PETrA: a Software-Based Tool for Estimating the Energy Profile of Android Applications

Dario Di Nucci[1], Fabio Palomba[1 3], Antonio Prota[1], Annibale Panichella[2], Andy Zaidman[3], Andrea De Lucia[1]
[1]University of Salerno — [2]University of Luxembourg — [3]Delft University of Technology

***Abstract*—Energy efficiency is a vital characteristic of any mobile application, and indeed is becoming an important factor for user satisfaction. For this reason, in recent years several approaches and tools for measuring the energy consumption of mobile devices have been proposed. Hardware-based solutions are highly precise, but at the same time they require costly hardware toolkits. Model-based techniques require a possibly difficult calibration of the parameters needed to correctly create a model on a specific hardware device. Finally, software-based solutions are easier to use, but they are possibly less precise than hardware-based solution. In this demo, we present PETRA, a novel software-based tool for measuring the energy consumption of Android apps. With respect to other tools, PETRA is compatible with all the smartphones with Android 5.0 or higher, not requiring any device specific energy profile. We also provide evidence that our tool is able to perform similarly to hardware-based solutions.**

***Keywords*-Energy Consumption; Mobile Apps; Estimation**

I. INTRODUCTION

Laptops, smartphones and tablets are becoming the most used devices for billions of users from all over the world [1], which rely on portable devices for running a number of mobile applications (*i.e.,* apps) essential for their daily activities. Unfortunately, the utility of mobile apps and, more in general of mobile devices, is threatened by their energy efficiency [2]. To deal with energy related issues, the research community proposed several solutions which are mainly concerned with hardware efficiency (*e.g.,* circuit efficiency [3]). However, recent findings have shown software to be a potential source of energy inefficiency for mobile apps (see *e.g.,* [4]).

In this context, correctly measuring the energy consumption of mobile devices becomes an important step to identify *energy leaks*, *i.e.,* portions of source code consuming too much energy. In the last five years, a number of approaches and tools for measuring energy consumption have been proposed. The vast majority of them can be classified as *hardware-based*, and rely on hardware toolkits to perform measurements. For instance, Hindle *et al.* devised GREENMINER [5], a hardware mining testbed based on an ARDUINO board with an INA219 chip [6]. 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.

While hardware-based solutions are supposed to be more precise in their measurements, they require costly hardware devices. Consequently, researchers are trying to find alternatives to approximate the energy consumption. One good choice is represented by *model-based* approaches, which define mathematical functions to estimate the energy consumption of mobile apps on a given hardware device. For example, Zhang *et al.* [7] proposed POWERBOOTER, 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. However, a not always easy-to-find calibration of the parameters of the model is essential to correctly estimate power consumption.

Finally, *software-based* approaches use only the system functionalities to estimate the power consumption, without constructing any specific model. For instance, Hao *et al.* proposed ELENS [8] a tool for 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 to better understand the application behavior.

Despite the fact that some techniques are showing good performance, there is a still lack of publicly available tools able to combine the cost-effectiveness of software and model-based tools and the reliability of hardware-based tools in order to quickly and efficiently measure energy consumption of mobile applications [9]. Moreover, most of these approaches are not easily usable in practice. For example, ELENS requires that a `Software Environment Energy Profile` (SEEP) is provided by the manufacturer of the specific Android device, but currently this scenario is not common although Hao *et al.* provide guidelines to develop a SEEP [8].

In this paper, we present a novel tool, coined PETRA (**P**ower **E**stimation **T**ool fo**r** **A**ndroid), able to measure the energy consumption of ANDROID apps relying on the tools and APIs provided with the publicly available `Project Volta`[2]. For this reason, PETRA is compatible with all the smartphones equipped with Android 5.0 or higher. Our tool provides estimations at the method level.

An empirical investigation that we have conducted to evaluate PETRA indicates similar performance to hardware-based solutions. In fact, the mean relative error with respect to the MONSON toolkit [10] 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.

Tool and Data Replication. The executable file and all the

[1]http://softwareprocess.es/static/GreenMining.html

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

data used in the experiment are available on the PETRA replication package [11]. Moreover, a video of the tool at work is available at `https://youtu.be/1rrFPDMIBbI`. The binaries and the data are released under the MIT license[3].

II. PETRA: A POWER ESTIMATION TOOL FOR ANDROID APPLICATIONS

This Section reports the main characteristics of PETRA, as well as details about its architecture and inner-working.

