PowDroid: Energy Profiling of Android Applications

Fares Bouaffar, Olivier Le Goaer, Adel Noureddine

PowDroid: Energy Profiling of Android Applications

Fares Bouaffar*, Olivier Le Goaer†, and Adel Noureddine‡
Universite de Pau et des Pays de l'Adour, E2S UPPA, LIUPPA
Anglet, France
*fares.bouaffar@univ-pau.fr, †olivier.legoaer@univ-pau.fr, ‡adel.noureddine@univ-pau.fr

***Abstract*—While the energy efficiency of mobile apps is receiving considerable attention in recent years, Android developers have little tools to assess the energy footprint of their applications. In this paper, we introduce PowDroid, our tool to estimate the energy consumption of Android application. It uses system-wide metrics and does not require access to applications' source code. We run PowDroid on a use-case scenario comparing the energy footprint of applications in different categories.**

***Index Terms*—Energy consumption, Android, Battery drain, Estimation, Tool**

I. Introduction

Android is the most popular operating system in the world, with over 2.5 billion active users spanning over 190 countries. With millions of apps available on the Play Store, energy efficiency is becoming as important a quality attribute as security. Android developers have become aware of both the negative impact on the planet and on the reviews the end-user leave on the store to blame the battery drain. Indeed, battery-limited devices powered by Android like smartPhones, tablets, smartWatches are particularly concerned by energy-heavy hardware components like the screen, GPS, Wi-Fi and so on. But as early noticed in [1], developers have little or no tools to address this hot concern.

In the Android development field, developers are still not able to compare their app with competing apps nor conduct studies of energy consumption at large. On one side, hardware-based solutions require some electrical skills to avoid to damage the device, are hardly scalable, and come at a great cost. On the other side, software-based solutions are often untrustworthy and quickly deprecated as the Android platform evolves. The only tool officially supported is the Energy Profiler[1] integrated within Android Studio IDE. However, it requires access to the source code (white-box measurement) and does not provide runtime energy consumption but rather displays an overview of the energy levels (*e.g.*, light, medium, etc.) of hardware components over the usage timeline of the application. In addition, no export functionality is available, which could have allowed for analysis and decision-making. To our knowledge, no research paper has still used this tool despite its potentially large audience.

To fill this gap, this paper introduces a simple command-line tool to evaluate the amount of energy consumed in joules by any application (black-box measurement) run in isolation in a given time interval. It can be used to perform energy benchmarks and to analyze which components are the most influential on the battery drain. This tool should allow researchers and engineers to get a better understanding of energy consumption in real-life Android apps.

The rest of this paper is organized as following: Section II lists the hardware and software solutions found in the literature. Section III describes how our tool works and how a developer can use it. Some results obtained with PowDroid are shown in Section IV before we conclude.

[1]https://developer.android.com/studio/profile/energy-profiler

II. Related works

There are some existing approaches which directly address the energy measurement of android apps, involving recurring elements.

A. Hardware-based approaches

In [2] , the authors used the Monsoon power monitor by Monsoon Solutions Inc. in order to measure energy (in terms of Voltage and Intensity) with the special aim of discovering which API are the most consuming in an android application. In his thesis [3], the authors used the Otii arc by QOITECH for measuring the components of the Open Source Smartwatch PineTime. Their work is probably transposable to an Android smartWatch, and to a lesser extent, to Android smartPhones. The Yocto-amp by Yoctupuce was encountered in two academic works. A master thesis [4] aimed at building a power model based on the gathered measurements. The implementation targeted Android-based platform. A PhD. thesis [5] aimed at evaluating and correcting code source defaults related to energy consumption in android applications.

B. Software-based approaches

To measure any Android app, Di Nucci et al. [6] introduced their own tool called PETrA, relying on raw data from Batterystats to compute the energy consumption of each method call. Huber et al. [7] also used Batterystats but to study energy consumption of PWAs (Progressive Web Apps) on two Android devices with four execution scenarios. In 2015, Qualcomm released Trepn Profiler, an Android app to analyze which installed apps consume resources. It was used in [8] to analyze and predict the energy consumption of any android applications. It was also used by Ahmad et al.

