser | com.browser.sogood.ui | 2.3 | 19.483 |
| 6 | AudioPlayer | com.bytemystery.audioplayer | 1.2 | 2.733 |
| 7 | Ï0,000 Quotes DB (FREE!)¨ | com.hmobile.quotesmegacollection | 3.0.4 | 11.067 |
| 8 | Android Music Player | com.jrtstudio.music | 4.0.4b3 | 13.823 |
| 9 | Android Antivirus | com.lab4apps.antivirus | 2.0.1 | 12 |
| 10 | Livo Recorder Lite | com.mp1.livolite | 3.0.7.a | 1 |
| 11 | Simple Weather | com.netthreads.android.weather | 1.1.3 | 13.143 |
| 12 | SimpleNews | com.prss.simplenews | 1.4 | 138 |
| 13 | 25000 Best Quotes | com.puissantapps.quotesapp.free | 1.0.7 | 1 |
| 14 | Classical Music Radio Lite | com.rslclasslite | 1.0.3 | 8.953 |
| 15 | news | com.segvic.news | 1.0.0 | 737 |
| 16 | Droid Notepad | com.williamkingdom.droidnotepad | 1.11 | 1.637 |
| 17 | Inspiring Quotes | com.xstudio.inspiringquotes | 1.2 | 12 |
| 18 | Battery Info | com.zgame.batteryinfo | 1.6 | 6.233 |
| 19 | Anime Radio Online | free.animeradioonline.gutisoft | 1.06 | 8848 |
| 20 | Wifi Radar | girsas.wifiradar | 1.06 | 185.189 |
| 21 | Battery Info Always | jp.dip.sys1.android.battery | 1.2.0 | 185 |
| 22 | Meridian Media Player Revolute | org.iii.romulus.meridian | 2.4.5 | 13.871 |
| 23 | Better Notepad | org.strive.notes | 0.0.5 | 18.317 |

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 to replace hardware-based solutions. More specifically, the study aims at addressing the following research question:

- **RQ$_1$:** *What is the accuracy of the estimations provided by `PETrA`?*

### A. Context Selection and Oracle Extraction

The *context* of the study consists of a set of 23 Android apps from `Google Play Store` of different categories and having different scope. Table I reports for each app (i) its name and its `Google Play Store` identifier, (ii) the specific

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

version taken into account, and (iii) the number of APIs used. 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 actual consumption of source code methods is publicly available. Indeed, to answer our research question we needed to quantify the actual energy consumption of the methods of an app. However, since we had not available any hardware-based tool to perform effective measurements, we had to look for alternative solutions.

Some datasets available provide data about the energy consumption of software changes [33] or system calls [34]. These datasets are not suitable for our scope since they do not provide detailed measures for source code methods. For this reason we used the dataset provided by Linares-Vasquez *et al.* [7], which reports the actual power consumption of the methods belonging to the APIs used by 55 mobile apps. The authors computed the measurements relying on the MONSOON toolkit [6]. 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.* [7] (more details on the measurement process later in this section). As a consequence of this choice, we had to limit the focus of our analysis to the methods belonging to an API.

### *B. Test Environment Setup and Energy Profiles Extraction*

Fig. 2: Test environment

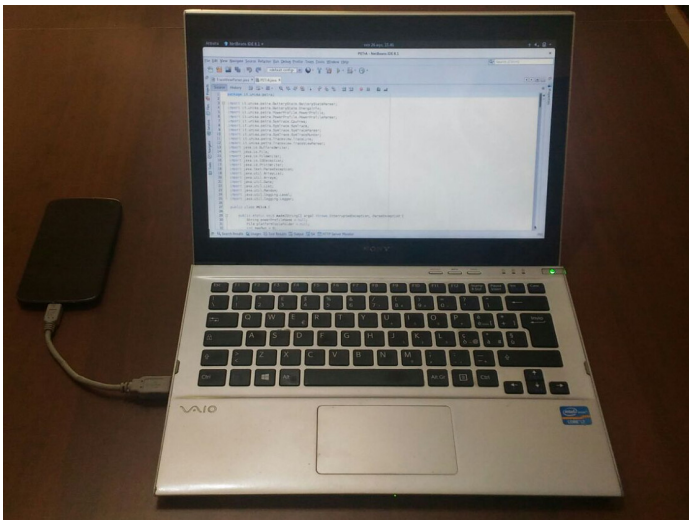


Figure 2 shows our test environment. Being PETRA a software-based approach it requires a simple test environment composed only by a smartphone and a PC. Despite this, when performing measurements of the energy consumption, having a well-isolated test environment is needed to avoid biases. To this aim, we carefully followed the guidelines coming from previous work in the field [4], [7], [17], [35]. The subsequent subsections detail each design choice.

