Preserving energy resources using an Android kernel extension: a case study

Conference Paper · May 2016

DOI: 10.1145/2897073.2897124

5 authors, including:

Luis Corral
Tecnológico de Monterrey
44 PUBLICATIONS **516** CITATIONS
SEE PROFILE

Ilenia Fronza
Free University of Bozen-Bolzano
75 PUBLICATIONS **530** CITATIONS
SEE PROFILE

Nabil El Ioini
Free University of Bozen-Bolzano
85 PUBLICATIONS **577** CITATIONS
SEE PROFILE

Andrea Janes
Free University of Bozen-Bolzano
75 PUBLICATIONS **895** CITATIONS
SEE PROFILE

Preserving Energy Resources using an Android Kernel Extension: a Case Study

Luis Corral
Instituto Tecnológico y de Estudios Superiores de Monterrey, Universidad Autónoma de Querétaro
E. Gonzalez 500
76130, Queretaro (Mexico)
lrcorralv@itesm.mx

Ilenia Fronza
Free University of Bozen/Bolzano
Piazza Domenicani 3
39100 Bolzano, Italy
Ilenia.Fronza@unibz.it

Nabil El Ioini
Free University of Bozen/Bolzano
Piazza Domenicani 3
39100 Bolzano, Italy
Nabil.ElIoini@unibz.it

Andrea Janes
Free University of Bozen/Bolzano
Piazza Domenicani 3
39100 Bolzano, Italy
Andrea.Janes@unibz.it

Peter Plant
Vertical-Life
Otto v. Guggenbergstrasse
39042 Brixen, Italy
peter@vertical-life.info

ABSTRACT

The autonomy of mobile devices is a requirement of utmost importance for end users. The autonomy is strongly related to the capacity of the built-in battery, in combination with the technical capabilities and the demand of energy of the diverse components of the device. As mobile equipment becomes more powerful and demanding, the need to find ways to optimize the overall energy consumption of the system grows as a critical research path. Software, as an instrumental component of a mobile system, it is also an attractive target to deploy energy saving approaches. Several techniques of software-based energy aware strategies have been explored, including solutions placed at operating system, compiler and application level. In this paper, we present an energy saving strategy at operating system level. Our approach is implemented in the form of kernel extensions that assess the status of the device, and enable economic profiles without user intervention. Our experiments showed that the power management kernel extension was able to significantly extend the battery runtime by 70% to 75%, at the expense of impacting the experience of the user with an estimated performance degradation of 20% to 30%.

CCS Concepts

•**Human-centered computing** → **Mobile computing;**
•**Hardware** → **Power estimation and optimization;**

WOODSTOCK '97 El Paso, Texas USA

DOI: 10.475/123_4

Keywords

Android, Energy, Kernel, Mobile

1. INTRODUCTION

Smartphones have become a central aspect of our daily life. In the past years, mobile devices have growth in capacity and power thanks to the integration of complex hardware. However, computing power, device performance, and resource usage is strongly related to higher energy consumption.

The autonomy of mobile technologies relies on the available power source and the way such power is managed. Therefore, it is important to research on power management strategies that may reduce the power intake of the device, and consequently extending the battery runtime. A way in which this can be achieved is by designing and implementing more efficient hardware, with electronic components of minimum energy footprint. A second approach is leveraging the capacity of Software as an instrumental component in the overall performance of the mobile device. In this way, one may modify the software running on a device to attempt to minimize the demands of the system, reducing in consequence the energy consumption. The necessary modifications can be implemented either at user space, application, or operating system levels.

In this study, we examine a low-level energy saving approach implemented in the form of a power management extension of the Android OS kernel[1]. This extension monitors the device's battery level, and depending on the current power state, it applies pre-defined settings that lower the power demand for the operation of the device. The lower the current battery power is, the more restrictive the applied settings are. In addition to the implementation of the power management extension, this study analyzes as well how the application of energy-aware settings affects the overall performance of the mobile device.

[1]https://www.android.com/

Android is a mobile operating system based on the Linux[2] kernel. It is designed to primarily run on mobile devices with touchscreens such as smart phones, tablet computers and smart watches [2]. It has been developed to run on a variety of different devices produced by manufacturers all over the world. This is possible thanks to the layered architecture of the operating system that allows for device-specific tailoring and customizations. Android's Linux kernel acts as the interface between the user space and the hardware components, operated through drivers. The device drivers controlling the hardware can be configured as modules, which can be loaded and unloaded while the system is running [4].

An important task of the kernel is to control the CPUs; this is done using the so-called CPU governor. A CPU governor controls how the CPU raises, and lowers its frequencies according to the current workload [1]. To carry out this task, there are different CPU governors designed for different purposes, such as performance or energy efficiency.

The goal of this work is to create a power saving system running at the kernel level, avoiding the overhead associated to sending messages from user space all the way down to the modules that interface with the hardware. All the necessary settings to reduce the power consumption of given hardware components can be directly set in the specific drivers at kernel level. These settings should not be set only once, but rather they should be induced progressively: as the battery level decreases, the settings will change accordingly to preserve more battery, especially at the lower charge stages. This allows to extend the battery runtime at the moment the device is almost out of charge at the expense of system performance. Since a way to preserve the energy of a mobile device is by inducing reductions in the performance of the involved hardware components, it is also important to measure the trade-off in terms of performance and usability.

The rest of the paper is structured as follows: Section 2 discusses the related work; Section 3 details our approach to implement an energy aware kernel module for Android; Section 4 describes the experiments that were conducted to investigate the impacts on the performance and the battery runtime of a mobile device resulting by the usage of the power management system, as well as the threats to validity; Section 5 draws conclusions and provides directions for further research.

2. RELATED WORK

Analyzing and improving the battery consumption of a mobile computing system at software level is a complex task, which resulted in a variety of studies pursuing this objective through different targets and approaches.

In general, this research field can be divided into two groups. The first group of studies focuses on the measurement of the battery consumption with a focus on the different hardware components of a mobile device. The goal is not to reduce the energy consumption with the implementation of a power saving system, but to understand first *how* the battery is drained. The second group of research works is related to studies trying to modify the software running in the mobile system with the goal of extending the battery runtime. In the rest of this Section, we discuss the most relevant software based approaches to reduce energy consumption.

