TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik - Institut für Software- und Multimediatechnik

TECHNISCHE BERICHTE
TECHNICAL REPORTS

ISSN 1430-211X

TUD-FI12-10 Dezember 2012

Claas Wilke et al.

Fakultät Informatik, Lehrstuhl Softwaretechnologie

Comparing Mobile Applications' Energy Consumption

Technische Universität Dresden
Fakultät Informatik
Institut für Software- und Multimediatechnik
Lehrstuhl Softwaretechnologie
01062 Dresden, Germany

Comparing Mobile Applications' Energy Consumption *

Claas Wilke
Software Technology Group
TU Dresden, Germany
claas.wilke@tu-dresden.de

Sebastian Richly
Software Technology Group
TU Dresden, Germany
sebastian.richly@tu-dresden.de

Christian Piechnick
Software Technology Group
TU Dresden, Germany
christian.piechnick@tu-dresden.de

Sebastian Götz
Software Technology Group
TU Dresden, Germany
sebastian.goetz@acm.org

Georg Püschel
Software Technology Group
TU Dresden, Germany
georg.pueschel1@tu-dresden.de

Uwe Aßmann
Software Technology Group
TU Dresden, Germany
uwe.assmann@tu-dresden.de

ABSTRACT

As mobile devices are nowadays used regularly and everywhere, their energy consumption has become a central concern for their users. However, mobile applications often do not consider energy requirements and users have to install and try them to reveal information on their energy behavior. In this paper, we compare mobile applications from two domains and show that applications reveal different energy consumption while providing similar services. We define microbenchmarks for emailing and web browsing and evaluate applications from these domains. We show that non-functional features such as web page caching can but not have to have a positive influence on applications' energy consumption.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging

General Terms

Measurement, Performance

Keywords

Android, Energy Benchmarking, Greenness of Mobile Apps

1. INTRODUCTION

Mobile devices are nowadays very popular and are used regularly and everywhere. They are able to fulfill several tasks such as emailing, web browsing, gaming, video capturing, uploading, and replay. However, due to their round-the-clock usage, their energy consumption has become an omnipresent problem. Hardware-intensive use cases such as video capturing or GPS navigation can drain the devices' batteries within hours leading to shorter device uptimes and thus, limited quality of experience [5, 7, 13].

Hardware vendors have tackled this problem by designing hardware being more energy-efficient, switching unutilized components into low power or sleep states. However, most software is still designed without considering these power modes and thus, often hinders hardware components from switching into power saving states. As a consequence, they consume more energy than necessary [6].

Building software applications for mobile devices is a highly competitive market. Often multiple applications providing similar services exist. In [11], we propose to introduce energy labels as additional guidance for users searching for appropriate applications supporting individual usage requirements. In this paper we demonstrate that such a process is necessary and sensible, by comparing mobile applications of two typical usage domains w.r.t. to their energy consumption. Based on the results we show that applications providing similar services can have different energy characteristics. We present two case studies: emailing and web browsing; and define microbenchmarks to evaluate the energy consumption of mobile applications.

The core contributions of this paper are:

- Two microbenchmarks for Android email clients and web browsers reusable to compare the profiled apps with other Android applications from the same domains.
- Measurement results showing that different applications can vary significantly in their energy consumption for similar use cases leading to scenarios where one application behaves better for some use cases, whereas the other application behaves better for other use cases. This results in a trade-off between different use cases and their significance for the application user.
- Profiling results showing that advertisement can have a major influence on an application's energy behavior depending on when and how often new banners are loaded and displayed.
- Measurements showing that caching can but not has to have a measurable influence on an application's energy consumption.

The remainder of this paper is structured as follows: Section 2 presents the profiling infrastructure used to perform our measurements. Section 3 elaborates the two case studies,

*A shorter version of this paper has been published at the ACM Symposium on Applied Computing (SAC 2013) [10].

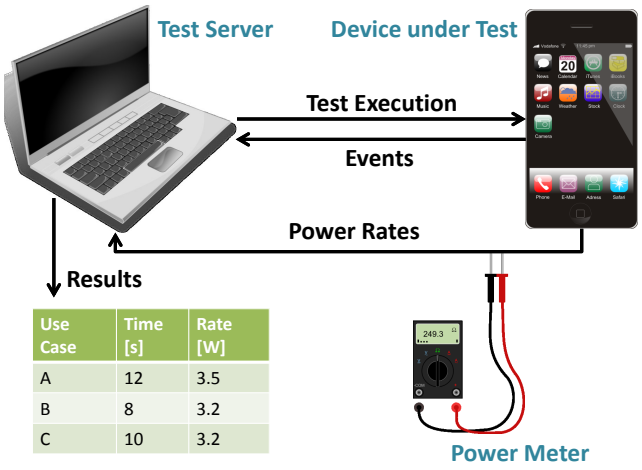


Figure 1: Illustration of our profiling infrastructure.

their use cases and the measured results for the respective applications. Afterwards, Section 4 discusses the limitations and threats to validity of our results. Section 5 presents related work. Finally, Section 6 concludes our work.