[9] in conjunction with the PowerTutor app [10] to evaluate the performance of dynamic analysis-based energy estimation. PowerTutor was used alone in the study of Saipullah et al. [11] to measure the energy consumption for image processing on Android smartphone.
In [12], the authors developed an application energy metering for Android smartphone called AppScope. The latter uses hardware power models and usage statistics for each hardware component. Finally, they use the Monsoon power monitor to verify the relevance of their results.

III. PowDroid

In this section, we describe our tool called *PowDroid*, a software-based approach for measuring and profiling the energy consumption of android applications.

A. Architecture

The general architecture of our approach is described in figure 1. The key idea is to pull system-wise battery data from any Android-powered device thought the Android Debug Bridge (ADB). Powdroid connects to the device using WiFi instead of a USB connection, in order to avoid having the device falls into a charging state while connected to USB (and thus preventing the proper observation of the battery drain in real time). However, it is still possible to use a USB cable by disabling its charging feature, as in PETrA [6].

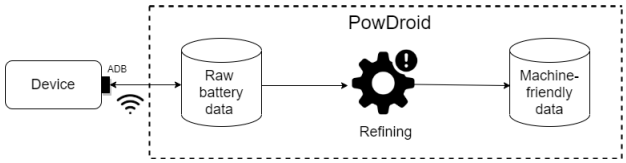

Fig. 1: PowDroid Architecture

The pulled data are produced by the Android system itself and serves as ground proofs in our approach. Hence, we do not introduce over-estimations than those already calculated natively by the Android platform. Instead, we just refine the raw data through a specific workflow to spit out a machine-friendly output. The output contains joules, of course, but also hardware components status along a fine-grained time sampling. We think that the resulting set of metrics is valuable to perform a large panel of analysis and statistics.

B. Components

Our architecture uses multiple components to collect and process metrics and calculates energy consumption.

1) Batterystats: is a tool included in the Android framework that collects raw battery data on a device. The corresponding ADB command is `adb shell batterystats`. To avoid unnecessary accumulation of data between test sessions, it is wise to clean the history with the command `adb shell batterystats reset`. At this stage, the data are located on the device in a file named `data/local/tmp/battery.txt`.

2) Bugreport: is also a tool included in the Android framework which generates a report file in a ZIP format on the device from the aforementioned `battery.txt`. The corresponding command is `adb bugreport [filename].zip`. A bugreport file is a rich format compatible with Battery Historian.

3) Battery Historian: Google's Battery Historian[2] converts the report from Bugreport into a pretty web-based visualization that can be viewed in a web browser. We reuse the Go script `local_history _parse.go` to convert a .zip file to a human-readable CSV file. We use the `--summary=totalTime` argument to produce results based on time, rather than the evolution of the battery charge.

4) Processing energy scripts: We write several python scripts to implement the workflow: *PowDroid.py* is the main program; *split.py* cuts a bulk CSV file along its different metrics, and *Merge.py* achieves the union of metrics in sync with the time windows.

C. Energy Metrics

Each time that the battery evolve or that the status of hardware components change, an entry is written in the bugreport file. This is called an energy-related event, delimited by a start time and an end time (of type Timestamp - TS). Thus, one can find in information about the states of voltage, Coulomb charge, top application packages, and so on.

We start with the calculation of the duration of a state (T_Δ), thanks to the start time and the end time:

$$T_{\Delta(event)}(ms) = endtime_{event}(TS) - starttime_{event}(TS)$$

The energy consumed is not the same for every application execution as it varies depending on different factors, such as the hardware components activated by the application, the complexity of the method call, the active wakelocks and so on. These factors influence the energy consumption, and can be sorted chronologically.

In addition to the time windows, we calculate the intensity of the current from the difference between two remaining charges (C_Δ) and the duration of the smartphone charge drain (T_w), by using this formula:

$$Intensity(mA) = \frac{C_\Delta(mAh)}{T_{w(phonecharge)}(hr)}.$$

With intensity, voltage and duration of event, we can calculate the power and energy consumption of our device for the monitored event. First, we calculate the power with:

$$Power(Watt) = Voltage(Volt) * Intensity(Amp).$$

Secondly, we calculate the energy consumption for each event in Joules:

$$Energy_{(event)}(Joule) = Power(Watt) * T_{\Delta(event)}(Second)$$

[2]https://github.com/google/battery-historian

We are then able to calculate the energy consumption of the application by adding every energy consumption of all events:

$$Energy_{(app)}(Joule) = \sum_{i=0}^{n} Energy_{(event_i)}(Joule)$$

In summary, our tool will generate one CSV file, crossing all events, their time windows and energy consumption. The final structure of our file is presented in Table I.

