Energy Profiling as a Service

Claas Wilke, Sebastian Richly, Sebastian Götz, and Uwe Aßmann

Fakultät Informatik, Institut für Software- und Multimediatechnik
Lehrstuhl Softwaretechnologie, Technische Universität Dresden
D-01062 Dresden
{claas.wilke, sebastian.richly, uwe.assmann}@tu-dresden.de, sebastian.goetz@acm.org

Abstract: Energy optimization of software systems has emerged as one of the biggest challenges for the development of future software applications. Especially the efficiency of mobile applications has become a major driver, as limited battery capacities force applications to cause as few energy consumption as possible to increase the devices' uptimes. Although many research groups have investigated approaches to energy optimization of mobile applications, still, each group is building and using its own methodology to evaluate the impact of their optimizations on the devices' energy consumption. This variety of different evaluation methodologies hinders the reproducibility of individual research group's findings as well as their comparability with similar research results. In this paper, we propose profiling as a service by exposing a hardware-based energy profiling infrastructure as a web service reusable for other research groups and application developers. We discuss possible use cases for our profiling service as well as limitations. Besides, we present its transparent integration into software development tools such as Eclipse.

1 Introduction

Within the last years, energy consumption emerged to be one of the central challenges for future software development. With the introduction of mobile devices such as smartphones and tablets, energy consumption became an omnipresent problem for mobile application users, and thus, also for mobile application developers. As existing studies show, the energy optimization of mobile applications can help to increase the uptime of mobile devices [PHZ12, VRSF+12], and hence, also the satisfaction of their users [WRG+13a].

Many research groups are investigating approaches to energy optimization of mobile devices [PHZ12, GJJW12, SH12, BS13]. For proper evaluation, each approach requires a framework for energy profiling of mobile devices. In consequence, almost every group has created its own methodology to profile the energy consumption of mobile devices [HSB12, WGR13, JSW13]. However, building profiling infrastructures for mobile devices and measuring their energy consumption in a comparable manner is still a challenge for many research groups. Whereas some groups are reusing existing software-based profiling approaches as the PowerTutor tool [ZTQ+10, BS13], others focus on developing their own software-based profilers [JSW13]. Again, other groups set up their own hardware-based energy profiling infrastructures [WGR13].

As the breakout discussion at the EASED@BUIS workshop 2013 showed [BGNW13], this lack of common approaches to energy profiling hinders the research groups in gaining representative, comparable measurements results for their work. The envisioned goal are reproducible measurement results, extensible by other groups focusing on similar problems around the energy optimization of mobile devices.

Besides the problem of various existing profiling approaches that hinder comparability, setting up an own energy profiling infrastructure is a resource intensive task, both in terms of development time and money, as measurement hardware is likely to cost several thousand euros, an investment not achievable by all research groups those budgets are commonly limited and inflexible.

Apart from research groups focusing on the energy optimization of mobile applications, first industrial application developers are already trying to build energy-efficient mobile applications. For them the task of energy profiling is even more challenging, as they typically have even more limited resources to design and set up their own profiling infrastructures and are thus, often forces to optimize their application without any energy profiling.

To address this lack of available profiling infrastructures and, in consequence, the lack of comparable profiling results, we propose the introduction of *energy profiling as a service*: a web service providing a publicly available profiling infrastructure at low cost. For re-executable and reproducible energy profiling, we introduce a methodology for unit-test-driven energy profiling of mobile applications, based on our JouleUnit framework [WGR13]. Besides, we present a prototypical realization of profiling as a service, encapsulating a hardware-based profiling infrastructure, which we set up in our research lab. The profiling service is packaged with integrated development environment (IDE)-based tooling that allows for an easy and transparent integration of profiling as a service into existing development processes for mobile applications. We highlight possible usage scenarios for profiling as a service and discuss limitations and advantages of our solution. This is—to the best of our knowledge—the first approach to provide a hardware-based energy profiling infrastructure for a large research community, which allows for comparable and reproducible energy profiling results of mobile applications.