2. PROFILING PROCESS

To comparably profile the energy consumption of mobile applications we designed the following profiling process (cf. Fig. 1): typical use cases are implemented as executable unit tests that are executed on an device under test (DUT). In parallel, power rate probes are picked from the device's battery and afterwards, associated with the executed test cases to compute their energy consumption. The individual steps are described in more detail below.

Each individual test case represents a set of user interface (UI) interactions of a user with the application under test (AUT). For example, a test case can represent the composition and sending of a mail, consisting of a sequence of button clicks and entered texts. Altogether, these test cases form an executable test suite (or microbenchmark) for the AUT. These tests can simply be adapted to other applications of the same domain (i.e., the code to click buttons or to enter texts has to be adapted to the UI elements of the new AUT, whereas the general intention of the test cases remains the same).

The execution of the test cases is controlled from a test server (e.g., a desktop PC) which deploys the test cases on a DUT and triggers their execution. In parallel, the energy consumption of the DUT is profiled with a power meter picking power rate probes at the DUT's power supply in front of the battery. On the test server, the measured power rates are associated with events logged during the test cases' execution (i.e., the start and stop of the test cases and the respective timestamps). Thus, the power rates are associated to individual test runs and use cases. To compute the energy consumption of the use cases, the idle energy consumption (i.e., the consumption of the DUT without executing any services and applications) is subtracted from the measured energy consumption.

For our experiments we profile Android applications as Android is nowadays the most broadly used as well as the most open mobile platform. We extended the existing Android Developer Tools [1] for our benchmarking process as described in [11]. Although implementation details may differ, similar profiling processes can be realized for other mobile platforms such as iOS or Windows Phone. As external power meter hardware we used a Yokogava WT210 with a measurement accuracy of $\pm 0.2\%$ for current and voltage and a maximum profiling frequency of 10Hz. As DUT we used an Asus Transformer TF101 running Android version 4.0.3 and a Google Nexus 7 running Android version 4.1.1. All power rates were profiled with a probe frequency of 10Hz. As the display brightness has a significant influence on the devices' energy consumption [7] but is not in the focus of our study, we set the tablets' display brightness to 55% and did not modify the brightness during our tests.

3. CASE STUDIES

We now present two case studies comparing mobile applications from the same domain w.r.t. their energy consumption. For both domains, we define a set of use cases representing a microbenchmark implemented as a set of unit test cases for each AUT. Afterwards, we present and interpret the measured results.

3.1 Email Client Case Study

The first case study consists of two different email clients.

3.1.1 Use Cases

We identified the following use cases:

- Setup an email account
- Drop an email account
- Check for new incoming mails
- Read an email
- Write an email
- Forward an email
- Delete an email

Besides the *setup* and *drop account* use cases, each use case is executed with varying test data: we designed five mails to parameterize the use cases: A short mail with a text of 460 characters (2KB), a longer mail containing 920 characters (3KB), a mail with a picture attachment of 3.2MB, a mail with a note (consisting of 110 characters), and a mail with a speech memo of 40 seconds speech (62KB). Where mails include attachments, the *read mail* use case includes the opening of the attachment. Besides, the *read* use case was executed twice during each iteration, to check, whether or not a second execution is more efficient due to caching of messages and/or attachments.

These combinations of use cases and test data resulted in a test set of 33 test cases for each AUT. The composition and reading of mails has been implemented such that they appear realistic as each character of a composed mail is

entered separately simulating the mails creation via a keyboard. Furthermore, the time to display opened mails for reading depends on their content's length.[1]

3.1.2 Tested Applications

For our study we compared the following email clients:

App	Version	Downloads
K-9 Mail	4.0.1	$> 1,000,000$
MailDroid	2.5.7	$> 500,000$
MailDroid pro	2.5.7	$> 50,000$

MailDroid is delivered in two different versions, a freely available version including advertisement banners and a professional version that can be purchased for about 15 Euros and excludes advertisement. As network traffic required for communication might influence the results, we decided to test both versions of MailDroid.

All use cases have been tested using the same mail account and the IMAP protocol. We disabled the pull/push functionality to avoid side effects due to non-triggered mail down- and uploads. Besides turning off the pull/push functionality, we used the default settings for the tested applications. For each application we started the profiling with a fully charged battery and disabled all unnecessary services and applications running in parallel on the Asus Transformer TF 101. Each test was profiled 50 times per application. As network communication device we used a local WiFi rooter connected to a 1GBit Internet connection.

3.1.3 Results

Fig. 2 shows the median execution times, power rates and energy consumption for all profiled use cases and applications. A Kruskall-Wallis test led to rejected null hypotheses[2] for all use cases (cf. Fig. 2(d)), leading to the conclusion that the number of measured values is statistically sufficient to compare the tested email applications.