Metric	Description	Unit
Start time	when the event start	Timestamp
End time	when the event ends	Timestamp
Duration	duration of the event	Millisecond
Voltage	electric voltage emitted by the battery.	Millivolt
Remaining charge	electric charge remaining in the battery	Milliampere-hour
Intensity	electric intensity calculated from remaining charge	Milliampere
Power	amount of energy during a given time, usually 1 second.	Watt
Consumed charge	amount of charge passing through the cross-section of smartphone	Milliampere-hour
Energy	total energy consumed when the application was run.	Joule
Top app	name of the package running in the foreground.	String
Wakelock	wakelocks acquired by the application.	String
Screen	indicates if the screen is active or not.	Boolean
GPS	indicates if GPS is on or not	Boolean
Mobile radio	indicates if mobile GSM is active for scanning or transmitting data	Boolean
Wifi on	indicates the status of wifi (On/Off)	Boolean
Wifi radio	indicates if the wifi is transferring data	Boolean
Camera	indicates if the camera is active or not	Boolean
Video	indicates if there is a video like video reading or a video call	Boolean
Audio	indicates if the audio component (like speakers) is active	Boolean

TABLE I: Metrics of the Output CSV file

IV. Evaluation

In order to evaluate PowDroid, we decided to compare the energy consumption of several Android apps from three categories: web browsers, camera, and weather applications. We chose these applications in order to get a wide spectrum of use cases using various hardware components (WiFi, GPS, Camera, CPU, GPU, etc.).

A. Experimental Setup

To conduct our experiments, we use a *Google Pixel 3a* device in debug mode. Accurate measures need to run an app in the maximum isolation as possible: First, we remove the SIM card to prevent from accidental phone calls and messages, and to avoid the continuous impacts of cellular network connectivity. Then,we disable running background services as Google Play Store and closed all forefront applications. Finally, we disable the Battery Saver Mode.

We then set up the *config.json*, the configuration file of PowDroid that contains all the directories paths where the generated files will be saved. The smartphone is then connected on the same WiFi network as our personal computer. Notice that the first time a device is plugged, a brief USB connection is required to get the INET address of the WLAN interface.

Once our smartphone is properly connected, we drag and drop the apk file to the PowDroid shell: the app is automatically installed and launched on the device. Then, the user drives the test session. PowDroid simply asks when the user is ready to test the application, until the user decides to stop the recording. Then PowDroid generates a CSV file containing all the metrics and data as described in Section III.

B. Experimental Scenarios

We conduct our experiments on three software categories: web browsers, camera and weather applications. For each category, we define an experimental but realistic scenario that is conducted for all applications. We have installed the latest versions of the applications available at the time of writing this paper.

For each application, we run them for 4 minutes, with the battery level at 50% at the start of the measurement (we recharge back to 50% before running the next experiment), screen brightness at 50%, and audio at 50%. We run each scenario three times for each application and average the results in joules. For web browsers, the scenario includes typing a URL, searching with a keyword, and selecting and watching a link from the search results, and repeat all for 3 different websites. For camera applications, we take pictures from both front and back cameras in normal, panoramic and HDR modes, and record a 10-second video clip. And for weather applications, we locate our position, get weather information for today, check monthly view and another day, and view the radar map.

C. Experimental Results

Table II outlines the energy consumption of all our tested applications and domains: web browsers, camera and weather applications. Overall, our results are consistent with recent comparative studies (such as for web browsers [3]).

For web browsers, our experiments found that Brave is the most energy efficient, while Firefox was the worst with a 33.8% increase in energy consumption (a increase of 108.12 joules for the same workload). However, this can be partially attributed to the default built-in adblock feature in Brave, which also blocks YouTube and video ads. However, Samsung Internet Browser achieved the second place in energy

[3]https://greenspector.com/en/what-are-the-best-web-browsers-to-use-in-2020/

TABLE II: Energy consumption of our different application domains

(a) Web browsers

Application	Energy (Joule)
Mozilla Firefox	427,78
Google Chrome	404,27
Microsoft Edge	386,18
Opera	368,33
Samsung Browser	345,93
Brave	319,66

(b) Camera applications

Application	Energy (Joule)
Open Camera	711,35
Google camera	683,45
Samsung S10 camera	661,92
ProCam X	660,54
Manual camera DSLR Pro	610,54

(c) Weather applications

Application	Energy (Joule)
AccuWeather	441,12
The weather channel	398,92
Weather Forecast	318,82
Advanced Weather	236,92

efficiency without blocking ads. Google Chrome and Mozilla Firefox were within a slim margin of 5.8% in energy consumption.