Kraiman et al. [11] designed an intelligent modular power management system suitable for any mobile platform. This system is based on the Advanced Configuration Power Interface (ACPI)[3] architecture and it includes mechanism to define the most efficient power management strategy for a particular mobile device. To test the power consumption of the different hardware components a testing framework has been created. This framework allows to estimate the energy consumption of the built-in hardware devices based on a linear regression model. The results have shown that the implemented system is able to reduce the overall power consumption and that the testing framework allows to detect energy resource leaks in applications.

The Android kernel has built-in power saving mechanisms and systems such as the ACPI used in the previous study. In another research by Motlhabi [13] these default power saving tools were subject of a comparison between the Android kernel and the parent kernel from Linux. This paper gives detailed insights the different power saving mechanism such as wake-locks, power states and the ACPI.

Another study by Bala and Garg [3] tries to reduce the energy consumption by the implementation of a learning engine. This engine gets as input a set of collected data resulting from the monitoring of the users behavior. The result shall be a customized power profile adapted to the needs of a specific user, allowing to save energy by predicting and reducing the minimum required resources. The same kind of concept was applied in a study conducted by Datta et al. [6]. Their paper describes the approach of deriving power saving profiles based on the usage patterns of the Android devices. Through the monitoring of specific hardware components data are collected, these data are used to generate multiple usage patterns over time and space, which define the user contexts. Each profile contains several system settings and intelligently preserves battery charge. The results have shown that the implemented learning engine can increase the overall battery runtime by 82%.

Ellis [7] published a research paper focused on high level power management. The study states that high level software should be more involved in power management to impact positively the power consumption. This is achieved through the implementation of a power-based API that allows a partnership between applications and the system in setting the energy consumption policy. The conducted experiment has shown promising results even though the system interface inhibits the full exploitation of power saving opportunities.

Another technique for high level power management is the code transformation of applications. This concept aims to optimize the energy conservation of a mobile device along with a relative small performance degradation. In particular, the approach (e.g., presented in [8]) proposes app modifications that increase the device idle times and inform the operating system about the length of each upcoming period of idleness. The results have shown that such transformations can reduce disk energy consumption from 55% to 89% with a negative impact on performance of only 8%.

Finally, in a work that precedes the present one, Corral et al. [5] compared available Android kernel-based modifications evaluating their impact on battery runtime. In this paper, we use the term "battery runtime" to describe the time a

[2]https://www.linux.com/

[3]http://www.uefi.org/acpi/specs

device can run on a fully charged battery and to distinguish it from "battery lifetime", which is also used to describe the life span of a battery from its manufacturing until its disposal. They performed four performance modifications as well as evaluation tests on each kernel, monitoring the battery consumption in background. In addition, a general performance test was run to see the impact of the applied kernel modifications to the overall performance of the optimized device. The results show that kernel level enhancements can improve battery runtime, showing in selected cases a positive impact in the performance of the device. According to the tests in this previous work, the analyzed custom kernels can reduce the battery consumption up to 33% for isolated tasks, improving the general performance of the device by up to 16%.

3. IMPLEMENTING AN ENERGY AWARE KERNEL MODULE FOR ANDROID

As stated in the introduction, the aim of this work was to create an Android kernel extension that monitors the device's battery level, and depending on the current power state, it applies pre-defined hardware settings that lower the power demand for the operation of the device.

The implementation of this study was performed on a LG Nexus 5 (Hammerhead) device[4], operated by the Android Version 5.01 Lollipop OS, which is based on the 3.4 `lollipop-release` kernel.

The first step was to implement the power management system running as a module at the kernel level. Since the Android Kernel source code is smaller than 1 GB, we decided to create patch files which can be applied directly to the kernel sources.

The kernel level power saving system is composed of a set of modules, hardware drivers, and a power management module itself (not to be confused with the Android built-in power management engine). The power management module controls the different involved device drivers by inducing specific settings, which aim to reduce the performance of the controlled hardware component, pursuing a reduction in the battery consumption. As the battery level of the mobile device decreases, the settings induced to the drivers are continuously set to higher power preservation levels, which results settings that, while are aggressively economic, they also result in a progressive performance degradation.

The principle behind the power management module is illustrated in Figure 1: our brand-new power management module acts as the controlling unit of the power saving tool. The primary duties of this module is to check periodically the energy level by communicating with the battery device driver and to activate a given power class according to the current battery capacity. A power class is a group of settings for the different device drivers.

We consider the following settings to influence the current power consumption in each power class:

- maximum screen brightness (0-255);
- vibration strength (0-30);
- maximum CPU frequency (960.0-1958.4);

[4]http://www.lg.com/us/cell-phones/lg-D820-Sprint-Black-nexus-5

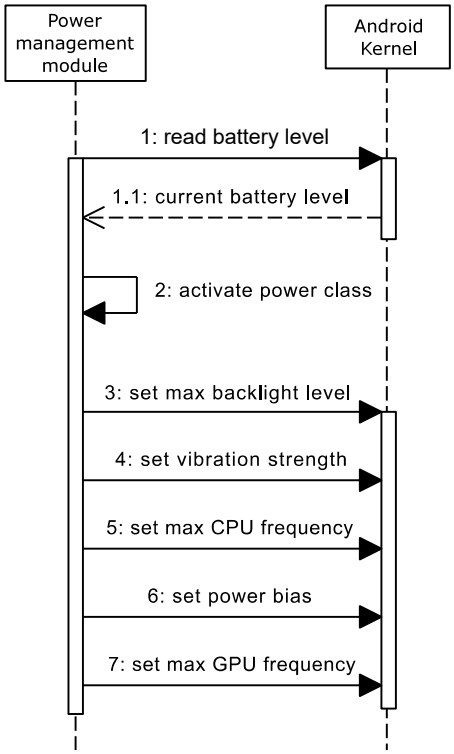

Figure 1: Power management module communication.

