Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Mobile Smartphone Speed Test

Bachelor thesis

Fabian Grob
grobfa@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Gino Brunner, Simon Tanner
Prof. Dr. Roger Wattenhofer

August 31, 2018

Abstract

When a new smartphone is released, there are multiple technology magazines which do reviews of the device. They usually test new features, the cameras, various other sensors and measure the performance by running different synthetic benchmarks. These benchmarks mainly focus on the maximum performance the System-on-a-Chip can deliver in the tested device. Most of the time the user does not use this maximum performance because usual every-day tasks such as sending messages, taking photos or browsing the Internet do not need high computation power. This real-world performance is rarely tested and so there are no numbers to easily compare the real-world speed of smartphones.

This thesis presents a two-device solution for measuring the real-world performance of an Android device. The solution consists of two apps: SpeedtestReplay and SpeedtestControl. The tested device runs SpeedtestReplay, the other device acts as measuring tool running SpeedtestControl, which records the other device with the camera. The taken images are analysed to get the time the tested device needs to start the specified app. The measurement is based on the assumption that as soon the tested app is started completely the screen remains static.

These apps are used to compare the real-world performance of three devices: Sony Xperia V, Sony Xperia Z1 Compact and Sony Xperia Z3 Compact, three similar devices from different generations. The results show that as expected the two newer devices perform much better but the difference between the Z1 Compact and Z3 Compact is bigger than one might expect from the synthetic benchmark results.

An additional test is performed between two devices with identical hardware. One of the devices is freshly reset, the other is used daily. Comparing just the app start times, the difference between the two devices was negligible. When switching between multiple apps there was a bigger difference.

Contents

Chapter 1

Introduction

1.1 Motivation

Every year manufacturers present new smartphones with faster processors and graphic chips. They usually perform better in intense tasks like games, but most users do not use their smartphone primarily for these tasks. Instead, they surf the web, write messages or take pictures.

To compare the performance of different devices, you can look at the specifications: The type of processor and graphic chip plus the amount of memory give you an idea how well a phone can perform. To measure this performance, there are benchmark tools, for Android the most known are Antutu Benchmark [1] and Geekbench [2]. There are also Android versions of 3DMark [3] and PCMark [4], two very popular Windows benchmark tools. At the end these benchmarks deliver a number as result. The task to find the more powerful system is then as easy as comparing numbers.

These benchmarks are synthetic and do not tell the whole story. There are more factors which influence the real-world performance such as the amount of running background processes, the screen resolution (higher resolution means more pixels to compute) or the speed of the storage memory. A widely known example are SSD in desktop computers: A system may have a very high 3DMark-result because it has a fast processor and graphics card but a harddrive leads to long waiting times. A SSD makes the computer more responsive and for basic office tasks like document processing or mailing a high 3DMark-result gives you not really an advantage.

To compare the real-world performance you can put two ore more devices side by side, press on a button and observe which finishes the task first. For a review you can publish the video so everyone can watch it. This is a very good comparison, but only for the phones beeing tested. To add a new phone to the comparison you have to do the whole procedure again, for a nice video ideally with every phone. You can also cut/stop the video, so that the tap on the phone is shown at the same time. A simpler way to compare the results is to measure the start time with a stopwatch or watch the video and calculate the time from the timestamps. If you repeat the test multiple times for more accurate results, it will take quite a while and the tester has to stop the times manually. At this the point starts the work of the the thesis.

1.2 Goals

The goal is to write an application for Android devices, where device A starts different apps, device B records the screen of device A and determines how long the apps needed to start. Apps on device A should start automatically, so the user only has to interact with device B.

As a result, device B should analyse the recorded video nearly in real time to detect the start and the complete load of the app started on device A.

1.3 Related Work

Ever since computer have existed, it has been important to know their performance. Even the first computers were benchmarked. They needed to know how much faster it can do algebraic operations than humans, so the performance was measured in operations per second. This unit is still used today as **fl**oating point **o**perations **p**er **s**econd or short FLOPS, the performance of super-computers is specified in this unit [5].

This unit may be usefull for scientific computers but the casual user of a personal computer is not interested in how many FLOPS his device has. He rather wants to know how long the system and applications need to start or how many frames per second are reached in the newest games. Popular benchmark tools take this fact into account. They measure the performance of common tasks and combine the results into a single number which then can be compared. In the general linguistic usage a personal computer means a computer tower with a monitor, a keyboard and a mouse. According to dictionary.com a personal computer is a “a compact computer that uses a microprocessor and is designed for individual use” [6], a smartphone matches these explanation too which means the performance can be measured in the same way. Therefore, it is not surprising that there are several benchmark apps for Android.

The most popular benchmarking apps from Google Play are Antutu Benchmark [1], Geekbench [2], GFXBench [7], 3DMark [3] and PCMark [4]. These benchmarks differ as they focus on different things: Geekbench run theoretical tests on the CPU, GFXBench measures the 3D graphics performance, 3DMark simulates the workload of games, PC Mark has some tests “based on everyday activities” [4], Antutu with over 10 million installations the most popular benchmark does a combination of all with additional storage tests.

All these benchmarks have one thing in common: They measure the performance based on an artificial workload. With DiscoMark there was once a benchmark app in Google Play, which measured the performance of a real-world benchmark. DiscoMark was developed by Gino Brunner for his master thesis at ETH [8]. It is a single device solution and uses Accessibility Services to measure the start time of the apps. In November 2017 Google started removing apps from the Google Play if the "application uses an Accessibility Service for any reason other than assisting users with disabilities" [9] which is exactly what DiscoMark does so it was removed from Google Play.

This leads to the fact that there are currently no apps in Google Play which can measure the real-world performance of an Android device. The solution created during this thesis should fit in this gap.

Chapter 2

Speedtest Application

As stated the benchmark is a solution which requires two devices so there are also two different apps needed. These two apps are presented in this chapter: SpeedtestReplay runs on the tested device, SpeedtestControl runs on the recording device. These two apps communicate over the wireless so it is required both devices are connected to the same wireless network. When no access point is available, one of the devices can connect to the hotspot created by the other devices.

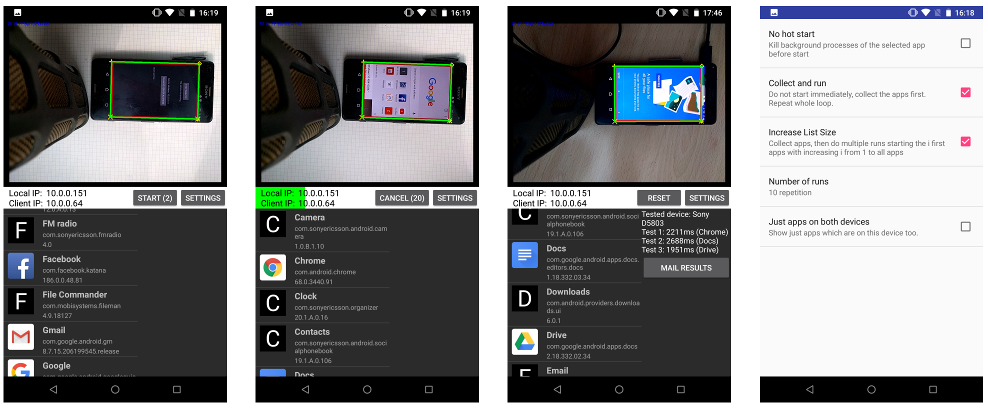
2.1 SpeedtestControl

SpeedtestControl is running on the device which acts as measuring tool. On top of the screen, the taken picture is visible (Figure 2.1 (a)). The largest red contour on the picture is marked, so the user can easily see if the other phone is detected correctly. The image is not rotated because test showed that the additional computations needed for the rotation reduce the number of frames per second.

