A Framework for Automated Measurement of Energy Consumption in Multiple Android Devices

Lujun Weng

Supervisor: Markus Wagner

Co-Supervisors: Brad Alexander, Mahmoud Bokhari

November 2018

Thesis submitted for the degree of Master in Computer Science

SCHOOL OF COMPUTER SCIENCE

THE UNIVERSITY OF ADELAIDE AUSTRALIA

Declaration

Except where stated this thesis is, to the best of my knowledge, my own work and my supervisor has approved its submission.

Signed by student: 

Date: 02/11/2018

Signed by supervisor: 

Date: 02/11/2018

Acknowledgements

I would like to express my deepest gratitude to my supervisor Dr. Markus Wagner, for his continuous support, encouragement and guidance throughout this research project.

I am also deeply grateful to Mr. Mahmoud Bokhari and Dr. Brad Alexander for their valuable suggestions and help.

I would also like to thank my family for their endless encouragement and support.

Abstract

Smartphones are very common now. Due to limited battery capacity of smartphones, increasingly more researchers and developers are interested in energy efficiency of these devices. When it comes to energy efficiency, energy consumption measurement is usually an important task, but the task could be time-consuming and involve many manual operations. Therefore, in order to ease the task, this research project focuses on investigating potential solutions to automating the process of energy consumption measurement of applications in Android devices. The goal is developing a prototype framework to automate the process. One core challenge of automating the measurement process is control over charging. Different solutions are proposed and analysed for the charging control. Two of them, programmatical and relay-based methods, are evaluated and used in the framework. The initial version of the framework utilises the programmatical method. Despite the method's inherent drawbacks, it is used for integrating other components with the framework. Later, it is replaced by the relay-based method, which introduces complexity but make the framework applicable to more devices. Therefore, the framework is improved based on this alternative and more functions are added to the framework. The framework is then verified in practice through a case study. Certainly, more studies need to be conducted for testing the framework and there are still important features that have not been added into the framework. All of them would be focused in the future development of the framework.

Contents

Chapter 1

Introduction

This chapter introduces the research project and the thesis. More specifically, a brief description is given for the project itself at first. After that, background, motivation and objectives of the project are provided. The structure of the thesis is outlined at the end of this chapter.

1.1 About the research project

This thesis is a report of the research project for my Master's degree on Computer Science in the University of Adelaide. It lasted for two semesters, from February to October 2018.

The project is about measuring energy consumption of applications in Android devices, which is a common but nontrivial task for energy-related work. The whole process could be time-consuming and involve many manual operations. Therefore, the project focuses on automating energy consumption measurement of application across multiple Android devices. Two general goals for this project are 1) to prove that the whole process of energy consumption measurement in Android devices can be automated and 2) to develop a prototype framework for automating the measurement process.

1.2 Background

Smartphones are everywhere now. According to this report [20], about 45% of American adults own smartphones and that is the statistics in 2012. With the ubiquitousness of smartphones, interests on energy efficiency of mobile phones have been growing in recent years due to their limited battery capacity. One direction towards energy efficiency is through energy management technologies which have been studied, developed and used in modern mobile devices by mobile phone manufacturers and operating system builders. They regard energy saving technologies and long battery duration as key features when they are marketing their products. Another direction researchers are growing interest on is through changing software code. It has been shown that energy efficiency of a particular mobile phone application can be improved by changing the code [11]. Researchers also started to look at how to automatically optimising software to achieve energy minimisation, because it is somewhat unrealistic to demand every developer to be aware of energy efficiency issue [5, 18]. Among these

and other energy-related studies, there is a common and key task, namely energy consumption measurement of a particular application in mobile phones.

1.3 Motivation

Researchers have been looking at energy efficiency of mobile phones for many years and methods are being proposed from time to time for estimating energy consumption of mobile devices [12]. The purposes of measuring energy consumption usually include code optimisation, detecting energy bugs and better battery management [11, 4, 3]. One common drawback of these energy-related approaches is that they are not automatic and cannot be applied at the time in multiple devices.

On the other hand, there are testing tools that can automate the testing of applications and control multiple devices simultaneously. For instance, the authors propose a framework called SAPIENZ in [17], which can test applications automatically across multiple devices in parallel. Using this framework, they tested the top 1,000 Google Play applications and found 558 previous unknown crashes [17]. OpenSTF is another similar framework with many features and a fancy user interface [6]. These tools mitigate the process of testing applications in many devices. However, they are not designed for measuring energy consumption.

Therefore, if we can bring in the features of these testing tools, the process of measuring energy consumption might be more easily applied to multiple applications and devices.

1.4 Objectives

In this research project, the general goal is to ease the process of energy consumption measurement in Android platform to make it automatic and even parallel. The reason why Android platform is chosen to be the objective platform is that Android phones are ubiquitous and Android operating system's source code is accessible. According to this report [21], Android smartphones took the largest amount of the market share with 81.7% as of the fourth quarter of 2016. It is also the platform many researchers base on and therefore hopefully this project can help more researchers. Ultimately, this can also benefit application developers with more energy efficient applications. More specifically, the aims of this master project are to

1. review current energy consumption measurement methods and propose a feasible measurement process;
2. build tools to automate the measurement process; and
3. conduct measurement simultaneously in multiple devices.

In order to guide the research at the beginning stage of this project, the following questions are proposed based on the objectives:

1. What are pros and cons in current methods?
2. What are the common steps in current methods?
3. What factors need to be configurable when setting up test environment?
4. What factors can be controlled or to what extent we can control them?

5. How can the measurement process be automatic and run in multiple devices simultaneously?

6. How can comparative testing of software configurations be integrated into the workflow of app developers?

1.5 Document structure

Chapter 2 Literature Review: provides a review of relevant methods of energy consumption measurement in Android devices as well as frameworks that are similar to the one developed in this project.

Chapter 3 Framework Design: provides a general design of the framework and analyse several solutions to control over charging.

Chapter 4 Implementation: describes and analyses the implementation of the framework, including major components and methods.

Chapter 5 A Case Study: provides an energy-related case study, with the data collected using the framework.

Chapter 6 Conclusion: provides a summary of key points of the thesis and future work.

Chapter 2

Literature Review

Since this research project draws ideas from both energy-related and test-framework-related research, this chapter for literature review will be divided into two sections. One is for methods of energy consumption measurement. It reports about recent work on energy efficiency and different approaches to measuring energy consumption in smartphones. The other section is for related test frameworks. Test frameworks usually enable automatic and parallel tests, but they do not have energy-related functions. Looking into these frameworks would provide useful ideas for this project.

2.1 Methods of energy consumption measurement

Hoque et al. [12] provides a very comprehensive survey of the energy consumption of mobile devices. More specifically, they first summarise a list of terminologies used in this field, which is helpful for people who want to have a general idea of key components in different types of solutions. Then, they classify different solutions according to their implementation and deployment strategies [12]. Also, they make a comparison among different types of solutions in terms of capabilities and performance.

Generally, there are three types of methods for the energy consumption measurement in mobile devices [12]. The first one is using external instruments. The external instruments can measure the current used by smartphones directly. This type of methods usually can provide high precision and accuracy [12]. Researchers already successfully conducted several energy-related studies by using this type of methods. For example, Hindle et al. [11] applies external meter to create a framework called Green Miner in the field of Green Mining. They demonstrate the effectiveness of Green Miner in power optimisation of graphical user interface (GUI) [11, 24]. Also, according to the report [11], Green Miner successfully helped to find an energy bug in a Reddit reader app. Another example of applying external instruments for energy consumption measurement can be found in the report of Banerjee et al. [3]. Banerjee et al. [3] utilise this kind of measurement technique to detect energy bugs and hotspots in mobile phones and they have successfully evaluated their framework in 30 free Android applications from Google Play [3].

The second type of method is model-based. According to Hoque et al. [12], the power consumption model can be based on utilisation, events and code analysis. Zhang et al. [23] and Pathak et al. [19] report in detail about utilisation-based and event-based models respectively.

The third type of method is using the fuel gauge and the battery APIs [12]. Modern smartphones usually come with battery fuel gauge, which is able to read

battery information like voltage and current. This kind of information then can be access through the battery APIs, such as BatteryManager in Android SDK [8]. Bokhari et at. [5] has already applied this method in deep parameter optimisation on Android smartphones for energy minimisation. In that application [5], they utilise the internal fuel gauge and the battery APIs to do energy minimisation of `Rebound` library [7].

2.2 Related test frameworks

For related test frameworks, one significant work is from Mao et at. [17], who propose a framework called Sapienz for multi-objective testing. This framework can test different applications in multiple mobile devices automatically and simultaneously. According to their report, Sapienz outperforms some widely-used tools and revealed 558 unknown crashes in the top 1000 Google Play apps [17]. Another framework helps this research project significantly is called Smartphone Test Farm (STF)[6]. It enables remote control and management over smartphones from browser. Some features include mouse and keyboard input, app upload, screen display etc. All these features are also important for this research project and provide ideas when implementing the framework in this project.

Chapter 3

Framework Design

One objective of this project is to build a framework that can help researchers to measure mobile phone's energy consumption easily and accurately. Therefore, this chapter will focus on the high-level design of the framework. The actual implementation of the framework will be described later, but it is based on the design introduced in this chapter.

This chapter will start with user stories and main requirements. They provide an overview over the project and how will the framework be used from user's perspective. After that, general architecture of the framework will be described and this section will also outline the main components in high level and interaction among the components. The last part of this chapter is for control over charging. It is one of the main requirements.

