

# WiFi to LTE handover in mobile phones

Johannes Alexander Berg



Thesis submitted for the degree of  
Master in Informatics: Programming and Networks  
30 credits

Department of Informatics  
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2019



# **WiFi to LTE handover in mobile phones**

Johannes Alexander Berg

© 2019 Johannes Alexander Berg

WiFi to LTE handover in mobile phones

<http://www.duo.uio.no/>

Printed: Representralen, University of Oslo

# Abstract

Today many devices like laptops and smartphones have access to both WiFi and LTE at the same time. The reliability and performance of wireless networks and the Internet is not perfect. Sometimes users need to manually disconnect devices from slow or unreliable WiFi networks in order to switch to LTE. The Android operating system already includes a way to automatically switch to using LTE when the connected WiFi network becomes unstable. In this thesis we analyse this switching between WiFi and LTE in Android. We do this by emulating various network conditions in order to provoke an Android phone to switch from WiFi to LTE.

We show that the Android phone sometimes can stay connected to WiFi network networks even when there is very poor Internet access. Informed by this we propose and implement a simple algorithm to disconnect from WiFi networks that are performing poorly. We show that our implementation is more consistent in switching to LTE from WiFi networks with poor performance, however it also leads to much higher overhead in terms of data usage.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Problem statement . . . . .	1
1.3	Scope and limitations . . . . .	2
1.4	Research method . . . . .	3
1.5	Main contributions . . . . .	3
1.6	Thesis structure . . . . .	4
<b>2</b>	<b>Background</b>	<b>5</b>
2.1	WiFi . . . . .	5
2.2	Cellular networks . . . . .	5
2.3	Handover . . . . .	6
2.3.1	Horizontal handover . . . . .	6
2.3.2	Vertical handover . . . . .	6
2.3.3	Handover control . . . . .	7
2.3.4	The handover process . . . . .	7
2.3.5	Soft vs hard handover . . . . .	7
2.3.6	The ping-pong effect . . . . .	8
2.3.7	The focus of this thesis . . . . .	8
2.4	Handover in Android . . . . .	8
2.5	Network performance metrics . . . . .	9
2.5.1	Bandwidth and throughput . . . . .	10
2.5.2	Delay . . . . .	10
2.5.3	Packet loss . . . . .	10
2.5.4	Signal strength and signal to noise ratio . . . . .	10
2.6	captive portals . . . . .	11
2.7	Related work . . . . .	11
2.8	summary . . . . .	16
<b>3</b>	<b>Methodology</b>	<b>17</b>
3.1	Experimental setup . . . . .	17
3.1.1	Hardware . . . . .	17
3.1.2	choice of operating systems . . . . .	20
3.1.3	Router configuration . . . . .	20
3.1.4	software tools . . . . .	20
3.2	WiFi to LTE handover experiments . . . . .	21
3.2.1	Signal strength . . . . .	21

3.2.2	Captive Portal . . . . .	22
3.2.3	Prerequisites to run to automate experiments . . . . .	23
3.2.4	Traffic control in linux . . . . .	23
3.2.5	Android automation with ADB . . . . .	24
3.2.6	Testing performance thresholds for triggering handover . . . . .	27
3.2.7	Testing how long it takes before handover is initiated. . . . .	27
3.3	Battery consumption . . . . .	27
3.4	Summary . . . . .	29
<b>4</b>	<b>Handover from WiFi to LTE in Android</b>	<b>31</b>
4.1	Handover threshold results . . . . .	31
4.1.1	Handover caused by delay . . . . .	32
4.1.2	Handover caused by Packet loss . . . . .	33
4.1.3	Handover caused by low throughput . . . . .	34
4.1.4	Handover caused by low signal strength . . . . .	35
4.2	How long does it take to make the handover decision? . . . . .	35
4.3	TCPdump analysis . . . . .	37
4.4	Captive portal . . . . .	42
4.4.1	Deauthentication from captive portal . . . . .	42
4.4.2	Summary of handover in Android . . . . .	42
<b>5</b>	<b>A new algorithm for initiating WiFi to LTE handover</b>	<b>45</b>
5.1	New handover algorithm . . . . .	45
5.2	File transfer for bandwidth estimation . . . . .	46
5.2.1	tuning parameters for the new algorithm . . . . .	46
5.2.2	Results of testing the new algorithm in comparison with existing handover in Android . . . . .	48
5.3	Overhead . . . . .	51
5.3.1	Data usage . . . . .	51
5.3.2	Energy usage . . . . .	52
5.4	Summary and conclusions . . . . .	53
<b>6</b>	<b>Conclusion</b>	<b>55</b>
6.1	Summary of contributions . . . . .	55
6.2	Future work . . . . .	55