A. Design Goals

Harman *et al.* [9] recently pointed out two main characteristics that should be met by an energy measurement tool, namely a (1) quick and (2) efficient estimation of the energy consumption. These two characteristics should contribute to the systematic use of such tools in the context of other processes (*e.g.,* software testing [9]). For this reason, we designed PETRA in order to face these two challenges. Specifically, the characteristics of our tool can be summarized as follow:

- **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 little influence on the estimation process. In this way, it is possible to minimize the *Hawthorne effect* in which the measurements are affected by the measurement process [9]. 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 output results that can be easily analyzed.

B. PETrA's inner workings

PETRA's workflow can be observed in Figure 1. PETRA starts a pre-processing phase, which entails installing the app (step 1), cleaning the app cache and resetting the Android tools that PETrA needs in order to make the estimations (step 2). Such pre-processing phases are needed to create an adequate test environment not influenced by the previous app executions. Subsequently, in step 3 of Figure 1, it exercises the app using (i) ANDROID MONKEY tool[4], or (ii) a ANDROID MONKEYRUNNER script[5]. ANDROID MONKEY generates pseudo-random streams of user events such as clicks, touches, or gestures, as well as a number of system-level events with the aim of performing stress-testing of the app under analysis. ANDROID MONKEYRUNNER is an API for controlling an Android device. It allows to write programs

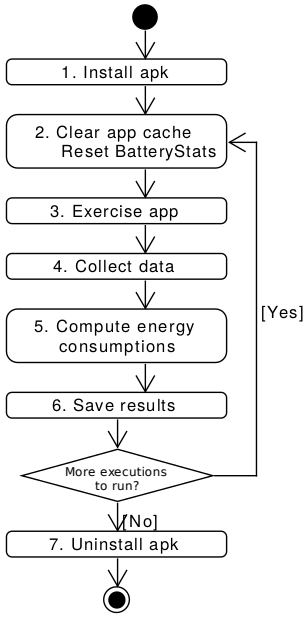

Fig. 1: PETRA: workflow

that install Android applications, run them while sending keystrokes, take screenshots of its user interface, and store screenshots on the workstation.

Afterwards, PETRA collects the data provided by the Android tools (step 4) and computes the energy consumed by the methods previously executed (step 5). At a high level, PETrA's estimation works as follows: given a time window T_w, we define a time frame T_Δ as a smaller time unit in which none of the smartphone components changed its state. In each time frame T_Δ, PETRA extracts the state of each hardware component (*e.g.,* CPU, WiFi, GPS) and the device voltage. Afterwards, it calculates the current intensity of the smartphone looking at the information contained in the `Power profile file`. Given current intensity, voltage, and time frame length PETRA calculates the energy consumed in Joule. The energy consumed by a method call is then calculated by summing up the energy consumed in each time frame T_Δ in which the method call was active:

$$J = \sum_{T_\Delta \in T_w} (I_\Delta \times V_\Delta \times T_\Delta) \quad (1)$$

where J 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. Further details are reported in the paper by Di Nucci *et al.* [13]. To perform the estimations, PETRA relies on some of the tools available from `Project Volta`[6], which are responsible for handling the communication with the hardware components of the smartphone the app is running on. Specifically, we rely on `dmtracedump`[7], `Batterystats`[8], and `Systrace`[9]:

[3]https://opensource.org/licenses/MIT
[4]http://tinyurl.com/gvnxdd3
[5]https://developer.android.com/studio/test/monkeyrunner/index.html
[6]http://tinyurl.com/lvj4lxq
[7] https://developer.android.com/studio/profile/traceview.html
[8]https://developer.android.com/studio/profile/battery-historian.html
[9]https://developer.android.com/studio/profile/systrace-commandline.html

- `dmtracedump` allows to show trace log files. For each method call it provides the entry and the exit time at microseconds granularity.
- `BatteryStats` is an open source tool of the Android framework able to collect battery data from the device under evaluation. `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.
- `Systrace` is a tool that can be used to analyze application performance. In PETRA, the information provided by `Systrace` is used to capture the frequency of the CPU in a given time window. Considering that CPU's consumption changes as its frequency varies, this information completes the one provided by `Batterystats` improving the estimations.

Considering these sources of information, collecting and merging all this information represented the main challenge during the development of the tool. In particular, when the time the app is exercised is long, the data extraction is not a trivial task: we mitigated this issue applying efficient streaming algorithms. As a consequence of the choices applied in this stage, PETRA requires to add the *Android SDK* and in particular the *Android Development Bridge* (`ADB`) to the `PATH` environment variables. `ADB` is a command line tool needed to allow the different tools to exchange information with the mobile device. Moreover, the `apk` of the app to be analyzed must be enabled for debugging.