Comparing execution times (cf. Fig. 2(a)), for some use cases K-9 Mail performs a bit faster than MailDroid (both the free and the professional version). Fetching and reading mails can be performed a bit faster with K-9 Mail which is majorly caused by its easier UI navigation (e.g., the first email account is opened by default on app start up and does not have to be selected manually). However, the faster performance for downloading and opening attachments cannot be simply explained by UI navigation. K-9 Mail performs significantly faster for these use cases.

Considering the median power rates (cf. Fig 2(b)) and energy consumption (cf. Fig 2(c)), MailDroid's free version behaves much worse than K-9 Mail and MailDroid pro for all tested use cases. As MailDroid pro does not show this additional energy consumption, it can be assumed that the loading of advertisement banners has a very bad influence on energy consumption due to additional network communication. This assumption matches with results from prior work, for example [5, 9] and the observation that energy consumption increases more significantly for longer test cases, where further advertisement banners are downloaded and displayed. The energy consumption of K-9 Mail and MailDroid pro differs only marginally for most of the tested use cases. In general, K-9 Mail behaves a bit better. However, these differences are only a few deci-Joules caused by its slightly shorter execution time for the use cases. The differences increase for the opening attachment use cases (due to MailDroid's longer downloading time). Surprisingly, both applications behave differently when handling picture attachments. MailDroid requires more time and thus, also more energy to fetch and open picture attachments, whereby K-9 Mail requires more time and energy to send and forward such attachments.

Summing up, whereas MailDroid consumes much more energy due to advertisement, K-9 Mail and MailDroid pro behave rather similarly. However, K-9 Mail comes for free whereas MailDroid pro costs more than 15 Euros. Thus, where free versions of email applications contain advertisement banners, users should avoid them and should use other freely available applications without advertisement instead, saving up to 75% of the energy spent to write, send and read emails.

3.2 Web Browser Case Study

The second case study comprises three web browsers.

3.2.1 Use Cases

For web browsers we defined the following use cases, based on scenarios evaluated and tested in [8].

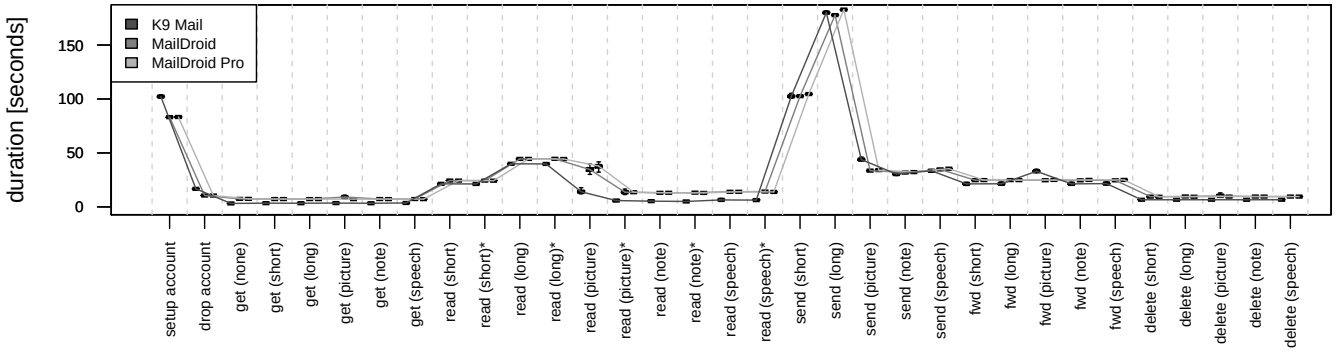
- Open a web page
- Open an image
- Download a file
- Performing a web search

As tested web pages we setup five dummy web pages on a test web server, consisting of raw HTML, HTML and JavaScript, HTML and CSS, HTML with embedded images and HTML with embedded video. Besides, we tested three popular web pages: Google.com, NYTimes.com and YouTube.com. All web pages were loaded with and without cleared browser caches. As test images we used two JPEGs (3.2MB/351KB) and two GIFs (4.3MB/330KB). As downloads we used two PDFs (1.5MB/233KB). The images and PDFs were also deployed on our test web server. Downloads and images have been opened two times (cached and uncached). For web search we used the default browser search engine by entering a keyword into the address field. As keyword we used the 2011's most often used keyword for news search: 'Fukushima' [2].