# List of Figures

1.1	xkcd: WiFi vs cellular [43]	2
2.1	WiFi handover related settings in Android	9
3.1	Setup	18
3.2	hardware setup	19
3.3	Transmit power configuration	22
3.4	This figure shows the steps we used to find out if handover is triggered at a boundary value for various network performance metrics. First the phone connects to Wifi, then we apply a command on the router to create packet loss, delay or other network conditions. We keep three variables, limit, upper_limit and lower_limit. Limit is used as a parameter to determine for example how many kbps of throughput to allow. Then based on if handover is detected or not, we update the variable_upper limit or lower_limit. Then we update limit and check if upper_limit and lower_limit have converged.	28
3.5	Procedure for measuring time until handover initiation.	29
4.1	Frequency of ping packets sent from the mobile phone when Smart network switch disabled. The X axis is time in seconds and Y axis is packets per second.	39
4.2	Frequency of ping packets sent from the mobile phone when Smart network switch is set to aggressive. The X axis is time in seconds and Y axis is packets per second.	40
4.3	Ping packets between phone and router captured by TCPdump with packet loss created by the router.	41
4.4	captive portal check in Android	42
5.1	Handover Initiation algorithm	46
5.2	Handover initiation algorithm	47
5.3	Data usage while adjusting parameters	52
5.4	Battery usage while adjusting parameters	53



# List of Tables

2.1	Generations of WiFi standards [40] [11] . . . . .	6
2.2	Generations of mobile communication systems . . . . .	6
2.3	Parameters for handover decisions. [44] . . . . .	12
2.4	Vertical handover algorithm performance metrics . . . . .	13
2.5	Classification of vertical handover decisions. [44] . . . . .	14
3.1	TP-Link Archer C7 V2 specifications . . . . .	19
4.1	Overview of experiments performed when testing the exist- ing handover in a Android. . . . .	32
4.2	Delay boundary values for Android phone . . . . .	33
4.3	packet loss values for Android phone . . . . .	34
4.4	throughput boundary values for Android phone . . . . .	35
4.5	Signal strength boundary values . . . . .	36
4.6	Time taken before making the handover decision. . . . .	37
4.7	Time taken before making the handover decision while streaming video. . . . .	38
4.8	Internet blocked by captive portal . . . . .	43
5.1	Delay boundary values for Android phone . . . . .	48
5.2	packet loss values for Android phone . . . . .	49
5.3	throughput boundary values for Android phone . . . . .	49
5.4	Time taken before making the handover decision. . . . .	50



# Listings

3.1	Capturing packets with tcpdump . . . . .	21
3.2	Getting signal strength example . . . . .	22
3.3	ndsctl example . . . . .	22
3.4	Throughput limit with tbf . . . . .	23
3.5	Setting delay with netem . . . . .	24
3.6	Setting loss with netem . . . . .	24
3.7	WiFi control through adb . . . . .	24
3.8	Broadcast with adb . . . . .	25
3.9	Android broadcast receiver . . . . .	25
3.10	read system log with adb . . . . .	25
3.11	Detecting WiFi handover in Android . . . . .	26



# Preface

I would like to thank my supervisors Dr. Audun Fosselie Hansen and Professor Pål Halvorsen for their invaluable guidance throughout this thesis.





# Chapter 1

## Introduction

### 1.1 Motivation

Smartphones often have access to both WiFi and cellular networks. There are many factors that can affect the quality of wireless networks. Poor network quality can prevent the use of resources over the Internet. In our opinion having to manually switch between networks in order to determine the best one is not optimal. This is illustrated in Figure 1.1 on the following page.

Network connectivity can fail in many different ways. I often want to access the Internet through my phone while waiting for the microwave oven to finish. This is usually impossible even with good signal strength. The phone stays connected to WiFi, but interference from the microwave oven makes it completely impossible to reach anything on the Internet. Another annoying situation is when you are waiting for a buss or similar outside of your home or office. At this distance your phone might still be able to maintain contact with the WiFi access point, but due to the low signal strength the connection can be very unreliable.

By 2021 smartphone traffic is expected to exceed PC traffic, and more than 63 percent of total IP traffic will be caused by mobile and wireless devices.[7] If it is possible to make any sort of improvement to the way mobile phones handle multiple network connections that would be quite useful.

