Energy Labels for Mobile Applications

Claas Wilke, Sebastian Richly, Georg Püschel, Christian Piechnick, Sebastian Götz, and Uwe Aßmann

Fakultät Informatik, Institut für Software- und Multimediatechnik
Lehrstuhl Softwaretechnologie, Technische Universität Dresden
D-01062 Dresden
{claas.wilke, sebastian.richly, georg.pueschel1, christian.piechnick, uwe.assmann}@tu-dresden.de, sebastian.goetz@acm.org

Abstract: In recent years the usage of mobile devices and the expansion of their functionality by installing further applications have become very popular. Their frequent usage causes much faster battery discharging, and thus drastically limits the uptime of the devices and their applications. Hence, investigating and reducing the power consumption of mobile applications is one of the current, central challenges in software engineering. In this paper we propose an approach for profiling the power consumption of mobile applications and comparing their consumption for similar services. We show by example that such differences can be identified for two well-known email clients. We envision a repository or market place that allows users comparing and selecting applications based on energy labels and their personal requirements.

1 Introduction

Mobile devices, such as smart phones and tablets, have become very popular within the last years. Nowadays, we use them regularly and everywhere, checking emails, appointments or obtaining other content from the Internet. Besides the general usage of mobile devices, adapting and extending their functionality with small, domain-specific applications (i.e., *apps*) has become a typical scenario. The extensive usage of mobile devices and their low energy budgets, however, make power consumption of individual apps a major concern. Often, devices consume so much energy that they run out of it within hours or a day. Thus, investigating whether the power consumption of mobile devices can be decreased by developing applications more intelligently or more resource-saving is a major research challenge in software engineering. This is especially important for mobile applications that are not only executed during direct user interaction but are running as background services as well (e.g., to check email or news feed accounts for new incoming messages).

Thus, to increase the uptime of mobile devices, users should be able to base their decision, which application they want to install, not only on the provided functionality and the community's rating (e.g., a five star grading system as used in the Android market *Google Play*), but also on an expectation of the application's power consumption during runtime. To provide this information, we are working on a methodology that allows the comparison of applications providing similar services w.r.t. their power consumption. We intend to establish a framework, which allows profiling the power consumption of applications and approximating their long-running power consumption based on this information. Besides profiled power rates for different services, the way users utilize them and the decision which services they use how frequently heavily influences the applications' power consumption (e.g., the decision how often a user checks its emails, will influence an email client's communication traffic and thus, its power consumption). Therefore usage profiles should be used together with the profiled power consumption to approximate applications' power consumption for different usage scenarios. This approximated power behavior then be used to provide an alternative app grading system (e.g., a labeling system similar to existing approaches such as the European Union energy label for electric devices).

In this paper we propose such an energy labeling process for mobile applications. We profile the power consumption of two different Android email clients and evaluate their consumption for different use cases. The core contributions of this paper are:

- A process to profile the power consumption of mobile applications for certain use cases and a methodology to approximate their long-time consumption based on these measurements and additional usage profiles.
- First power measurement results showing that an application can consume different amounts of energy for different services and that different applications can consume different measurable amounts of energy for similar services.

The remainder of this paper is structured as follows. In Section 2 we present our energy labeling process for mobile applications. In Section 3 our power consumption profiling infrastructure is described and we investigate the power consumption of mobile applications by using an email case study. Section 4 discusses work related to our approach. Finally, in Section 5 we conclude this paper and give an outlook onto future works.

2 Comparing Power Consumption of Mobile Applications

Traditionally, to compare mobile applications, the only rating criterion is the apps' popularity within a community (e.g., star rankings as in Google Play). To extend this approach by power consumption as an additional criterion, we propose a process that allows for the investigation and approximation of the power consumption of mobile applications. The overall process consists of five steps and is shown in Figure 1. In the following we elaborate the individual steps and discuss how they could be realized if the process would be implemented for a mobile application market place.

To compare applications providing similar services, a prerequisite is to identify these services and to define a service model that expresses how these services are typically interconnected. Thus, during **(1) service modeling**, general use cases for a domain of applications are specified (e.g., use cases for email clients such as *checking for new mails*, *reading a mail*, *writing a mail*). Afterwards, based on these services, abstract test cases are defined.

