# *E-MANAFA*: Energy Monitoring and ANAlysis tool For Android

Rui Rua
HASLab/INESC TEC, Portugal
University of Minho, Portugal
rui.a.rua@inesctec.pt

João Saraiva
HASLab/INESC TEC, Portugal
University of Minho, Portugal
saraiva@di.uminho.pt

## ABSTRACT

This article introduces the *E-MANAFA* energy profiler, a plug-and-play, device-independent, model-based profiler capable of obtaining fine-grained energy measurements on Android devices. Besides having the capability to calculate performance metrics such as the energy consumed and runtime during a time interval, *E-MANAFA* also allows to estimate the energy consumed by each device component (e.g. CPU, WI-FI, screen). In this article, we present the main elements that compose this framework, as well as its workflow. In order to present the power of this tool, we demonstrate how the tool can measure the overhead of the instrumentation technique used in the *PyAnaDroid* application benchmarking pipeline, which already supports *E-MANAFA* to monitor power consumption in its Android application automatic execution process.
**Video demo:** shorturl.at/hmyz5

## 1 INTRODUCTION

Since the beginning of the century, we have witnessed a revolution in computer systems portability. Manufacturers focused on developing a wide variety of devices, which can offer complex and demanding features over several hardware components, such as GPS, Bluetooth, among others. As a consequence, the computational effort needed by such non-wired devices also demands a lot from a critical hardware resource: the battery. Given the limited capacity of current batteries, consumers are becoming more energy aware regarding the apps [23] that they install.

Furthermore, improving the efficiency of mobile devices and mobile apps has become one of the most important concerns for software developers [23]. As a consequence, a lot of research work in green software is being done in the context of the Android ecosystem. Such research aims at analyzing the energy consumption of mobile apps in multiple ways, such as by measuring/estimating the energy consumed by an application or block of code [4, 5, 21], by detecting energy-expensive coding patterns [6, 22, 25], or by defining green/red APIs [19, 29].

Despite many recent efforts, there is still a lot of difficulty in identifying and repairing energy issues in Android applications. Although there are several techniques for energy analysis in the Android ecosystem, all of them have several limitations that make their use suitable only in certain execution contexts. Energy issues are more difficult to repair than non-energy issues [16] and typically only reveal themselves in specific contexts [17]. As a result, finding a reliable tool to estimate energy consumption in a given execution context can be extremely complicated.

This work is financed by the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within project UIDB/50014/2020.

In order to fill all the applicability and replicability flaws of the existing tools, this article presents *E-MANAFA* [7], a device-independent profiler that can be used in state-of-the-art devices to obtain reliable estimates of fine granularity. *E-MANAFA* is also the first tool that is able to quantify energy consumption estimates of each device component and record events that occurred during time intervals of the monitoring process, such as component state changes, execution of background jobs, etc.

*E-MANAFA* can be easily installed on any Android development machine as a Python package and can be used as a 3rd party library or command line tool. It can also be combined with instrumentation techniques to perform method consumption estimates of Android applications. Its code is open-source and is open to community contributions. Its operation is based on open-source tools and services, which are part of the Android platform.

The rest of this paper is organized as follows: Section 2 presents the related work. Section 3 presents the components on which *E-MANAFA* is based, as well as its functioning. Afterward, Section 4 presents some results and use-cases in which *E-MANAFA* can be used. Finally, in Section 5 we present some final conclusions.

## 2 RELATED WORK

Both academia and industry have been developing tools aiming to ease and improve the process of measuring or estimating software energy consumption in the Android ecosystem. The most accurate and reliable tools tend to be hardware-based artifacts, such as Monsoon [20] or ODroid [14]. However, such tools are not easily available for developers and researchers, incurring a considerable monetary and setup-time cost. They also imply disassembling the device to bypass the battery, incurring the risk of physically damaging the device and voiding the manufacturer's warranty. Furthermore, it also limits the reproducibility and parallelization of studies that consider several different mobile devices.