- powersave bias (0-1000): setting this value higher will "bias" the governor — the driver in Android that regulates the CPU frequency — toward lower frequencies. The ondemand governor will scale the CPU to a frequency lower than its "target" speed according to this value;
- maximum GPU frequency (0-5).

We defined ten power classes, each representing a range of power levels. A power class can be identified by the range it stands for, e.g., power class P8 represents all battery levels ranging from 80 to 89. The selection of ranges including ten units, respectively eleven units for the 9^{th} power class should balance the number of updates done to the different parameters. The power classes and their characteristics are described in Table 1.

The choice of evaluating a power management module with ten power classes (and their respective settings) is arbitrary, but our selection was based on the following considerations:

- having a too high number of power classes results in frequent parameter updates: every time a new battery level is detected, the new power class settings have to be transferred to the different hardware components. Considering that every update is associated to a cost in battery consumption, it is advantageous to keep low the number of power classes;
- having a too low number of power classes results in inefficiencies because some available values would need to be skipped. In addition to this, the procedure of

Table 1: Settings for the different power classes.

	Max screen brightness (0-255)	Vibration (%)	Max CPU frequency (GHz)	Powersave bias (‰)	Max GPU frequency (0-6)
P9	default	default	default	50	0
P8	150	default	1958.4	50	1
P7	100	default	1728.0	100	1
P6	40	30	1574.4	100	2
P5	30	30	1267.2	100	3
P4	20	30	1190.4	200	3
P3	10	0	1190.4	200	4
P2	5	0	1036.8	300	4
P1	3	0	1036.8	300	5
P0	1	0	960.0	300	5

how to associate a given range to a power class would be more complex;

- the limited amount of available frequencies for GPU and CPU settings reduce the benefit of more than ten power classes.

The power saving system includes a set of device drivers that are already present in every stock Android kernel. Some of them had to be modified so that the configuration parameters already present inside the driver module become accessible from outside.

The following drivers are part of the power saving system: the screen backlight driver, the vibration driver, the graphics processing unit (GPU) driver and central processing unit (CPU) driver.

The power saving system is composed of five components:

- the overall Power management module controlling the other four components;
- the power module, reading the current battery level;
- the backlight module (based on the Texas Instruments LM3630A Backlight driver chip[5]), to set the back-light level;
- the CPU module, to set the maximum CPU frequency and the powersave bias;
- the GPU module (based on the kernel graphics support layer provided by the Qualcomm Adreno GPU[6]), to set the maximum GPU frequency.

Every component represents a kernel module which is either loaded into the kernel at run time or once at boot time. The power management module communicates with all other components by either retrieving or providing values via function calls. The internal structure of the power management component includes a timer structure which runs periodically to take the appropriate actions. Unfortunately, the native libraries used to build the kernel module do not provide timer structures that are able to run along a non-atomic environment that uses schedules, thread sleeps, and Mutex locks. Therefore, a simple timer structure has been built, running in its own thread and calling the a `read_battery_state()` function repeatedly. The delay of the function calls is achieved by a thread sleep of 2 minutes.

The power management module also defines all available power classes and the associated values of each class as defined in Table 1. Every power class is represented by its own C file and has the same basic structure. The selection of the power class is done by a simple calculation on the basis of the current battery level which is divided by 10. The resulting value is the identifier of the current power class. For example, the current battery level of a given smartphone is 55%, when applying 50/10, we get 5, which establishes that P5 as the current power class. In contrast to the hardware driver modules, loaded into the kernel at boot time, the power management module can be loaded at any time once the system has booted. This has a practical reason since in this way, the module can be modified during development without rebuilding the whole kernel, which can take up to 20 minutes.

Once the module has been loaded into the kernel, all data structures and variables are setup that are used at a later point in execution time. The kernel modules include the possibility to be launched with parameters. In case a numeric value grater or equal than 0 has been passed, the periodic timer is not going to be started but the power class associated to the passed parameter value is directly activated (see Figure 2). Within the scope of a timer iteration, the battery level is red, and it passed to the routine that checks whether a new power class needs to be activated. In case the power class is already activated, the power class gets updated. After the activation, the update is executed, and the timer thread is put to sleep for the defined amount of time. Once the thread wakes up again, a new timer iteration will start.

The power management module and the modified Android kernel are available on https://github.com/pplant/powerm_kernel. The source code can be built using the make files included in the project folders. We built the kernel and the module on Ubuntu 14.10[7].

Before building the kernel and module code, it is required to setup the tool chain to build the code for an ARM processor architecture. The building toolchain we used is `arm-eabi-4.6`. The kernel building process will result in the creation of the `zImage`, which contains the kernel executable. This file is embedded into the so-called `boot.img` file. To use the kernel, the previously created file needs to be flashed into the device by using the Android Bootloader and Fastboot (a tool that is part of the Android Software Development Kit[8]). Once the kernel is running in the device, the power management module can be built, which creates a kernel object file (with extension `.ko`). This is the file that is then transferred into the device using the Android Debug Bridge[9]. The last step is to load the `.ko` file into the kernel using the adb `push` command.

Figure 3 illustrates the deployment diagram of our prototype.

[5]http://www.ti.com/product/lm3630a
[6]https://developer.qualcomm.com/software/adreno-gpu-sdk/gpu
[7]http://www.ubuntu.com/
[8]http://developer.android.com/sdk/index.html
[9]http://developer.android.com/tools/help/adb.html