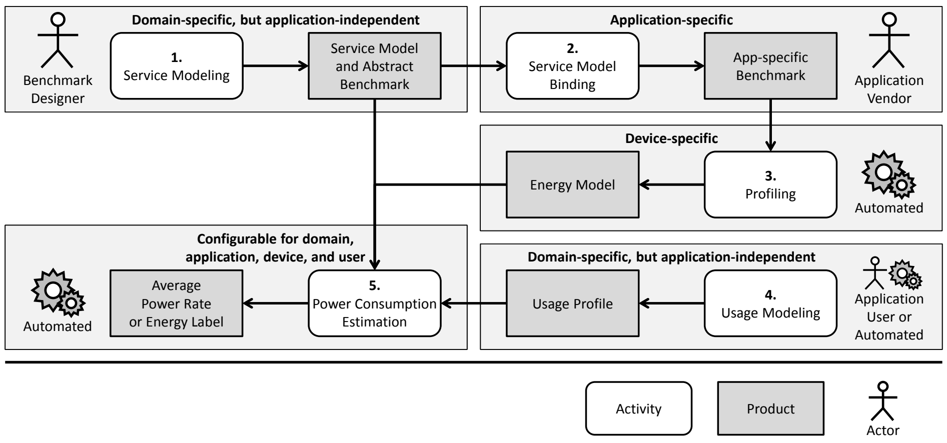
Domain-specific, but application-independent
Benchmark Designer
1. Service Modeling
Service Model and Abstract Benchmark
Application-specific
2. Service Model Binding
App-specific Benchmark
Application Vendor
Device-specific
Energy Model
3. Profiling
Automated
Configurable for domain, application, device, and user
Automated
Average Power Rate or Energy Label
5. Power Consumption Estimation
Domain-specific, but application-independent
Usage Profile
4. Usage Modeling
Application User or Automated
Activity
Product
Actor

Figure 1: Energy label computation process.

They describe paths through the service model (e.g., that an email account has to be selected before an email can be opened) as well as test data for benchmarking (e.g., the name and login of an email account).[1] Together, the service model and the abstract test cases form an abstract benchmark which specifies how to test an application of a certain domain (e.g., email clients). The technical realization of these test cases for a specific application under test (AUT) (e.g., which buttons, text fields, etc. have to be used to run these tests for certain email applications) is not part of the service model. In the context of a market place providing energy labels, the service modeling would be realized by a *benchmark designer*. A benchmark designer has sufficient domain knowledge for a specific domain of applications to extract typical services and to design abstract test cases. Although theoretically *application vendors* could be responsible to design such a benchmark, a neutral institution or the market place provider should be responsible for service modeling to avoid benchmarks designed in advantage for specific applications within the market place.

If a new application should be deployed in a market place or energy labels should be provided for an existing application, the abstract test cases have to be concretized for this application. Thus, during **(2) service model binding**, the transitions and activities of the service model are bundled to sequences of application-specific user interface (UI) interactions (e.g., the activity *browsing inbox* is bound to a click event on the button *"inbox"*). The result is a set of concretized test cases for the AUT ready for execution. We are currently working on an automated approach that derives the app-specific benchmark from the service model binding and generates the concretized benchmark code. As *application vendors* or developers typically know their application best, they should be responsible to deploy their application together with a test case binding in the market place to support comparison with competing applications (which requires trusting that vendors will not realize dummy bindings causing less power consumption).

[1]Classically, a test case represents an execution of an application under test (AUT) in a well-defined context comparing the expected with the execution result. However—in the context of this paper—a test case does not compare expected and observed outputs, but is used to utilize the AUT for power consumption profiling.

Once concrete test cases are available, they are executed during **(3) profiling**, resulting in an energy model. This process step is described in more detail in Section 3. The resulting energy model describes the energy behavior of the AUT in the context of the specified use cases. The energy model can be considered as metadata defined relative to the original service model (e.g., activities such as *reading an email* or *sending an email* are annotated with power rates expressing the average power rate of the AUT when performing these activities). The executable test cases from the previous activities should be usable to perform this activity in an automated way. However, application vendors should be able to investigate the results to identify potential for optimization of their applications.