In the center is a white bar. The IP address of the local device is displayed here, as this must be entered manually on the other device. In a future version of the app this may be solved by a QR-Code which can be scanned by the other device. If the devices are connected, the address of the client is shown too. The white bar does not state white all the time. It is also used as a progress bar for the test (Figure 2.1 (b)).

On the right side of this bar there are two buttons. The function of the Settings-Button is quite obvious: It opens the settings (Figure 2.1 (d)). All settings and their effects are described in Table 2.1. The other button has different functions depending on the configuration and the state of the test. To make it easier for the user, the text of this button is changed based on the current function. If "Collect and run" is disabled, this button stays hidden, otherwise it is used to start the previously selected app, the number in the brackets is the number of selected apps (Figure 2.1 (a)). While the test is running, the button is used to cancel the test, in brackets showing the the runs left from the current test (Figure 2.1 (b)). When some results are shown, the button is used to clean up the results.

At the bottom left, the alphabetically sorted list of apps of the other device is shown. It is used to choose the apps to be tested. At the bottom right, the results of the measurements since the last reset are shown. The list of results is scrollable and if there are any results, an additional button appears to share them via mail (Figure 2.1 (c)).



Tested device: Sony D5803
Test 1: 2211ms (Chrome)
Test 2: 2688ms (Docs)
Test 3: 1951ms (Drive)

No hot start
Kill background processes of the selected app before start ☐

Collect and run
Do not start immediately, collect the apps first. Repeat whole loop. ✓

Increase List Size
Collect apps, then do multiple runs starting the i first apps with increasing i from 1 to all apps ✓

Number of runs
10 repetition

Just apps on both devices
Show just apps which are on this device too. ☐

(a) select apps (b) running a test (c) showing results (d) settings

Figure 2.1: Screenshots of SpeedtestControl

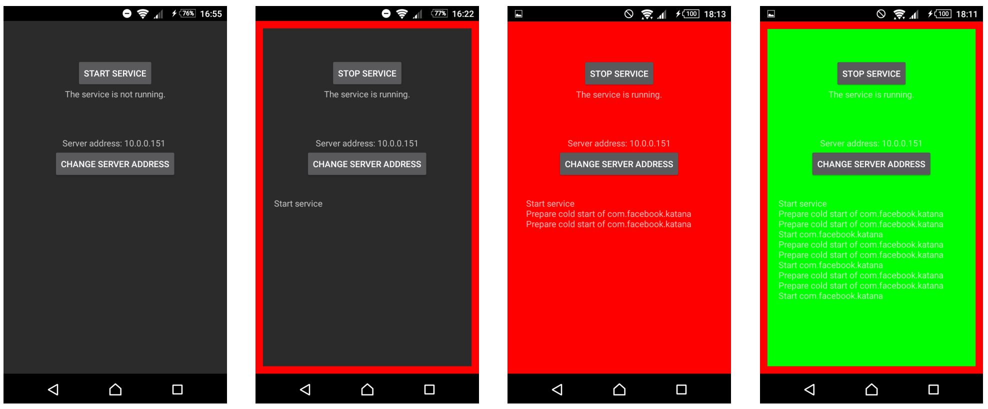
2.2 SpeedtestReplay

SpeedtestReplay is the app running on the tested device. It consists of a visible activity and a background service. The interface of the activity can be divided in a top and a bottom half. In the top half there are two buttons (Figure 2.2 (a)): The first button is used to start or stop the background service, the other one to change the address of the measuring device. The background color of the app is used as visually detectable indicator of the current measuring phase (Figure 2.2 (c) and (d)). These phases are explained in detail in section 4.1.

The background service manages the communication and draw the red frame on the screen (Figure 2.2 (b)) so it needs the permission to draw over other apps. This permission is checked by the service at startup. It also checks if a server address is defined. If one of these checks fails, the service is stopped immediately, otherwise the first action of the service is connecting to the other device and send a list of the installed apps.

Option	Effect
No hot start	No hot starts are performed. This should be reached by killing all background processes before starting the app, but it has not the same effect for all apps.
Collect and run	When tapping an app in the list, the test is not started immediately. The chosen apps are collected and the test only starts when the "START" button is tapped. One run consists of a start of each app.
Increase List Size	This is only available when "Collect and run" is active. The collected apps are tested in a different order: It performs multiple runs. The first run is just done with one app. Afterwards each run contains all the apps of the previous run plus one additional app. These runs are performed until every selected app is started at least once.
Number of runs	Each app is tested a specified numbers of time. Selectable are 1, 5, 10 or 20 runs. This setting is ignored, when "Increase List Size" is active.
Just apps on both devices	In the app list only apps are shown which are installed on the local device as well. This option has no influence on the performed test.

Table 2.1: Settings of SpeedtestControl



The service is not running.

Server address: 10.0.0.151

(a) initial state

The service is running.

Server address: 10.0.0.151

Start service

(b) service started

The service is running.

Server address: 10.0.0.151

Start service
Prepare cold start of com.facebook.katana
Prepare cold start of com.facebook.katana

(c) red background

The service is running.

Server address: 10.0.0.151

Start service
Prepare cold start of com.facebook.katana
Prepare cold start of com.facebook.katana
Start com.facebook.katana
Prepare cold start of com.facebook.katana
Prepare cold start of com.facebook.katana
Start com.facebook.katana
Prepare cold start of com.facebook.katana
Prepare cold start of com.facebook.katana
Start com.facebook.katana

(d) green background

Figure 2.2: Screenshots of SpeedtestReplay

Chapter 3

Device Recognition

SpeedtestControl gets the results from the recorded images. To analyse if the screen on the tested device is static, the app needs to know which part of the image represents the screen. The task of recognize the device in the recorded video is described in this chapter.

Finding a device in is a classical task of computer vision. There exist different libraries for computer vision, SpeedtestControl uses with OpenCV [10] one of the most popular. Nvidia even labeled OpenCV as "a de-facto standard API for computer vision" [11]. Google itself has a mobile vision library too [12] but it is more made for predefined tasks like face detection than for individual purpose.

3.1 Find Device

To measure the time in the video, the phone screen has to be recognized in the video. A simple way to this is by placing the phone on a single color background, there you can easily filter the background and the rest of the image is the phone. Such a background should be easy to find, but it must also be ensured that there are no other objects in the recorded image.

Another method is to use object detection. There are multiple object detection algorithms which mostly use deep neural networks. With such an algorithm you can detect what objects there are in the video, if one of the object is a smartphone or a tablet the goal is reached. These algorithms are very powerful but they need some training and the computational effort is quite high.