Consequently, several attempts of producing more user-friendly and accurate tools that can be integrated with the typical development environment have emerged in the literature. Android Studio offers an energy profiler [9], to allow developers to analyze their app's energy consumption. It provides a GUI that allows visualizing the variation of the power values being consumed by the CPU, GPS and Networking during an app execution. This tool was specially conceived for app developers, being suitable for performing single-app analysis. However, it is IDE-dependent, which limits its capability to be ported to automatic processes aiming at analyzing a large set of executions or corpus of applications, since it is built into Android Studio and relies on its GUI and graphical representations to present the power behavior of an app during its execution.

One of the most used profilers to conduct energy-related studies in the literature is Qualcomm's Trepn Profiler [8], that allows measuring power consumption and system resources usage in devices

**Table 1: Comparison among the state-of-the-art tools for energy profiling in the Android ecosystem**

| Tool | Open source | Plug And Play | Calibration | Device dependent | API version | Device disassembly | Sample rate | Per-comp. estimates | Library |
|---|---|---|---|---|---|---|---|---|---|
| **Android Profiler** [9] | ✓ | ✓ | ✗ | ✗ | ≤ 18 | ✗ | 1 ms | ✗ | ✗ |
| **Petra** [21] | ✓ | ✗ | ✓* | ✓* | ≥ 18 | ✗ | 1 ms | ✗ | ✗ |
| **Trepn Profiler** [8] | ✗ | ✓ | ✗ | ✗ | ≤ 26* | ✗ | 100 ms | ✗ | ✓ |
| **PowDroid** [3] | ✓ | ✓ | ✗ | ✗ | ≥ 21 | ✗ | 1 ms | ✗ | ✗ |
| **Greenscaler** [4] | ✓ | ✗ | ✓ | ✗ | All | ✗ | 1 ms | ✗ | ✓ |
| **Monsoon** [20] | ✗ | ✗ | ✗ | ✗ | All | ✓ | 1 ms | ✗ | ✓ |
| **ODroid** [14] | ✗ | ✗ | ✗ | ✗ | All | ✓ | 1 ms | ✗ | ✓ |
| **ORKA** [28] | ✗ | ✗ | ✓* | ✗ | All | ✗ | 1 ms | ✗ | ✓ |
| **Power Tutor** [30] | ✓ | ✓ | ✗ | ✓ | ≤ 11 | ✗ | 1 s | ✗ | ✓ |
| ***E-MANAFA*** | ✓ | ✓ | ✗ | ✗ | ≥ 28 | ✗ | 1 ms | ✓ | ✓ |

with Snapdragon chipsets. However, this profiler is not been updated since 2015 and was even recently removed from the Google Play Store, while there are no reports of the tool providing accurate results in recent devices and platform versions. Besides Trepn, there are other profilers such as GreenScaler [4] or Petra [21] that can be used to estimate the energy consumption on the Android platform. These tools require calibration or tuning for different device models and do not provide warranties that the tool can work with different devices or Android platforms. GreenScaler is a model-based approach for Samsung Galaxy devices that can estimate energy consumption based on system calls and CPU measurements. Petra is a monitoring tool that collects dynamic data from Android services such as *batterystats* [10]. More recently, a new tool named Powdroid [3] emerged from the literature, being specially designed for performing black-box energy analysis of Android apps, limiting its portability for white-box testing scenarios.

Table 1 presents a comparison of the state-of-the-art tools for monitoring energy consumption on Android. These tools are compared according to different criteria, including whether their source code is openly available, require calibration, work only for a limited set of devices, sample rate, provide per-component energy estimates, among others.

As can be observed in Table 1, *E-MANAFA* presents the best possible values in practically all criteria, except for *API version*. However, according to data reported in 2021, *E-MANAFA* can be used in more than 69% of the devices used globally [24], and this figure will tend to increase over time.

## 3 THE *E-MANAFA* FRAMEWORK

This section presents the *E-MANAFA* tool, describing the foundations on which it was implemented, as well as how it works as a third-party library or CLI tool. *E-MANAFA* was developed to be a plug-and-play energy monitoring tool, aiming to provide an intuitive and pleasant installation and usage experience for any type of user. Thus, we strive for the *E-MANAFA* to be adopted by researchers and practitioners to perform energy consumption analysis of applications. Especially analysis that covers a large corpus of applications or a large set of executions.