Switching from one network to another is called a handover, and Android phones already includes the ability to automatically switch from WiFi to LTE when the WiFi network becomes unstable. However, our subjective experience as users of android phones is that the phones sometimes gets stuck on poor WiFi networks even when the most aggressive handover settings are applied. In this thesis we investigate experimentally whether this actually is the case.

### 1.2 Problem statement

Today phones can be connected to WiFi networks without internet access or WiFi networks that give almost no throughput. Phones often have

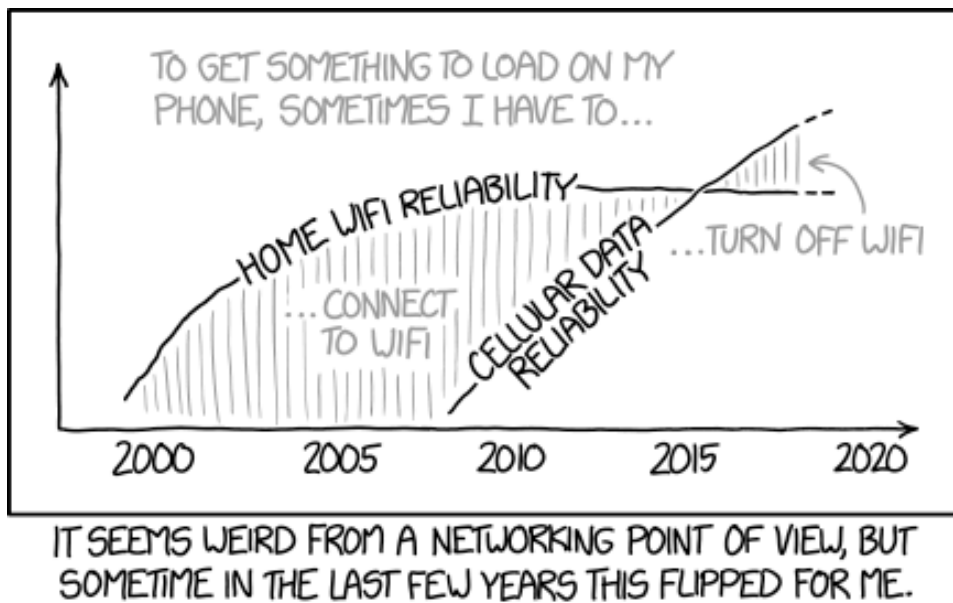


Figure 1.1: xkcd: WiFi vs cellular [43]

access to both WiFi and mobile network connections. In this thesis we will examine handover from WiFi to LTE in Android phones. More specifically we will attempt to answer the following two research questions:

1. Do Android phones stay connected to WiFi networks with poor performance or do they automatically initiate handover?
2. Can we improve the automatic handover in Android?

To answer the first research question we need to investigate if there are thresholds for how poor performance of the WiFi network can become before handover is triggered and also how long it takes from losing Internet connection in a WiFi network until handover is triggered in the phone.

If it turns out that Android phones actually stay connected to WiFi networks with poor performance instead of handing over to LTE, then we can propose a new system for handover and test the new system. For example, if it currently takes a long time from Internet connectivity becomes unavailable until Android switches over to LTE, then we could try to reduce this time. Another possibility would be that we find situations where the Internet is not reachable through WiFi but Android for some reason does not switch over to LTE. In that case we could attempt to find a way to detect that these situations have occurred.

### 1.3 Scope and limitations

In this thesis we are focusing on Samsung smartphones with the Android 7.0 operating system. For the mobile network a 4G LTE connection will be used, and for WiFi we will use IEEE 802.11ac-2013 in the 5GHz band.

There are many different manufacturers of Android based smartphones, and the handover process might be handled differently in Android based smartphones made by other manufacturers and it could change in newer versions of Android.

## 1.4 Research method

The ACM task force on the core of computer science proposed three research paradigms for computer science [8]. These are theory, abstraction and design.

In the theory paradigm one follows four steps to develop a coherent and valid theory. These are: (1) characterize objects of study(definition), (2) hypothesize possible relationships among them(theorem), (3) determine whether the relationships are true (proof) and (4) interpret results.

The abstraction paradigm is used in the investigation of a phenomenon and consists of the steps: (1) form a hypothesis, (2) construct a model and make a prediction, (3) design an experiment and collect data (4) analyse results.

The final paradigm is design, which is rooted in engineering and consists of four steps. These are described in the report in the following way; (1) state requirements, (2) state specifications, (3) design and implement the system, (4) test the system.

The research method for this thesis support the design paradigm through proposing an system for automatically initiating handover from WiFi to LTE and then implementing and testing this system. In our methodology we analyse an existing system for handover in Android, then we develop a prototype for a new system and test the prototype in comparison to the existing system.