3.1 User stories

The following three user stories are what the project mainly deals with. Also, several main requirements can be identified from them. The first user story can be described as follows:

1. A user issues a command to measure an application's energy consumption in a device
2. The framework runs the application in the device and meanwhile records the energy consumption
3. The framework gets back the result

The second use story can be described as follows:

1. A user issues a command to measure an application's energy consumption in multiple devices
2. The framework runs the application in these devices and meanwhile records the energy consumption in each device.
3. The framework gets back results from all these devices

The third use story can be described as follows:

1. A user issues multiple commands to measure multiple applications in multiple devices

2. The framework queues and runs these applications in the devices and records the energy consumption

3. The framework gets back all results from these devices

The three user stories are somehow progressive. The latter user story can be built upon on the former one. A core function of the framework that can be identified here is able to run an application, record the energy consumption and get back the result in a particular device.

3.2 Requirements analysis

This section provides a general requirements analysis. According to the user stories described in the last section, three general requirements of the framework are listed as follows:

1. Users shall be able to interact with the framework.

2. The framework shall be able to communicate with multiple connected Android devices.

3. The framework shall be able to control Android devices.

Specifically, the first requirement requires the framework to provide a user interface. The user interface could be in many forms, such as command line, graphical user interface and even plugins of integrated development environments. Furthermore, the user interface shall provide users with a simple operation for the core function that is running a specific application under test in devices, recording the energy consumption and getting back the result.

For the second and third requirements, they concern the interaction between the framework and Android devices. Two key words in the statements are "communicate" and "control". How they are defined determines required functions for the second and third requirements. The word "communicate" means the framework shall be able to identify devices, send messages to devices and fetch relevant data from devices. The word "control" means the framework shall be able to run specific tasks in Android devices and change devices' status to some extent. The tasks can be the app under test as well as various services belonging to the framework.

While looking at the third requirement more closely, there are some more requirements worth mentioning here. In order to reduce the noise from the system during the measurement of energy consumption, it is required that some aspects of Android devices be controlled. The aspects include charging, CPU frequency, screen brightness, network connection, airplane mode etc. Control over these aspects is an important part of the framework, especially control over charging. Control over charging is also essential for restoring battery capacity automatically.

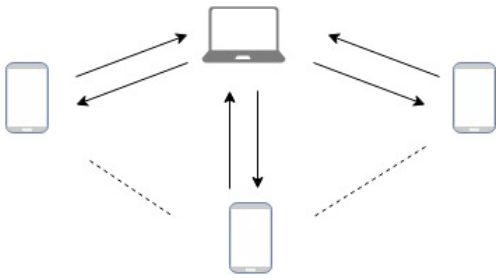
Therefore, whether control over charging can be achieved is a key factor that determines if the whole measurement process can be automated. As can be seen in the following chapters, it will influence the implementation and some sections contribute to this topic particularly.

3.3 General architecture

From the user stories and requirements analysis, the framework shall provide an interface for users to interact with the framework and more importantly with devices.

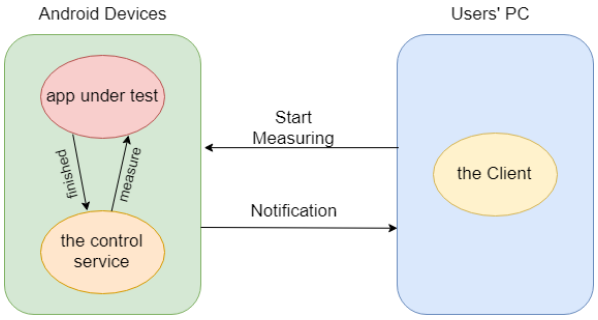
Assuming users use a PC to interact with the framework, the framework should enable mobile devices and a PC to communicate in bi-direction. For example, the framework should be able to tell a device to run a particular application and record the energy consumption, and the device should be able to inform the PC to pull back the result when the recording finishes. Therefore, the core is communication with multiple mobile devices. Figure 3.3.1 shows the general model of communication.

Figure 3.3.1: The general communication model



In order for successful communication to happen, there should be two services running in the PC and devices respectively and a communication protocol for the two services. In the thesis, the service running in the PC is called "the client" and the service running in devices is called "the control service". These two services, along with app under test (AUT), consists of three main components of the framework. In general, the client is responsible for interacting with users and communicating with the control service; the control service is responsible for measuring energy consumption of the app under test and communicating with the client. Figure 3.3.2 shows the three components and interaction among them.

Figure 3.3.2: Main components in the framework and their interaction



3.4 Control over charging

Control over charging is important for automating the energy measurement process and therefore a key part of the framework. It is needed when the measurement is ongoing or devices are being recharged. The selected solution to it will influence the

framework design and implementation directly. Therefore, three candidate solutions are introduced in this section.

3.4.1 Programmatical Method

Investigation shows that it is possible to stop charging an Android device while the device is still connecting to a power source. If this is true, it can be utilised to stop charging a device while keep the connection on. In this case, it acts as if there is no the restriction and the PC and devices can communicate as normal.

Similarly, this solution also comes with drawbacks. It is unclear if this method can be applied in every mobile device and every version of Android system. Since the fragmentation is severe in Android ecosystem, it is impossible to test every device. Therefore, this can never be proved or disproved. Another issue of this solution is that it only works for rooted devices. In other words, every device needs to be rooted before applying this method.

3.4.2 Programmable USB Hub

A programmable USB hub can be controlled by software to switch on or off certain port. Therefore, instead of devices being connected directly to the PC, a programmable USB hub can be used as an agent to control connections. Every mobile device connects to the PC through the programmable USB hub and in this way, the service in the PC will be able to stop charging a device without manually unplugging the USB cable.

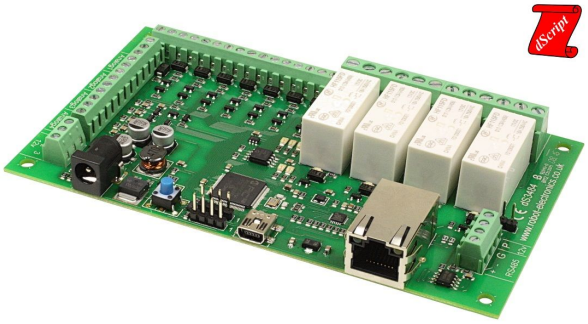
However, this solution also introduces some issues. One is the communication channel disappears as the service tells the USB hub to switch off a connection to a device. Consequently, the device cannot send any message back to the PC via USB cable and the PC has no idea of when to switch on the connection again. One possible way to deal with this is using a fixed timeout, but it is not as effective as a client-controlled reconnect and it can be difficult to estimate an efficient timeout length. However, it would certainly complicate the framework further. Another issue is that a programmable USB hub could be expensive. A programmable USB hub with 8 ports would cost about 450 US dollars [1]. This is only one USB hub. It definitely would cost more if later more and more mobile devices are added.

3.4.3 Relay-based Solution

Relay-based solution is somehow similar to using programmable USB hub. Instead of programmable USB hub with controllable ports, relay-based solution uses relay boards, which have relays on them. These relays could be controlled in different ways, such as WiFi, USB cable and Ethernet. Figure 3.4.1 shows a relay board controlled through Ethernet. Usually, these boards will come with built-in services for relay control.

Relay boards function the same way as programmable USB hub by introducing an extra layer between the PC and Android devices. Therefore, relay boards have the same advantages as programmable USB hub and they are usually cheaper. Furthermore, relay boards could potentially keep the communication channel on while cutting off power source, given USB cable is processed properly. By contrast, programmable USB hub shuts down both power source and communication.

Figure 3.4.1: Ethernet relay board with four relays [13]



Chapter 4

Implementation

This chapter is about the implementation of the framework. The goal of the implementation is to develop a framework that satisfies the requirements listed and analysed in Chapter 3. This chapter will describe various components of the implementation in detail.

According to the design and analysis in Chapter 3, several key components are important for implementation. First, there are services in both Android devices and the PC. Services in Android devices' side are the ones the framework uses to communicate and launch various tasks. Meanwhile, the framework also needs to provide an interface through services in the PC for users to interact with. Second, it is also important part of implementation how services in Android devices and the PC communicate with each other. Several options are possible, such as designing a new communication protocol based on TCP/IP or using an existing tool like Android Debug Bridge. In this implementation, Android Debug Bridge is used. Lastly, control over charging and other factors are considerably significant for energy consumption measurement and therefore they are key part of the implementation, especially control over charging. Therefore, two methods for control over charging are evaluated, implemented and reported in detail. Also, control over other factors like CPU frequency and screen brightness is described after that.

The remaining of this chapter is organised as follows. The services in both Android devices and PC will be described first, including their functions and interaction. Then how these services communicate will be given. After that, implementation of control over charging and other factors will be analysed and reported in detail. Last, a discussion will be given for the implementation's pros and cons as well as potential future improvements.

4.1 Components in Android devices

There are mainly three components involved in Android devices in this implementation. They are control service, log service and app under test. Originally, there are only two components in Android devices' side, as demonstrated in Figure 3.3.2. However, when it comes to actual implementation, it was found that it is better to separate the log function, which is responsible for recording relevant information during the measurement, from the control service. Therefore, control service in Chapter 3 has a more general meaning than the one here.