Although the energy model is sufficient to express the AUT's energy behavior, workloads are required to predict the AUT's power consumption at runtime. As we intend to compare applications based on their round-the-clock energy behavior, it is insufficient to predict the power consumption for individual activities. Instead, we need information on how often and how long users intend to perform certain activities. Thus, during **(4) usage modeling** the usage behavior of a certain user is described. The usage profile enriches the service model with further metadata: activities are annotated with average durations (e.g., the average time to browse mails in the inbox) and transitions are annotated with statistical information (e.g., how often an email is written). We consider two different options to construct a usage profile: First, *application users* can estimate their usage behavior by creating a profile manually. This process can be simplified by answering predefined questions (e.g., "How often do you check your inbox per day?") whose answers are used to generate the actual usage profile. The second possibility is to derive a usage profile by gathering all necessary information automatically. This can be achieved by installing a special profiling app which constantly monitors the user's behavior. However, this approach requires that users already have applications installed that fulfill services similar to the application they are looking for, which might not always be the case. A solution for this problem would be the provision of default usage profiles (e.g., "heavy" and "standard" email usage profiles).

After modeling an application-domain's use cases as well as profiling its power consumption on a specific mobile device and selecting or creating a usage profile, the gained model and metadata can be combined to **(5) estimate the power consumption** of applications. By exchanging or altering the usage data, the approximation can be adapted to other usage scenarios. Moreover, for approximations for other mobile devices, or for other applications of the same domain the energy model can be exchanged as well. This way, the approach allows for easy adaptation to other user, software and hardware contexts. This final activity can be realized in a rather automated way; the users are only responsible to specify queries against the market place's database.

3 Investigating Power Consumption of Similar Android Applications

After presenting a process for energy labeling of mobile applications, we now focus on the profiling activity which is a prerequisite for the entire process. We present our profiling infrastructure that executes JUnit test cases on Android devices while profiling their power consumption in parallel and apply it to a small email case study.

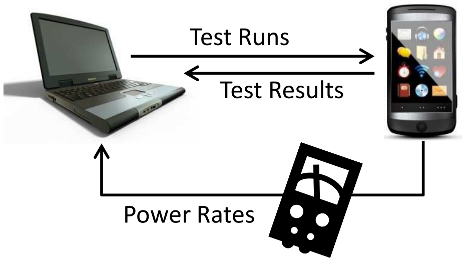

Figure 2: Schematic profiling illustration.

3.1 Profiling Infrastructure

To profile applications' power consumption for certain use cases, we follow the approach illustrated in Figure 2: Test cases are deployed and started on a mobile device issued from a test server. In parallel, power rates a profiled with a power meter bypassing the device's battery. On the test server, the measured power rates are associated with events logged during the test case execution. This way, the power rates can be associated to individual test runs and thus, to individual use cases. We decided to profile Android applications for our experiments. Thus, in the following, we describe how we implemented the outlined profiling infrastructure for Android. Although implementation details may differ, similar infrastructures could be implemented for other platforms such as iOS or Windows Mobile.

To implement an Android profiling infrastructure, we decided to reuse and extend the existing Android testing infrastructure provided by the Android SDK[2] and the Android Development Tools (ADT) that extend the Android SDK for Eclipse.[3] The ADT allows testing Android applications with an Android-specific extension of the JUnit framework. Test cases can be implemented in Java on a desktop PC. Afterwards, the test cases are compiled into an Android test application that is shipped and deployed via an USB connection onto the device under test (DUT). On the DUT the test application instruments the AUT and runs workloads by simulating UI interactions such as pressing buttons or entering texts. Besides the execution of unit test cases on the AUT, the test application is also able to access information from the device's operating system services such as its CPU utilization, its WiFi connection or battery status. Thus, in theory, the ADT test runner would be sufficient to profile the power consumption of Android applications. However, profiling the device's power consumption via its built-in sensors introduces a probe effect during testing (i.e., the device utilizes the hardware for the profiling as well). Even more important is the fact that for all tested Android devices the refresh rates of the battery sensors are too low. For many devices, new probes are only available every thirty seconds or even just once in a minute, which is too slow for reasonable power consumption profiling, as many activities on mobile devices usually last at most a few seconds (e.g., a network communication or the rendering of an image to be displayed).

[2]http://developer.android.com/sdk/ (visited in June 2012)

[3]http://developer.android.com/guide/developing/tools/adt.html (visited in June 2012)