## 1.5 Main contributions

The main contributions we have done during the work on this master thesis is firstly to perform experiments to evaluate the existing solution to automatic handover from WiFi to LTE in Android phones. We find that Android does manage to handover to LTE, but that it often takes more than a minute of being connected to a WiFi network with practically no Internet access before handover is initiated. We also attempt to analyse how the handover mechanism in Android works and why it performs the way it does.

Secondly, informed and motivated by the results of analyzing handover in Android and related works, we have proposed an alternative way to automatically initiate handover. Finally, we compare it with the existing solution in Android.

## 1.6 Thesis structure

This thesis is organized as follows. Chapter 2 covers the relevant background and related work. We describe how handover algorithms can be defined, and how different handover algorithms can be classified. Then we cover metrics for measuring network performance, which can be used as part of the handover decision. Finally, we look at some other relevant studies of handover algorithms.

Chapter 3 describes the methodology we used to answer the research questions. It covers the hardware and software setup and the procedure for how we performed experiments.

In chapter 4 we analyse the results of performing the experiments described in chapter 3 with the existing handover mechanism in Android. We look at how handover was triggered by packet loss, high delay and low data throughput. We also look at how long it takes to trigger handover in Android.

In chapter 5 we propose a new algorithm for initiating handover, which is motivated by the results presented in the previous chapter. We also present the results of performing experiments to test how handover is triggered and how long it takes to trigger handover on the proposed algorithm and compare the results of this with the results of the testing the existing Android handover mechanism.

Finally in chapter 6 we conclude the thesis with a summary of what we have presented and the most important conclusions in addition to discussing potential future work.

## Chapter 2

# Background

The focus in this thesis is preventing mobile phones with access to both WiFi and LTE from being stuck on WiFi network with no or poor Internet access. We start this chapter by briefly cover what LTE and WiFi are. Then we discuss the basics of handover. Switching from one network to another is called a handover, and We will elaborate on how handover can be defined and classified and the different kinds of handover algorithms that exist. We also discuss the options for automatically switching from WiFi to LTE that already exist in Android. After this, we also discuss and some metrics that can be used to evaluate network performance. These can potentially be used as parameters for the decision of initiating handover. Finally we will discuss existing work that has been done on handover algorithms.

### 2.1 WiFi

Wireless local area networks (WLAN) are wireless computer networks where two or more computers form a network using wireless communication. WLANs are typically at the scale of a building and are commonly used within homes and companies. WiFi is a set of communication standards for WLAN. It contains specifications for the data-link layer and physical layer of the Open Systems Interconnection (OSI) model. These specifications are for implementing wireless communication in the 2.4GHz, 5GHz, and 60GHz frequency bands. Table 2.1 on the next page shows an overview of WiFi generations. In addition to the ones listen in the table the WI-FI Alliance has introduced 802.11ax, which is also know as WiFi 6 and will be the next generation WiFi standard. The router we are using in this thesis is using the 802.11ac wave 1 standard.

### 2.2 Cellular networks

Cellular networks are a form of wireless wide area networks (WWAN). There are currently four generations of mobile communications standards. Each generation includes several technologies. An overview of generations

<b>Standard</b>	<b>release year</b>
Legacy 802.11	1997
802.11b	1999
802.11a	1999
802.11g	2003
802.11n	2009
802.11ac wave 1	2014
802.11ac wave 2	2016

Table 2.1: Generations of WiFi standards [40] [11]

<b>Generation</b>	<b>Technologies</b>
1G	NMT, AMPS, TACS
2G	GSM, IS-136, PDC, IS-95
3G	WCDMA/HSPA, cdma2000, TD-SCDMA
4G	LTE

Table 2.2: Generations of mobile communication systems

of mobile communication standards can be seen in Table 2.2. Specifications for these can be found on the 3GPP website [36]. In addition to the generations listed in the table, the first specifications for the fifth generation was completed in 2017 [1].

## 2.3 Handover

Handover or handoff is the process where a network node changes or attempts to change its attachment point to a network. Within the literature, the terms "handoff" and "handover" are used interchangeably according to [31]. In this thesis the Android phone takes the role of the network node. Handover can be classified into different types based on various aspects involved [24]. The following subsections discuss some of these types and how they relate to handover from WiFi to LTE.

### 2.3.1 Horizontal handover

In this type of handover mobile nodes move from one access point to another of the same type. For instance, a node might handover from one WiFi access point to another, or from one cell to another when moving between the border region of two cells in a cellular network.