App under test (AUT) is the application whose energy consumption will be measurement by the framework. In principle, app under test should remain as independent

as possible. In other words, the framework should not have any dependency on the app under test. However, in this implementation, code should be injected into the app under test in order for the control service to know if the app already finishes. Furthermore, app under test is the target that the framework deals with. Therefore, it is regarded as important part of the framework. The code that is injected into app under test will be demonstrated further in the next section.

Log service is responsible for recording relevant data during the measurement. It was part of the control service in the original design, but became an independent component when implemented. The following is a list of types of information the service will collect during the energy consumption measurement.

1. **Timestamp**: Obtained from `System.currentTimeMillis`, it is the current system time. The unit is milliseconds.

2. **Device Name**: Combination of `BUILD.MANUFACTURER` and `BUILD.MODEL`. It is the identification for devices.

3. **Android Version**: It is the build version of current Android operating system, identical to the value of `Build.VERSION.RELEASE`.

4. **Power Source**: It is the current power source of devices. The value will be one of `Battery`, `AC`, `WIRELESS`, `USB` and `UNKNOWN`, and it is determined by comparing the value of `BatteryManager.EXTRA_PLUGGED` with these predefined values.

5. **Temperature**: Obtained from `BatteryManager.EXTRA_TEMPERATURE`, it is the current battery temperature. The value will be -1 if it is not available. The unit is tenths of a degree in Celsius.

6. **Voltage**: Obtained from `BatteryManager.EXTRA_VOLTAGE`, it is the current battery voltage level. The value will be -1 if it is not available. The unit is millivolts.

7. **Capacity**: Obtained from `BatteryManager.BATTERY_PROPERTY_CAPACITY`, it is the current remaining battery capacity. It is an integer represented as the percentage of total battery capacity.

8. **Battery Level**: It is the current battery level ranging from 0 to 1.0, calculated by dividing `BatteryManager.EXTRA_LEVEL` by `BatteryManager.EXTRA_SCALE`.

9. **Charge Counter**: It is the current battery capacity in microampere-hours, obtained from `BatteryManager.BATTERY_PROPERTY_CHARGE_COUNTER`.

10. **Current Average**: It is the average battery current in microamperes, obtained from `BatteryManager.BATTERY_PROPERTY_CURRENT_AVERAGE`. Positive and negative values represent net current entering and discharging respectively.

11. **Current Now**: It is the instantaneous battery current in microamperes, obtained from `BatteryManager.BATTERY_PROPERTY_CURRENT_Now`. Positive and negative values represent net current entering and discharging respectively.

12. **Energy Counter**: It is the Battery remaining energy in nanowatt-hours, obtained from `BatteryManager.BATTERY_PROPERTY_ENERGY_COUNTER`.

13. **Screen Brightness**: It is the current screen backlight brightness, ranging from 0 to 255 (inclusive). The value is obtained from the system file `/sys/class/leds/lcd-backlight/brightness`, but root access is needed.

14. **CPU0-3 Frequency**: It is CPU frequency in each core. The value is obtained from a system file for each core. For example, frequency for CPU core 0 comes from `/sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq`. Root access is no needed.

All these types of information will be recorded every 250ms (4Hz), which is the maximum sampling frequency of the battery fuel gauge. Also, they are stored in CSV format. Partial wake lock is acquired during the whole process of measurement to prevent the log service from being killed by the operating system [10]. Not all these types of information are available in every device. Special value like 0 or -1 would be returned when a type of information is not supported by a device.

Control service is the main service running in Android devices. It is responsible for coordinating with different components of the framework as well as controlling some aspects of the operating system like screen brightness and CPU frequency. For example, when a command is issued by a user, the control service will receive the command, set up environment accordingly and run the app under test and the log service. After the measurement completes, the control service will notify the client to pull back the result. In the initial implementation, the only aspect controlled by the control service is charging. The successful implementation of control over charging is the starting point for other aspects of the environment.

4.1.1 Interaction among the components

Three main components in Android devices' side are app under test, log service and control service, as described above. They have their own role and function, but they need to interact with each other as well. Therefore, interaction among these components is important for understanding the framework.

For the control service and the log service, their relation and interaction is relatively simple. The control service will start the log service at the beginning of an energy consumption measurement and stop the log service at the end of the measurement. The log service has no dependency on the control service. However, they indeed share a common storage for collected data so that the control service is able to let users pull back the data.

Similarly, interaction between the log service and app under test is also straightforward. App under test has no idea of existence of the log service while the log service will record related information when energy consumption of the app under test is measured.

Only relation between the control service and app under test has somewhat complexity. The control service can run the app under test as simply as starting the log service. The real problem comes from how the control service is going to know when the app under test finishes running. The temporary solution in the initial implementation is to utilise broadcasted `Intent` [9], which is like a message that only some registered receivers could receive. Therefore, the control service is registered as the receiver of an intent that will be broadcasted by the app under test after it finishes. However, this solution requires several lines of code be injected into the app under test for intent broadcast, which means source code of the app under test must be accessible. Accessibility to source code would not be available in every case and hence this solution is only temporary. It would be better to have a solution that does not need to inject any code into the app under test.

In the initial implementation, the app under test informs the control service it has already finished through broadcasting a special intent. This method needs to inject few lines of code into the app under test, which is not effective. Later, it is simplified

by using Android APIs directly. The key method is `startActivityForResult`, which will start the app under test on the top of the control service in this case. Once the app under test finishes, a dedicated method in the control service will be invoked and hence control is transferred back to the control service. In this way, there is no need to hack the app under test and inject code into it.

4.2 Command line tool for users

For the user interface of this framework in this version of implementation, an interactive command line tool is implemented. The interactive command line is based on Node.js, which is a Javascript runtime built on Chrome's V8 JavaScript engine. It is selected mainly because of its popularity and simplicity. Also, it has a third-party library called adbkit which supports Android Debug Bridge directly. The effectiveness of adbkit has already been demonstrated as a core library of the open source project Smartphone Test Farm, which enable control and management of smartphones in browser. Another important third-party library is vorpal, on which the interactive command line interface is based.

There are two commands that have been already implemented for the implementation. One is `list` command, which will return a list of available smartphones and their status. Besides, users will be notified when a smartphone goes online or offline. The other command is `test` command, which will deploy a app under test onto a specified device or all devices, tell the control service in Android devices' side to measure the energy consumption of the app and pull back results after being notified that the results are ready.

4.3 Two-way communication

Communication between the PC and devices is a key part of this framework. As outlined in Section 3.3, two-way communication is needed here, which means the PC should be able to tell devices what to do and devices can notify the PC of their status. In general, this kind of communication can be implemented via USB cable or wirelessly. Communication through USB cable can be the simplest solution in that connections can be established easily while wireless communication might need more manual steps to connect devices to the PC. From the experiments in Section 4.4.1, we know that disabling charging programmatically does not always work. However, it is important that the device is not charging while we are measuring energy consumption. Therefore, the strategy is to use the programmatic method to make the framework work basically and then add wireless communication for those devices that cannot use the programmatic method.

4.3.1 Method

The method that used to implement communication relies on Android Debug Bridge or ADB. It is an official command-line tool that helps communicate with a device [2]. It is most widely used for installing and debugging apps. This does also mean that it can be used to tell a device what to do, that is the communication from the PC to a device.

As for how to send message back to the PC from a device, the command logcat of ADB is used. It is originally used for "Print log data to the scree" [2]. However, through monitoring the output of logcat and filtering out useless information, one can

also get messages from a particular application in a device. This makes it possible for a device to notify the PC of its status.

There are three major advantages of using ADB for communication. Firstly, ADB supports operations such as installing and running an application natively. These operations are also needed by the framework to deploy an application into a device before starting measuring energy consumption. Secondly, ADB helps handle the low-level connection. From user's perspective, once connection is established through ADB, we do not need to care about how messages are sent or obtained so that the framework can focus on how to handle these messages. Finally, ADB supports wireless mode. As mentioned above, wireless connection can be very useful when programmatic method to disable charging is not available. Sooner or later, more than one method will be used to deal with the restriction proposed in Section 3.3. Moreover, the use of ADB in wireless mode is the same as USB mode.

4.4 Control over charging

This chapter reports about the implementation of control over charging. In Section 3.4, three possible solutions are introduced. They have their own advantages and disadvantages. For example, the programmatical method is the simplest one without introducing any extra devices. However, not every Android smartphone support this method and root access is needed. Using programmable USB hub might be a good candidate without the issues of the programmatical. However, programmable USB Hub could be expensive and extra service needs to be set up for the hub to work. Relay-based solution is somehow similar to using programmable USB hub. It usually costs less than using programmable USB hub and has no restrictions as the programmatical method, but it will introduces complexity and USB cables also need special process.

Nevertheless, all the three methods are worth investigating in the actual implementation. However, due to limited time and resources, only the programmatical method and relay-based solution are evaluated and implemented. The remaining of this section is mainly about these two methods and their implementations.

4.4.1 Programmatical method

It is required that a device cannot be charging when measuring energy consumption of an app in that device. Meanwhile, the device and the PC still need to communicate. For example, the device will need to notify the PC when it finishes the measurement of energy consumption so that the PC can deploy another measurement in that device. Therefore, there should be a method that can stop charging a device while the PC is still able to get message from the device.

In Section 3.3, three solutions are analysed for this issue. They all have their pros and cons and the programmatical one would be the simplest if it could work in every device. Although it is impossible to test every existing device, experiments can certainly be conducted to test devices I have.