As explained in step 6 of Figure 1, the results are then saved as CSV file in the same folder of apk. The steps [2-6] are repeated for each execution. Finally, the apk is uninstalled from the device (step 7).

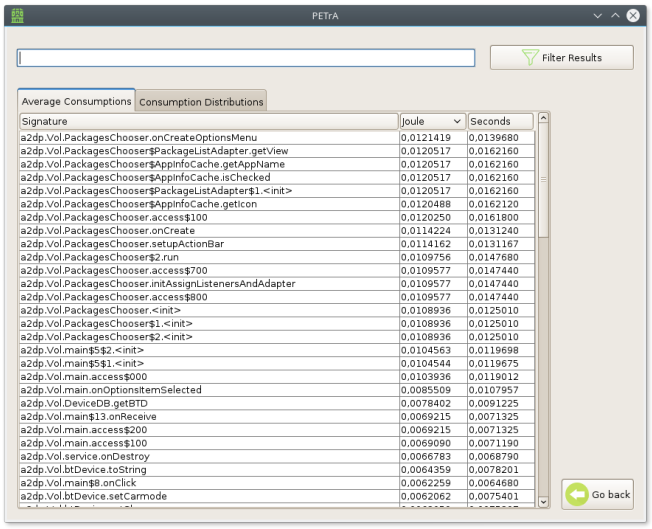

Fig. 2: PETRA: Results Table View

C. PETrA at Work

Being a software-based approach, PETRA does not require any additional hardware equipment and, consequently, any strong experience in the setup of the test bed. Indeed, a developer interested in computing the energy consumption of her mobile applications with our tool simply needs to connect her smartphone and run the executable `jar` file via command line using the following command:

```
java -jar PETrA-1.0.jar
```

Subsequently, PETRA shows the configuration view. Using this window, the developer can customize PETRA with respect to (i) the location of the `apk` file, (ii) the ANDROID MONKEY options, (iii) the ANDROID MONKEYRUNNER script location, (iv) the number of times (*i.e.,* runs) the energy measurements must be computed, (v) the location of the ANDROID SDK, and (vi) the location of the XML file containing the power profile of the device. Regarding the ANDROID MONKEY configuration it is possible to set the number of interactions (*e.g.,* keystrokes, gestures) to send to the app and the time that must elapse between an interaction and the next one.

Once the input parameters have been configured and the *Start Energy Estimation* button is clicked, the process described in the previous section is executed. When the estimations have been computed, the *Statistics* button is enabled.

This button allows the user to see the results of the computations though the table depicted in Figure 2. The table reports, for each method of the app under analysis, (i) its signature, (ii) the average Joules consumed during the tests, and (iii) the average time in which it was executed, expressed in seconds. Note that the methods are sortable by both Joules consumed and execution time. Moreover, the results can be filtered by using the field available in the top part of the view, thus, allowing the user to select methods having specific names or belonging to specific packages.

Additionally, the user can analyze the results of the top 5 *energy-greedy* methods by looking at the distribution of the energy estimations computed during the tests. In particular,

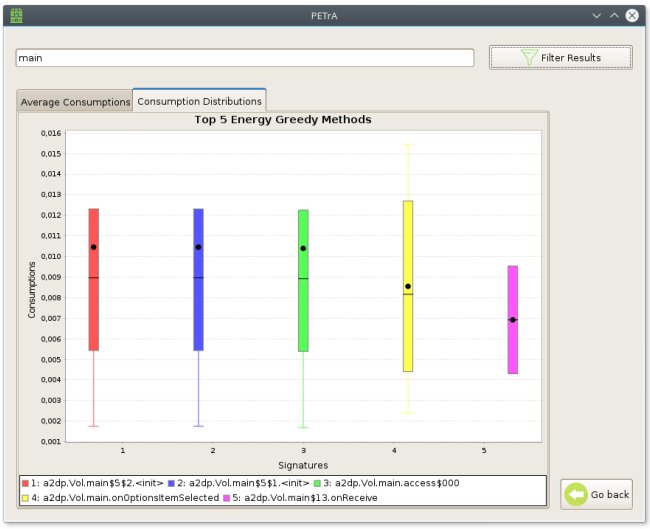

Fig. 3: PETRA: Results Plot View

clicking on the *Consumption Distribution* tab of the window, a new view, depicted in Figure 3, is shown. To better study the energy consumption estimations, five box plots (*i.e.,* one for each top 5 greedy method) are reported.