These two methods detect the whole other device and nothing is needed on the recorded device, but they do not make use of the fact that the two devices can communicate. The recording device B can ask device A to show an easy detectable pattern on the screen, which then can be detected. For example, there are apps where a QR-Code is detected before the camera has finished focusing the image.

3.2 Track Device

Once the device is recognized, the system has to follow the device. The solution should not just work in a laboratory-like environment, so the assumption that the device always is at the same place in the video can not be made. There are several ways to track the device.

3.2.1 Optical Flow

The Optical flow is a vectorfield where visual points move from one frame to another. Under the assumption that the tested device is in a static environment, which means neither the device nor anything else in the captured video is moved during the capture, the movement of the image can be estimated from the optical flow. Since version 3.0 OpenCV has some tracking algorithms implemented, one of them is called MEDIANFLOW [13] which is based on the Optical flow tracking of the Lucas-Kanade method [14]. On the Nexus 5X this algorithm was not fast enough to keep a sufficient high frame rate.

3.2.2 Use Other Sensors

Most Android devices have more sensors than just the camera. From the nearly 2800 listed Android smartphones on "geizhals.de" [15] currently more than 95% have an accelerometer and about 50% have a gyroscope. These sensors can be used to compute the movement of both devices and so estimate the new position of the phone in the video image. The results of these sensors are accurate for tracking the device because most of the current virtual reality applications are based on it. Google has a platform for building augmented reality apps called ARCore [16], but it works only on a few devices which are powerful enough and support Android 7.0 or higher.

3.2.3 Visual Mark

As stated before, detecting patterns like a QR-Code can be done very fast. Because this is so fast, device A can show a pattern on the screen and device B recognize the pattern in every frame.

This method is used by the final app. The app needs the permission to draw over other apps and draw a red frame over the whole screen. The device now just has to find the biggest red contour in the image which works quite well. Obviously, it does not work if there is a bigger red object than the recorded device in the image but there is another issue with this solution: The color of the recorded video depends also on the illumination. Under some circumstances, the red frame was recorded partly yellow. It could be solved by turning the lights on, change the brightness of the recorded screen or reposition the recorded phone.

3.3 Implementation

The implementation of the device tracking is done in OpenCV, which has functions for most of the tasks. It is implemented in the following steps.

Get red parts of the image

A binary image is generated where the parts of the source image are marked.

Find largest contour

First in the binary images all contours have to be found. In a loop through all contours the biggest is found.

Approximate a polygon around the contour

The contour is approximated with a polygon. If the polygon has four edges then it is be recognized as the screen.

Chapter 4

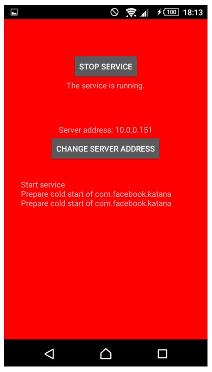
Measurement

4.1 Procedure

The procedure of the measurement can be divided into three phases:

- Initialisation: Prepare the start of the tested app and recognize the phone in the video
- Synchronisation: Show a visual mark which identifies the moment of the app start
- Waiting: Wait until the recording device believe that the app is loaded completely

4.1.1 Initialisation Phase



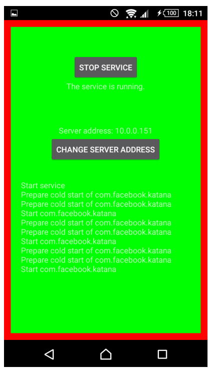
The service is running.

Server address: 10.0.0.151

```
Start service
Prepare cold start of com.facebook.katana
Prepare cold start of com.facebook.katana
```

In the initialisation phase, the recorded device shows a mostly red screen. Here the initialisations are done for the app start. The app-start-intent is created, if “No hot start” is selected, the background processes of the selected apps are killed. When the red screen is recognized by the recording device, it knows that the initialisation is done and can send the request to start the app.

4.1.2 Synchronisation Phase



The service is running.

Server address: 10.0.0.151

```
Start service
Prepare cold start of com.facebook.katana
Prepare cold start of com.facebook.katana
Start com.facebook.katana
Prepare cold start of com.facebook.katana
Prepare cold start of com.facebook.katana
Start com.facebook.katana
Prepare cold start of com.facebook.katana
Prepare cold start of com.facebook.katana
Start com.facebook.katana
```

In the synchronisation phase the app is shown with a green background. The recording device does not know when network message are received, so just from the network messages it is unclear when the app is started. To make the moment visible when the app-start-intent is started, a green screen is shown 2000ms before. So the recording app starts the stopwatch at the moment the green screen appears and then subtracts 2000ms from the measured time when it recognizes that the app is loaded completely.

4.1.3 Waiting Phase

In the waiting phase the tested app is starting. The recording device has to detect when the app is loaded completely. The only evidence to recognize this are the recorded images, so it assumes that the load is complete when the image does not change for 5000ms. The recorded images are noisy so they are usually not identical so the images have to be similar enough with a given threshold. The app uses two metrics to decide if the two images are similar enough.

Structural similarity

The structural similarity (or short SSIM) was developed in 2004 [17]. The base idea behind the algorithm was to measure the quality of lossy compressed images with adoptions to the human perception which extracts structural information from a scene. The implementation code is based on source code from an OpenCV tutorial [18].

Edges similarity

As an additional metric, the similarity of the edges of both images are compared. To compute this, the edges of both images are computed with the Canny edge detector [19]. Then the edges of the first image are dilated, the metric is then the percentage of the edges of the second images which are part of the dilated edges of the first image.

The idea behind this additional metric is to detect small contour changes. SSIM blurs the image at the beginning heavily, so contour changes are lost. Such small changes are for example streets loaded in Google Maps.

4.2 Accuracy

To measure the accuracy of the Tester App, the tested device was filmed parallel with a Canon EOS 600D at 60 frames per second. The times were then compared with the timestamps of the video. The accuracy was tested with two apps (Google Chrome and Facebook), the results were taken from two different

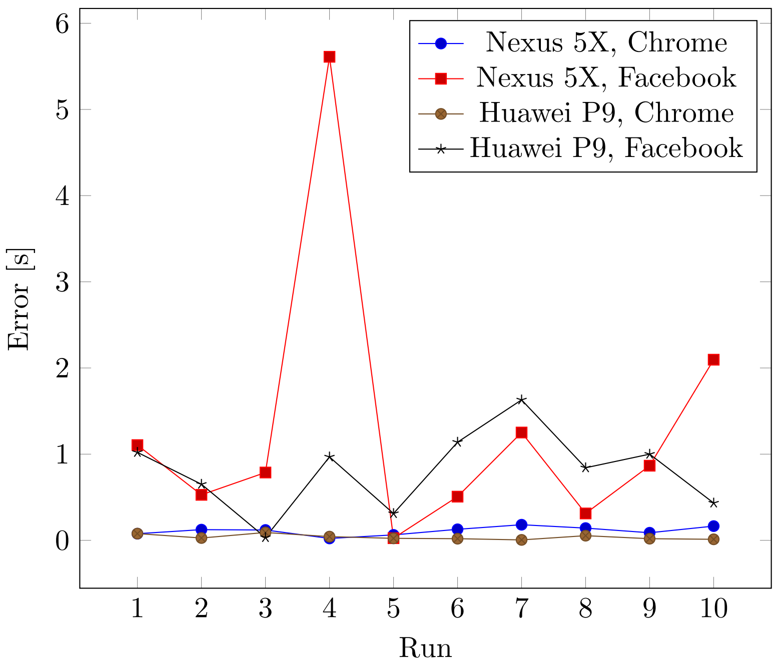


Figure 4.1: Difference between measured time and exact time

phones (Huawei P9, Google Nexus 5X). For each combination of app and measuring device there are 10 measurement results. The results of these tests are shown in Figure 4.1.