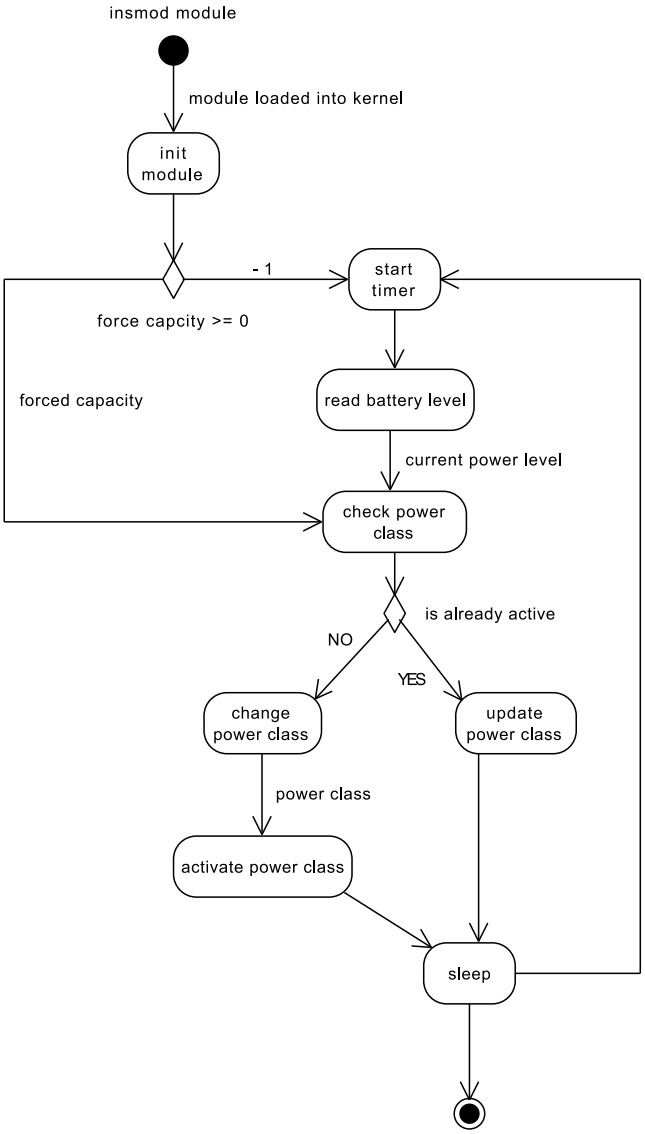

Figure 2: Basic execution flow.

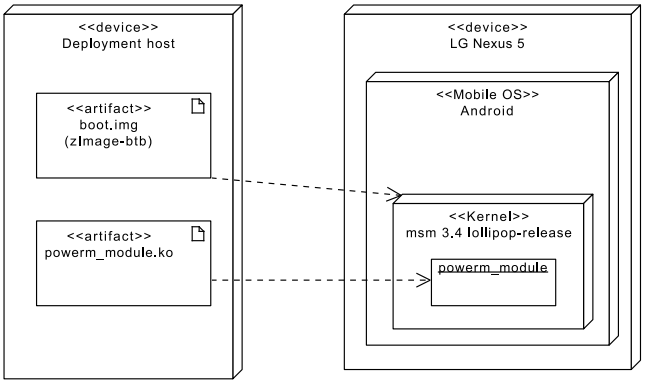

Figure 3: Power management module deployment.

4. EVALUATION

The following step is the investigation of the impact of the power management system described in Section 3 on the performance and the battery runtime of a mobile device.

The developed system reduces the energy consumption by progressively decreasing the resources available to different hardware components, which, as a consequence, also reduces the performance of the smart phones. Therefore, we performed two experiments to investigate both aspects: the battery gain and the performance loss.

4.1 Experiment 1: Performance

This experiment has the goal to investigate the impact of our power management system on the performance. To measure this trait, we focused on CPU and GPU performance, since vibration and LCD back-light brightness are more related to usability.

To measure the impact of the power management system on the performance of a smart phone, we used "Geekbench 3"[10] to evaluate the performance using simulated workloads. While the simulated workloads were executed on the phone, we switched the mobile phone to the ten different stages defined by the power classes (see Table 1). Then, we compared the measured performance on each power class with the values measured on a smartphone (with the same hardware and software) not running the power management system. The experiment has been planned and executed according to the experiment process described in [16] using the "one factor with two treatments" structure.

We used three new LG Nexus 5 devices in this experiment, all the phones were acquired at the same time, to minimize the source of variation induced by worn components. To reduce the possibility of occurring background tasks and operations that are not related to the benchmark testing, all installed applications except the two benchmark tools were either disabled or removed.

Before each experiment, we switched off the connectivity services such as radio, 2G, 3G, Bluetooth and GPS. Unfortunately, it was not possible to switch off the wifi during the CPU benchmark tests using the "Geekbench 3" tools, since this application requires a working Internet connection once a test has been completed. During the whole experiment the devices were attached to a power source to guarantee a constant power supply.

First all GPU benchmark tests have been conducted, followed by the CPU benchmark tests. The three devices executed one run without treatment and one run with the treatment.

To measure the performance of the CPU, we collected the data throughput in megabyte per seconds during SHA1 encrypting [10] and the megaflops (million floating point operations per second) during the application of a blur filtering in an image. To measure the performance of the GPU, we collected the obtained average frames per second as we played an animation.

The design of this experiment applies the general design principles described in [16]:

- randomization: to comply with this principle, we randomized the order of the applied treatments;
- blocking: during the experiment execution there is a possibility that other processes falsify the final results. Therefore, all applications except of the benchmark tools have been removed;

[10]http://www.primatelabs.com/geekbench/

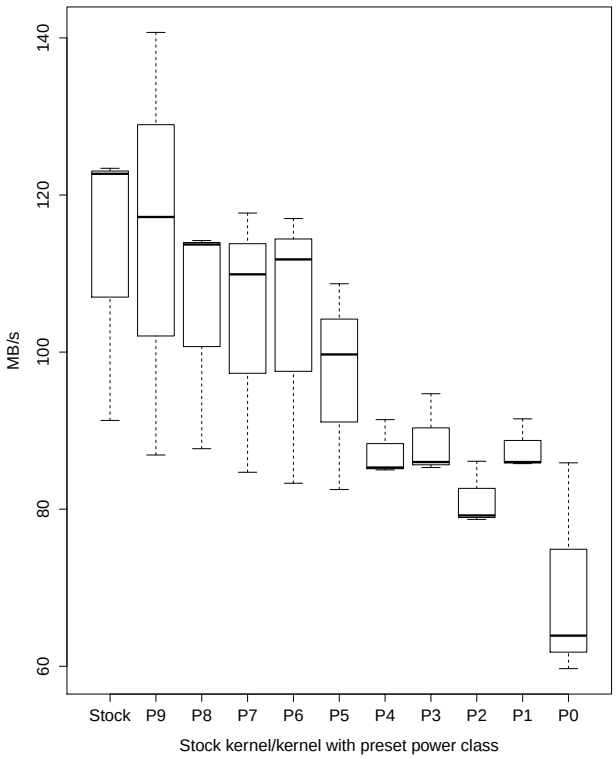

Figure 4: Boxplots of the performance results for the CPU performance test using SHA1 encrypting.