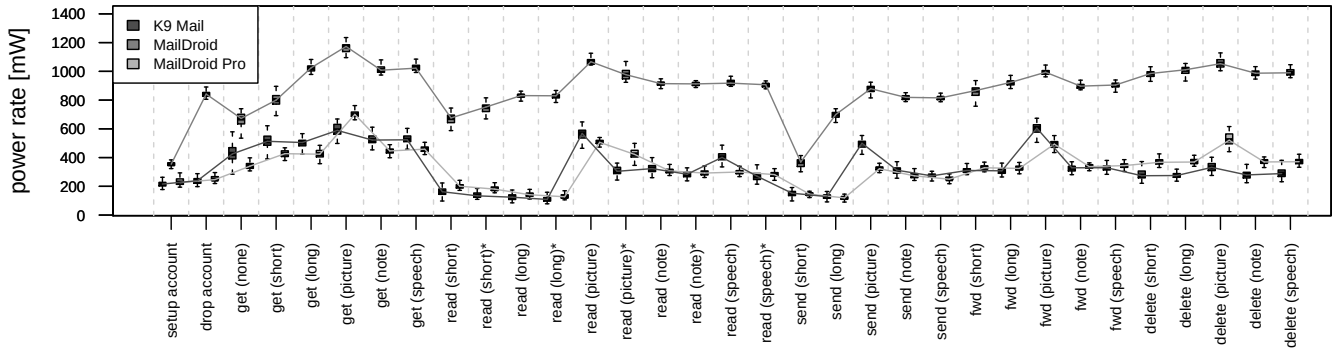
These use cases resulted in a test set of 30 test cases per AUT. The entering of URLs and keywords into the browsers' address field has been implemented such that they appear

[1]Further information on the implemented test cases, their source code and the measurement results can be obtained from http://www.qualitune.org/?page_id=532.

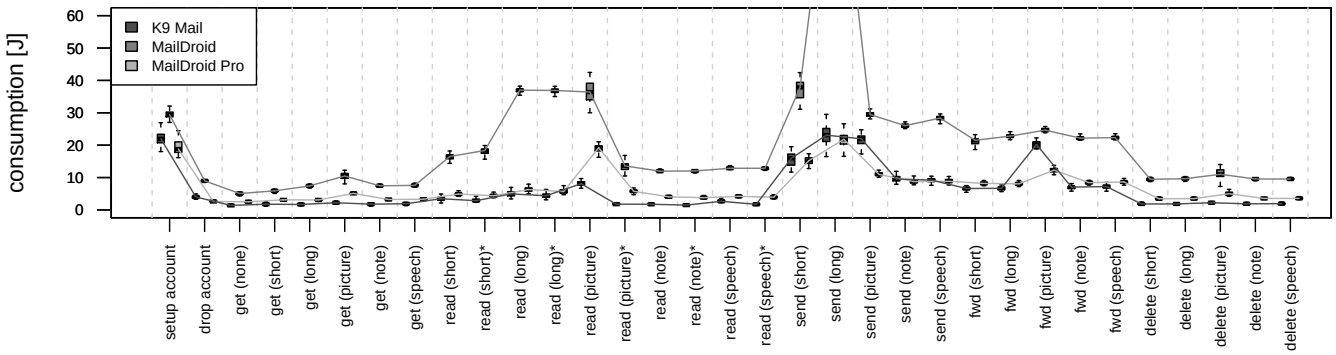
[2]In a Kruskall-Wallis test, the null hypothesis assumes that there are no measurable differences between the investigated applications.



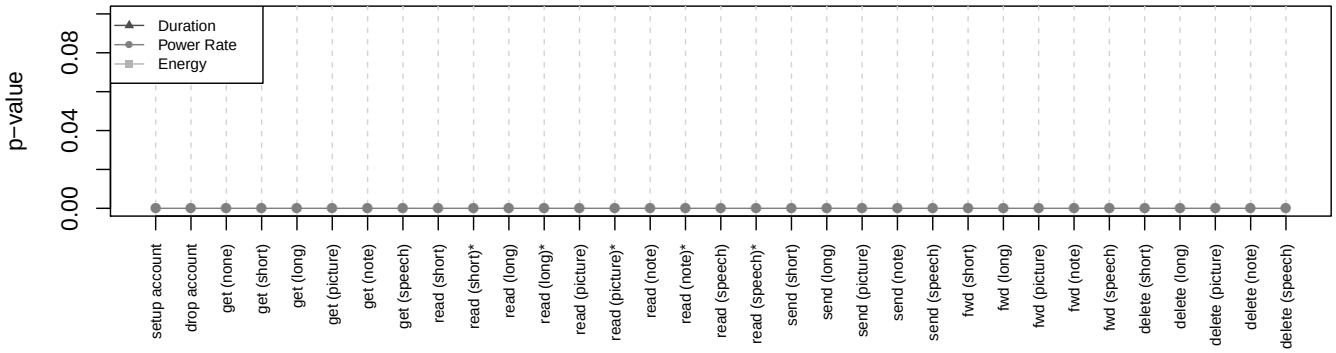
(a) Median execution time.



(b) Median power rates.