As you can see, the results of Google Chrome are much more accurate than the ones for Facebook. The reason is that Facebook loads first the structure of the page and afterwards downloads the images to fill the structures with content. A few of these images are very small so they are within the tolerance of the similarity which leads to a lower measurement than the manual measurement. Chrome is started with the default start page when opening a new tab. In contrast Chrome loads all contents in the background and shows the final screen in one rush, which is easier to detect.

There is also a difference between the phones. Due to some implementation of OpenCV, the analysed frames on the Huawei P9 have a resolution of 960x720 whereas on the Nexus 5X the resolution is 1024x768. This and the faster CPU (Appendix A.1) give the Huawei P9 a faster capture frame rate: The Huawei P9 analyses at a frame rate of around 30fps, the Nexus 5X just at a frame rate of around 20fps. The cameras are not identical which can make a difference too.

As a conclusion, the accuracy of the results depends more on the tested app than the device used for measuring.

Chapter 5

Experiments

5.1 Setup

The goal of a benchmark is to measure the performance and compare the results with other devices. The most common comparisons are done between the top-devices of different manufacturers. Many people want to know which is the fastest device. The focus in the following experiments is part of other important comparisons: The one between devices of different generations. The manufacturer wants to sell the new device so it has to be better than the old one.

5.1.1 Devices

In the tests, three generations of Sony Devices are compared: Xperia V, Xperia Z1 Compact and Xperia Z3 Compact. These phones are not the newest on the market but are very similar in their size. At their release time their synthetic benchmark scores were some of the highest.

Table 5.1 shows the technical specifications and some synthetic benchmark results of the three devices. Before the test, all the devices were reset. Afterwards all the apps were installed, started and configured so that the apps can load completely and not just to a login screen.

For the last test another Xperia Z3 is used. This device is special because it is used daily since over three years and runs some test in the non-reset state. This device is compared with the reset device in a special run to examine how much the data generated by daily usage (e.g. multiple background-services, fuller storage, more messages in WhatsApp to load) makes the device slower. Seven apps from the main test set are installed and used on this device. To not falsify the results, the state of the device remains as untouched as possible which means the test is restricted to these seven apps.

Devicename	Sony Xperia V	Sony Xperia Z1 Compact	Sony Xperia Z3 Compact
Release	2012, December	2014, January	2014, September
Android	8.1.0 (Custom)	5.1.2 (Stock)	6.0.1 (Stock)
Resolution	1280x720		
CPU	ARMv7	Snapdragon 800	Snapdragon 801
Cores	2	4	
Max. Clock	1512 MHz	2150 MHz	2457 MHz
GPU	Adreno 225	Adreno 330	
Max. Clock	400 MHz	450 MHz	457 MHz
RAM	1GB	2 GB	
Storage	8GB	16 GB	
Antutu Score[1]	23194	65323	65056
Geekbench 4 SC[1]	599	997	999
Geekbench 4 MC[1]	967	2657	2694

Table 5.1: Specifications and synthetic benchmark results of the phones

[1]Detailed results can be found in the Appendix A.1

5.1.2 Apps

The measurement is tested with a few popular apps. Google Play has "Apps-Top-Charts" in which the apps are ordered by the number of downloads. There are also apps which are preinstalled on many devices, so they do not appear in the charts, but are popular too. These are mainly apps from Google. The list of all apps can be found in Table 5.2.

Apps-Top-Charts			
	Facebook Messenger		Helix Jump
	Instagram		MeteoSwiss
	SBB Mobile		Snapchat
	Spotify		Tomb of the Mask
	Whatsapp		Wish
Popular preinstalled apps			
	Facebook		Gmail
	Google Chrome		Google Maps
	Play Store		Youtube

Table 5.2: Used apps to test the measurement

5.2 Test Methodology

To compare the devices, each runs through three different tests. For all results, the recording device is the Google Nexus 5X.

Single App

As a first test, on each device each app is started 20 times. If the app was started immediately before then the previously allocated memory is still available and the app performs a hot start [20], which means the app is loaded nearly instantly. To prevent this, before starting the app the background processes are cleaned. This test measures the start time of an app on each device. The “No hot start” and “Number of runs” options of the settings 2.1 is used to perform this test.

App Switching

The second test should deliver similar results like the first one. Instead of manually cleaning up all the apps are started in a row. Afterwards the test start again from the first app. The idea is to make use of the limited resources of the phone. Every app has some allocated memory which stays assigned until the user closes the app manually or the memory is needed for another app [20]. So if the number of other apps used between two usages is high enough the previously allocated memory is no longer available and a cold start is performed. The “Collect and run” and “Number of runs” options of the settings 2.1 is used to perform this test.

Instant Start Size

The last test wants to find out, how many apps can perform a hot start. To measure this, it performs multiple runs where an increasing number of apps are started in a row. In each run, there is one additional app, which should be the only one which does no hot-start. As soon as one of the previously started app does a cold start the limit is reached. The “Increase List Size” option of the settings 2.1 is used to perform this test.

5.3 Results

5.3.1 App Start Times

When analysing the results of the Single App-Test (Figure 5.1) we see that the Xperia V usually takes the longest to start the app. The only difference are the two games Helix Jump and Tomb of the Mask, where it is very fast. Analyzing the app start of these two apps, it gets clear why the Xperia V has the faster results: Both apps have a static loading screen before showing the app. This loading screen needs on the Xperia V more than five seconds, so the results are wrong. In reality, the loading screen for both games is shown for about 20 seconds on the Xperia V. Even on the other two devices the loading screen sometimes needs more than five seconds, so the results from these two apps are just shown in the first figures 5.1.