The remainder of this paper is structured as follows: in Sect. 2 we present our unit test-driven JouleUnit approach for mobile application energy profiling. Following, in Sect. 3 we explain how we encapsulated our profiling infrastructure behind a web service. Afterwards, Sect. 4 illustrates the service's integration into the Eclipse IDE. Sect. 5 discusses different use cases realizable with profiling as a service and Sect. 6 outlines some limitations for use cases and profiling scenarios being not addressable by our profiling infrastructure. Finally, Sect. 7 concludes this paper.

2 Energy Profiling Based on Unit Testing

To allow the eased execution of energy profiling for mobile applications, we propose to base energy profiling on unit testing as outlined in [WGR13]. The use cases (or test sce-

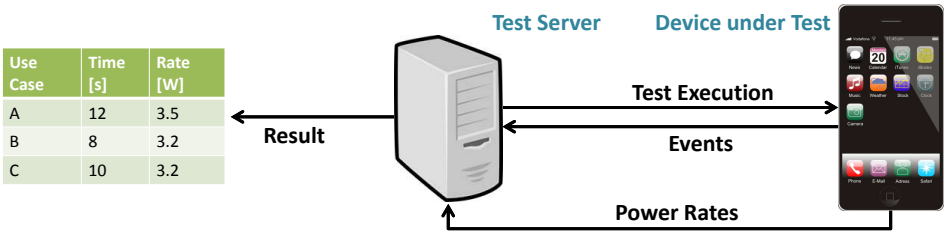

Figure 1: Energy profiling based on unit testing.

narios) to be profiled are expressed in the form of executable unit tests and are deployed and executed on a real mobile device from a test PC (cf. Fig. 1). Each test case comprises a set of user interface (UI) commands to be executed on the device under test (DUT) (e.g., pressing buttons and entering texts). During their execution, timestamps indicating the begin and termination of the individual test cases are logged and the energy consumption of the DUT is profiled in parallel by bypassing its power supply through an external power metering device. After the test run, the logged timestamps of the test cases are used to correlate them with the profiled power rates and, thus, to compute their energy consumption.

Although this solution allows for coarse-grained energy profiling by measuring the device's energy consumption as a whole only, the approach proved to be sufficient to evaluate energy consumption differences in the same use cases for similar mobile applications as well as the impact caused by source code modifications such as insertion of advertisement banners [WRG+13b].

A major advantage of the reuse of unit testing for energy profiling is that the learning curve for mobile application developers is rather low. They can reuse their existing tooling for mobile application testing to retrieve additional information such as the test cases' energy consumption. Besides, they can also reuse existing functional unit tests for energy profiling with the limitation that these test cases represent realistic usage behavior (e.g., w.r.t. the execution time of sequences of UI commands). Furthermore, screen recording tools to create functional unit tests can be reused for the creation of energy test cases. Finally, existing techniques to evaluate the coverage of functional unit tests can be applied to evaluate the coverage for energy testing.

3 Energy Profiling as a Service

Similar to unit testing, energy profiling is likely to result in high execution times, especially when applications shall be profiled under realistic conditions with representative use cases and profiling coverage. Moreover, in contrast to test cases, profiling should be executed multiple times during each run to reduce the impact of measurement outliers and thus, results in more representative profiling results. Unfortunately, these time requirements contradicted with our other research activities. Furthermore, as our experiments started to run longer than a single working day we also wanted to supervise (and probably maintain)