**Choice of the Smartphone.** Table II reports the characteristics of the phone used in the experiment. Specifically, we selected a factory re-setted `LG Nexus 4` having Android 5.1.1 Lollipop as operating system, and equipped with 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 of having the same smartphone used in the paper by Linares-Vasquez *et al.* [7] in order to conduct a fair evaluation. Moreover, it is worth noting that this particular hardware allow to be connected via a data cable, namely a cable where the USB charging can be disabled[8]. 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, we firstly modified the settings of such apps in order to enable the debug mode. To this aim, for each app we manually added in the `AndroidManifest.xml` file the option `android:debuggable="true"`. Then, we re-generated the `apk` file, *i.e.,* the executable version of the app, using the ANDROID STUDIO IDE [36].

Once having a debuggable and exacutable version of the apps, we ran PETRA over them. As explained in Section II, our approach receives as input a set of test cases for exercising the app under consideration and measuring the energy consumption

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

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) |

at method-level granularity. In the context of this experiment, we exercised the apps in our dataset by using exactly the same `Monkeyrunner`[9] test cases used by Linares-Vasquez *et al.* [7]. This was needed to conduct a fair evaluation between the energy profiles extracted using our approach and the oracle provided using the MONSOON toolkit [6].

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 [6], 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 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. Each run costs around five minutes since, as reported by Choudhary et al. [37], this is the time needed by Monkey to achieve code coverage convergence. The results achieved after 10 runs (*i.e.,* the joules consumed by the methods in each run) have been aggregated using the mean operator. Therefore, the final output consisted of a unique value representing the average energy consumed by the methods belonging to an API exercised during the test execution.

### C. Data Analysis and Metrics

Once we extracted the energy profiles using `PETrA`, we answered $\mathbf{RQ}_1$ by comparing the energy profiles computed using `PETrA` with the oracle provided by Vasquez *et al.* in their empirical study [7]. To evaluate to what extent the energy consumption provided by our approach is close to the actual values, we used a set of metrics widely used in the area of cost estimation [38] [39]. Specifically, we used the *Mean Magnitude Relative Error* (MMRE) [38] defined as follow:

$$MMRE = \frac{1}{N}\sum_{i=1}^{n} MRE_i \tag{3}$$

where $n$ is the number of energy estimations computed by PETRA on each app, and $MRE$ indicates the *Magnitude Relative Error* [38] and has values in the range defined by the following formula:

$$MRE_i = \frac{|y_i - \hat{y_i}|}{y_i} \tag{4}$$

where $y_i$ is the actual energy value e $\hat{y_i}$ is the energy estimation provided by `PETrA` for the method $i$.

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

Besides estimating the mean error in the estimations provided by our approach, we also computed the $PRED(x)$ metric, namely the *Relative Error Deviation Within x%* [40]. This measure gives an indication of how many estimation errors of our approach are within x% of the actual values provided by the oracle. In particular, $PRED(x)$ is defined as the average fraction of the $MREs$ off by no more than x as defined by Jorgensen [41].

$$PRED(x) = \frac{1}{n}\sum_{i=1}^{n}\begin{cases} 1 \; if \; MRE_i \leq x \\ 0 \; otherwise \end{cases} \tag{5}$$

In the field of cost estimation, the parameter x is usually set to 25, *i.e.,* the estimated cost is within the 25% of the actual cost of a project [39]. 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 [27]. For this reason, an analysis done in this way would have been too coarse-grained. Thus, we verified whether PETrA can achieve a lower estimation error, by setting x=5; 10; 20; 25, 30; 50. In this way, we were able to control how the estimation errors of our approach were 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 $PRED(5)$ is equal to 0.40 and $PRED(0.10)$ is 0.42, 2% of the estimation errors are between $0.05$ and $0.10$.

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.

## IV. Analysis of the Results

In this section we describe the results achieved to answer our research question.