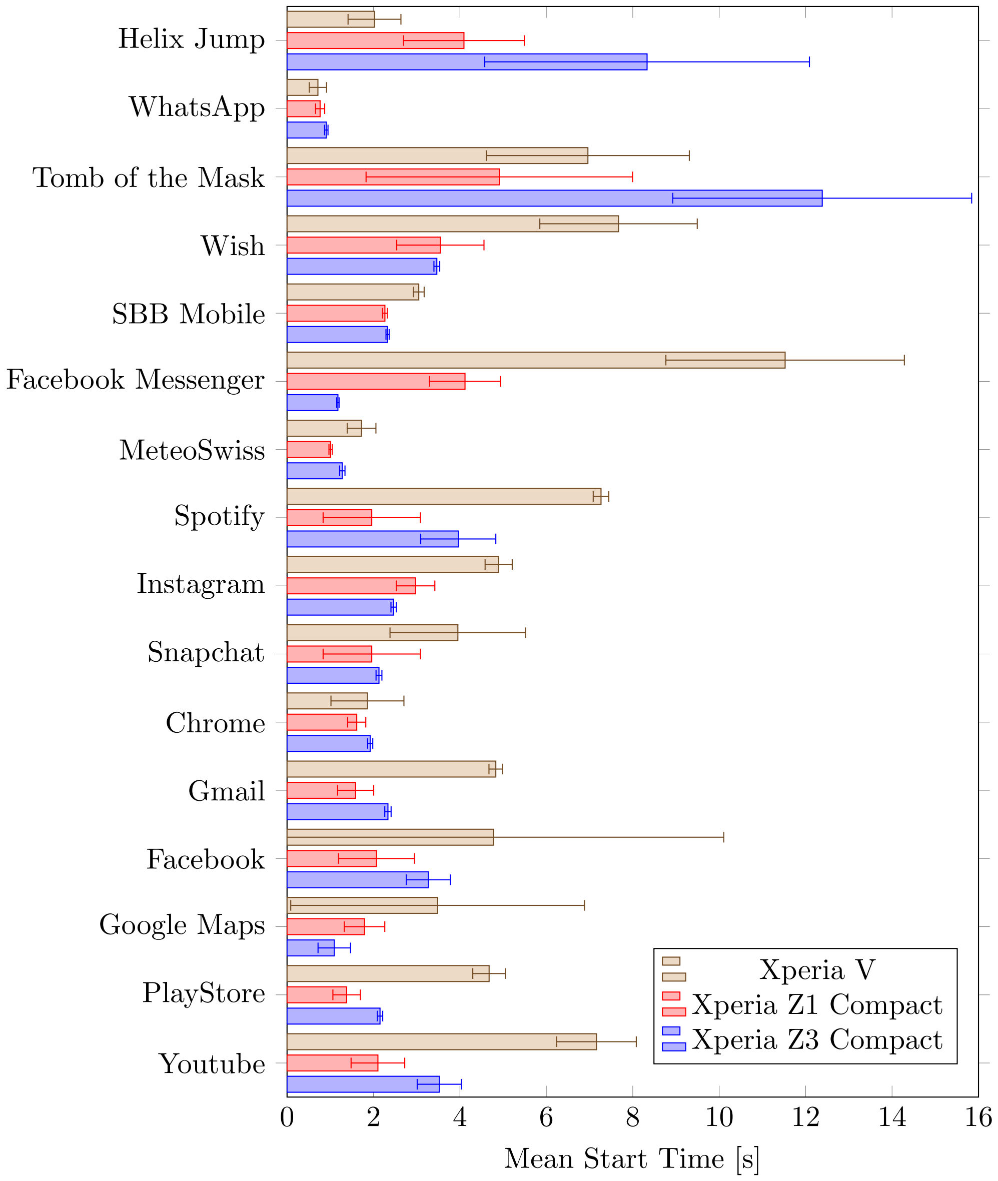


Figure 5.1: Results of the Single App-Test

Taking a closer look at the results there are a few results which are very inconsistent. It seems that killing the background services does not always work so sometimes hot starts are performed. Figure 5.2 shows the mean times without respecting these values. The order of the results does not change much, the Xperia V still has the worst performance and between the other two devices there is no clear winner identifiable.

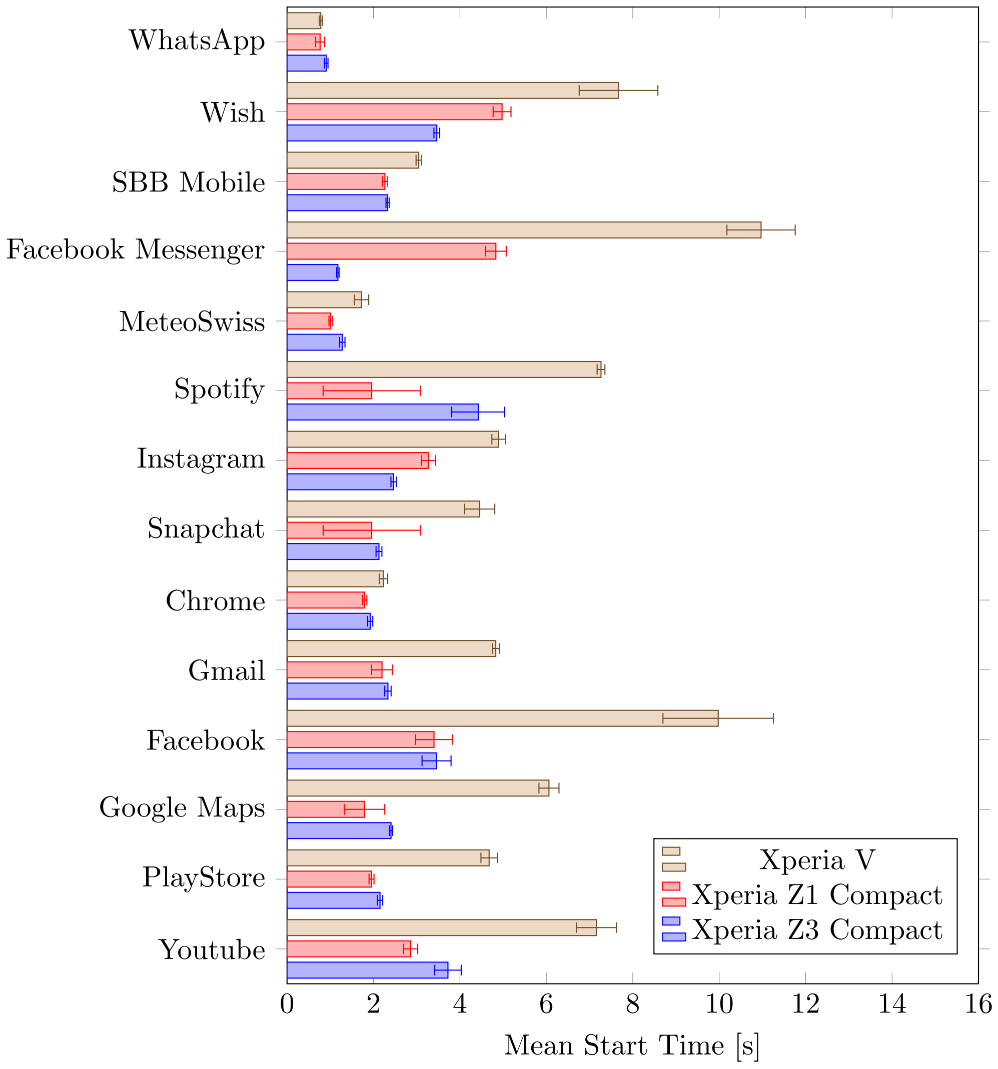


Figure 5.2: Cleared results of the Single App-Test

The results of the App Switching-Test shows that loading different apps in a row is the better way to prevent hot starts. In this test the Xperia Z3 Compact is significantly faster than the Xperia Z1 Compact, in most tests the apps load more than one second faster (Figure 5.3). In contrast to the Single App-Test at start time there are other apps open on the device, this behaviour can be seen in a side by side comparison. In most tests the Xperia V is again the slowest device. Wish, Google Maps and Youtube break this pattern but when analysing the start of these three apps, all load data in the background while showing a static loading screen. On the Xperia V, the loads in the background take that long that the time required to show this loading screen is measured. For these apps, the waiting-time of five seconds is too short to get correct results.