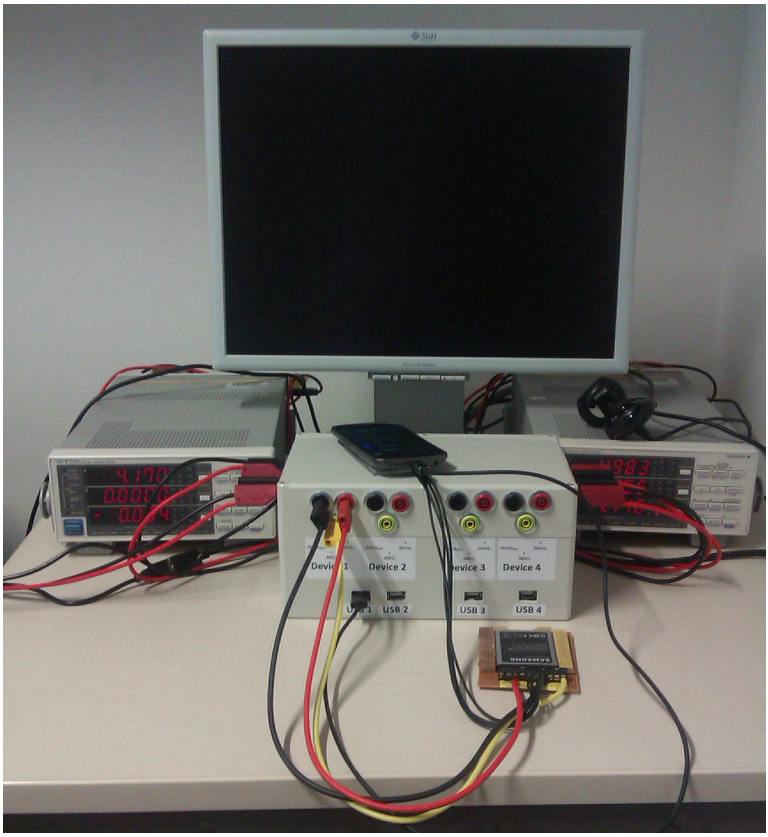

Figure 2: Our profiling server with two power meters connected to one testing device.

our profiling runs from home. Thus, we experienced high pressure to automate unit test-based energy profiling as much as possible. That is why we started the development of an energy profiling service.

We set up a server connected to multiple Android devices and energy measurement hardware bypassing the power supplies of these devices. Fortunately, for Android, the Android Debug Bridge (ADB) [Goo13b] provides tooling to remotely install and uninstall applications, as well as to trigger the execution of test cases. Everything that is required are two files: one file representing the application under test (AUT), and one file representing the executable unit tests, packed as a second application installable on the DUT via the ADB as well. Furthermore, where necessary the ADB allows to remotely configure the DUT via touch and key events (e.g., to enable and disable the display backlight and the WiFi module). We implemented a profiling service that takes the AUT as well as the testing application together with a set of parameters (e.g., how often to profile which test cases) and coordinates the profiling on the DUT fully automated. Afterwards, profiling results are stored within a relational databse that allows the eased persistence, propagation and

inspection of testing results. To further supervise our test runs we set up a web cam that can be used to observe executed profiling runs as well as a Java application that allows to interfere with the DUT and thus the profiling runs remotely by triggering UI commands on the DUT (e.g., to reset an application when accidentally stuck by and unexpected UI event such as a toast message).

The setup of our profiling server allows us to easily trigger and execute profiling jobs remotely. Besides, the profiling service allows us to batch multiple profiling jobs (even for different applications) to be executed on the DUT subsequently and automatically or to profile the same application on multiple DUTs subsequently.

4 Integrating the Profiling Service into Eclipse

To further ease the creation and profiling test cases and their execution, we designed an integration of our profiling service into the Eclipse IDE. We decided for Eclipse, as the existing tooling for Android development and testing provided with the Android Development Tools (ADT) is also Eclipse-based [Goo13a]. Therefore, application developers can reuse the existing tooling to program both their applications and their test cases. Afterwards, they can simply trigger the energy profiling from the IDE and get the results visualized within the IDE; similar as they are used to execute JUnit test cases resulting in a green bar within Eclipse.

Therefore we implemented a new run configuration for Android JUnit tests, where testers can specify on which device and how often (and possibly under which conditions, e.g., display backlight) they want to execute their profiling runs. Afterwards, the AUT as well as the test cases are deployed on the profiling server and the profiling is triggered automatically. Finally, once the profiling runs terminated, the profiling results are visualized in Eclipse in two additional views designed for this purpose (cf. Fig. 3).

The first view (cf. Fig. 3, upper part) allows for the investigation individual test runs in a graphical manner. On a time axis, the test runs are illustrated by colored regions. Besides, the profiled power rate values as well as the hardware utilization for CPU, network and display backlight of the DUT is illustrated as measured values over the profiling time.