*E-MANAFA* is a purely software and model-based profiler, which allows estimating the energy consumption of Android devices at a low level of granularity. Like other state-of-the-art profilers [3, 4, 8, 21], despite providing system-level measurements, *E-MANAFA* can be used to estimate energy consumed by applications. Thus, energy measurements must be carried out in controlled environments, where precautions have been taken to minimize interference from system processes and other applications [27].

*E-MANAFA* is the only profiler known to the authors capable of providing quantitative estimates of the energy consumed by each device component. *E-MANAFA* can be used alongside instrumentation techniques to estimate source code consumption. The tool has already been integrated into the *PyAnaDroid* execution pipeline, which precisely uses *E-MANAFA* together with code instrumentation to benchmark Android applications.

*E-MANAFA* is a device-independent tool, since its operation is based on resources present in the Android SDK and platform. The tool can be used in any operating system where the Android SDK can be installed. Furthermore, it requires no calibration, since it relies on data provided by the manufacturer and system services to estimate energy consumption and this data is extracted from the device when the profiler is executed. To be executed in a device, our tool relies on 4 Android artifacts, namely: the *power_profile.xml* file, the *batterystats* service, the *perfetto* [13] service, and (eventually, the utility *logcat*. We describe each of them in the following sections. Figure 1 presents a high-level overview of the architecture of the profiler.

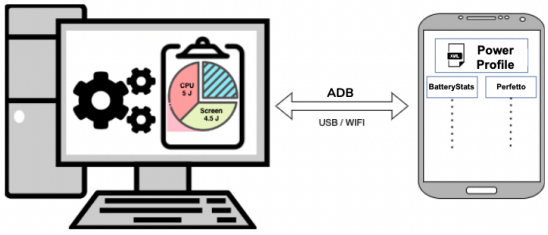


**Figure 1: *E-MANAFA* architecture diagram**

### 3.0.1 *Power Profile*

In order to compute the energy consumption of an application during execution time, *E-MANAFA* uses the *power_profile* [12] XML file, which is present on all devices running an Android platform. This file is provided by the manufacturers and contains an estimate of the energy consumption of each hardware component of the device, in all its possible execution states. The values present in this file are dependent on the platform and can vary significantly. This file is located inside the APK */system/framework/framework-res.apk*, being extracted by *E-MANAFA* before each execution with the tool *apktool* [2]. Although most manufacturers provide accurate values for all possible states of the hardware components, Google also provides a guide to derive these values.

### 3.0.2 *Batterystats*

*BatteryStats* is a tool that collects information related to the device's battery, organizes it in the form of logs and divides it into sections, the main one being the Battery History. This service is running on every device after Android 5.0 and is automatically flushed after every device's full charge.

The Battery History section shows a timeline of everything that has happened on the device since the last moment the service was restarted. It records the occurrence of all types of events relevant to the analysis of energy consumption, such as changes in components' state (for example, screen or wi-fi), applications executing in

the foreground and background, etc. *E-MANAFA* uses this tool to determine the state of each hardware component at a given instant of time during the monitoring session.

#### 3.0.3 *Perfetto*

*Perfetto* is an open-source stack for performance analysis of different platforms and it can be used to trace or profile events from several data sources, such as CPU, battery or memory-related events. *Perfetto* on Android is based on platform services that are available since Android 9 (P). *E-MANAFA* uses *Perfetto* to trace CPU frequency scaling events, since this tool can record an event every time the in-kernel *cpufreq* scaling driver changes the frequency. On most Android devices the frequency scaling is per cluster (group of cores), being common to see groups of CPUs changing the frequency at the same time. Also, many state-of-the-art devices have heterogeneous CPUs, having clusters of CPUs suitable for different processing tasks that work at different frequencies. *Perfetto* is able to detect individual CPU frequency change events per core with a microsecond granularity.

#### 3.0.4 *Logcat and app instrumentation*

*Logcat* is a command line that is able to fetch logs of system messages. *E-MANAFA* uses this tool to inspect the logs registered during a profiling session in order to detect method traces of executed apps.*E-MANAFA* inspects the device logs, searching for messages that are emitted with specific patterns in order to identify app traces. In order to record app traces, we developed a library that, at compile-time, injects trace calls on methods of the app's source code [1], using an Aspect-Oriented [15] approach applied on Android native projects.