Method

The method used to stop charging is by modifying a file in Android OS. The file's path is `/sys/class/power_supply/battery/charging_enabled`. This file contains 1 or 0, which means "enable charging" and "disable charging" respectively. Therefore, when a device is connecting to a power source, changing the file's content to 0 could

possibly stop charging. However, modifying this file needs root access. That is why a device should be rooted first before this method can be potentially effective in that device.

General Rooting Process

As mentioned above, Android devices must be rooted for using the programmatic method to stop charging. Therefore, I tried to root every device I have in order to test if this programmatic method can be applied. The rooting process varies in different devices and different versions of Android system, but a series of common steps can still be summarised as follows:

1. Unlock bootloader. A bootloader is responsible for loading and starting up the operating system, which is Android OS in this project. Manufacturers usually "lock" the bootloader so that it cannot be modified and users have to stick to the original Android OS. In the next step, a custom recovery needs to be flashed for rooting, which requires the bootloader to be unlocked. Therefore, unlocking bootloader is the very first step. Google shall give the steps to unlock bootloader for most devices in few minutes, although some devices' bootloader cannot be unlocked.

2. Flash TWRP. TWRP is a custom recovery for Android phones [22]. It can be used to install custom ROMs as well as root devices. To flash TWRP, the first thing is to download the specific TWRP image from the official website for the device to be rooted. Then, the device should be rebooted into fastboot mode and use "fastboot" command to flash the custom recovery.

3. Install Magisk. After TWRP recovery has been ready, one can enter the TWRP recovery mode to install SuperSU or Magisk to root the device. Both of them should work well and Magisk is used for this project. After this, apps in the device should have root access.

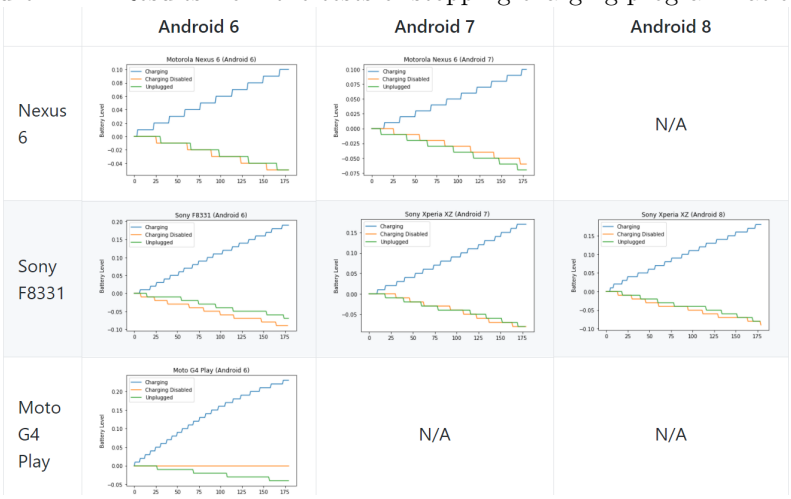
The above three steps are common for rooting Android phones. However, details of each step usually vary across different devices and Android versions. Also, there will extra steps for specific devices and Android versions. For example, only one command is needed to unlock the bootloader of Nexus 6 while the official guide should be followed for Moto G4 Play. It is always better to search the Internet and investigate first before starting any step.

Validation of the method

In order to test if the method works in a device, a heavy calculation task can be run for a while in three different conditions in the device to compare their energy consumption. Specifically, for each device with a certain version of Android OS, calculation of tanh function runs for 30 minutes when the device is connecting to a AC power source, charging is disabled using the above method and the device is unplugged. Energy-related information is being recorded while the calculation is running. It is expected that the results should be similar when charging is disabled using this method and the device is unplugged, and the result for connecting to an AC power source should show the opposite trend. Figure 4.4.1 shows the test results for the devices I have with different versions of Android OS.

For each line graph, the vertical axis represents battery level, the blue line represents the device is connecting to a AC power source, the green line represents the

Figure 4.4.1: Results from the tests of stopping charging programmatically



device is unplugged and the yellow represents charging is disabled by this method. As can be seen from Figure 4.4.1, this method works in Nexus 6 and Sony F8331 with at least two different versions of Android OS as their line graphs meet expectation. Although it does not work in Moto G4 Play with Android 6 and possibly in other devices either, it provides the initial solution on which the following two-way communication can be built.

4.4.2 Relay-based solution

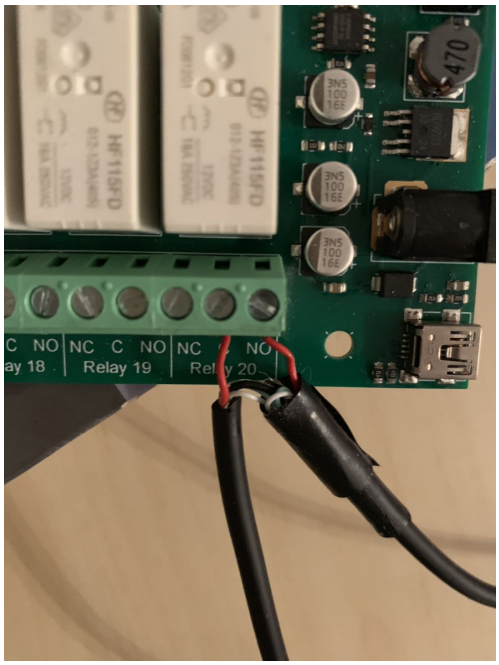
The relay-based method introduces a relay board into the framework. In the case of programmable USB hub, smartphones are connected to the PC via ports of programmable USB hub. Similarly, smartphones are connected to the PC via relays in the relay board. However, compared with programmable USB hub, relay-based solution gives more flexibility and usually costs less. The only inconvenience is USB cable needs to be process for the relay-based solution. More specifically, the wire for charging within USB cable, which is usually the red one, is cut and connected via a relay in the relay board. Figure 4.4.2 shows an example of how a USB cable is connected with a relay in a relay board.

Comparison of Relays

There are several different relay boards which mainly differs in the way of how relays in the board are controlled. Table 4.1 compares three different relay boards from Devantech Limited. The three relay boards are WIFI8020 [15], USB-RLY16L [14] and ds3484 [13]. An 8-port programmable USB hub from Acroname is also put in the table and compared with the relay-boards.

As shown in Table 4.1, WIFI8020 has the lowest price per relay/port. It also provide built-in services for controlling relays. As long as devices and the server are in the same network as WIFI8020, they have access to the built-in services. This is the same with ds3484, but ds3484 seems to have more functions than WIFI8020 after going through ds3484 document quickly. Meanwhile, the price of ds3484 almost doubles compared with WIFI8020. USB-RLY16L and Acroname Programmable USB 3.0 Hub are almost the same, except their price. Both of them need a service to be

Figure 4.4.2: Example of a USB cable being connected with a relay in a relay board



Name	WIFI8020	USB-RLY16L	ds3484	Acroname Hub
Control	WiFi	USB	Ethernet	USB
Price(Euro)	135.29	62.39	59.94	387.47
Number of relay/port	20	8	4	8
Price per relay/port	6.76	7.80	14.99	48.43
Setup	connected to the same network	need a service in the PC	connected through cable	need a service in the server

Table 4.1: Comparison of three relay boards and a programmable USB hub

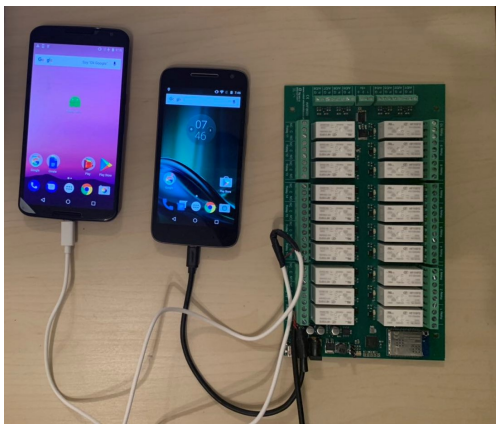
set up in the server, which means demanding more time for developing and testing. Considering pros and cons of the different relay boards, WIFI8020 is selected for this version of implementation.

Using the WiFi-based relay board

Switching on/off a relay in the WiFi-based relay board, WIFI8020, is fairly simple, given devices are connected to the same network as the board. According to WIFI8020's technical document [15], there are several options for controlling the relays. For this implementation, control through HTML commands is used, mostly because this way is the simplest when it comes to coding. For example, if the relay board's IP address is `192.168.0.200` and the number of a relay to be controlled is 2, only a `HTTP GET` request being sent to the following two URL addresses is enough to activate or deactivate the relay respectively. The number at the end of each URL address represents how long the relay will be activated or deactivated for. It is ten seconds in the examples and zero means no time limit. Figure 4.4.3 shows two smartphones are being connected with the relay board.

```
192.168.0.200/io.cgi?DOA2=10
192.168.0.200/io.cgi?DOI2=10
```

Figure 4.4.3: Two smartphones being connected with the relay board



4.5 Control over other factors

Besides charging, there are another four aspects of the Android operating system that can be controlled. They are screen brightness, CPU frequency, airplane mode and WiFi. Control over airplane mode and WiFi is relatively simpler than screen brightness and CPU frequency. After relevant permissions are declared in the manifest file, airplane mode and WiFi can be controlled directly through Android APIs. Therefore, the remaining of this part will be focused on methods for control over screen brightness and CPU frequency. Root access is needed for controlling both factors.