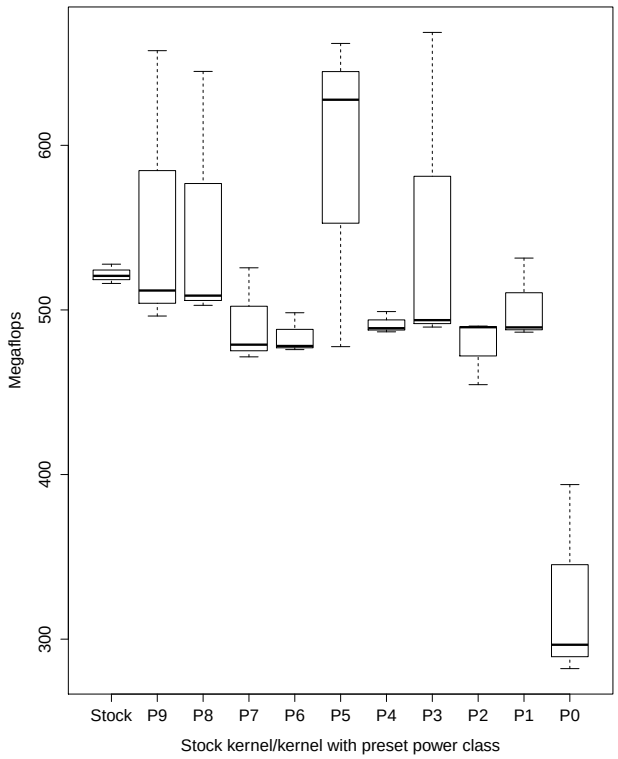

Figure 5: Boxplots of the performance results for the CPU performance test using blur filtering.

- balancing: this principle is respected by balancing the runs between the different devices, such that every device experiences the same number of executions. In addition to that, both treatments are applied the same amount of times.

The results of the performance experiment for the CPU are depicted in Figures 4 and 5.

In general, the results of this experiment suggest that the performance is negatively affected when the power management system is running. All observed variables except blur filtering have shown a downward trend with the simulated decreasing power level.

The data resulting from the SHA1 encryption reveal that the data range covered by the values with the treatment is larger. At the beginning the untreated and the treated device were behaving similar, but along with the decrease in hardware resources, the algorithm throughput decreased as well.

Comparing the behavior of the algorithm throughput in Figure 4 along the different power classes, it seems that during the upper power classes, ranging from P9 to P6 the values are decreasing slowly. Even though the maximum available CPU frequency has changed from 2265.6 GHz in P9 to 1574.4 GHz in P6 and the Power save bias from 50‰to 100‰, the algorithm seems to have enough resources to keep a rate over the 100 MB/sec. The algorithm seems to be stronger affected between power class P6 and P4. The reason for this is a further reduction of the available resources (see Table 1).

The values of the power classes P3 and P1 are showing a abnormal behavior by a increased value with respect to their predecessor. This phenomena can be retraced to the settings done in P3 and P1 which are not changing in comparison to their predecessor.

The blur filtering algorithm does not reveal a clear trend, since the results seem to vary independently of the decreasing battery level. The comparison of the power classes in Figure 5 show a slight downward trend including values which are breaking the order (P5, P3 and P1). The algorithm performance seems to decrease whenever the Powersave bias is reduced (P7, P4 and P4, see Table 1). The reduction of the maximum CPU frequency seems not to have a direct impact on the algorithm throughput.

The result of the performance experiment for the GPU is depicted in Figure 6.

The GPU benchmark test results suggest that running the power management system reduces the animation quality in terms of displayed frames per second. The comparison of the treatments observing the average frames per second shows a decreasing trend, whereas the values are constantly decreasing.

The interpretation of the results of this experiment indicate that the application of the power management system decreases the performance of the device in terms of CPU and GPU. Among the different measurements, the performance reduction spans between 18% and 30%.

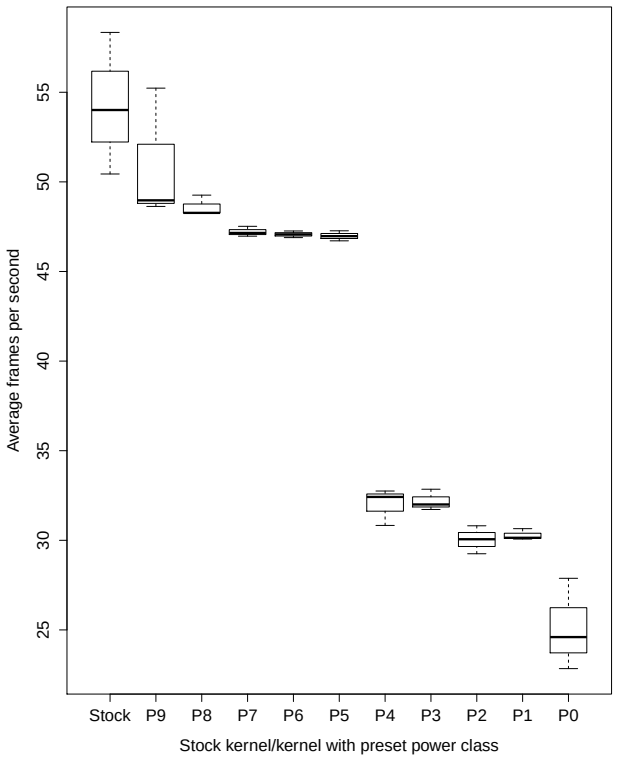

Figure 6: Boxplots of the performance results for the GPU performance test observing the average frames per second.