The second view (cf. Fig. 3, lower part) provides tabular feedback for all profiled test cases as their duration, their average power rate and energy consumption. Besides, for test cases profiled multiple times, average results and outliers are enlisted as well. Finally, the view provides capabilities to compare the profiling results with older profiling results for the same test cases by indicating decreasing or increasing trends.

By providing an Eclipse-based front-end for our profiling server we allow the easy triggering of profiling runs from a well-known environment. Furthermore, the propagation and graphical presentation of profiling results within Eclipse further increases the transparency of profiling test runs triggered from the IDE and helps to identify bottlenecks or energy-intense operations within the profiled use cases.

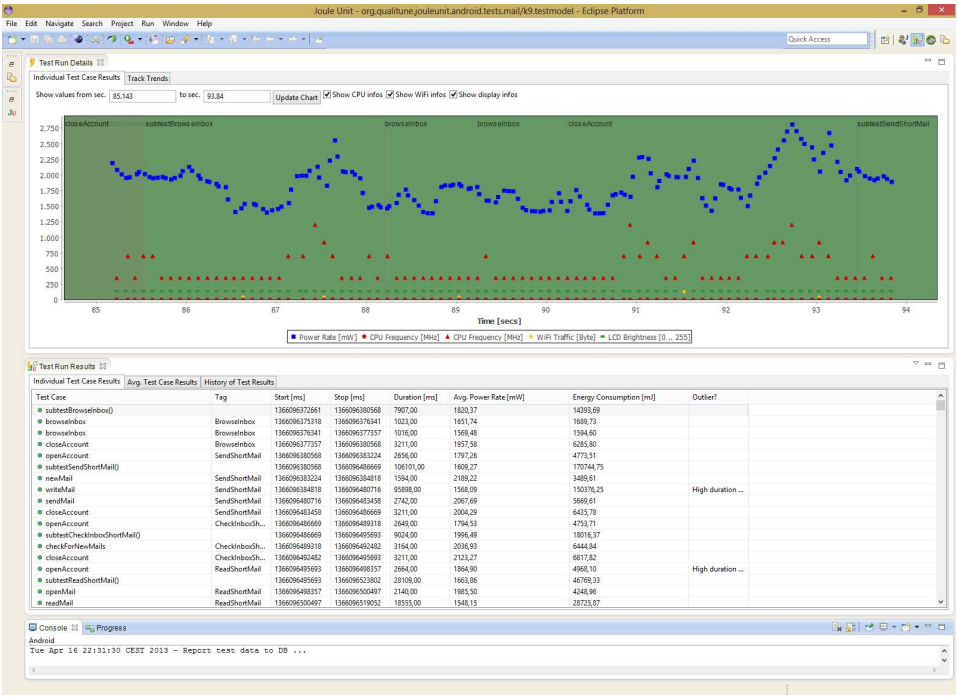

Figure 3: Presentation of profiling results in Eclipse.

5 Use Cases for Energy Profiling as a Service

In this section we discuss possible use cases for energy profiling and testing emerging from our profiling service.

First, research groups can use such a profiling infrastructure for their individual research projects (e.g., to evaluate the impact of design patterns [BS13] on applications' energy consumption, or how much energy refactorings can decrease applications' energy consumption [GJJW12]). The major benefit for such groups is that they can reuse an existing instead of setting up their own profiling infrastructure, allowing them to focus onto their central research questions and challenges. If different groups are using the same profiling services, this leads to more comparable and reproducible results as identified as required at the EASE@BUIS workshop [BGNW13]. Moreover, research groups not able to invest into their own hardware-based profiling infrastructure, are not forced to fully concentrate onto software-based profiling which is known to be less accurate in most cases [HSB12].

Second, besides research groups, application developers can apply the profiling service to evaluate and optimize their applications. They can use the service to test their applications in multiple ways:

1. They can test and profile their applications in an exploratory manner by profiling

the functionalities they expect to be candidates for energy optimization on multiple Android devices with different hardware characteristics conneted to our profiling server. Once they have identified functionalities with unexpected high consumptions, they can aim to optimize and validate their optimizations based on the profiling service. Besides, they can apply coverage criteria to evaluate which functions remain for further profiling and optimization.