Thus, we extended the ADT testing infrastructure with support for external power consumption profiling. JUnit test cases derived from the service model are deployed as a test application on the DUT. During their execution, events (e.g., the beginning or termination of the individual test cases) are logged. In parallel, an external power meter is used to profile the DUT's power consumption between its battery and the device itself. Afterwards, the collected power rates and the logged events from the AUT are correlated on the test server to compute the power consumption of the tested activities. To allow this correlation, the system clock of the test server and the DUT are synchronized using the Network Time Protocol (NTP) before starting the test sequence. The power consumption of the AUT is computed by measuring the DUT power consumption without the application and only a reasonable set of background services running (i.e., required OS services only), and afterwards, the device's power consumption while executing the AUT. The AUT's power consumption can be assumed as being the difference between profiled base power consumption and profiled power consumption when running the AUT. Although this might not be always the case as a minimum set of OS services is running in parallel, we argue that the computed power rates are sufficient for our process as we intend to compare the power consumption rates of different applications relatively and are not interested in the most possible precise measurements—as long as the results are sufficient to compare different AUTs. By executing the same test cases multiple times, variations of power consumption measurements can be minimized and statistically eliminated. Although it is likely that the event logging on the DUT might introduce some extra power consumption and thus, a probe effect, our investigations showed that reasonable logging activity does not cause an extra power consumption being high enough to influence our measurement results. Thus, this probe effect can be considered as being too small to be relevant and can be ignored for our measurements. As external power meter hardware we used a Yokogava WT210. Anyhow, any other power meter that can be connected with a PC can be used instead.

3.2 An Email Client Case Study

To investigate whether or not our comparison process for mobile applications is generally possible, we defined a simple case study to evaluate the general concerns of our approach. Especially we were interested in answering the following research questions: Is power consumption of mobile applications big enough to be measured with available measurement hardware? Can services of the same application cause different power rates being measurable? Can different applications providing similar services cause different amounts of power consumption being measurable? To answer these questions we defined a few email client use cases, general enough to be executed on different email client apps. Each of these use cases relates to a central email client service. The defined use cases for our investigation are: (1) browsing an email account's inbox and checking for new mails, (2) reading an email, (3) opening and viewing an attachment, and (4) executing a client's background service that is responsible to check the email account for new mails while the user is using other apps or the device is running in a standby mode. To evaluate these use cases for different Android applications, we decided to investigate two different email

Use case	AUT	$\overline{p}\,[W]$	$s_p\,[W]$	$\overline{t}\,[sec]$	$s_t\,[sec]$
Check inbox	K-9 Mail	0.356	0.022	7.96	0.06
	MailDroid	0.433	0.023	10.40	0.14
Read mail	K-9 Mail	0.259	0.019	16.01	0.55
	MailDroid	0.338	0.012	17.91	0.04
Open attachment	K-9 Mail	0.286	0.017	14.06	0.54
	MailDroid	0.420	0.031	20.71	0.13
Background service	K-9 Mail	0.037	0.011	10.00	0.00
	MailDroid	0.152	0.029	10.00	0.01

Table 1: Average power consumption $\overline{p}$, corrected sample standard deviation s_p, average execution time $\overline{t}$, and corrected sample standard deviation s_t for different use cases, each profiled five times for both apps.

clients available for free via the Android market place. The first application we chose is the popular K-9 Mail app[4] (version 4.107) that has already been downloaded by more than 1,000,000 Android users. As a second application we decided for the MailDroid app[5] (version 2.31), another popular email client installed more than 500,000 times.

3.3 Profiling Results

The above-mentioned use cases have been implemented for both email clients and have been tested on an ASUS Transformer TF101 tablet PC with Android version 3.2.1. For both applications the same email account was used to execute the test cases. All four use cases were profiled five times for each AUT. We started the profiling by using a fully charged battery and executed the test cases for both applications alternately to reduce influences on the measurements caused by a more or less discharged battery. Before each test run we profiled the TF101's base consumption for ten seconds with a minimum of running background services (e.g., the network and keyboard service) and the LCD brightness being set onto 50%. The WiFi connection has been set enabled for the required Internet communication. The power rates have been profiled using a Yokogava WT210 power meter with a probe frequency of $4Hz$.

Profiling the tablet's base consumption resulted in an average consumption of $2.694W$ with a standard deviation being less than $0.001W$. For the power consumption profiling of the executed use cases, this base consumption has been subtracted from the profiled power rates, assuming that all additional power consumption during the test runs was caused by the currently executed AUT.