For camera applications, we also observe an energy increase of 16.5% (100.81 joules) between the most efficient of our experiments (Manual camera DSLR Pro) and the least (Open Camera).

Finally, we tested 4 weather application with even bigger variation: an increase of 86.19% (204.2 joules) between Advanced Weather (the most efficient), and AccuWeather (the least efficient). This can be explained by the presence of ads in all applications, except Advanced Weather which does not contain any ads.

These results for ad-supported applications, for the different categories, are consistent with state-of-the art studies which show that ads are known to have a huge impact on energy consumption in mobile applications [13].

D. Limitations

Although our tool allows energy profiling of Android applications in an easy and automated approach, it exhibits a few limitations.

First, our tool monitors the energy consumption of the entire Android smartphone as reported by *Batterystats*, and does not monitor individual applications directly. This can be alleviated by isolating and running a single application during profiling. However, this is not a fail-proof approach as some Android background services still run and impacts our profiling.

Second, we rely on the hardware and battery information provided by Android through *Batterystats*, which does not offer a real time and fine-grained estimations. Energy metrics are based on battery drain, and the energy consumption of individual hardware components are not mapped in our approach.

V. Conclusion

We presented, in this paper, PowDroid, our automated Android energy profiling approach and command-line tool. We also presented a preliminary study comparing the energy consumption of multiple applications in different categories across a common workload. Our findings are consistent with recent studies.

For future works, we aim to study the influence of each hardware component (*i.e.*, energy of WiFi, GPS, CPU, etc.) on the global energy consumption, and being able to isolate the energy consumption of individual applications automatically during runtime.

References

[1] A. Hindle, "Green software engineering: The curse of methodology," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 5, 2016, pp. 46–55.

[2] 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*. New York, NY, USA: ACM, 2014, p. 2–11.

[3] L. Berglund, "What is draining the battery on the pinetime smartwatch?" p. 23, 2020.

[4] C. Petropoulos, "Power measurement infrastructures and power analysis of an android based smartphone," 2016.

[5] M. A. Ait Younes, "Évaluation et correction des défauts de code liés à la consommation d'énergie dans les applications mobiles android," 2017.

[6] D. Di Nucci, F. Palomba, A. Prota, A. Panichella, A. Zaidman, and A. De Lucia, "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)*, 2017, pp. 3–6.

[7] S. Huber, L. Demetz, and M. Felderer, "Pwa vs the others: A comparative study on the ui energy-efficiency of progressive web apps," in *Web Engineering*, M. Brambilla, R. Chbeir, F. Frasincar, and I. Manolescu, Eds. Cham: Springer International Publishing, 2021, pp. 464–479.

[8] Y. Hu, J. Yan, D. Yan, Q. Lu, and J. Yan, "Lightweight energy consumption analysis and prediction for android applications," *Science of Computer Programming*, vol. 162, pp. 132–147, 2018, special Issue on TASE 2016.

[9] R. W. Ahmad, S. H. A. Hamid, A. Gani, M. S. Obaidat, J. Shuja, F. Rehman, and A. U. R. Khan, "Performance assessment of dynamic analysis based energy estimation tools," in *2018 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, 2018, pp. 1–12.

[10] 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*. New York, NY, USA: ACM, 2010, p. 105–114.

[11] K. M. Saipullah, A. Anuar, N. A. Ismail, and Y. Soo, "Measuring power consumption for image processing on android smartphone," *American Journal of Applied Sciences*, vol. 9, pp. 2052–2057, 2012.

[12] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha, "Appscope: Application energy metering framework for android smartphone using kernel activity monitoring," in *2012 USENIX Annual Technical Conference (USENIX ATC 12)*. Boston, MA: USENIX Association, Jun. 2012, pp. 387–400.

[13] J. Gui, S. Mcilroy, M. Nagappan, and W. G. J. Halfond, "Truth in advertising: The hidden cost of mobile ads for software developers," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1, 2015, pp. 100–110.