2. They can use the profiling infrastructure for debugging. If users report unexpected behavior, they can try to emulate the reported problems with the profiling service and investigate the profiling results to identify the problem and a possible bug fix.

3. Next they can use the profiling service for impact analysis during the introduction of new features. For example they can profile their application with and without integrated advertisement banners, known to be energy-inefficient in most use cases [WGR13]. They can also evaluate the impact on modifications of synchronization mechanisms or simply evaluate the energy consumption while running their applications as background services (e.g., for email clients polling for new messages in the background frequently).

4. Finally, the profiling service can be applied as the back-end for regression testing and continuous integration. Once developers have conducted a representative set of energy test cases for their applications, they can trigger the execution of this test cases after each modification or during a nightly build and thus, can avoid the release of changes to their apps that introduce new energy bugs leading to frustrated users and negative application ratings [WRG+13a].

6 Limitations of our Approach

In this section we discuss limitations of our profiling service w.r.t. possible energy profiling scenarios for mobile devices and illustrate which of these limitations can be addressed.

- Currently, our profiling service is limited to the profiling of Android-operated devices. However our profiling framework JouleUnit that we use for energy profiling could be used for other mobile operating systems (OSs) (e.g., iOS) as well. However, the remainder of the service (application deployment and test case triggering) would have to be reimplemented for other OSs.

- As our profiling service uses external measurement hardware and as our DUTs are wired to our profiling server, the provided energy profiling is limited to in-the-lab tests. Meaning, in-the-field tests (e.g., such as profiling the energy consumption of GPS modules in the context of real usage or energy profiling of the DUT's camera module while taking pictures) are not possible. For such tests we propose to use existing software-based profilers such as PowerTutor [ZTQ+10].

- Currently, our profiling service supports energy profiling at the battery level and thus, for the whole device as once only. Although such profiling can be consid-

ered as rather coarse-grained, it was shown to be sufficient for comparative measures [HSB12]. However, as our profiling infrastructure was designed generically, other devices allowing profiling on the chip (e.g., as specific-purpose hardware as provided by ARM [ARM12, Ch. 12]) or mobile devices modified for profiling on chip as also discussed in [HSB12] could be integrated into the profiling infrastructure.

- Similarly, the currently used Yokogawa WT210 wattmeters allow the energy profiling with a probe frequency of 10 Hz only. But similarly to the profiled devices, other measurement hardware could be integrated into our profiling service, allowing higher probe frequencies if financially affordable.

- Besides limitations to the kinds of devices to be profiled, the probe frequency and the supported use cases, there are also some security issues to be considered. Every software developer should at least raise an eyebrow then someone offers the opportunity to deploy an executable application on a remotely available device running an OS having as much security issues as current Android versions. Executed unit tests under Android are likely to be able modifying files from the file system, uninstalling other applications, downloading files from the web or placing malicious code somewhere on the DUT. Currently, we do not handle such security issues. We simply assume that the testing code to be executed will not do such things. However, for future work these security issues should definitely be addressed and somehow handled for our profiling service.

- Finally, the limited available devices and measurement hardware within our profiling infrastructure cause some limitations w.r.t. the availability of the profiling service for other research and development facilities. However, we argue that using a profiling infrastructure being available for research at specific points in time is at least better than having no profiling infrastructure at all. Batching and scheduling of profiling jobs can help to improve the situation and to utilize the profiling infrastructure at both day and night. However, for how many research groups and application developers the currently existing infrastructure will scale is a task to be evaluated by future work.

7 Conclusion

In this paper we propose to address the lack for hardware-based energy profiling approaches in the domain of mobile application optimization by providing our infrastructure as a profiling service. We documented how we set up our profiling server and how other researchers as well as application developers can use our profiling service and energy profile their applications in a unit test-driven way. We presented additional tooling for the Eclipse IDE that allows for the transparent integration of our profiling service into the IDE.

We invite other research groups in participating and using our profiling infrastructure as well as giving feedback and proposals for improvements of our profiling services. We do

not claim that are profiling service can address all profiling issues existing within the community. However, we think that the service can help by giving research groups the ability to energy-evaluate their applications—an ability they probably would not have without our profiling service.