Table 1 shows the profiling results. Except for the profiling of the background service, all use cases resulted in average power rates having standard deviations of less than 10%. The major reason for the background's service higher standard deviation may be the relative short profiling time for such a long running service. As can be seen, different use cases

[4]http://code.google.com/p/k9mail/ (visited in June 2012)
[5]http://groups.google.com/group/maildroid/ (visited in June 2012)

of the same application caused different amounts of power consumption. The most power was consumed by activities requiring communication hardware (i.e., WiFi) as by the *check inbox* use case. Therefore, also the assumption that application's power consumption can vary for different provided services seems to be correct. Another observation is that all use cases caused higher power consumption rates when executed with the MailDroid application. K-9 Mail outperforms MailDroid in all specified use cases consuming between 18% (check inbox) and 75% (background service) less power than MailDroid. Although differences in power consumption are only deci-Watts, a t-test showed that the observed differences can be considered as being significant. Thus, it can be assumed that the differences have not been caused by measurement errors. One explanation for the higher power consumption caused by the MailDroid application is that we profiled its trial version where an advertisement banner is display in each view of its UI. Loading the banner from the Internet could cause the additional power costs. Similar observations have already been made by Pathak et al [PHZ12], showing that advertisement can cause up to 75% of app's total power consumption! An investigation if the commercial version of MailDroid behaves better than the free version remains a target for future work.

3.4 Threats to Validity and Limitations

Although the measurements presented above generally support our research questions, some threats to validity in the context of these measurements should be considered which are shortly discussed in the following. We differentiate threats related to the profiling infrastructure and to the proposed energy labeling process in general.

3.4.1 Profiling

Charging via USB power connection As mentioned above, the test cases are executed on the DUT via a USB connection from a test runner PC. Unfortunately, on Android devices a USB connection will automatically cause battery charging which cannot be disabled via software settings. As we need the USB connection to run our test cases, we cannot unplug the cable from the device. Cutting the power wire of the USB cable is not working either as the power wire is also required to identify the device from the PC and vice versa. Thus, we ignored the battery charging via USB connection during our test runs. We measured the charging rate of the DUT's battery before running our test cases and measured a constant charging rate of $0.6W$. Thus, the measured base power consumption of the DUT can be considered as being about $0.6W$ too low. Although the precise values of our measurements are incorrect, we argue that the results are still sufficient to show that the MailDroid app consumes more energy than the K-9 Mail competitor, because both tests ran under the same circumstances and are therefore comparable. For future test runs we plan to profile the USB charging rate in parallel to the battery's power rate of the DUT.

Imprecise measurements Measurements were done with a probe frequency of $4Hz$. Thus, power rate peaks lasting only $250ms$ or shorter may not be profiled appropriately.

However, we argue that for our coarse-grained test scenarios the probe frequency is sufficient. As triggered events such as WiFi communication typically last longer than $250ms$. The standard deviation of our results shows that it is unlikely that we missed many power peaks in one of the test runs as otherwise the computed standard deviation of the measured values should be higher. Anyhow, our approach is currently not able to track short term events such as high CPU utilization for a few milliseconds only. For future work we plan to use more fine-grained probe frequencies instead. Another concern for the results' quality is that each individual test run was only profiled five times which can be considered as too less for significant statistical results. However, we argue that the small measurements variances suggest stable measurements over multiple test runs. Based on the variances we computed the minimal number of necessary measurements resulting in five runs for all our measurements. Thus, our number of measurements can be assumed as only just sufficient. As a consequence, we plan further test runs for future work to confirm our findings.

Generalizability for other mobile devices Our measurements were performed for one specific Android device, an Asus Transformer TF101. It is very likely that other mobile devices from other vendors will behave differently and have different power rates while executing the same applications and use cases. Thus, the question whether the same differences between K-9 Mail and MailDroid will occur on other mobile devices remains a target for future work. We assume that similar phenomena will be observable, but the total power rates as well as the relative differences between the apps may vary on other devices.

Small power consumption differences for similar use cases of different applications Although we showed that the two profiled applications consume different amounts of power, the differences are rather low (between $0.077W$ and $0.152W$ for our four use cases). However, we argue that even these small differences can be important for long-run scenarios. Especially the differences of the background services' power consumption that may run 24/7 can be important (e.g., according to our measurements when using MailDroid instead of K-9 Mail as email client, the background service will consume $2.76Wh$ additional energy every day, more than 10% of the TF101's total battery capacity!).

Influences from helper applications In some cases, Android applications delegate services to other applications via so-called *Intents*. For example, in the open attachment example the attachment is opened by another application, depending on the file type (e.g., an image or PDF viewer). Thus, the power consumption of this use case depends on the application to open the attachment as well. As in our scenario we executed all test cases on the same DUT, the same applications were used for attachment viewing. However, in more heterogeneous scenarios such application interference should be considered as well.