Screen Brightness. There are two files related to screen brightness in the operating system, which are `/sys/class/leds/lcd-backlight/max_brightness` and `/sys/class/leds/lcd-backlight/brightness`. The two files both contain only one integer value ranging from 0 to 255 inclusive. As can be seen from the file names, one file is for setting the maximum brightness and the other one reflects the current brightness. Therefore, in order to change the screen brightness, both files should be changed. If the file of maximum brightness is not changed, the screen brightness could still be changed by the operating system.

CPU Frequency. Since CPU in current smartphones normally has multiple cores, steps for changing the frequency is applied in core level not whole CPU level. Take one CPU core `cpu0` for example. The following steps will set the core's frequency to 1728000.

```
echo "userspace" > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
su cat /sys/devices/system/cpu/cpu0/cpufreq/ scaling_available_frequencies
su stop mpdecision
su echo 1728000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_min_freq
su echo 1728000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
su echo 1728000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
```

Other cores' frequency can be changed in a similar way.

All these factors will be controlled accordingly by the framework before launching target app for measurement, as part of environment setup. During the whole process of energy consumption measurement, the framework will also make sure these factors remain unchanged. At the end, the framework will restore these factors to a proper status after the measurement finishes.

4.6 Discussion

Currently, the framework can measure energy consumption of an app in multiple Android devices with a user issuing one command. More specifically, a user can interact with the framework to list all connected devices and specify all or one device to run measurement task. The framework then can launch several services in the specified devices to measure energy consumption of the app under test. After that, the framework will return measurement results to the user. Basically, the framework satisfies the user stories and requirements listed at the beginning of Chapter 3. Also, it proves the possibility of automating the whole process of energy consumption measurement in Android devices.

However, the framework is still at its early stage and more like a prototype currently. Firstly, the framework is not robust enough. It is mostly tested in experimental not practical scenarios. Better error handling should be implemented for this purpose. Secondly, the functionality of the framework is not complete. Features like scheduling multiple measurement jobs are planned but not yet implemented. Also, the information returned for each device's status is still limited and more detailed information are needed to have a better knowledge of each device. Finally, the user interface is restricted to command line currently. Potential extension to this could be GUI or plugin of IDEs, which would make the framework more user-friendly.

In summary, the framework can work in a basic way, however, improvements are needed to make the framework more robust, capable and user-friendly.

Chapter 5

A Case Study

This chapter is an energy-related case study by using the framework for data collection. There are two benefits from conducting this case study. One is testing the framework in practice. Before the case study, only dummy applications are used for testing. The other benefit is identifying potential future improvements through the case study. Required features for the framework are mostly identified from previous work and analysis of relevant experiments. A practical case study might give more insights into that.

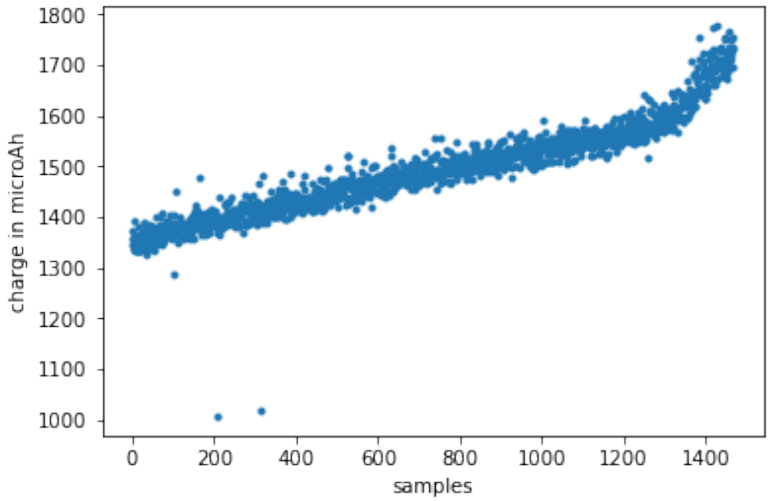
5.1 Statement of the problem

When comparing two variants of a target app by their energy consumption, multiple runs are usually needed due to various sources of noise. For example, Figure 5.1.1 shows multiple energy consumption measurements for a specific app. As can be seen from the figure, the overall trend is increasing rather than horizontal. Also, an unusual increase can be observed at the end of the figure. These are examples showing how noisy energy consumption measurement can be. Ideally, more runs will make it more confident to determine which variant is better. However, it is impossible to run too many times due to limited resources and time. Therefore, one question is how many runs are needed for each variant to be significant enough when comparing two variants by their energy consumption.

5.2 Experiment

The experiments are conducted by running variants of target apps in an Android device and measuring their energy consumption. The device used in the experiments is Nexus 6 with Android version 7.1.1. Each experiment can be divided into three stages. The first stage is environment preparation. During this stage, the framework will automatically set up the environment for latter energy consumption measurement. For our specific experiments, the device's screen brightness is set to 0, its airplane mode is turned on and its CPU frequency is set to 1728000Hz. The second stage is energy consumption measurement of the target. Two components are launched by the framework in this stage, which are log service and app under test respectively. For the experiments here, the data is collected as the app under test runs repeatedly from 100% to 10% battery level. In order to reduce the influence of energy-saving technology potentially launched by the operating system, data is not collected for

Figure 5.1.1: Multiple energy consumption measurements for a specific app in an Android device. (Conditions remain unchanged during all measurements.)



battery level ranging from 10% to 0%. The third stage is restoring stage, where the device's screen is turned on, the device is recharged and the data is fetched.

There are three target app, busy loop, NOP and rebound. The busy loop app uses an infinite loop to keep the app busy for a specified period of time. The NOP app uses `Thread.sleep()` to let the app idle for a specified period of time. For these two target apps, different periods of time are regarded as variants.

The rebound app comes from experiments of Bokhari et al. [5], where the `Rebound` library is optimised for energy minimisation. For the experiments here, each run of the rebound app executes all test cases of `Spring` class in the library for five times. Eight configurations from Bokhari et al. [5] are used to generate variants of the rebound app. Configuration R0 corresponds to the original configuration of `Rebound` library; configuration R1 to R3 come from experiments where fitness function is fed by the battery fuel gauge; configuration R4 to R5 are from Jiffy model experiment, where fitness function is computed using energy vs. Jiffy model; configuration R6 to R7 are from LOC experiment, where energy is estimated using the line of code.

5.3 Data analysis

This section analyses the data collected in the experiments. Through comparing the data, we can obtain some preliminary insights into the problem.

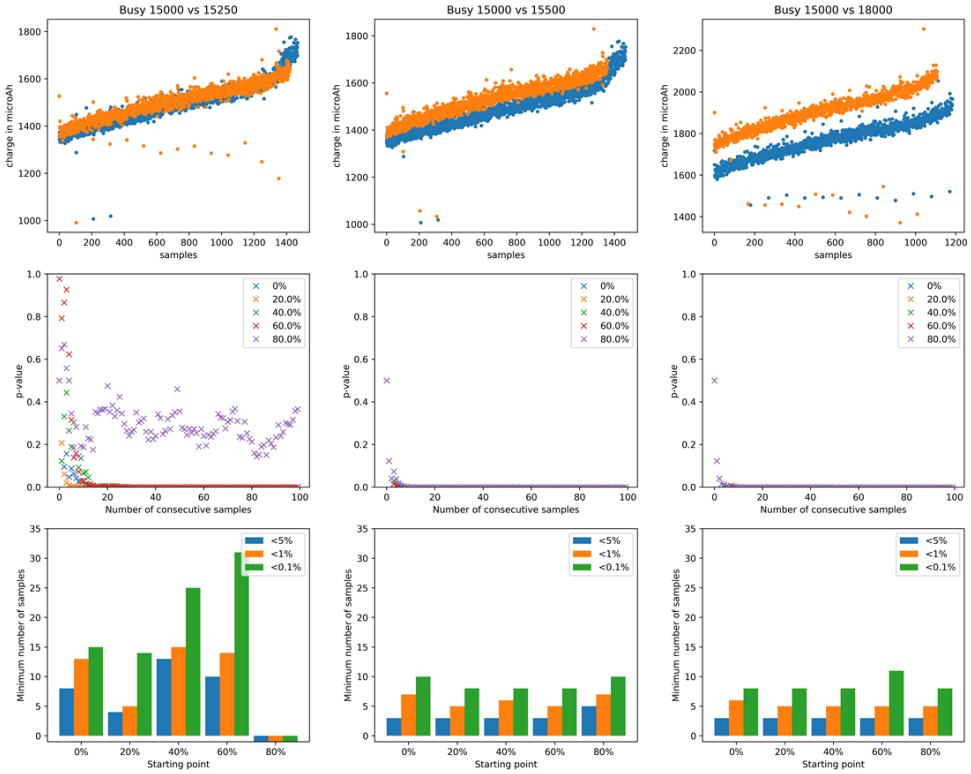
Busy loop and NOP app

Busy loop and NOP are somehow similar. They are compared and analysed together in this part to get some insights into the minimum number required for distinguishing their variants. Figure 5.3.1 shows three pairs of busy loop's variants with increasing amount of energy consumption difference. Three columns of the figure are analysis of the three pairs respectively. In each scatter plot of the first row, it visualises the raw data from two variants. Each dot corresponds to a sample of energy consumption

measured in each run of that variant. All samples are plotted as the order of appearance in log file. For each pair of two variants, we want to know how many samples we need in order to distinguish the two variants. In each of the second row's plots, we use statistical test to calculate p values with different starting points of every data file and different numbers of samples. The statistical test is Mann-Whitney rank test [16], which is a nonparametric test and can be used to determine if two independent samples come from the same distribution. Lower p value means it is more statistically significant that the variant represented by blue dots consume less energy than the one represented by orange dots. Samples for the statistical test are selected from different starting points, including 0%, 20%, 40%, 60% and 80%. The number of selected samples ranges from 0 to 100. In each bar chart of the third row, it shows the minimum number of samples that are needed to distinguish two variants statistically. In other words, we want to know since when that all the p values afterwards are less than some specified p values.