### 2.3.2 Vertical handover

In vertical handover mobile nodes move from an access point of one type to a different type. This thesis focuses on handover from WiFi to LTE which is a vertical handover. According to [44], vertical handover algorithms need

to allow seamless roaming among multiple access network technologies, and they also need to be designed to provide the quality of service that is required by a wide range of applications.

### **2.3.3 Handover control**

Handover can also be classified in several ways depending on how it is controlled. Handover is either mobile-initiated or network-initiated depending on who initiated the handover process. Furthermore handover can be either mobile-controlled or network-controlled depending on who has primary control of the process.

Handover can also be classified based on where it obtains information used to make the decision to initiate handover. If a handover is network controlled and information collected by the mobile node is used to assist, then the handover is mobile-assisted. In the opposite situation where handover is mobile-controlled and uses information from the network, we can qualify the handover as network-assisted. The final alternative is that the network and mobile node does not assist each other.

### **2.3.4 The handover process**

The handover process consist of several stages. It can be divided into initiation, decision and execution. In the initiation stage handover is triggered based on some criteria like low signal strength or network congestion. After the initiation stage the process enters the decision stage. During this stage the process makes a decides which access point to hand over to. To make this decision several parameters can be used. For example, signal strength. The final stage is execution. In this stage communication is established with the new access point and and data is re-routed through the new path. [25]

### **2.3.5 Soft vs hard handover**

Handover can also be classified as soft or hard. The terms soft and hard handover are defined for the UMTS system [23]. Hard handovers are also known as break-before-make. In this type of handover the connection to the old cell is released before a connection to the new cell is established.

Soft handovers are also known as make-before-break. In this variant connection to the old cell is not broken before connection to the new cell is established. It is possible that both connections will be used simultaneously for a long period of time in soft handover.

The terms soft and hard handover are also used in the context of vertical handover. Here an example of a soft handover could be that the mobile terminal executes handover from WiFi to LTE, but retains connection to the WiFi access point while using the LTE connection.

### 2.3.6 The ping-pong effect

The ping-pong effect was introduced in the context of horizontal handover in cellular networks. When a mobile node is moving at the boundary between the two base stations, rapid changes in received signal strengths could force the node to hand over from one base station to the next and then back to the original again. The mobile node may continue to "ping-pong" between the base two stations until it moves sufficiently far into the coverage of one of them [31].

The ping-pong effect is also used to describe undesired handing back and forth between networks in the context of vertical handover. For example, some of the vertical handover algorithms surveyed in [44] use heuristics to eliminate the ping-pong effect.

### 2.3.7 The focus of this thesis

In this section we have elaborated on the different kinds of handover and some possible ways to classify handover. So how does this thesis fit into the world of handover algorithms? The focus is handover from WiFi to LTE, which is classified as a vertical handover. The mobile phone is the one initiating handover and the decision is only based on information gathered by the phone itself, so the handover can be classified as mobile initiated and mobile controlled. Focus is mainly on the initiation part of the handover process. More specifically the goal is that the decision to initiate handover should be taken if the connected WiFi network no longer is providing sufficient quality. The decision stage of choosing which network to handover too is left to the Android system. This means that when the handover decision is made we terminate connection to the WiFi access point, meaning that the handover is break before make or hard handover.

## 2.4 Handover in Android

Our experience as users is that Android based phones sometimes gets stuck on poor WiFi networks even with the most aggressive handover settings. Figure 2.1 on the facing page shows the menu for the handover related settings that are available in Android 7.0. On the left we can see smart network switch, which is found under the WiFi section of the settings app. When enabled this setting is supposed to make the mobile phone use a mobile network like LTE for Internet connectivity Instead of WiFi when WiFi becomes unstable without the need for the user to manually disconnect from WiFi. There is also an option to toggle Aggressive Switching to use mobile networks when WiFi is only slightly unstable. For Android 8.0 and newer, Samsung have renamed smart network switch to adaptive WiFi in their phones[41].