*PyAnaDroid* [26] already automated the process of injecting the library in Android Projects in order to obtain instrumented APKs of apps that can be analyzed with *E-MANAFA*. Log inspection can be enabled/disabled in *E-MANAFA*, according to the type of the profiling session (app-level or system-level profiling).

#### 3.0.5 *E-MANAFA workflow*

*E-MANAFA* integrates three main services that we described previously: *batterystats*, *perfetto* and *logcat*. The energy monitoring process can be summarized as the process of managing these 3 components and consulting the power profile file. The total consumption of the session and the consumption per component is calculated based on the information collected from these artifacts and is computed according to the following equations 1 and 2.

$$P(c,t) = I(S(c,t)) * V(t) \tag{1}$$

$$J(\Delta t) = \sum_{t=0}^{t_n} \sum_{c \,\epsilon\, C} P(c,t_n) * (t_{n+1} - t_n),\ t_n \,\epsilon\, \Delta t \tag{2}$$

Equation 1 computes the power consumed by a device component $c$ in a given timestamp $t$. In this equation, the instantaneous voltage $V(t)$ is calculated at the system level and its value at a given time instant can be given by the data obtained with *batterystats*. $S(c,t)$ consists in determining the state of the component $c$ at the instant $t$, which can be inferred by analyzing the *batterystats* logs or, in the case of the CPU, by inspecting the *perfetto* logs to determine the CPU frequency at instant $t$. Finally, the electrical current values I, in Amperes, are estimated from the data in *power_profile.xml* since this file contains the current consumed by each component at each state. The exception is the CPU component, in which case the file contains current values that are consumed at different frequencies. If the CPU frequency at instant $t$ is not specified in this file, we estimate its current consumption value via linear interpolation.

Equation 2 estimates the energy consumed by a device, in Joules (J), in a given time interval $\Delta t$. This value is obtained by adding the consumption obtained for each time interval belonging to $\Delta t$ between which the device components maintained their constant state. For each of these intervals $[t_n, t_n{+}1]$, we calculate the product of the power $P(c,t)$ by the duration $t_n + 1 - t_n$, which corresponds to the calculation of the energy consumed by each component at a given time frame in $\Delta t$ .

## 4 ENERGY PROFILING WITH *E-MANAFA*

*E-MANAFA* tool is already being used to monitor, analyze and optimize the energy consumption of Android applications, namely in identifying energy hot-spots in Android applications and energy-greedy libraries. The tool produces results in the form of JSON files, that can also be interpreted in an interactive and user-friendly graphical form developed for such purpose. Figure 2 shows the result of such results inspector, where the pie chart shows the overall energy consumption of device components and in the (bottom) methods section, the energy consumption for each app method invoked is presented.

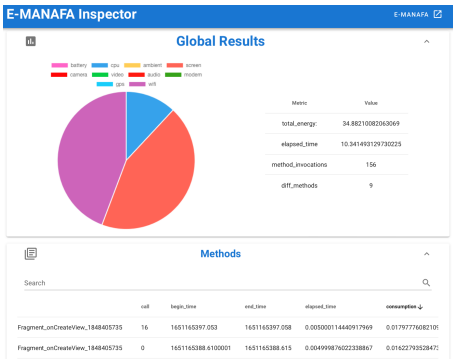


**Figure 2: *E-MANAFA* Results Inspector**

In order to demonstrate the granularity level of energy consumption measurements that *E-MANAFA* obtains, we present results regarding the execution of two versions of the same app: the original application and its instrumented version using the aspect-oriented, compile-time approach. Such approach instruments app methods with calls to *Log* class methods containing the method id to trace method calls. This analysis allows us to present the capability of *E-MANAFA* to detect the energetic impact of minor changes performed in the functional behavior of Android applications, more specifically in detecting differences in terms of CPU energy consumption between 2 similar versions. For this purpose, we used a sample app containing only 50 methods in its source code.

In order to ensure that the results are consistent and independent of the tests performed, each version was executed 25 times with 2

different testing frameworks: Monkey [11] and Droidbot [18]. The test parameters were manually calibrated in order to generate tests with similar duration (approx. 75 seconds). Figure 3 contains the distribution of the values obtained for runtime and CPU energy for each of the versions of the application and execution of the framework. The runtime values are presented in seconds (left diagram), while the energy values are presented in Joules (right diagram).