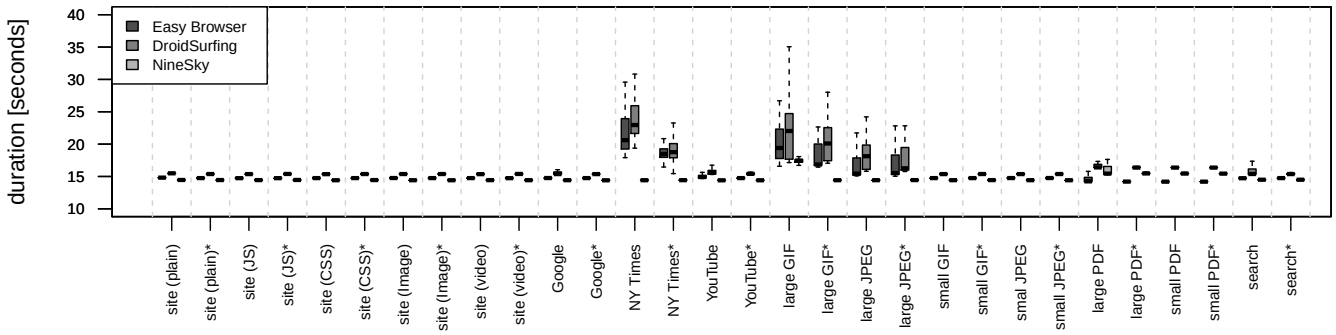


(c) Median energy consumption.

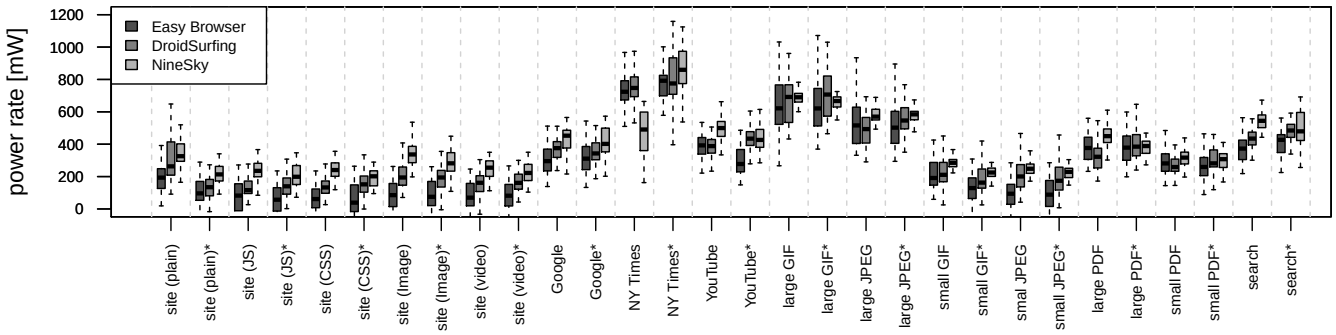


(d) Kruskall-Wallis test.

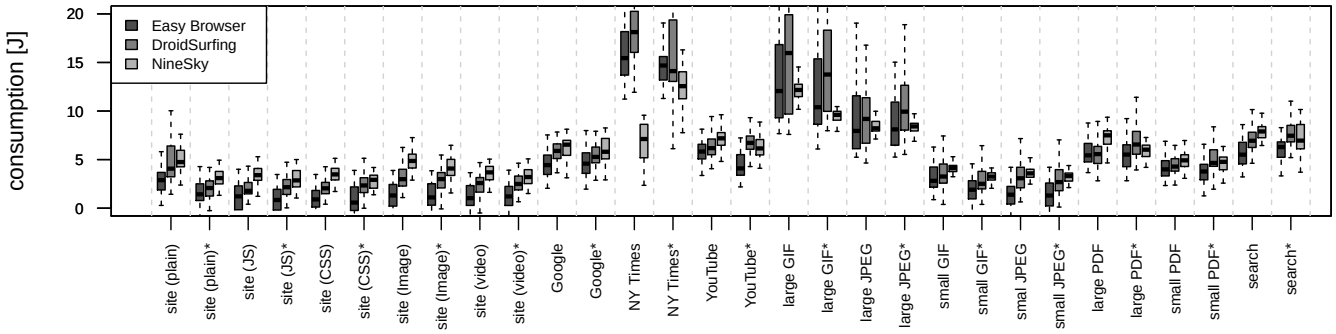
Figure 2: Android email clients' median execution time, power rates and energy consumption for all use cases (* denotes cached mails). Below the p-values for the respective Kruskall-Wallis tests.



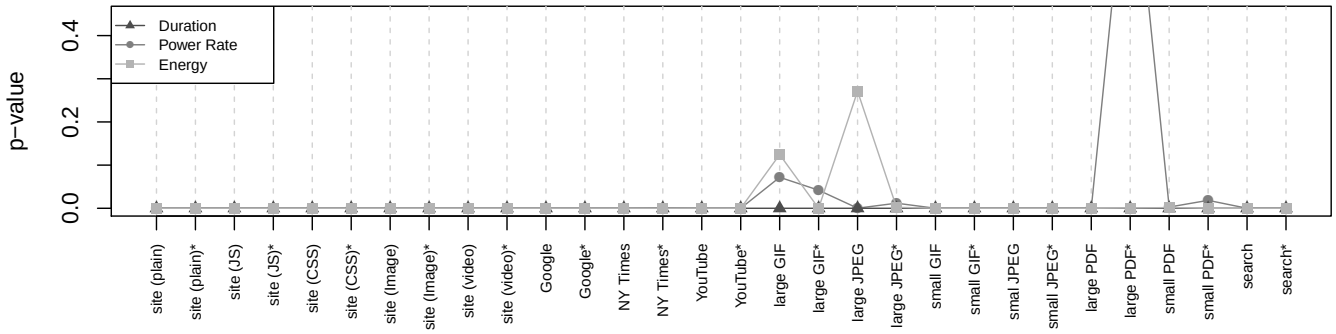
(a) Median execution time.