On the right we can see that under the developer options there is a setting to more aggressively hand over to cellular when WiFi signal strength is low. The developer options contain options that meant for debugging and development purposes. In order to access developer



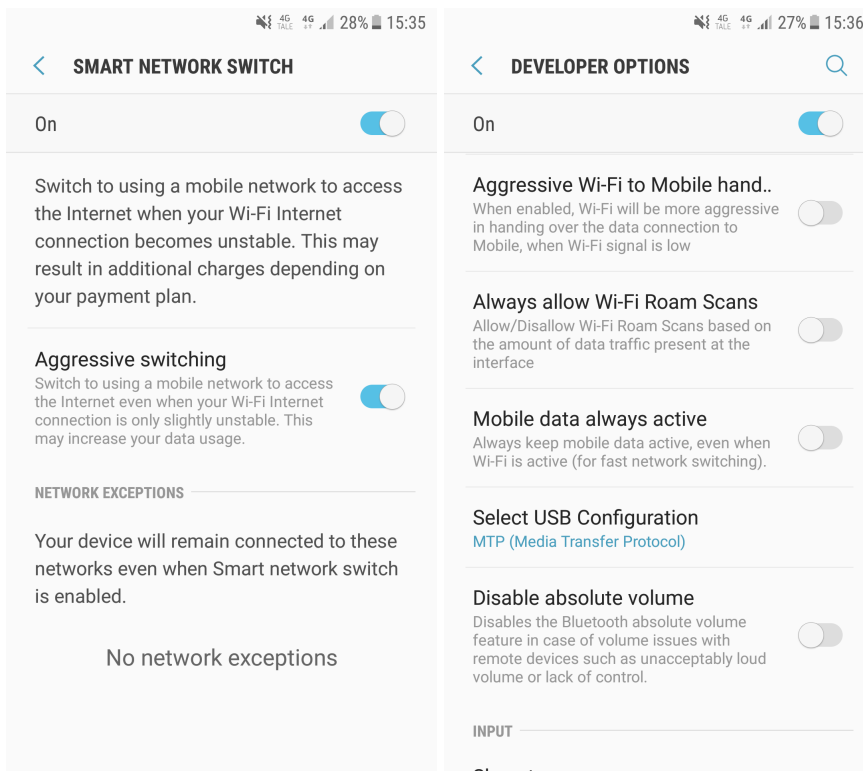


Figure 2.1: WiFi handover related settings in Android

options one must first enable developer mode. On Samsung devices this is done by touching build number under software info in the settings app[19]. The fact that the setting to more aggressively switch over from WiFi to cellular is only accessible in developer options suggests that it is not yet meant to be used by ordinary users and that it still might be under development.

In this thesis we will investigate if these settings actually can prevent the problem of getting stuck on a WiFi network with poor connectivity, and why they how they solve or don't solve the problem. By using experiments in a controlled setup we can investigate if Android phones actually do get stuck on poor networks without initiating handover and if applying these settings makes any difference. If we find that the mobile phone does get stuck, then we can use the results as a basis for comparison that we can use in our attempt to improve the handover behavior.

## 2.5 Network performance metrics

Handover algorithms need some basis for decision making. The algorithms must decide when to initiate handover and there might be several available candidate networks to choose from. Network performance metrics gives a way to quantitatively compare different networks, and when evaluating handover algorithms it can be useful to investigate how the algorithms

cope with various performance problems in the networks.

In [17] Hanemann et al. have created a study on network performance metrics and their composition. There they provide explanations of some network performance metrics. The metrics relevant for this thesis will be explained in this section.

### **2.5.1 Bandwidth and throughput**

In [9], Comer defines both throughput and bandwidth. Throughput is the capacity of a network. It is a measurement of how much data can be transferred through the network in a unit of time, and is often measured in bits per second.

Bandwidth, on the other hand, is related to the number of signal levels in a transmission medium, and determines the maximum amount of data that theoretically can be sent in a given time. 802.11ac can achieve a data rate of is 3.46Gbps using eight antennas [20], and the advertised maximum data rate of the TP-Link Archer C7, which we are using in this thesis, is 1.3Gbps for the 5Ghz band. Even though there is a difference between throughput and bandwidth, as Comer explains in [9], network bandwidth is generally used to refer to data rate in the networking industry.

In a multi-hop path the link with the lowest throughput will be the limiting factor, so the WiFi link itself will not always be the limiting factor.

### **2.5.2 Delay**

The delay metric can be either one way delay (OWD), which measures how much time it takes to transfer a packet from the sender to the destination, or it can be round trip time(RTT) which is the time it takes to send it takes to send a packet from sender to receiver and to send a reply back again.

### **2.5.3 Packet loss**

Packet loss indicates how many of the sent packets that are lost. It is usually measured as the percentage of packets lost compared to the total number of packets. There are various reasons that packets can get lost in the network. For example, transmission errors or congestion.

### **2.5.4 Signal strength and signal to noise ratio**

WiFi signal strength is typically measured in three different ways, milliWatts (mW), Received Signal Strength Indicator (RSSI) and decibel-milliwatts. A potentially challenging scenario for handover algorithms is when you far enough away from the access point that you have poor throughput but not so far away that the mobile phone has lost contact with the access point.