3.4.2 Energy Labeling Process

Coarse-grained use cases Although our measurements show that different use cases of the same application as well as the same use cases executed on different applications can cause different power rates, the results presented above are rather coarse-grained. Each profiled use case consists of a complete UI interaction beginning with starting the application, clicking through its menus and finally performing an action (e.g., checking the inbox). For future work we plan to define more fine-grained test cases that perform similar tasks with different parameters (e.g., sending emails of different sizes) to get a better understanding on how these parameters can influence the use cases' power consumption.

How to compare applications with limited similarity Another question that should be raised in context of the proposed energy labeling process is the question how applications can be compared that provide a set of similar services but other individual services as well. Of course, only the services provided by both applications can be considered for comparison. Thus, if an application *A* provides additional services not provided by a compared application *B*, these services are considered as not existing. The easiest way to reflect this during power consumption estimation is to set their usage rate in the applied usage profile to zero—meaning they will never be used after the app's installation. Another related problem are apps being totally different from all other applications available. In this case a label comparing the app w.r.t. similar application would not be possible. However, it is still possible to compare applications from similar genres (e.g., games with other games, having different service models and usage profiles).

Similarity in application workflow Binding test cases for different apps to the same service model requires that their workflow is similar enough to allow the abstraction of their common behavior in a service model. This might be the case for the discussed email example as well as other standard use cases (e.g., web browsers or new feeders). However, in other scenarios this might be more complicated. We plan to investigate the limitations of a common service model in further case studies and considering optional or alternative transitions within future versions of our proposed service models.

Combinatorial explosion of possible configurations Of course, the separation of the power consumption into an app-specific energy model and a user-specific usage profile on a highly heterogeneous platform like Android with hundreds of different devices and millions of apps available leads to a combinatorial explosion when all different settings shall be profiled and stored in energy models. Future work will show how different power consumption of individual apps will behave on different mobile devices. As we are only interested in relative results comparing different applications, it might be sufficient to investigate a small subset of representative devices to compute appropriate energy labels (expressing only relative power behavior!) in all realistic usage scenarios.

Power rates as comparison criterion As profiling results show, different apps do not only differ in their average power rates but also in their average execution time for different use cases. This raises the question whether or not average power rates are right comparison criterion. For short term use cases such as reading a mail or opening an attachment also a use case's total power consumption would be an appropriate criterion. However, for long running scenarios (i.e., background services) power rates are more appropriate as execution time is irrelevant for these cases. An optimal solution could be a comparison considering both apps' long run power consumption as well as execution time for use cases involving active user interaction.

3.5 Power Profiling for Large Developer Communities

A central question that arises once the proposed process should be used in an ecosystem where many different stakeholders develop applications, share them in a central repository and compete against each other is whether it is possible to adapt the process without requiring each developer having his own profiling infrastructure, as power metering hardware is still rather expensive. We see three different solutions for this profiling problem:

Better built-in battery sensors Future mobile devices might get better built-in battery sensors that may be sufficient for appropriate power consumption profiling. As power consumption is a major concern, it is likely to expect that hardware vendors might consider the introduction of better battery sensors. Once built-in sensors provide probe frequencies of $1Hz$ and higher, the external measurement hardware becomes obsolete. However, this solution depends on the decisions of hardware vendors and not of the software developers aiming to build more energy-efficient mobile applications. Furthermore, the usage of built-in sensors might introduce further probe effects during the test runs that have to be considered during energy modeling.

Power profiling as a cloud service Besides waiting for better built-in sensors it would be possible to provide an existing power metering infrastructure as a Web or cloud service in a way that it could be reused remotely by other software developers. However, this solution comes together with some drawbacks. First, it requires a Web service with sufficient resources for all developers intending in power testing their applications. Second, using remote testing, developers are not able to control that the DUT performs in the way they expect it to do. They can introspect the devices status via screen-capturing tools or similar solutions but they are not able to investigate the DUT's physical environment (e.g., whether or not the mobile is in an upright position or the current lighting of the environment).

Application of power consumption models Another alternative solution would be to use mathematical models that approximate the device's power consumption based on hardware utilization information. Related work shows that building such models based on consumption profiling training data can result in systems appropriate enough to approximate the device's power consumption with errors of less than 5% [ZTQ^{+}10, KB11, ZGFC11]. These models could be used to replace the external measurement hardware. Although this would lead to less accurate results, they could be sufficient to compare different applications' average power consumption.

4 Related Work

Besides our approach, several other works exist that allow for the profiling and estimation of mobile devices' power consumption. However, most of these approaches focus on power consumption of individual hardware components whereas our approach focuses on the comparison of different software applications' power consumption.