(b) Median power rates.



(c) Median energy consumption.



(d) Kruskall-Wallis test.

Figure 3: Android web browsers' median execution time, power rates and energy consumption for all use cases (* denotes cached pages). Below the p-values for the respective Kruskall-Wallis tests.

realistic as each test case includes seven seconds for the entering of URLs into the address field.

3.2.2 Tested Applications

We tested the following web browsers:

App	Version	Downloads
Easy Browser	1.1.6	$> 50,000$
NineSky Browser	2.5.1	$> 500,000$
Droid Surfing	1.2.8	$> 10,000$

Besides the enlisted browsers, we also tried to test Google Chrome, Opera Mobile and Firefox. The testing of Google Chrome failed due to a null pointer exception within the app every time we entered an URL from the simulated UI testing code. Thus, we had to exclude Google Chrome from our study. For Opera Mobile we were able to run the tests. However, we were not able to implement the test cases to get notified from the app's UI after the loading of a page completed, as Opera Mobile uses its own UI widgets instead of Android's default widgets. Thus, we got measurement results for Opera Mobile but they did not reflect the real energy consumption of the profiled use cases. That's why we excluded the measured results for Opera Mobile. Finally, for Firefox we were able to run all the use cases. Unfortunately, Firefox timed out for the loading of all tested web pages several times during the test run. Thus, we had to exclude the measured results for Firefox as well as they included large standard deviations and errors. However, in future work we should be able to overcome these problems and to profile Firefox, Google Chrome and Opera Mobile as well.

All browsers have been tested under their default configuration w.r.t. caching and versioning of web pages. For each application we started the profiling with a fully charged battery and disabled all unnecessary services and applications running in parallel. As test device we used the Google Nexus 7. Each test case was profiled 50 times per application. As network communication device we used a local WiFi rooter connected to a 1GBit Internet connection.

3.2.3 Results

Fig. 3 shows the median execution times, power rates and energy consumption for all use cases and applications. A Kruskall-Wallis test led to rejected null hypotheses for all use cases except for the loading of PDFs and large images (cf. Fig 3(d)). This lead to the conclusion that browsers vary in their energy consumption for the same use cases. he absence of differences differences for downloads may be explained by the simplicity of the task to download a file which is probably implemented similarly for all tested browsers.

Comparing execution times, the browsers behave rather similarly (cf. Fig. 3(a)). A major outlier is the NineSky Browser that performs better than the Easy Browser and Droid Surfing for all use cases that include the loading of larges images.

Considering their energy consumption, the browsers behave almost similarly for the specially designed web pages. Surprisingly, the Easy Browser and Droid Surfing behave better than the NineSky Browser, although both include advertisement which the NineSky Browser does not. However, for use cases with longer execution times (due to the increased number of loaded banners) this effect is compensated; which is the case for real web pages and large images. In general Easy Browser requires a less energy than Droid Surfing. The reason might be the difference in their polling strategies for new advertisement banners. The Easy Browser loads a banner once a web page is opened and then every sixty seconds. Droid Surfing in contrast, loads a banner on app start up and then every twenty seconds, which can lead up to three times the banner traffic of the Easy Browser.

More surprising are the results for the caching mechanisms of the tested applications. The Easy Browser does no provide any caching mechanism. Thus, no caching effects are observable. Anyhow, the Easy Browser behaves similarly or even better than the other teste browsers for the loading and reloading of web pages and images. Caching in Droid Surfing leads to reduced execution times for real pages (NY Times, YouTube) and large images. However, energy consumption reduces only marginally. The effect of caching in NineSky Browser is only observable for large images. The loading time of the large GIF is reduced by about five seconds, leading to reduced energy consumption as well. However, for the other tested images, the effect is not observable, as obviously, their loading time is too short for being affected by the caching mechanism. But as it is unlikely that JPEGs loaded on mobile devices exceed the size of 3.2MB, it can be concluded that caching in NineSky Browser has no effect on the loading on JPEG images. In general, the Easy Browser behaves best for the loading of small images, although only marginally better than both Droid Surfing and the NineSky Browser. For the loading of large images the Easy Browser and the NineSky Browser show similar energy consumption, whereas Droid Surfing behaves worse which, again might be caused by its additional advertisement traffic.

For PDF downloads, the execution times and energy consumption of all tested browsers differ only marginally, probably due to the simplicity and similar implementation of this task in the tested applications, as discussed above.