## 2.6 captive portals

Captive portals are used to restrict access to public WiFi networks. When a user connects to a WiFi network they get redirected to a web portal where they can log in to gain access to the Internet. Captive portals are relevant to this thesis because they restrict Internet access and therefore makes it impossible to reach the Internet for devices that have not been authenticated. If the captive portal somehow is not detected when connecting to WiFi, then the result is that the phone could end up connected to the unusable WiFi without handing over to LTE. Another potential problem with captive portals is that a device might become deauthenticated after using the network for some time and therefore be prevented from accessing the Internet. The phones should be able to detect that Internet access is no longer available, and maybe do a handover to LTE.

We typically find captive portals in places like hotels or airports. So a potential scenario where a captive portal causes problems would be that you are visit a hotel you have stayed at before and your phone connects to the WiFi which is known from the previous visit. Internet access is restricted by the captive portal and you need to get a password from the staff before you can log in. In this situation the phone should correctly detect the captive portal and prompt the user to authenticate or disconnect.

## 2.7 Related work

Much work has been done on vertical handover algorithms. [44] is a survey on vertical handover decision algorithms in fourth generation heterogenous wireless networks. In [44] Yan et al. compare many vertical handover algorithms, but before that they start the paper by presenting background on vertical handover algorithms. They presented several parameters that have been proposed in literature to use in the vertical handover decision algorithms. An overview of these parameters is presented in table 2.3 on the next page.

They also presented a classification on vertical handover decision algorithms based on the criteria used to classify them. Table 2.5 on page 14 gives an overview of their classification of vertical handover algorithms.

Finally Yan et al. also presented some metrics for evaluating vertical handover algorithms. These are handover delay, number of handovers, handover failure probability and throughput. These are explained in more detail in Table 2.4 on page 13.

In [5] Busanelli et al. present two vertical handover algorithms and an experimental performance analysis of these. The scenario is very similar to the scenario considered in this thesis. They are testing a RSSI based vertical handover algorithm and a hybrid RSSI/goodput algorithm. Their algorithm is running on a Windows 7 notebook and the vertical handover is performed between WiFi and UMTS instead of WiFi and LTE as in this thesis.

Received signal strength (RSS)	RSS is used as the main criterion in the majority of existing horizontal handover algorithms and is also an important criterion for vertical handover algorithms.
Network connection time	The network connection time is the duration that a mobile node is connected to a point of attachment.
Available bandwidth	This parameter is a measure of available data transmission resources. We discuss this parameter more in detail in Section 2.5.1 on page 10.
Power consumption	limited power is a critical issue for mobile devices. Handover decision algorithms should prefer networks that maximize battery life if possible.
Monetary Cost	It should be taken into account that different network have different associated costs. For this thesis in particular LTE networks will often charge per megabyte or have a data cap. WiFi networks on the other hand will often not have these kinds of charges.
Security	Applications have differing requirements for confidentiality or integrity of transmitted data. The security level of a network might be taken into account when making the handover decision.
User preferences	User preference may differ from the decision made by the handover decision algorithm.

Table 2.3: Parameters for handover decisions. [44]

handover delay	This is a measure of the total time of the handover process, from initiation to completion. Handover delay is affected by the complexity of process and is especially important for voice and multimedia which are sensitive to delay or interruptions.
number of handovers	This metric is related to the ping-pong effect. It is usually preferred to reduce the number of handovers because frequent handovers is a waste of network resources. If handover is executed and it becomes necessary to handover back to the original network in a short period of time, then we have a superfluous handover, which should be minimized.
handover failure probability	Handover can fail in two different kinds of ways. It can fail when a mobile node moves out of range of the target network before the handover is completed, and it can fail if the target network has insufficient resources to complete the handover.
throughout	This metric measures the data rate that is delivered to a mobile node on the network. Networks that provide higher throughput are usually preferred when selecting between multiple candidates.

Table 2.4: Vertical handover algorithm performance metrics

RSS based algorithms	This class of algorithms use RSS as the main criterion for taking the handover decision.
Bandwidth based algorithms	This class of algorithms use available bandwidth for a mobile terminal as the main criterion for taking the handover decision.
Cost function based algorithms	Algorithms in this class combine various parameters in a cost function. Then the function is used to calculate a score for the candidate networks which can be compared to decide which network to use.
Combination algorithms	Algorithms in this class use a richer set of inputs than the other classes for making handover decisions.