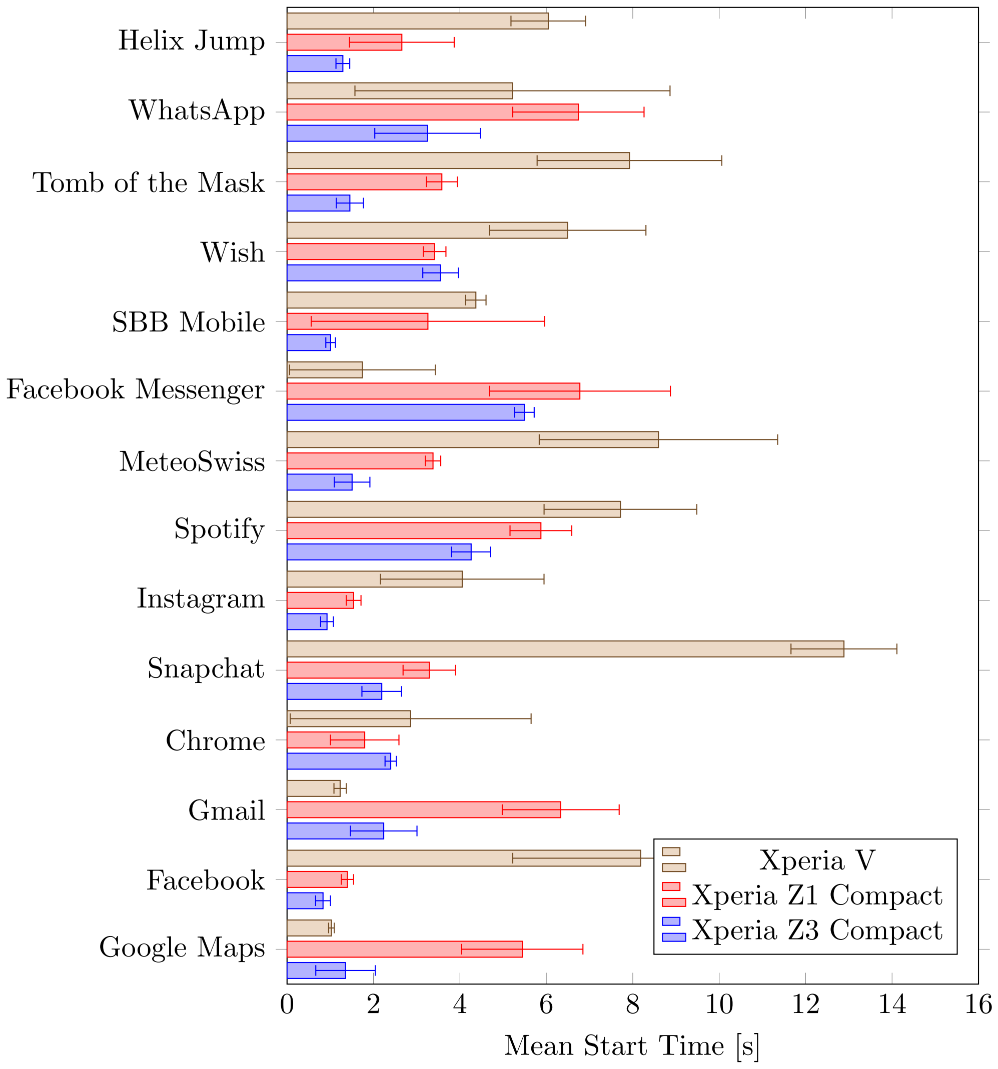


Figure 5.3: Results of the App Switching-Test

5.3.2 Instant Start Size

The results of the Instant Start Size test are similar to the start time test: The Xperia V is behind the other two whereas the Xperia Z1 Compact and the Xperia Z3 Compact perform similar. The number of apps which start instantly is not fix but depends on app (Figure 5.4). With simpler apps like the calculator, the dialer or the clock, all devices allow more instant start apps. Even the Xperia V allowed more than ten simple apps which all start instantly.

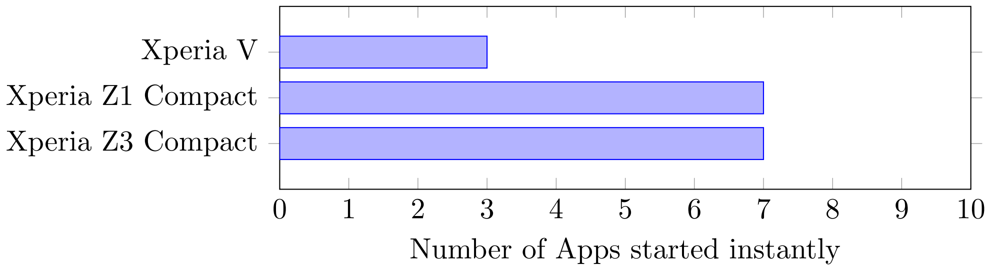


Figure 5.4: Instant Start Size

To find a difference between the Xperia Z1 Compact and the Xperia Z3 Compact, the mean times a hot start needed is compared (Figure 5.5). Here some differences are seen, in fact the Xperia Z3 Compact performed the hot start on average 300ms faster. In a classical side-by-side-comparison, this difference is visible too, but the difference vanishes when animations are disabled.

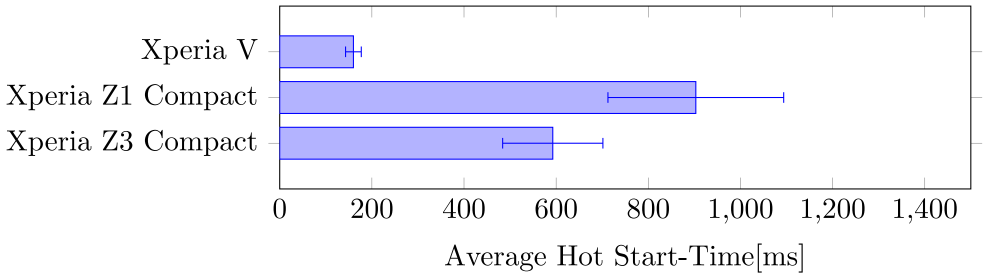


Figure 5.5: Mean time needed for an instant start

5.4 Discussion Of The Results

In the synthetic benchmarks, the newer devices deliver better results (Table 5.1). The difference between the Z1 Compact and the Z3 Compact is negligible, the Xperia V is far behind. The ranking of the devices of the tests is the same and in most tests the Xperia V is far behind. But while the results of Z1 Compact and the Z3 Compact in Antutu and Geekbench are similar, the App Switching Test shows a significant difference.

To find an explanation, we must watch the difference between the two devices. From the specifications the largest difference is the higher clock speed of the graphics chip. This gives more frames in 3D-applications but should not affect the app start results. Another difference are the preinstalled apps which are similar for both devices. The last difference is the Android version. In the changes of Android 6 [21] Google does not mention performance improvements but it is likely they have done some.