Summing up, the energy consumption of mobile browsers varies while loading the same pages and images. Advertisement banners can increase the browsers' energy consumption, however, other implementations can behave even worse although excluding advertisement. For the loading of images, caching showed no measurable effect on the energy consumption of both Droid Surfing and the NineSky Browser (except for the case of large GIFs) and the Easy Browser behaves even better although it does not provide any caching mechanisms.

4. THREATS TO VALIDITY

In the following, we discuss some threats to validity that should be considered for our measurement results.

USB connection powering: The test cases are executed on the DUT via a USB connection from a test runner PC. Unfortunately, the USB connection provides the DUT with energy during each test run which cannot be disabled while running the tests. Thus, we used a second power meter to profile the power rate at the USB cable during all test runs and added the power rate to the DUT's profiled energy consumption which was rather low for all test runs and only

of small variation ($0.09W$..$0.11W$).

Generalizability for other mobile devices and hardware settings: Our measurements were performed for each case study on one specific Android device, an Asus Transformer TF101 for the email and a Google Nexus 7 for the browser case study. It is very likely that other mobile devices from other vendors, and even other hardware configurations on the same device (e.g., using 3G instead of WiFi) will behave differently and have different power rates while executing the same applications and use cases. We assume that similar differences between the tested applications will be observable, but the total power rates as well as the relative differences between the apps may vary on other devices.

Influences from helper applications: In some cases, Android applications delegate services to other applications via so-called *Intents*. For example, downloaded files are opened by another application, depending on the file type (e.g., an image or PDF viewer). Thus, the energy consumption of this use case depends on the application to open the files as well. Thus, for our tests we designed a dummy application that simply receives the open file intent but does not do anything to exclude this effect from the test runs.

Limited determinism of test cases: For some test cases, their determinism is limited. First, network communication can vary for each test run as data packages can be lost and have to be resent again. Besides, the used WiFi router and network hardware can influence the throughput and thus, the application's runtime behavior as well. Furthermore, the content of real web pages such as Google or New York Times can change between the individual test runs (e.g., by new news to appear). We tried to limit this factor by shuffling the individual test runs of all AUTs. However, these factors can influence our test results. Nevertheless, we assume our results to be quite representative as we executed each use case 50 times for each AUT which was confirmed by the performed Kruskall-Wallis tests.

Representativeness of test cases: The use and test cases of our study have been designed manually and it can be questioned whether or not they do represent realistic and average usage scenarios of the test applications. Thus, to get a better impression on how users use their applications and to derive more realistic and appropriate scenarios, we are currently conducting a user study evaluating representative usage scenarios for Android applications such as mail clients and web browsers.

Influence of application settings: Some of the AUTs allow several settings to configure them w.r.t. the users' needs (e.g., for some of the browsers, caching can be enabled or disabled). We tested the applications with their default settings only. Of course, adapting their settings could have led to more energy-efficient results. However, we argue that most users will use applications under their default settings only. Thus, their default settings should be considered for the comparative test runs. Future work could evaluate the impact of altering the settings w.r.t. energy efficiency.

Influence of testing code: Of course, the code used to run our workloads while profiling the applications' energy consumption causes additional energy consumption to be executed. However, this energy consumption can be considered as similar for all tested applications, as similar sequences of UI commands have to be executed for each AUT. Furthermore, as we are more interested in relative than absolute results, this energy consumption can be considered as negligible.

5. RELATED WORK

The most important related work was done by Pathak et al. [5, 6] who investigate Android applications' energy consumption while executing typical usage scenarios (e.g., web browsing or chess gaming). Although they profile applications' energy consumption—in contrast to our work—they do not compare energy consumption for applications from the same usage domain, but focus on the identification of energy bugs within the profiled applications.

Thiagarajan et al. [8] investigate the energy consumption required by mobile devices to download, render, and display web pages during web browsing. They show that web pages not especially designed for mobile devices can cause large communication overhead as well as parsing and rendering of large JavaScript and CSS files can significantly increase energy consumption. They derive a set of recommendations to design web pages being more energy-efficient for mobile web browsing.

Xiao et al. [12] investigate the energy consumption of different mobile YouTube use cases executed on Nokia Symbian phones. They define use cases for upload, download and replay of videos and show that energy consumption via WCDMA communication is about 1.5 times more costly than WiFi communication.

Palit et al. [4] present a methodology to profile average energy consumption of mobile applications. They focus on profiling average power rates for typical application use cases executed on different mobile platforms (e.g., Android and Blackberry) and identify parameters of hardware configurations influencing the devices' energy consumption [3]. Our approach in contrast targets to compare the energy consumption of similar applications on the same mobile devices.

In a study on European mobile network traffic, Vallina-Rodriguez et al. show that today up to 80% of Android network traffic is used for advertisement services and that efficient caching could reduce the energy consumption of advertisement-included applications by up to 50% [9]. In this paper we confirmed these assumptions by showing that advertisement can increase energy consumption of mobile email applications by up to 75%.