Acknowledgements

This research has been funded within the project ZESSY #080951806, by the European Social Fund (ESF) and Federal State of Saxony and within the Collaborative Research Center 912 (HAEC), funded by the German Research Foundation (DFG).

References

[ARM12] ARM Holding. ARM® DS-5 – Version 5.9 – Using ARM Streamline, 2012.

[BGNW13] Christian Bunse, Marion Gottschalk, Stefan Naumann, and Andreas Winter. *2nd Workshop on Energy Aware Software-Engineering and Development (EASED@BUIS2013)*. Number 4/2013 in Oldenburg Lecture Notes in Software Engineering. Carl von Ossietzky University, Oldenburg, 2013.

[BS13] Christian Bunse and Sebastian Stiemer. On the Energy Consumption of Design Patterns. In *2nd Workshop on Energy Aware Software-Engineering and Development (EASED@BUIS2013)*, number 4/2013 in Oldenburg Lecture Notes in Software Engineering, pages 7–8, Oldenburg, 2013. Carl von Ossietzky University.

[GJJW12] Marion Gottschalk, Mirco Josefiok, Jan Jelschen, and Andreas Winter. Removing Energy Code Smells with Reengineering Services. In *Proceedings of the First Workshop for the Development of Energy-aware Software (EEbS2012)*, volume 208 of *Lecture Notes in Informatics*, pages 441–455, Bonn, 2012. Gesselschaft für Informatik.

[Goo13a] Google Inc. ADT Plugin. Website, May 2013. http://developer.android.com/tools/sdk/eclipse-adt.html.

[Goo13b] Google Inc. Android Debug Bridge. Website, May 2013. http://developer.android.com/tools/help/adb.html.

[HSB12] Hagen Höpfner, Maximilian Schirmer, and Christian Bunse. On Measuring Smartphones' Software Energy Requirements. In *Proceedings of the 7th International Conference on Software Paradigm Trends (ICSOFT2012)*, pages 165–171, Setétubal, Portugal, 2012. SciTePress.

[JSW13] Mirco Josefiok, Marcel Schröder, and Adreas Winter. An Energy Abstraction Layer for Mobile Computing Devices. In *2nd Workshop on Energy Aware Software-Engineering and Development (EASED@BUIS2013)*, number 4/2013 in Oldenburg Lecture Notes in Software Engineering, pages 17–18, Oldenburg, 2013. Carl von Ossietzky University.

[PHZ12] 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*, pages 29–42, New York, 2012.

[SH12] M. Schirmer and H. Höpfner. Towards Using Location Poly-Hierarchies for Energy-Efficient Continuous Location Determination. In *GI-Workshop on Foundations of Databases*, Bonn, 2012. Gesselschaft für Informatik.

[VRSF+12] N. Vallina-Rodriguez, J. Shah, A. Finamore, Y. Grunenberger, H. Haddadi, and J. Crowcroft. Breaking for Commercials: Characterizing Mobile Advertising. In *Proceedings of the Internet Measurement Conference (IMC2012)*, New York, 2012. ACM.

[WGR13] Claas Wilke, Sebastian Götz, and Sebastian Richly. JouleUnit - A Generic Framework for Software Energy Profiling and Testing. In *Software Engineering Green By Software Engineering Workshop (GIBSE 2013)*, New York, 2013. ACM.

[WRG+13a] Claas Wilke, Sebastian Richly, Sebastian Götz, Christian Piechnick, and Uwe Aßmann. Energy Consumption and Efficiency in Mobile Applications: A User Feedback Study. In *Submitted for GreenCom2013*, 2013.

[WRG+13b] Claas Wilke, Sebastian Richly, Sebastian Götz, Christian Piechnick, Georg Püschel, and Uwe Aßmann. Comparing Mobile Applications Power Consumption. In *Software Engineering Aspects of Green Computing (SEGC) track at the 28th ACM Symposium on Applied Computing (SAC2013)*, New York, 2013. ACM.

[ZTQ+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*, pages 105–114, New York, 2010. ACM.