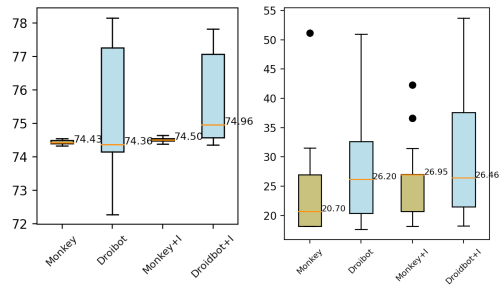


**Figure 3: Comparison between non-instrumented and instrumented (+I) app versions in terms of runtime and CPU energy consumption, respectively**

As can be seen in Figure 3, the instrumented version consistently produced higher energy values than the original version, when comparing the median values (the horizontal lines inside the boxes), top whiskers, and first percentile. Comparing the medians, the instrumented version consumed 34.3% more CPU energy when executed with Monkey and 1% with Droidbot, although the runtime values of the tests are very approximate and consistent and the top whiskers and first percentile of the energy plot are similar. Furthermore, we can conclude that *E-MANAFA* is able to detect the instrumentation overhead since the instrumented tests consumed more energy than the non-instrumented versions.

## 5 CONCLUSIONS

This article introduces the *E-MANAFA* tool, an open-source, plug-and-play profiler capable of providing fine-grained energy measurement on Android devices. In order to present its main features, a comparison with other state-of-the-art tools and some illustrative results were presented.

*E-MANAFA* is the first energy monitoring tool for the Android ecosystem that can provide quantitative estimates of the energy consumed by each device component. It can be easily installed as a Python package or used as a 3rd party library or even via command-line. It can be combined with instrumentation techniques to perform method-level consumption estimates of Android applications.

*E-MANAFA* was developed for helping developers diagnose energy hotspots of their applications and for researchers aiming to carry out empirical studies regarding energy performance. To encourage its adoption, the tool was integrated into the *PyAnaDroid* pipeline in order to benchmark Android apps' performance.