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

| # | MMRE | PRED(0.05) | PRED(0.10) | PRED(0.20) | PRED(0.25) | PRED(0.30) | PRED(0.50) | over estimations | under estimations |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.04 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | $< 0.01$ |
| 2 | 0.01 | 0.99 | 0.99 | 0.99 | 0.99 | 1.00 | 1.00 | 0.99 | 0.01 |
| 3 | $< 0.01$ | 0.92 | 0.92 | 0.92 | 0.92 | 0.92 | 0.94 | 0.92 | 0.08 |
| 4 | $< 0.01$ | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | $< 0.01$ |
| 5 | 0.01 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | $< 0.01$ |
| 6 | $< 0.01$ | 0.89 | 0.91 | 0.94 | 0.96 | 0.97 | 0.99 | 0.87 | 0.13 |
| 7 | $< 0.01$ | 0.99 | 0.99 | 1.00 | 1.00 | 1.00 | 1.00 | 0.99 | 0.01 |
| 8 | $< 0.01$ | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 | 0.01 |
| 9 | $< 0.01$ | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | $< 0.01$ |
| 10 | $< 0.01$ | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | $< 0.01$ |
| 11 | $< 0.01$ | 0.98 | 0.98 | 0.99 | 0.99 | 0.99 | 0.99 | 0.98 | 0.02 |
| 12 | $< 0.01$ | 0.32 | 0.32 | 0.38 | 0.38 | 0.38 | 0.43 | 0.28 | 0.72 |
| 13 | $< 0.01$ | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | $< 0.01$ |
| 14 | 0.01 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | $< 0.01$ |
| 15 | $< 0.01$ | 0.11 | 0.13 | 0.17 | 0.17 | 0.30 | 0.56 | 0.11 | 0.89 |
| 16 | $< 0.01$ | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 | 1.00 | 0.99 | 0.01 |
| 17 | $< 0.01$ | 0.92 | 0.92 | 0.92 | 0.92 | 1.00 | 1.00 | 0.92 | 0.08 |
| 18 | 0.01 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | $< 0.01$ |
| 19 | 0.01 | 0.98 | 0.98 | 0.99 | 0.99 | 0.99 | 1.00 | 0.98 | 0.02 |
| 20 | 0.03 | 0.95 | 0.95 | 0.95 | 0.95 | 0.95 | 0.95 | 0.95 | 0.05 |
| 21 | $< 0.01$ | 0.97 | 0.97 | 0.97 | 0.97 | 0.97 | 0.97 | 0.96 | 0.04 |
| 22 | 0.01 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | $< 0.01$ |
| 23 | 0.02 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | $< 0.01$ |
| **Overall** | $< 0.01$ | **0.91** | **0.92** | **0.92** | **0.92** | **0.93** | **0.95** | **0.91** | **0.09** |

Table III shows the MMRE achieved comparing the estimations provided by PETrA with those achieved by Linares *et al.* in their empirical study [7] using an hardware-based solution relying on Monsoon [6]. In all the considered applications PETrA provides an energy estimations within $0.05$. In particular on 14 out 23 apps the estimations overall error is less than $0.01$. Looking at $PRED(x)$, we can see that overall 91% of the estimations is within a 5% of error, while 95% is within the 50% of error. It is worth to note that PETrA rarely underestimates the energy consumption. Indeed less than 9% of the energy values is underestimated.

## V. Threats to Validity

The main threats related to the relationship between theory and observation (*construct validity*) are due to imprecisions in the measurements we performed.

As briefly explained in Section IV, we empirically evaluated the accuracy of the approach on 23 mobile apps comparing the power estimation of the tool with the oracle provided by Linares-Vasquez *et al.* [7]. The validation revealed that in 91% of the cases the estimations of our tool are within the 5% of the actual values. Therefore, we believe that the data provided by

the tool are close enough to the actual energy consumption. Moreover, we aggregated the results given by PETrA in 10 runs using the mean operator, that may affect the results of the study. In order to be more confident about our findings, we repeated the experiment by aggregating the energy consumption using the sum, *i.e.,* the final output was a unique value representing the sum of the energy consumption of the methods exercised during the 10 runs.

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 by presenting descriptive statistics and using proper statistical tests in order to assess the significance of our findings.

Finally, regarding the generalization of our findings (*external validity*) we considered 23 apps of different category. However, further studies aiming at replicating our work on larger datasets are desirable and part of our future agenda.

## REFERENCES


[1] 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.

[2] 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.

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

[4] 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

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

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

[7] 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

[8] 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.

[9] 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

[10] 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.

[11] 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.

[12] 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.

[13] 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

[14] 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.

[15] 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.

[16] 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.

[17] 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

[18] 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

[19] 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

[20] 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

[21] 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

[22] 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

[23] 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

[24] 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

[25] 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

[26] 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

[27] 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

[28] 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.

[29] 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.

[30] 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

[31] 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

[32] 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.

[33] 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

[34] 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.

[35] 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.

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

[37] S. R. Choudhary, A. Gorla, and A. Orso, "Automated test input generation for android: Are we there yet? (e)," in *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, ser. ASE '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 429–440. [Online]. Available: http://dx.doi.org/10.1109/ASE.2015.89

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

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

[40] 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.

[41] 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.