III. Evaluation

The ability of PETRA in providing correct estimations has been studied on a set of 54 mobile applications from the dataset provided by Linares-Vasquez *et al.* [14]. The applications were exercised using the same test data (*e.g.,* Monkeyrunner scripts), previously used (included in the dataset). Due to space constraints, in this Section we summarize the main findings achieved in the empirical study [13]. Further details are reported in our replication package [11].

As done in other work [14], the experiment has been conducted using a `LG Nexus 4`. The *goal* was to analyze how close the estimations of PETRA were with respect to the ones provided by a hardware-based tool used in previous research [14], such as the MONSON toolkit [10]. To this aim, we compared the estimations of the two tools on 414,899 API calls belonging to the 321 APIs in terms of two widely known metrics, *i.e.,* Mean Magnitude Relative Error (MMRE) [15] and PRED(x) [16].

It is worth noting that to isolate the behavior of an application being executed on the smartphone, we adopted a number of precautions. In particular, we 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.

The results show that the estimations produced by the tools are quite close to each other. Specifically:

- **Error Magnitude.** The average estimation error achieved using PETRA is 4% with respect to actual value computed using the MONSOON toolkit. Moreover, 95% of the PETRA's estimation errors are within 5% of the actual values.
- **Error Reasons.** A significant usage of the network capabilities or sensors negatively impacts the measurement error.
- **Error Type.** Most of the estimations are over-estimations (*i.e.,* 89%). In the remaining cases, the use of sensors and network produces underestimations.

IV. Conclusion Remarks

In this demo, we presented PETRA, a software-based tool for the estimation of the energy consumption of Android apps at method level granularity. Unlike other tools proposed in the literature, it uses reliable tools coming from the Android Toolkit and does not exploit energy models that need to be calibrated. Moreover, it is publicly available. We also reported the main findings of the empirical study conducted to evaluate the correctness of the estimations of PETRA, which show that our tool performs similarly to the MONSOON toolkit. Our research agenda will focus on designing and developing new techniques that better estimate the energy consumption exploiting those components such as sensors and network. With this aim, we will work on providing more precise information on the state of components, exploring new Android tools (*i.e.,* `GPU Monitor`[10] and `Network Monitor`[11]) and designing new services able to capture those components not analyzed by the ANDROID TOOLS, *e.g.,* sensors.

References

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

[2] 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), IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*. IEEE, 2013, pp. 134–141.

[3] H. Jabbar, Y. S. Song, and T. T. Jeong, "Rf energy harvesting system and circuits for charging of mobile devices," *IEEE Transactions on Consumer Electronics*, vol. 56, no. 1, pp. 247–253, February 2010.

[4] S. Hasan, Z. King, M. Hafiz, M. Sayagh, B. Adams, and A. Hindle, "Energy profiles of java collections classes," in *Proc. of the International Conference on Software Engineering (ICSE)*. ACM, 2016, pp. 225–236.

[5] 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 *Proc. of the Working Conference on Mining Software Repositories (MSR)*. ACM, 2014, pp. 12–21.

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

[7] 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 IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, 2010, pp. 105–114.

[8] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan, "Estimating mobile application energy consumption using program analysis," in *Proc. Int'l Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 92–101.

[9] M. Harman, Y. Jia, and Y. Zhang, "Achievements, open problems and challenges for search based software testing," in *Proc. Int'l Conf. on Software Testing, Verification and Validation (ICST)*. IEEE, 2015, pp. 1–12.

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

[11] D. Di Nucci, F. Palomba, A. Prota, A. Panichella, A. Zaidman, and A. De Lucia. (2016, 11) Petra: a software-based tool for estimating the energy profile of android applications. [Online]. Available: https://doi.org/10.6084/m9.figshare.4233767.v1

[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] D. Di Nucci, F. Palomba, A. Prota, A. Panichella, A. Zaidman, and A. De Lucia, "Software-based energy profiling of android apps: Simple, efficient and reliable?" in *Software Analysis, Evolution, and Reengineering (SANER), 2017 IEEE 24rd International Conference on*. IEEE, 2017, p. To appear.

[14] 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 *Proc. Working Conference on Mining Software Repositories (MSR)*. ACM, 2014, pp. 2–11.

[15] L. C. Briand and I. Wieczorek, *Resource Estimation in Software Engineering*. John Wiley & Sons, Inc., 2002.

[16] 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.

[10] https://developer.android.com/studio/profile/am-gpu.html

[11] https://developer.android.com/studio/profile/am-network.html