4.2 Experiment 2: Battery runtime

The second experiment was performed to investigate if the developed system has a positive effect in the battery runtime, and if it has it, how longer a smartphone can run with one complete battery charge compared to a mobile device not running the kernel extensions. Like the previous experiment, this exercise has been planned and executed according to the experiment process described in [16].

To create a constant workload draining the battery, we built a small Android application which simulates different scenarios, using the available phone resources.

The advantage of using a battery draining application is to reduce the run times to completely discharge the device's battery. Moreover, shorter cycles lower the possibilities of confounding factors which could negatively affect the final results. Another benefit is that such an application allows to create a constant workload over multiple runs that can not be achieved in a real world situation.

The battery drainer application is composed of four tasks including three different 3D animations, a SHA1 encryption algorithm, a blur filter algorithm, and a vibration task. This app targets to keep all those components busy, which are also included in the power management system.

The three animations create an elevated workload mainly targeting the graphics processing unit (GPU). Each animation runs for two minutes, showing one or more objects with different surfaces, textures and details.

The SHA1 algorithm and the blur filtering algorithm are for targeting the central processing unit. The first algorithm is taking as input a 150 MB text file containing random words. This file gets red line by line and encrypted by the SHA1 algorithm. The second algorithm continuously applies a blur filter to a 4000 $\times$ 4000 pixel image for about four minutes.

The last task lets the device vibrate for one minute in intervals of one second.

Running all tasks one after each other creates a workload which increases the temperature of some hardware components. To avoid overheating leading to a reboot of the system, every task is followed by a one minute break.

During the test runs of this experiment, we logged the time required to completely discharge the battery of a given device. This measurement allows to compare the different tests runs of different treatments, but also of the same treatment. Moreover, we also logged the current battery level every three minutes. This makes possible to investigate the behavior of the battery consumption applying the two treatments. In addition to that, it is possible to observe the discharging behavior influenced by the settings of the different activated power classes (see Table 1) whenever the power management system is running.

The devices involved in this experiment are two LG Nexus 5 smart phones, acquired at the same time. Also in this experiment, we applied the principles of randomization, blocking, and balancing [16].

Before the experiment was conducted, both involved devices were prepared by removing all installed applications to reduce the occurrence of unwanted background tasks to a minimum. In addition, all connectivity services such as WiFi, Bluetooth, 3G, 2G and GPS were switched off. Before each test runs, the devices were completely recharged using the battery charger shipped with the Nexus 5 devices.

Every run began by starting the battery drain application, which collected all the data of interest. Depending on the treatment, the power management system needed to be started. After each test run the resulting logs file were transferred and archived, before the device was prepared for the next iteration.

We performed 60 experiments in total, distributed between two treatments and two devices. The results of the obtained runtimes with and without treatment are depicted in Figure 7.

The total run times with the running system show that on both devices the run times are typically ranging between 28.300 sec ($\sim$7.9 hours) and 29.000 sec ($\sim$8 hours) on device A, and on the other device B between 27.000 sec ($\sim$7.5 hours) and 27.500 sec ($\sim$7.6 hours). The values resulting from the cycles without the treatment span between 15.500 sec ($\sim$4.1 hours) and 17.000 sec ($\sim$4.7 hours) on device A, while on device B the run time without treatment ranged from 14.800 ($\sim$4.0 hours) sec to 16.600 sec ($\sim$4.5 hours).

Figures 8 and 9 show the discharging processes of the different executions.

On both devices we can observe that the test runs including the power management system initially behaves similar to the executions without the treatment. This trend continues until a battery capacity of 65% is reached. At this point, the activated power classes include more restrictive settings that slow down the discharging process. In the meantime, the energy consumption of the executions not

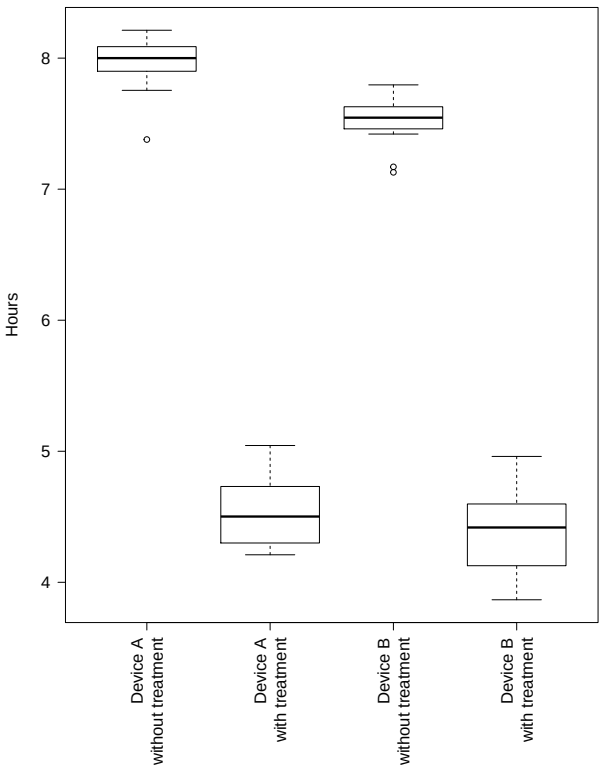

Figure 7: Boxplots of the battery runtime for the two tested devices with and without power management in hours.

running the power saver tool proceeds to decay constantly until the energy resources are completely exhausted. The test runs with the treatment constantly decrease the performance of the device which is progressively slowing down the discharging process. The performance settings reach their minimum level in the two lowest power classes.