By comparing three columns of Figure 5.3.1 horizontally, we can see that it is easier to distinguish two variants as the gap of energy consumption of two variants gets larger. More specifically, it becomes easier to tell the difference between two variants visually from left to right column. This is then verified statistically that can be seen from the second and third rows.

Figure 5.3.1: Comparing variants of busy loop app with increasing amount of difference between running periods. (Value zero in bar charts means data is unavailable for that item.)

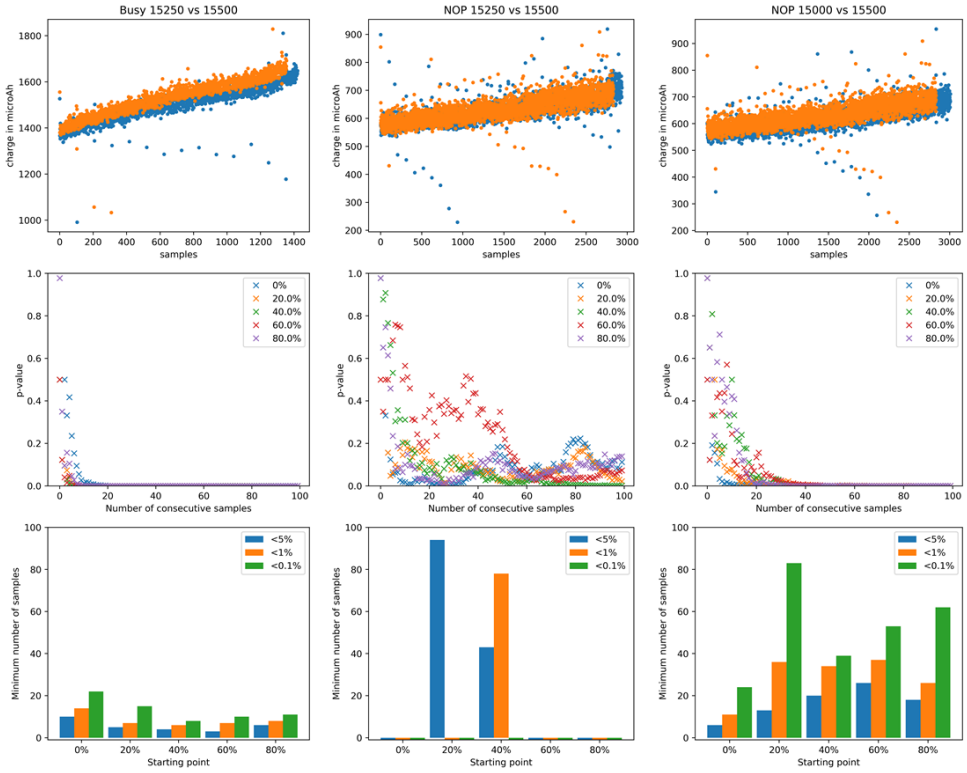


Similar results can be observed from NOP app, as shown in Figure 5.3.2, which is plotted in the same way as Figure 5.3.1. The second and third columns compare

two pairs of variants from NOP app with different gaps of runtime. The comparison shows a similar result as busy loop app, that is as the gap gets larger it is easier to distinguish two variants. Another thing shown in Figure 5.3.2 is that with the same difference of runtime it is relatively easier for busy loop app to distinguish two variants than for NOP app. One possible reason could be that busy loop app is more computationally intensive than NOP app and therefore busy loop app is less sensitive to noise from the system than NOP app.

As can be seen in the second column of Figure 5.3.2, it is almost impossible to distinguish two variants of NOP app with 250ms difference of runtime. However, if the runtime of the two variants is doubled, they become more distinguishable again. This can be observed from Figure 5.3.3. The reason could be that noise from the operating system is relatively constant and longer runtime makes it harder to be influenced by the noise.

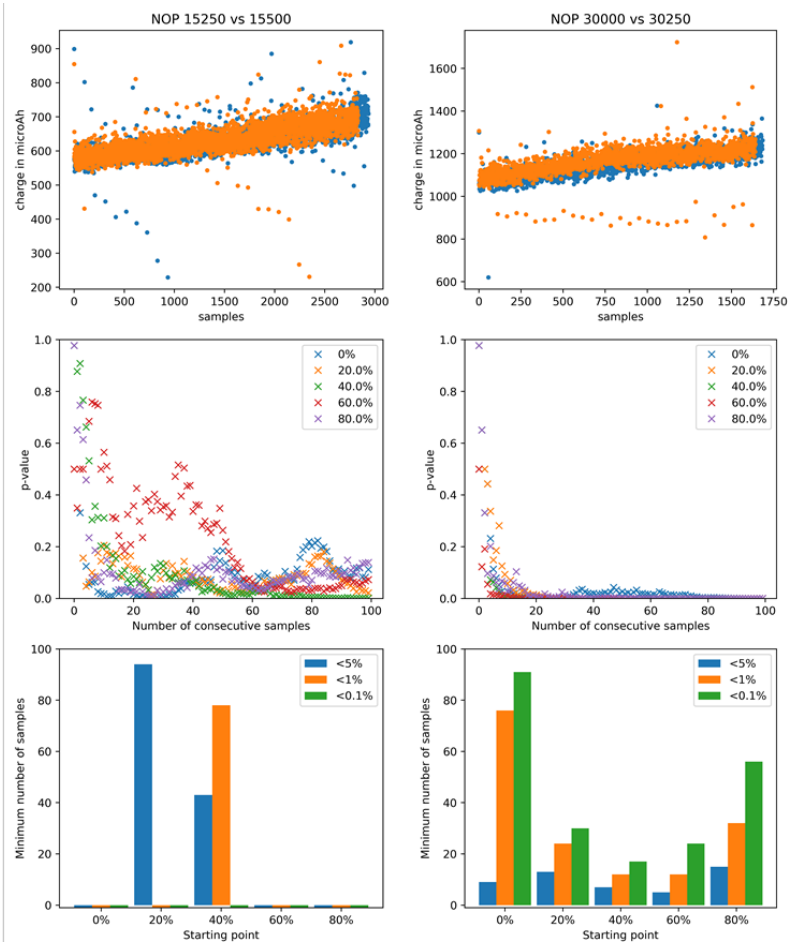
Figure 5.3.2: Comparing NOP and busy loop apps with the same difference of run time.(Value zero in bar charts means data is unavailable for that item.)



Rebound library

This part compares variants of `Rebound` library and analyses the required number of samples that are needed to distinguish two variants significantly. Figure 5.3.4 compares original configuration of `Rebound` library with other seven configurations in a similar way to the ones of busy loop and NOP apps. Raw data of pairs of variants is plotted in the first column. The second and third columns show statistical results of

Figure 5.3.3: Comparing variants of nop app with run time doubled.



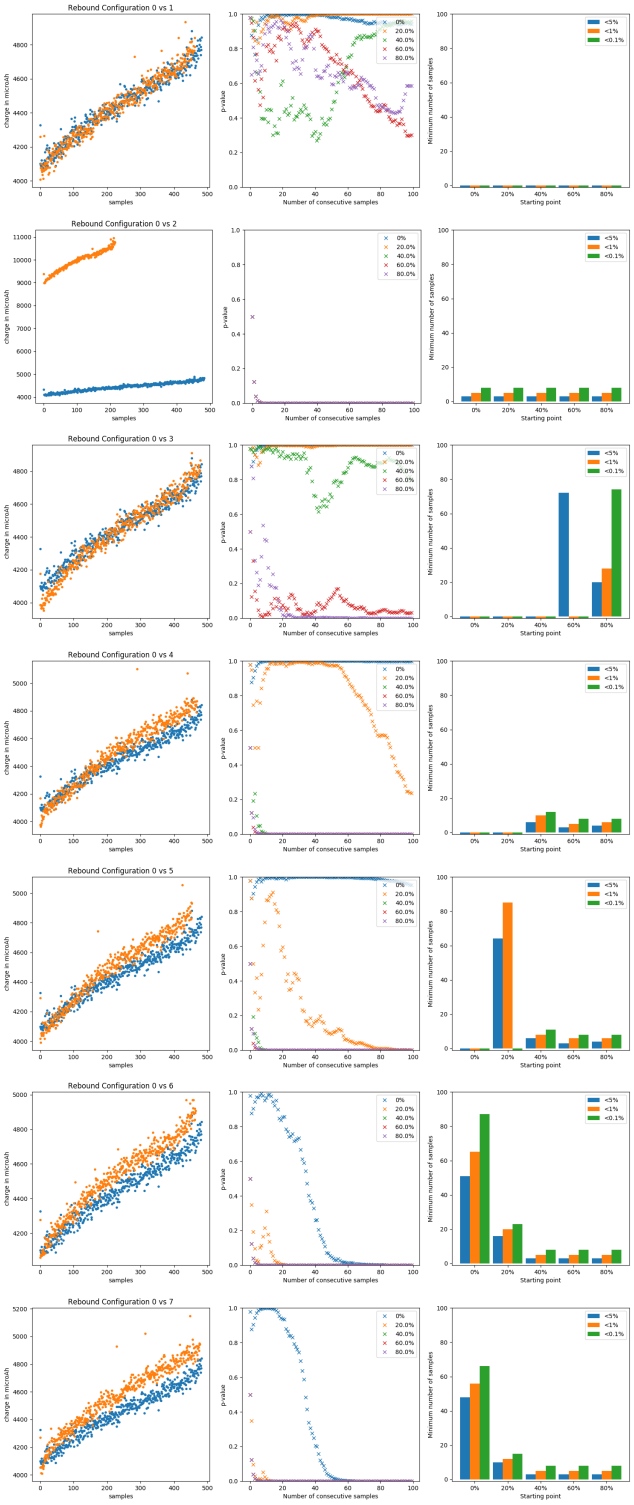
comparisons. Similarly, it can be observed that required number of samples decreases as difference of energy consumption between two variants gets larger.