5.5 Used Phone vs Reset Phone

For the first comparison 20 starts in a row of each app are tested. Compared are the hot start times. As the first start is a cold start, the mean values do not include the first time. The results are shown in Figure 5.6. Except in Google Maps the difference between the hot start times is less than 100ms. This difference is measurable but in everyday usage not noticeable.

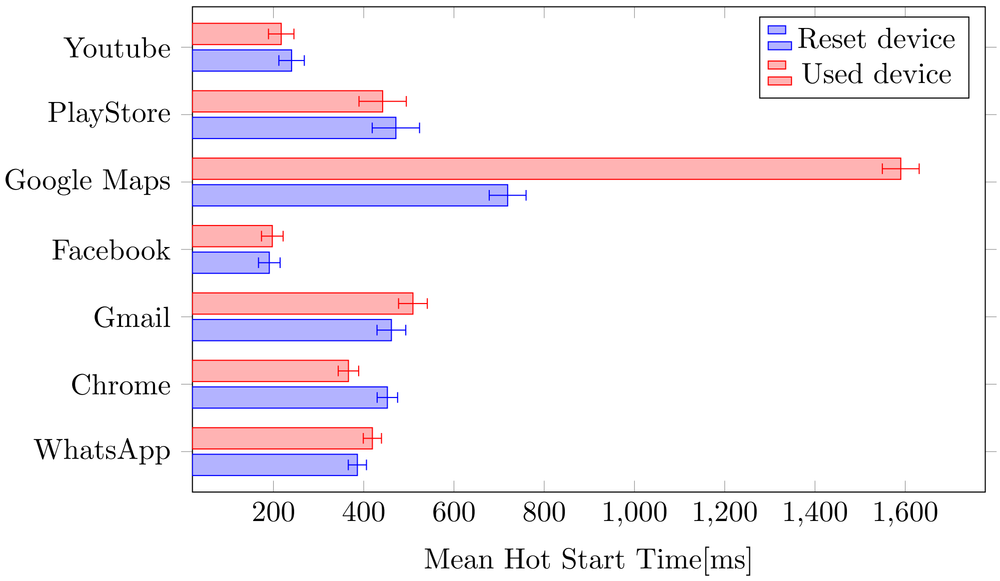


Figure 5.6: Used vs Reset: Hot Starts

Compared to the other apps the difference in Google Maps is quite huge. The individual times per run show that in the first five runs the app needed much more time to load (Figure 5.7). The cause of those slow starts is unknown, there were probably some background tasks of Google Maps like finding the location or downloading offline maps running which caused the first slow five runs.

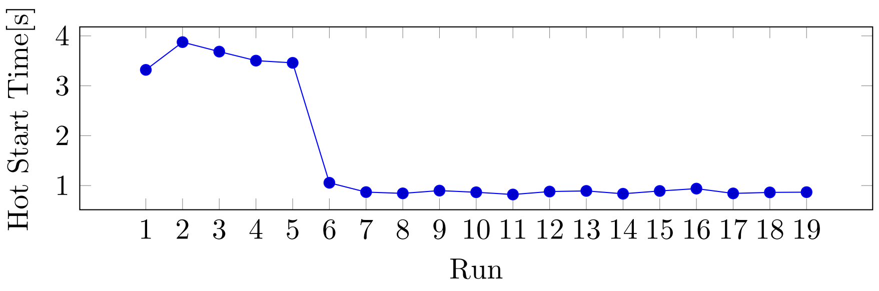


Figure 5.7: Used device: Hot start times of Google Maps

In the second test app switching (Section 5.2) was tested. To have more load two preinstalled apps are added to the test. The reset device performed during the whole test no clear cold start (Figure 5.8), the peak from Whatsapp in Run 7 is probably an error of measurement. The difference between the two devices in this test is much bigger: the used device had just two runs without a clear hot start (Figure 5.9). The apps on the used device have more data to load (e.g. the reset device has no messages in WhatsApp) which means that even the same apps are used the number of apps which can perform an hot start at the same time differ.

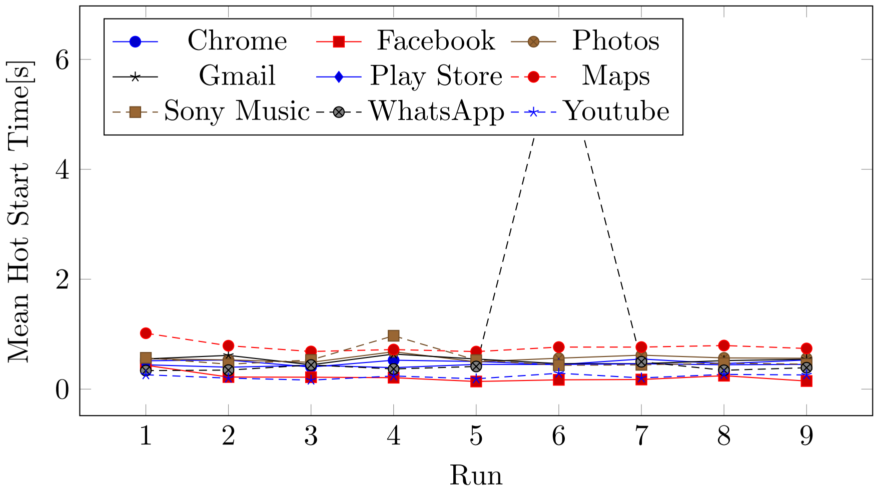


Figure 5.8: Reset device: Switch test

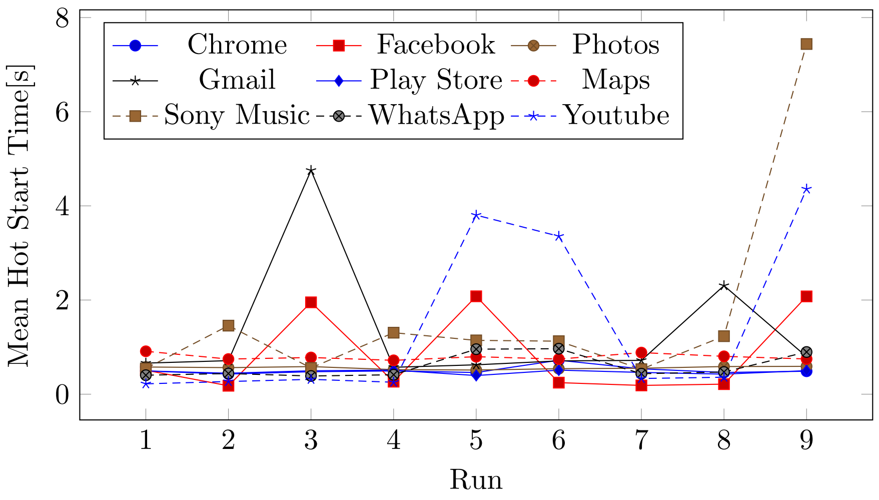


Figure 5.9: Used device: Switch test

Chapter 6

Conclusion

The results show that older top-of-the-line devices are behind when it comes to gaming performance but still are very usable for everyday tasks. Although the tests missing a current top device, because the results are durations in milliseconds and not a theoretical value in points an interpretation is possible without comparison to other devices. If a user usually does not use more than about 8 apps, the first start times of the apps may be slower, but if the apps are started once, future starts are very fast. To load apps even faster, it can help disabling animations.