In [PANS11] Palit et al. present a methodology to profile average power consumption of mobile applications. Their testing infrastructure can be considered as similar to our approach, consisting of a test server, a mobile device under test and an external power meter. However, they focus on profiling average power consumption rates for typical application use cases executed on different mobile platforms (e.g., Android and Blackberry). Our approach in contrast targets to compare the power consumption of similar applications on the same mobile devices.

Carroll et al. present an approach to profile an individual smart phone's power consumption in [CH10]. They execute different use cases on the mobile such as playing music, sending emails or phone calls and profile the device's power consumption using external power metering hardware in parallel. In contrast to our approach, Carroll et al. try to estimate the consumption impact of individual hardware components whereas our approach estimates the impact of different software applications.

Kjærgaard and Bluck [KB11] compute mathematical power consumption models for smart phones. Their *PowerProf* tool executes multiple benchmarks on Nokia Symbian phones, each utilizing different hardware devices such as CPU, display, and WiFi. In parallel, the tool profiles the device's power consumption by using the software interface of the battery sensors. A genetic algorithm is used to transform profiling data into a mathematical model predicting the phone's power consumption based on data on its hardware utilization. In contrast to our approach, Kjærgaard und Bluck focus on hardware profiling and a model to predict the hardware's power consumption based on utilization. Thus, to predict power consumption of software applications their approach requires a profiling of the hardware utilization cause by the AUT beforehand.

A similar approach to Kjærgaard and Bluck has been developed by Zhao et al. [ZGFC11]. The power consumption of Android devices is profiled via their software-based battery interface and used to compute linear regression equations expressing the devices power consumption based on its hardware components' utilization.

Another software-interface based profiling approach has been implemented by Rice and Hay [RH10]. Test cases are executed via a test server and the profiling results are traced by using the phone's logging system. In contrast to our work, Rice and Hey focus on profiling the power consumption caused by the phone's WiFi communication.

Zhang et al. developed a profiling tool [ZTQ$^+$10] that can be considered as rather similar to our approach. Test cases are executed on Android devices based on a test server, whereby the device's power consumption is profiled using external measurement hardware. The results are used to compute linear regression equations similar to Zhao et al. The equation system is used as a basis for an Android application called *PowerTutor*. PowerTutor is able to approximate the power consumption for applications currently executed on the device based on their CPU time and the device's hardware utilization. Although the approach can be used to estimate the current device's power consumption, it does not allow for an offline estimation of software applications' average power consumption where usage profiles can be varied to adapt power consumption to different usage scenarios. However, the PowerTutor tool would be the perfect candidate to replace our external power metering with a software-based approximation as discussed in Section 3.5.

Pathak et al. [PHZ$^+$11, PHZ12] propose another approach for power consumption profiling of mobile devices. AUTs are instrumented with additional logging code for system call tracing. Afterwards the applications are deployed on the DUT and typical usage scenarios (e.g., Internet browsing or chess gaming) are executed. The recorded system traces are used to compute the application's power consumption by replaying the traces on a finite state machine model that represents the device's hardware components and their different power states. Pathak et al. use their approach to investigate several Android applications' power consumption and show that network communication and input/output operations cause large amounts of their total power consumption. In contrast to our approach, for power consumption approximation Pathak et al. use system traces and a power consumption model of their DUT. Thus, they do not profile the applications' real power consumption but approximate them from power consumption models created beforehand. Furthermore, they investigate the power consumption of some mobile applications, but do not propose an approach to systematically compare mobile applications providing similar services.

Besides power profiling of mobile applications, further approaches exist that focus on power consumption of laptops, desktop PCs or server systems. Rivoire et al. developed the *JouleSort* benchmark for desktop applications as well as server systems [RSRK07, Riv08]. Based on classical sorting benchmarks the system under test is evaluated based on the elements sorted per Joule instead of performance and throughput. In contrast to our approach JouleSort focuses on desktop applications. Furthermore, the approach focuses on sorting algorithms whereby our approach aims to provide a benchmark infrastructure for arbitrary kinds of mobile applications.

In [LL06] Lafond et al. propose an approach that profiles the power consumption of individual software instructions. A large subset of all Java bytecode instructions is profiled and used as a basis for application's power consumption predictions. A similar approach has been proposed by Seo et al. [SMM07, SEMM08]. Although both approaches focus on applications' power consumption estimation, they focus on another level of abstraction than our approach. Our approach focuses on coarse-grained profiling of activities where measurement errors are likely to be rather small in contrast to profiling errors for individual instructions. Furthermore, our approach considers varying usage behavior for different usage contexts which is not supported by Lafond and Seo.