Another observation from Figure 5.3.4 is that 40% might be the best starting point for comparing two variants. As busy loop and NOP apps, data collected around batter level of 10% is not reliable since uncertainty of the operating system and its energy-saving technology. Also, from the last few rows of Figure 5.3.4, it can be observed that data overlaps at the beginning of each variant's measurement. Therefore, beginning and ending part of each data file might not be good choices.

For verifying that starting point with 40% is better than others, extra analysis is conducted upon the collected data for `Rebound` library. Analysis is conducted by using rank sum test to compare each pair of the eight `Rebound`'s variants. Table 5.1 shows the results with starting point 0% and significance level 5%. Items in the first row and column correspond to eight variants of `Rebound` library. The internal items of the table represent the minimum number of samples needed to distinguish two variants statistically. '101' is a special number, which means more than 100. Tables for all combinations of starting point and significance level are listed in Appendix A.

In order to compare all combinations more easily, all tables in Appendix A are

Figure 5.3.4: Comparing original configuration of Rebound library with other seven configurations.(Value zero in bar charts means data is unavailable for that item.)



summarised further into Table 5.2 by calculating the median value from upper-right (diagonal excluded) part of each table. As can be seen from Table 5.2, starting point 40% and 80% are equally best. However, samples collected at the end might be influenced more by the device's operating system. Also, starting from 40% will provide more samples. From the above, starting with 40% could be the best to compare two variants of `Rebound` library.

	R0	R1	R2	R3	R4	R5	R6	R7
R0	101	101	3	6	6	101	57	51
R1	101	101	3	6	101	101	24	36
R2	3	3	101	3	3	3	3	3
R3	6	6	3	101	14	6	6	6
R4	6	101	3	14	101	101	6	19
R5	101	101	3	6	101	101	9	29
R6	57	24	3	6	6	9	101	100
R7	51	36	3	6	19	29	100	101

Table 5.1: Comparing eight Rebound variants in pairs with starting point 0% and significance level 5%. (Value 101 in the table means more than 100.)

		starting point				
		0%	20%	40%	60%	80%
significance level	5%	7.5	4.5	3.0	4.0	3.0
	1%	14.0	11.0	5.0	8.5	5.5
	0.1%	32.5	14.0	9.0	11.5	8.5

Table 5.2: Minimum number of samples of different starting point and significance level. (The numbers are medians of upper-right (diagonal excluded) part of tables in Appendix A

5.4 Discussion

As mentioned at the beginning of this chapter, energy consumption measurement of a specific app in an Android device can be noisy. Usually, multiple energy consumption measurements are needed if we are going to compare two variants of a specific app. The experiments try to find out the minimum number of measurements needed for three target apps in order to distinguish two variants of them statistically. Some numbers are obtained for the three target apps from the experiments. However, there is no experiments yet to verify that these numbers are indeed enough to compare two variants. Due to time constraints, this kind of experiments is not conducted in this project, but it would certainly be a focus of future work. Also, more similar experiments could be conducted across a variety of target apps, devices and Android versions. Although the diversity of Android platform makes it impossible to examine every combination, it would still be very helpful and potentially give more insights if more devices, target apps and Android versions are investigated.

Chapter 6

Conclusion

This research project tries to prove the whole process of energy consumption measurement of a particular application in Android smartphone can be automated by developing a prototype framework for this purpose. It started with validation of the programmatical method for control over charging, one major part of the framework. Experiments demonstrate that the programmatical method can be effective in some Android devices. Therefore, it was selected as the method for controlling charging in the initial version of the framework. After investigating the programmatical method, major components such as control service in smartphone's side and command line tool in the PC are implemented and added into the framework.

The initial version of the framework aims to make sure all components of the framework can work together and interact with each other properly. Its functionality is very limited and every component only achieves its minimum requirement. For example, there are few commands implemented for users to interact with the framework. Also, this version has two significant issues. For setting up environment, only control over charging is implemented through the programmatical method. Even worse, the programmatical method is not always effective in smartphone and root access is needed for this method. Another issue is in order for the framework to work properly, some code should be injected into app under test to notify the framework when the app finishes. These two issues were addressed in the subsequent version of the framework.

The relay-based version of the framework improves over the initial version. The improvements include implementation of control over more environmental factors, relay-based control over charging and alternative way of interaction between control service and app under test. In this version, the framework can control CPU frequency, screen brightness, WiFi connection and airplane mode. Root access is still need for changing CPU frequency and screen brightness. For control over charging, relay-based solution is used. More specifically, a WiFi-based relay board is introduced into the framework. Smartphones then are connected via the relay board to the PC. In this solution, no root access is needed and it will work for almost every smartphone. Also, there is no need to inject some code into app under test in this version of the framework. To put it simply, native APIs are used directly to enable notification of end of running.

As of now, the framework can deploy an app under test into multiple Android smartphones, configure several environment factors and measure energy consumption of the app under test. A case study is used to show that the framework works in practical scenario. However, the framework is still a prototype and at its early

stage. Due to limited time, some planned features do not have chance to be added, such as integration with IDE and more comprehensive information for smartphones' status. These planned but never implemented features, coupled with other features like schedule of jobs and restoring from errors, will definitely be the direction for future work. It is to be hoped that these and more features will be implemented in the framework and the framework can help researchers and developers in practice.

Bibliography

[1] I. Acroname. Programmable industrial usb 3.0 hub (8 ports), 2018. Accessed on 17 May 2018.

[2] Android. Android debug bridge, 2018. Accessed on 17 May 2018.

[3] A. Banerjee, L. K. Chong, S. Chattopadhyay, and A. Roychoudhury. Detecting energy bugs and hotspots in mobile apps. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 588–598. ACM, 2014.

[4] M. Bokhari and M. Wagner. Optimising energy consumption heuristically on android mobile phones. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, pages 1139–1140. ACM, 2016.

[5] M. A. Bokhari, B. R. Bruce, B. Alexander, and M. Wagner. Deep parameter optimisation on android smartphones for energy minimisation: a tale of woe and a proof-of-concept. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1501–1508. ACM, 2017.

[6] CyberAgent. Smartphone test farm. `https://openstf.io/`, 2018.

[7] Facebook. Rebound. `http://facebook.github.io/rebound/`. Accessed on 16 October 2018.

[8] Google. Batterymanager. `https://developer.android.com/reference/android/os/BatteryManager`. Accessed on 15 October 2018.

[9] Google. Intent. `https://developer.android.com/reference/android/content/Intent`. Accessed on 15 October 2018.

[10] Google. Keep the device awake. `https://developer.android.com/training/scheduling/wakelock`. Accessed on 15 October 2018.

[11] A. Hindle, A. Wilson, K. Rasmussen, E. J. Barlow, J. C. Campbell, and S. Romansky. Greenminer: A hardware based mining software repositories software energy consumption framework. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 12–21. ACM, 2014.

[12] M. A. Hoque, M. Siekkinen, K. N. Khan, Y. Xiao, and S. Tarkoma. Modeling, profiling, and debugging the energy consumption of mobile devices. *ACM Computing Surveys (CSUR)*, 48(3):39, 2016.

[13] D. Limited. Ethernet relay. `https://robot-electronics.co.uk/products/relay-modules/ethernet-relay/ds3484.html`, 2018. Accessed on 15 October 2018.

[14] D. Limited. Usb-rly16l - 8 x 16a latching relay. http://www.robot-electronics.co.uk/usb-rly16l-8-x-16a-latching-relay-module.html, 2018. Accessed on 15 October 2018.

[15] D. Limited. Wifi8020 - 20 x 16a wifi relay. http://www.robot-electronics.co.uk/wifi8020-20-x-16a-relay-module.html, 2018. Accessed on 15 October 2018.

[16] H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pages 50–60, 1947.

[17] K. Mao, M. Harman, and Y. Jia. Sapienz: Multi-objective automated testing for android applications. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, pages 94–105. ACM, 2016.

[18] C. Pang, A. Hindle, B. Adams, and A. E. Hassan. What do programmers know about software energy consumption? *IEEE Software*, 33(3):83–89, 2016.