We tested the normality of the obtained battery runtimes (i.e., the battery runtimes with and without treatment for device A and B) using the Shapiro-Wilk normality test [15], which is recommended as the best choice for testing the normality of data [9]. The null hypothesis for this test is that data are normally distributed; if the chosen alpha level is 0.05 and the p-value is less than 0.05, then the null hypothesis that the data are normally distributed is rejected. In three cases, results do not allow us to reject the null hypothesis that our samples come from a population which has a normal distribution (p-values=0.02, 0.11, 0.15, and 0.58). We did not consider this result as an indicatin of normality distribution as, for small sample sizes, normality tests have little power to reject the null hypothesis and therefore small samples most often pass normality tests [14]. Moreover, the variance of the populations is not equal. Therefore, in order to decide whether the population distributions are identical, we used the Mann-Whitney-Wilcoxon Test [12], which does not assume the population distribution to follow a normal distribution, and to have equal variance. The null hypothesis is that data collected with and without treatmend are identical populations. In both cases, at 0.05 significance level, we concluded that the data originating from a device with treatment and the data originating from a device without treatment are non-identical populations.

Based on the results of the executed experiments and the statistical analysis, we concluded that the implemented system is able to improve the discharging time or in other terms, to extend the battery runtime. However, even though the results seem to be promising, we can not generalize them to all Android devices, and especially not to smartphones running different operating systems. The execution and evaluation of the experiments suggest that the collected data can be very fragile, in the sense that small side effects can affect negatively the validity of the results.

4.3 Threats to validity

The performed study faces the following threats to validity:

- internal validity: during a test run there are many different scenarios which could negatively affect the resulting data. The main problem is to control unwanted background tasks increasing the workload of the CPU. Even if the different treatments are executed multiple times, it is not possible to guarantee the complete absence of confounding factors. We tried to minimize this impact by uninstalling or deactivating all apps that were not involved in the experiments. Moreover, we deactivated all connectivity services;

- construct validity: a problem is that the battery within the used devices could be affected by a large number of experiments, e.g., recharging cycles or by the warming up of the device. We can not fully exclude that a decreased energy consumption is the result of an unwanted side effect;

- external validity: the major external threat to validity relies in the involved devices. The data we obtained are based on two devices of the same manufacturer and model. Therefore the results can not be generalized to all smartphones running the Android operating system. To enhance external validity, it is necessary to execute the developed system on different devices, covering the available devices on the market. A reason for using only one type of device was that the power management system implemented in the scope of this project is based on a kernel which can only be used on specific devices such as the LG Nexus 5.

5. CONCLUSIONS AND FUTURE WORK

This study suggests that kernel level power management helps to reduce the energy consumption of a mobile device. Therefore, it can be used as a basis for a variety of further research and enhancements. The major issues faced during this study were a result of the complexity of integrating a power management system in the kernel. In general, the most relevant difficulties raised from finding the right entry points to connect the power management module with the specific hardware drivers. When starting with the implementation of our power management tool, we recognized that the kernel source code is scarcely documented. There is a documentation package coming along with the kernel, however, it contains only a general overview of specific hard-

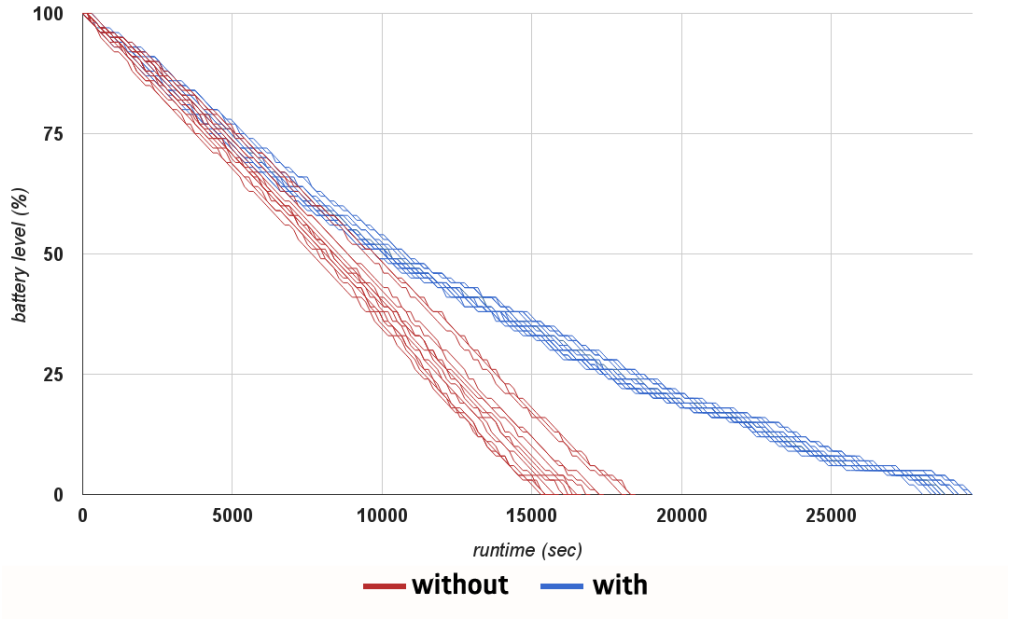

Figure 8: Discharge process without and with treatement on device A showing the battery level (y-axis) over time (x-axis).

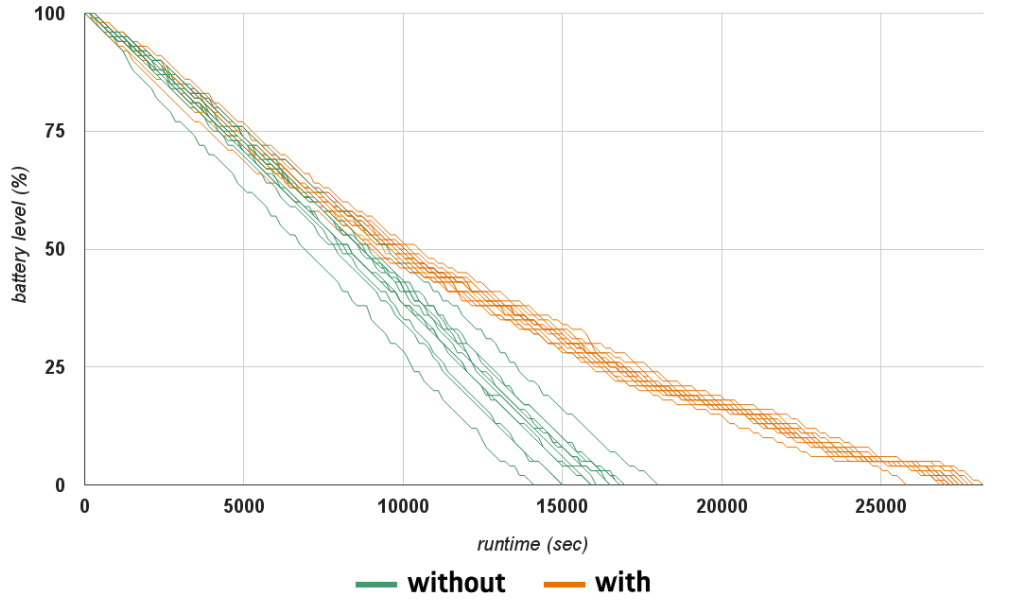

Figure 9: Discharge process without and with treatement on device B showing the battery level (y-axis) over time (x-axis).