5 Conclusion and Future Work

In this paper we proposed a process for comparing mobile applications' power consumption consisting of five steps. Use cases are modeled abstractly for a certain application domain. Afterwards, these use cases are refined to concrete test cases for each application under test. Execution and profiling of the test cases results in an energy model which together with a usage profile can be used to approximate an app's power consumption as well as to compute energy grades comparing similar apps w.r.t. their power consumption. We have shown, how Android apps can be profiled by extending the Android Development Tools with external power metering hardware and have profiled the power consumption of two Android apps (K-9 Mail and MailDroid) while executing the same test cases on both of them. We demonstrated that similar apps can indeed consume different amounts of energy, which is especially important when running as a background service as this use case can be assumed to be the most important impact factor for the apps' total power consumption in a 24/7 service scenario.

For future work we plan to improve our profiling infrastructure as well as performing more detailed case studies (e.g., for mail clients, news readers and web browsers). Besides, we plan to address the threats to validity as discussed in Section 3.4. We intend to implement the complete energy labeling process presented in this paper, resulting in a repository where users are able to compare different apps based on their power consumption in the context of personalized usage scenarios. We also plan to evaluate how energy labels influence users in their decisions which apps to install on their mobile devices. Another interesting task is to adapt the process to compare different versions of the same application. Thereby, the same process could be used for energy regression testing and optimization of specific mobile applications. Finally, based on our profiling results we are considering strategies or implementation patterns that can be identified for the development of future, more energy-efficient mobile applications.

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). Furthermore, we thank Waltenegus Dargie and his team for providing the required power metering hardware, Doreen Fiss for her assistance to statistically investigate our measurement results and the reviewers of this paper for their very helpful and encouraging remarks.

References

[CH10] A. Carroll and G. Heiser. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX annual technicalconference*, pages 21–21, Berkeley, CA, 2010. Usenix Association.

[KB11] Mikkel Baun Kjærgaard and Henrik Bluck. Unsupervised Power Profiling for Mobile Devices. In *Proceedings of the 8th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (Mobiquitous 2011)*, Berlin / Heidelberg, 2011. Springer.

[LL06] Sébastien Lafond and Johan Lilius. An Energy Consumption Model for an Embedded Java Virtual Machine. In *Architecture of Computing Systems - ARCS 2006*, volume 3894 of *Lecture Notes in Computer Science*, pages 311–325. Springer, Berlin / Heidelberg, 2006.

[PANS11] R. Palit, R. Arya, K. Naik, and A. Singh. Selection and Execution of User Level Test Cases for Energy Cost Evaluation of Smartphones. In *Proceeding of the 6th international workshop on Automation of software test*, pages 84–90, New York, 2011. ACM.

[PHZ$^+$11] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y. M. Wang. Fine-grained power modeling for smartphones using system call tracing. In *Proceedings of the sixth conference on Computer systems*, pages 153–168, New York, 2011. ACM.

[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.

[RH10] A. Rice and S. Hay. Decomposing power measurements for mobile devices. In *IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 70–78, Los Alamitos, CA, 2010. IEEE Computer Society Press.

[Riv08] Suzanne Marion Rivoire. *Models and Metrics for Energy-Efficient Computer Systems.* PhD thesis, Stanford University, 2008.

[RSRK07] Suzanne Rivoire, Mehul A. Shah, Parthasarathy Ranganathan, and Christos Kozyrakis. JouleSort: a balanced energy-efficiency benchmark. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data (SIGMOD '07)*, pages 365–376, New York, 2007. ACM.

[SEMM08] Chiyoung Seo, George Edwards, Sam Malek, and Nenad Medvidovic. A Framework for Estimating the Impact of a Distributed Software System's Architectural Style on its Energy Consumption. In *Proceedings of the The Working IEEE/IFIP Conference on Software Architecture (WICSA 2008)*, pages 277–280, Los Alamitos, CA, 2008. IEEE Computer Society Press.

[SMM07] Chiyoung Seo, Sam Malek, and Nenad Medvidovic. An Energy Consumption Framework for Distributed Java-Based Systems. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated Software Engineering*, New York, 2007. ACM.

[ZGFC11] X. Zhao, Y. Guo, Q. Feng, and X. Chen. A System Context-Aware Approach for Battery Lifetime Prediction in Smart Phones, 2011.

[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.