[19] A. Pathak, Y. C. Hu, and M. Zhang. Bootstrapping energy debugging on smartphones: a first look at energy bugs in mobile devices. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, page 5. ACM, 2011.

[20] L. Rainie. Two-thirds of young adults and those with higher income are smartphone owners. *Race/ethnicity*, 65(830):11, 2012.

[21] Statista. Android version market share distribution among smartphone owners as of september 2017, 2017. Accessed on 14 March 2018.

[22] T. TWRP. About, 2018. Accessed on 17 May 2018.

[23] 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. ACM, 2010.

[24] L. Zhong and N. K. Jha. Graphical user interface energy characterization for handheld computers. In *Proceedings of the 2003 international conference on Compilers, architecture and synthesis for embedded systems*, pages 232–242. ACM, 2003.

Appendices

Appendix A

List of pairwise statistical test of Rebound library's variants

	R0	R1	R2	R3	R4	R5	R6	R7
R0	101	101	3	6	6	101	57	51
R1	101	101	3	6	101	101	24	36
R2	3	3	101	3	3	3	3	3
R3	6	6	3	101	14	6	6	6
R4	6	101	3	14	101	101	6	19
R5	101	101	3	6	101	101	9	29
R6	57	24	3	6	6	9	101	100
R7	51	36	3	6	19	29	100	101

Table A.1: Comparing eight Rebound variants in pairs with starting point 0% and significance level 5%. (Value 101 in the table means more than 100.)

	R0	R1	R2	R3	R4	R5	R6	R7
R0	101	101	3	6	6	101	57	51
R1	101	101	5	8	8	101	71	58
R2	101	101	5	9	101	101	33	43
R3	5	5	101	5	5	5	5	5
R4	8	9	5	101	18	10	8	10
R5	8	101	5	18	101	101	8	26
R6	101	101	5	10	101	101	24	35
R7	71	33	5	8	8	24	101	101

Table A.2: Comparing eight Rebound variants in pairs with starting point 0% and significance level 1%. (Value 101 in the table means more than 100.)

	R0	R1	R2	R3	R4	R5	R6	R7
R0	101	101	8	11	101	101	101	69
R1	101	101	8	13	101	101	53	55
R2	8	8	101	8	8	8	8	8
R3	11	13	8	101	101	14	11	13
R4	101	101	8	101	101	101	19	35
R5	101	101	8	14	101	101	30	44
R6	101	53	8	11	19	30	101	101
R7	69	55	8	13	35	44	101	101

Table A.3: Comparing eight Rebound variants in pairs with starting point 0% and significance level 0.1%. (Value 101 in the table means more than 100.)

	R0	R1	R2	R3	R4	R5	R6	R7
R0	101	36	3	10	101	74	17	10
R1	36	101	3	101	65	27	3	3
R2	3	3	101	3	3	3	3	3
R3	10	101	3	101	65	6	3	3
R4	101	65	3	65	101	18	3	3
R5	74	27	3	6	18	101	9	3
R6	17	3	3	3	3	9	101	81
R7	10	3	3	3	3	3	81	101

Table A.4: Comparing eight Rebound variants in pairs with starting point 20% and significance level 5%. (Value 101 in the table means more than 100.)

	R0	R1	R2	R3	R4	R5	R6	R7
R0	101	42	5	42	101	88	20	12
R1	42	101	5	101	79	35	11	5
R2	5	5	101	5	5	5	5	5
R3	42	101	5	101	78	15	5	5
R4	101	79	5	78	101	101	5	5
R5	88	35	5	15	101	101	11	7
R6	20	11	5	5	5	11	101	101
R7	12	5	5	5	5	7	101	101

Table A.5: Comparing eight Rebound variants in pairs with starting point 20% and significance level 1%. (Value 101 in the table means more than 100.)

	R0	R1	R2	R3	R4	R5	R6	R7
R0	101	101	8	60	101	101	23	16
R1	101	101	8	101	101	39	13	10
R2	8	8	101	8	8	8	8	8
R3	60	101	8	101	92	41	9	8
R4	101	101	8	92	101	101	9	8
R5	101	39	8	41	101	101	15	10
R6	23	13	8	9	9	15	101	101
R7	16	10	8	8	8	10	101	101

Table A.6: Comparing eight Rebound variants in pairs with starting point 20% and significance level 0.1%. (Value 101 in the table means more than 100.)

	R0	R1	R2	R3	R4	R5	R6	R7
R0	101	101	3	101	8	7	3	3
R1	101	101	3	101	10	9	3	3
R2	3	3	101	3	3	3	3	3
R3	101	101	3	101	3	3	3	3
R4	8	10	3	3	101	101	4	3
R5	7	9	3	3	101	101	5	3
R6	3	3	3	3	4	5	101	7
R7	3	3	3	3	3	3	7	101

Table A.7: Comparing eight Rebound variants in pairs with starting point 40% and significance level 5%. (Value 101 in the table means more than 100.)

	R0	R1	R2	R3	R4	R5	R6	R7
R0	101	101	5	101	11	9	5	5
R1	101	101	5	101	12	10	5	5
R2	5	5	101	5	5	5	5	5
R3	101	101	5	101	6	7	5	5
R4	11	12	5	6	101	101	6	5
R5	9	10	5	7	101	101	98	5
R6	5	5	5	5	6	98	101	9
R7	5	5	5	5	5	5	9	101

Table A.8: Comparing eight Rebound variants in pairs with starting point 40% and significance level 1%. (Value 101 in the table means more than 100.)

	R0	R1	R2	R3	R4	R5	R6	R7
R0	101	101	8	101	13	12	8	8
R1	101	101	8	101	17	13	8	8
R2	8	8	101	8	8	8	8	8
R3	101	101	8	101	11	10	8	8
R4	13	17	8	11	101	101	85	9
R5	12	13	8	10	101	101	101	9
R6	8	8	8	8	85	101	101	27
R7	8	8	8	8	9	9	27	101

Table A.9: Comparing eight Rebound variants in pairs with starting point 40% and significance level 0.1%. (Value 101 in the table means more than 100.)

	R0	R1	R2	R3	R4	R5	R6	R7
R0	101	101	3	101	3	3	3	3
R1	101	101	3	101	7	7	7	7
R2	3	3	101	3	3	3	3	3
R3	101	101	3	101	8	3	3	3
R4	3	7	3	8	101	101	101	5
R5	3	7	3	3	101	101	101	9
R6	3	7	3	3	101	101	101	10
R7	3	7	3	3	5	9	10	101

Table A.10: Comparing eight Rebound variants in pairs with starting point 60% and significance level 5%. (Value 101 in the table means more than 100.)

	R0	R1	R2	R3	R4	R5	R6	R7
R0	101	101	5	101	5	6	5	5
R1	101	101	5	101	9	9	9	9
R2	5	5	101	5	5	5	5	5
R3	101	101	5	101	9	9	5	5
R4	5	9	5	9	101	101	101	8
R5	6	9	5	9	101	101	101	10
R6	5	9	5	5	101	101	101	16
R7	5	9	5	5	8	10	16	101

Table A.11: Comparing eight Rebound variants in pairs with starting point 60% and significance level 1%. (Value 101 in the table means more than 100.)

	R0	R1	R2	R3	R4	R5	R6	R7
R0	101	101	8	101	8	9	8	8
R1	101	101	8	101	11	13	12	12
R2	8	8	101	8	8	8	8	8
R3	101	101	8	101	12	14	9	8
R4	8	11	8	12	101	101	101	15
R5	9	13	8	14	101	101	101	14
R6	8	12	8	9	101	101	101	24
R7	8	12	8	8	15	14	24	101

Table A.12: Comparing eight Rebound variants in pairs with starting point 60% and significance level 0.1%. (Value 101 in the table means more than 100.)

	R0	R1	R2	R3	R4	R5	R6	R7
R0	101	101	3	21	4	4	3	3
R1	101	101	3	17	3	3	3	3
R2	3	3	101	3	3	3	3	3
R3	21	17	3	101	3	5	3	3
R4	4	3	3	3	101	101	101	7
R5	4	3	3	5	101	101	101	8
R6	3	3	3	3	101	101	101	7
R7	3	3	3	3	7	8	7	101

Table A.13: Comparing eight Rebound variants in pairs with starting point 80% and significance level 5%. (Value 101 in the table means more than 100.)

	R0	R1	R2	R3	R4	R5	R6	R7
R0	101	101	5	41	6	6	5	5
R1	101	101	5	28	5	5	5	5
R2	5	5	101	5	5	5	5	5
R3	41	28	5	101	7	7	7	5
R4	6	5	5	7	101	101	101	9
R5	6	5	5	7	101	101	101	9
R6	5	5	5	7	101	101	101	15
R7	5	5	5	5	9	9	15	101

Table A.14: Comparing eight Rebound variants in pairs with starting point 80% and significance level 1%. (Value 101 in the table means more than 100.)

	R0	R1	R2	R3	R4	R5	R6	R7
R0	101	101	8	101	9	9	8	8
R1	101	101	8	95	8	8	8	8
R2	8	8	101	8	8	8	8	8
R3	101	95	8	101	11	15	9	8
R4	9	8	8	11	101	101	101	17
R5	9	8	8	15	101	101	101	14
R6	8	8	8	9	101	101	101	18
R7	8	8	8	8	17	14	18	101

Table A.15: Comparing eight Rebound variants in pairs with starting point 80% and significance level 0.1%. (Value 101 in the table means more than 100.)