ware drivers without giving a detailed description of the API or the included routines.

The lack of documentation made it very difficult to understand the structure of the code and its behavior, especially when we had to modify existing code. Furthermore, the effort of searching for deeper documentation proved to be rather difficult since in most cases, the search results were related to user space topics. This slowed down the whole development process, and prevented us from including certain features such as radio services. However, the absence of the radio services cannot only be attributed to the fact of missing documentation, but also to the way radio services are embedded into the Android kernel. Even if the kernel source code is licensed as Open Source it does not necessarily mean that all the code running within the kernel is accessible. Some modules such as the baseband drivers are

proprietary, making impossible to induce any changes.

5.1 Future Work

Further research efforts may compare a user space power saving tool with the implemented system. Such a study could reveal important insights about the different energy footprints caused by tools running on different layers. An additional research path can be followed on the selection of the settings defined in the power classes. By changing them, it would be possible to optimize the ratio between performance tradeoffs and energy consumption. The comparison between the increase in battery runtime and the performance degradation suggests that the ratio does not behave in a linear fashion. This opens the doors for conducting a further study examining the behavior of the previously described ratio by decreasing or increasing the strength of the power saving settings. Between all the additional research approaches that can be taken using the same system, there is a variety of possibilities to enhance it. An idea for a further enhancement is to include more components in the system such as the radio drivers. A further enhancement could focus on increasing the usability of the power management tool. An approach could be an advanced mechanism for reducing the LCD backlight as described in [11]. This would allow definitely to preserve more energy and increase the battery runtime.

5.2 Conclusions

In this study we have implemented and analyzed a new approach to reduce the energy consumption of smartphones. The resulting system has been examined by conducting two experiments which suggest that the implemented tool is capable to extend the battery runtime by 70% to 75% with an estimated performance degradation of only 20% to 30%. These are promising results, and further work is worth in this direction.

The reduction of energy consumption of a mobile device through CPU usage adaptations is difficult to envision and even harder to implement and prove. There are many interdependencies between the different CPU settings: lower frequencies allow lower voltages and low voltages reduce the energy consumption. However, even though the user does not recognize immediately the reduced CPU performance, the device might take more time to perform a certain calculation, which might lead to a higher energy consumption. At the same time, lowering the clock speed of a CPU under a specific workload can force a second CPU to turn on, which increases the energy consumption instead of reducing it.

In a world where many businesses, entertainment, communication, and socialization rely upon the utilization of mobile devices, finding the right balance between performance and energy efficiency requires innovation, along with long lasting development, testing and measurement cycles, which set up a timely and challenging research area.

6. REFERENCES

[1] CPU Governors, Hotplugging drivers and GPU governors. Online: http://androidmodguide.blogspot.it/p/blog-page.html. Accessed: 2015-09-02.

[2] J. Annuzzi, L. Darcey, and S. Conder. *Introduction to Android Application Development: Android Essentials*. Addison-Wesley Professional, 4th edition, 2013.

[3] R. Bala and A. Garg. Battery Power Saving Profile with Learning Engine in android Phones. *International Journal of Computer Applications*, 69(13):38–41, May 2013.

[4] D. Bovet and M. Cesati. *Understanding The Linux Kernel*. Oreilly & Associates Inc, 2005.

[5] L. Corral, A. Georgiev, A. Janes, and S. Kofler. Energy-aware performance evaluation of android custom kernels. In *Green and Sustainable Software (GREENS), 2015 IEEE/ACM 4th International Workshop on*, pages 1–7, May 2015.

[6] S. Datta, C. Bonnet, and N. Nikaein. Power monitor v2: Novel power saving android application. In *Consumer Electronics (ISCE), 2013 IEEE 17th International Symposium on*, pages 253–254, June 2013.

[7] C. Ellis. The case for higher-level power management. In *Hot Topics in Operating Systems, 1999. Proceedings of the Seventh Workshop on*, pages 162–167, 1999.

[8] T. Heath, E. Pinheiro, J. Hom, U. Kremer, and R. Bianchini. Code Transformations for Energy-Efficient Device Management. *IEEE Transcations on computers*, 53:2004, 2004.

[9] T. HJ. *Testing for normality*. New York: Marcel Dekker, 2002.

[10] Information Technology Laboratory, National Institute of Standards and Technology. Secure hash standard (shs). Online: http://dx.doi.org/10.6028/NIST.FIPS.180-4, 2015.

[11] E. Kreiman, D. Emil, and C. Lupu. Using Learning to Predict and Optimise Power Consumption in Mobile Devices. Online: http://www.doc.ic.ac.uk/teaching/distinguished-projects/2010/e.kreiman.pdf, 2010. Accessed: 2015-09-02.

[12] H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *Ann. Math. Statist.*, 18(1):50–60, 03 1947.

[13] M. B. Motlhabi. Advanced Android Power Management and Implementation of Wakelocks. Online: http://www.cs.uwc.ac.za/~mmotlhabi/apm2.pdf, 2013. Accessed: 2015-09-02.

[14] D. Öztuna. Investigation of four different normality tests in terms of type 1 error rate and power under different distributions. *Turkish Journal of Medical Sciences*, 36(3), 2006.

[15] S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 3(52), 1965.

[16] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.