It shows also that there is a limit how old the device can be: Even a new operating system does not help the Xperia V. The new OS gives the device more security patches and a better compatibility with newer apps, but the phone is missing some raw hardware performance to compete with newer devices. If the Xperia V would have more RAM, more apps would fit into memory which would lead to a improved user experience.

The comparison with the non-reset device shows that everyday usage does not really impact the real-world performance. There was a obvious difference in the App Switching-test, but even the used device performed most of the time hot starts.

6.1 Future Work

Most of the false results come from the waiting time which was too small. Five seconds were enough in most tests, for more exact results this time can be raised, but this leads also to a longer time needed for each test run so the waiting time is a compromise between accuracy and responsiveness.

If the app knows at start how the finished app looks like, the waiting time can be reduced. A possible solution may be a first run with a much longer waiting time. The app then stores then the final image. In future runs he just has to compare the taken image with the final image of the first run. This solution would not work with all apps because there are apps like Facebook which show different content, even when there is very little time between two starts.

Another variant would be a online database in which these final images are stored, so the first long run is not required. For apps with random content, the database can store the static parts of the screen or propose a good waiting time.

As a benchmark is more valuable with more results. These can be collected in the future to obtain more references to which a device can be compared.

Bibliography

[1] AnTuTu. *Antutu Benchmark - Google Play.* [Online; accessed 22-August-2018]. 2018. URL: https://play.google.com/store/apps/details?id=com.antutu.ABenchMark.

[2] Primate Labs Inc. *Geekbench 4 - Google Play.* [Online; accessed 22-August-2018]. 2018. URL: https://play.google.com/store/apps/details?id=com.primatelabs.geekbench.

[3] Futuremark Oy. *3DMark - The Gamer's Benchmark - Google Play.* [Online; accessed 22-August-2018]. 2018. URL: https://play.google.com/store/apps/details?id=com.futuremark.dmandroid.application.

[4] Futuremark Oy. *PCMark for Android Benchmark.* [Online; accessed 22-August-2018]. 2018. URL: https://play.google.com/store/apps/details?id=com.futuremark.pcmark.android.benchmark.

[5] Top500. *Top500 JUNE 2018.* [Online; accessed 28-August-2018]. 2018. URL: https://www.top500.org/lists/2018/06/.

[6] Dictionary. *personal computer.* [Online; accessed 29-August-2018]. 2018. URL: https://www.dictionary.com/browse/personal-computer.

[7] Kishonti Ltd. *GFXBench Benchmark.* [Online; accessed 29-August-2018]. 2018. URL: https://play.google.com/store/apps/details?id=com.glbenchmark.glbenchmark27.

[8] Gino Brunner. "Android Benchmark". In: (2016).

[9] Mishaal Rahman. *Google is Threatening to Remove Apps with Accessibility Services from the Play Store.* [Online; accessed 29-August-2018]. 2017. URL: https://www.xda-developers.com/google-threatening-removal-accessibility-services-play-store/.

[10] OpenCV team. *OpenCV Website.* [Online; accessed 23-August-2018]. 2018. URL: https://opencv.org/.

[11] Nvidia. *Mobile Visual Computing @ GPU Technology Conference.* [Online; accessed 23-August-2018]. 2013. URL: http://developer.download.nvidia.com/GTC/PDF/GTC2012/PresentationPDF/SB140-Computational-Photography-Visual-Computing.pdf.

[12] Google. *Mobile Vision.* [Online; accessed 23-August-2018]. 2018. URL: https://developers.google.com/vision/.

[13] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. “Forward-backward error: Automatic detection of tracking failures”. In: *Pattern recognition (ICPR), 2010 20th international conference on*. IEEE. 2010, pp. 2756–2759.

[14] Bruce D Lucas, Takeo Kanade, et al. “An iterative image registration technique with an application to stereo vision”. In: (1981).

[15] Geizhals. *Handys ohne Vertrag mit Betriebssystem: Android*. [Online; accessed 23-August-2018]. 2018. URL: https://geizhals.de/?cat=umtsover&xf=148_Android.

[16] Google. *ARCore Overview*. [Online; accessed 28-August-2018]. 2018. URL: https://developers.google.com/ar/discover/.

[17] Zhou Wang et al. “Image quality assessment: from error visibility to structural similarity”. In: *IEEE transactions on image processing* 13.4 (2004), pp. 600–612.

[18] OpenCV. *Video Input with OpenCV and similarity measurement*. [Online; accessed 28-August-2018]. 2018. URL: https://docs.opencv.org/2.4/doc/tutorials/highgui/video-input-psnr-ssim/video-input-psnr-ssim.html#videoinputpsnrmssim.

[19] John Canny. “A computational approach to edge detection”. In: *IEEE Transactions on pattern analysis and machine intelligence* 6 (1986), pp. 679–698.

[20] Google. *App Startup Time*. [Online; accessed 28-August-2018]. 2018. URL: https://developer.android.com/guide/components/activities/activity-lifecycle.

[21] Google. *Android 6.0 Changes*. [Online; accessed 28-August-2018]. 2018. URL: https://developer.android.com/about/versions/marshmallow/android-6.0-changes.

Appendix A

Appendix Chapter

A.1 Results Synthetic Benchmarks

Each device was tested three times, these are the highest values.

Value	Huawei P9	Nexus 5X	Xperia V	Z1 Compact	Z3 Compact
Antutu All	114471	82549	23194	65323	65056
Antutu CPU	55519	33509	10193	37266	36141
CPU-Math	16318	8961	3847	8106	8225
CPU-Common	8714	6131	2785	4472	4753
CPU-Multi	30487	18417	3561	24688	23163
Antutu GPU	17915	23104	0	5526	6608
GPU-Marooned	2409	4831	n/a	2452	3029
GPU-Coastline	4358	9040	n/a	3074	3579
GPU-Refinery	11148	9233	n/a	n/a	n/a
Antutu UX	33355	20421	10597	17509	17349
UX-Data-Sec	6792	3296	1515	2421	2024
UX-Data-Proc	8360	4368	1705	2542	2775
UX-Img-Proc	3374	4574	218	3831	4095
UX-User-Exp	14829	8183	7159	8715	8455
Antutu MEM	7682	5515	2404	5022	4958
MEM-RAM	2581	1992	1705	3115	2992
MEM-ROM	5101	3523	699	1907	1966
Geekbench 4 SC	1730	1209	599	997	999
Geekbench 4 MC	5026	2699	967	2657	2694
SC Crypto	1317	868	29	46	51
SC Integer	2055	1312	601	958	981
SC FP	1516	980	347	630	684
SC Memory	1470	1411	1120	1872	1753
MC Crypto	4516	2852	57	179	197
MC Integer	6741	3910	1110	3406	3527
MC FP	4650	1792	643	2206	2302
MC Memory	1859	1487	1360	2265	2071