6. CONCLUSION

In this paper we presented two case studies, comparing mobile applications of the same domain w.r.t. their energy consumption. We defined two microbenchmarks for emailing and web browsing.

We compared three mail clients (K-9 Mail, MailDroid, and MailDroid pro) and showed that advertisement can increase the their energy consumption by about 75%. Besides, we identified only minor differences in use cases' energy con-

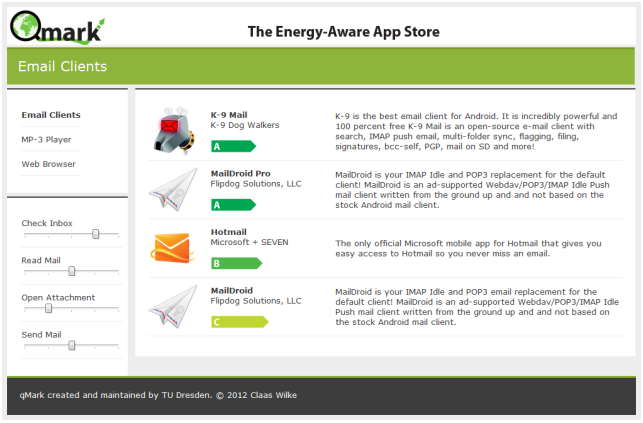


Figure 4: Prototypical implementation of an energy-aware app store.

sumption for K-9 Mail and MailDroid pro caused by more efficient UI navigation.

In a second study we compared three web browsers (Easy Browser, Droid Surfing, and NineSky Browser) for loading web pages as well as images and downloading PDF files. We identified major differences in the browsers' energy consumption which in some cases are caused by advertisement banners. However, both the Easy Browser and Droid Surfing behaved better than the NineSky Browser for some use cases although including advertisement. Furthermore, we showed, that caching has no significant influence on the loading of web pages, PDF files and images (except for the loading of a large GIF file).

The microbenchmarks defined in this paper can be reused to compare the investigated applications with further applications from the same domain. Besides, we plan further cases studies such as MP3 and podcast players. For future work we are also working on an app store providing energy labels as further guidance for users looking for energy-efficient applications (cf. Fig. 4) [11].

7. ACKNOWLEDGMENTS

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

Furthermore, we would like to thank Sebastian Betker, Sebastian Herrlich, Kevin Seppelt, Lukas Siedel, Robin Unger, and Elisa Zschorlich for their help during a first implementation of the test cases used in this study.

8. REFERENCES

[1] Android Developer Tools. Website, Sept. 2012. http://developer.android.com/guide/developing/tools/adt.html.

[2] Google Zeitgeist 2011. Website, Sept. 2012. http://www.googlezeitgeist.com/en/top-lists/global/google-news-fastest-rising.

[3] A. Abogharaf, R. Palit, K. Naik, and A. Singh. A Methodology for Energy Performance Testing of Smartphone Applications. In *Proceedings of the 7th International Workshop on Automation of Software Test (AST2012)*. IEEE, 2012.

[4] 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. ACM, 2011.

[5] 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, 2012.

[6] A. Pathak, A. Jindal, Y. C. Hu, and S. P. Midkiff. What is keeping my phone awake?: characterizing and detecting no-sleep energy bugs in smartphone apps. In *Proceedings of the 10th international conference on Mobile systems, applications, and services (MobiSys '12)*, pages 267–280. ACM, 2012.

[7] A. Shye, B. Scholbrock, and G. Memik. Into the Wild: Studying Real User Activity Patterns to Guide Power Optimizations for Mobile Architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 168–178. ACM, 2009.

[8] N. Thiagarajan, G. Aggarwal, A. Nicoara, D. Boneh, and J. Singh. Who killed my battery?: analyzing mobile browser energy consumption. In *Proceedings of the 21st international conference on World Wide Web*, pages 41–50. ACM, 2012.

[9] 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.

[10] C. Wilke, S. Richly, S. Götz, C. Piechnick, G. Püschel, and U. Aßmannn. Comparing Mobile Applications' Energy Consumption. In *SEGC - Software Engineering Aspects of Green Computing Track at the 28th ACM Symposium on Applied Computing (SAC2013)*. ACM, 2013.

[11] C. Wilke, S. Richly, G. Püschel, C. Piechnick, S. Götz, and U. Aßmannn. Energy Labels for Mobile Applications. In *Proceedings of the 1st Workshop for the Development of Energy-aware Software (EEbS 2012)*, volume 208 of *Lecture Notes in Informatcis (LNI)*. Gesselschaft für Informatik, 2012.

[12] Y. Xiao, R. Kalyanaraman, and A. Yla-Jaaski. Energy consumption of mobile youtube: Quantitative measurement and analysis. In *Proceedings of the 2nd International Conference onNext Generation Mobile Applications, Services and Technologies (NGMAST'08)*, pages 61–69. IEEE, 2008.

[13] Z. Zhuang, K. Kim, and J. Singh. Improving energy efficiency of location sensing on smartphones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 315–330. ACM, 2010.