## REFERENCES

[1] 2022. *E-Manafa Hunter-Debug.* shorturl.at/cemX9
[2] APKtool. 2022. *APKtool - A tool for reverse engineering APK files.* shorturl.at/efZ67
[3] Fares Bouaffar, Olivier Le Goaer, and Adel Noureddine. 2021. PowDroid: Energy Profiling of Android Applications. In *2nd International Workshop on Sustainable Software Engineering (SUSTAINSE).* Melbourne, Australia.
[4] Shaiful Chowdhury, Stephanie Borle, Stephen Romansky, and Abram Hindle. 2019. GreenScaler: Training Software Energy Models with Automatic Test Generation. *Empirical Softw. Engg.* 24, 4 (Aug. 2019), 1649–1692. https://doi.org/10.1007/s10664-018-9640-7
[5] Marco Couto, Jácome Cunha, Joao Fernandes, Rui Pereira, and Joao Saraiva. 2015. GreenDroid: A tool for analysing power consumption in the android ecosystem. In *2015 IEEE 13th International Scientific Conference on Informatics.* IEEE, 73–78.
[6] L. Cruz and R. Abreu. 2017. Performance-Based Guidelines for Energy Efficient Mobile Applications. In *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft).* 46–57. https://doi.org/10.1109/MOBILESoft.2017.19
[7] E-MANAFA. 2022. *Energy Monitor and ANAlyzer For Android.* shorturl.at/ewY68
[8] Google. 2016. *Trepn Profiler Android app.* shorturl.at/bioT7 Last visit: 2021-02-10.
[9] Google. 2021. *Android Profiler.* shorturl.at/KVY05
[10] Google. 2021. *Profile battery usage with Batterystats.* shorturl.at/MVWX7
[11] Google. 2021. *UI/Application Exerciser Monkey.* shorturl.at/KW123
[12] Google. 2022. *Measuring Power Values.* shorturl.at/GPQT1
[13] Google. 2022. *Perfetto: System profiling and trace analysis.* https://perfetto.dev
[14] LTD. HARDKERNEL CO. 2022. *ODroid.* https://www.hardkernel.com
[15] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. 1997. Aspect-oriented programming. In *ECOOP'97 — Object-Oriented Programming*, Mehmet Akşit and Satoshi Matsuoka (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 220–242.
[16] Xueliang Li, Junyang Chen, Yepang Liu, Kaishun Wu, and John Gallagher. 2022. Combatting Energy Issues for Mobile Applications. *ACM Transactions on Software Engineering and Methodology* (04 2022). https://doi.org/10.1145/3527851
[17] Xueliang Li, Yuming Yang, Yepang Liu, John P. Gallagher, and Kaishun Wu. 2020. Detecting and Diagnosing Energy Issues for Mobile Applications. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Virtual Event, USA) *(ISSTA 2020).* Association for Computing Machinery, New York, NY, USA, 115–127. https://doi.org/10.1145/3395363.3397350
[18] Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen. 2017. DroidBot: a lightweight UI-Guided test input generator for android. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C).* 23–26.
[19] Mario Linares-Vásquez, Gabriele Bavota, Carlos Bernal-Cárdenas, Rocco Oliveto, Massimiliano Di Penta, and Denys Poshyvanyk. 2014. Mining Energy-greedy API Usage Patterns in Android Apps: An Empirical Study. In *Proceedings of the 11th Working Conference on Mining Software Repositories* (Hyderabad, India) *(MSR 2014).* ACM, New York, NY, USA, 2–11. https://doi.org/10.1145/2597073.2597085
[20] Monsoon. 2021. *Monsoon Power Monitor.* https://www.msoon.com
[21] D. Di Nucci, F. Palomba, A. Prota, A. Panichella, A. Zaidman, and A. De Lucia. 2017. PETrA: A Software-Based Tool for Estimating the Energy Profile of Android Applications. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C).* 3–6. https://doi.org/10.1109/ICSE-C.2017.18
[22] Rui Pereira, Marco Couto, Jácome Cunha, Joao Fernandes, and Joao Saraiva. 2016. The influence of the java collection framework on overall energy consumption. In *2016 IEEE/ACM 5th International Workshop on Green and Sustainable Software (GREENS).* IEEE, 15–21.
[23] Gustavo Pinto and Fernando Castor. 2017. Energy efficiency: a new concern for application software developers. *Commun. ACM* 60, 12 (2017), 68–75.
[24] Android Police. 2021. *Google's latest Android version distribution numbers show 11 in dead heat with 10.* shorturl.at/fiwz9
[25] Rui Rua, Marco Couto, Adriano Pinto, Jácome Cunha, and João Saraiva. 2019. Towards using Memoization for Saving Energy in Android. In *Proceedings of the XXII Iberoamerican Conference on Software Engineering, CIbSE 2019, La Habana, Cuba, April 22-26, 2019.* Curran Associates, 279–292.
[26] R. Rua, M. Couto, and J. Saraiva. 2019. GreenSource: A Large-Scale Collection of Android Code, Tests and Energy Metrics. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR).* 176–180.
[27] Rui Rua, Tiago Fraga, Marco Couto, and João Saraiva. 2020. Greenspecting Android Virtual Keyboards. In *Proceedings of the IEEE/ACM 7th International Conference on Mobile Software Engineering and Systems* (Seoul, Republic of Korea) *(MOBILESoft '20).* Association for Computing Machinery, New York, NY, USA, 98–108. https://doi.org/10.1145/3387905.3388600
[28] Benjamin Westfield and Anandha Gopalan. 2016. Orka: A new technique to profile the energy usage of Android applications. In *2016 5th International Conference on Smart Cities and Green ICT Systems (SMARTGREENS).* 1–12.
[29] L. Zhang, C. Stover, A. Lins, C. Buckley, and P. Mohapatra. 2014. Characterizing Mobile Open APIs in smartphone apps. In *2014 IFIP Networking Conference.* 1–9. https://doi.org/10.1109/IFIPNetworking.2014.6857130
[30] Lide Zhang, Birjodh Tiwana, Robert P. Dick, Zhiyun Qian, Z. Morley Mao, Zhaoguang Wang, and Lei Yang. 2010. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS).* 105–114.