Table 2.5: Classification of vertical handover decisions. [44]

In [22] Inzerilli et al. proposes a location based handover algorithm with the goal of limiting the ping-pong effect and maximizing goodput. The algorithm they propose is a mobile controlled, soft and vertical between WiFi and UMTS. They then evaluate the performance of their algorithm versus a power based algorithm using simulation. Their results show that the location based algorithm reduces the number of handovers compared to the power based algorithm, thus reducing the ping-pong effect. They also show that there is a reduction in cumulative received bits with the location based algorithm versus the power based one. So the power based algorithm needs a waiting time constraint between handovers to increase stability and reduce number handovers. On the other hand this waiting time constraint should not be applied to the location based algorithm to limit the reduction in cumulative received bits.

Some studies have also looked specifically at vertical handover in android Phones. [28] and [13] are two studies that propose methods for selecting the best wireless interface in a mobile device, and they also made implementations for Android based mobile phones. A system for policy oriented real-time switching of wireless interfaces on mobile devices, MultiNets, has been presented in [28] by Nirjoin et al. The mobile devices in this paper were using WiFi and 3G cellular interfaces, while this thesis covers mobile phones with WiFi and 4G interfaces.

MultiNets has three different modes. Energy saving mode is meant to choose the interface that saves the most energy. Offload mode is for offloading data traffic from the cellular network to WiFi, and performance mode is for selecting the interface with the fastest data connectivity. The choice of interface in performance mode is based on a correlation between signal strength and bandwidth.

In [13] Foremski et al. propose a method for automatically selecting the optimal network interface, in terms of transmission speed and energy usage, for devices with both WiFi and LTE interfaces. They propose a new method for estimating available bandwidth in order to select the best link. Their method for bandwidth estimation consists of the following steps: 1. The client registers at the server and obtains a password to be used for further authentication. 2. The client sends a PING request and receives a 50B response from the server. By default this step is repeated five times with a timeout of one second. 3. The average time between sending the requests and receiving responses is treated as RTT. 4. The client sends a start request and the server replies with data, by default 100 packets of 1KB. 5. The client calculates available bandwidth based on the time it took to receive data.

Both [28] and [13] focus on selecting the best wireless interface. This thesis is different in that it focuses on the handover initiation decision instead of selecting between handover candidates. Another difference is that [13] uses a server component in its bandwidth estimation. The algorithm in this thesis will not use a server component.

## 2.8 summary

In this chapter we discussed background and related works about handover algorithms. This includes the basics of handover algorithms and how they can be classified in different ways. Additionally, we discussed how the focus of this thesis fits into these classifications. The chapter also includes a discussion of the options for handover from WiFi to LTE that exist in Android today. To the best of our knowledge, there does not exist detailed documentation of how these are implemented. This further motivates investigating how the handover mechanism works in Android. Finally, we discussed related studies about vertical handover algorithms. The next chapter elaborates the experimental setup and the experiments performed in order to answer the research questions. We analyse how handover works in Android, and if Android devices get "stuck" on a network with poor connectivity instead of doing a handover to an alternative network.



# Chapter 3

## Methodology

In this thesis we are investigating how well the existing handover mechanism in Android phones work. Does Android phones sometimes remain connected to WiFi networks with no connectivity when LTE is available? In section 1.4 on page 3 we discussed how this thesis follows the design research paradigm made by the ACM task force on the core of computer science. This research paradigms consists of the steps: (1) state requirements, (2) state specifications, (3) design and implement the system, (4) test the system. In this chapter we are designing some experiments, which we first use to investigate handover in Android. The results of these experiments can then be used to inform the design of the new system, and in the end we also repeat the experiments in this chapter in order to test the new system.

This chapter elaborates the setup used when performing experiments and it explains the procedure of how the experiments were performed. The experiments are meant to discover if there are any boundary values where performance lower than this value will trigger handover from WiFi to LTE. For example, will handover be triggered when throughput is below some threshold? We also perform experiments testing how long it takes to initiate handover after performance of the connected WiFi network becomes poor. Automatic handover is not that useful if it takes several minutes from the network becomes unusable until handover is initiated.

### 3.1 Experimental setup

We can see an overview of the setup in Figure 3.1 on the following page. TP-link AC1750 is functioning as a router and WiFi access point. It is connected to a modem which provides Internet access. The mobile phone can then connect to the Internet via WiFi provided by the router and also by the cellular LTE connection.

#### 3.1.1 Hardware

The hardware resembles a typical network setup found in a home or office, and should be representative of the networks that an Android phone would

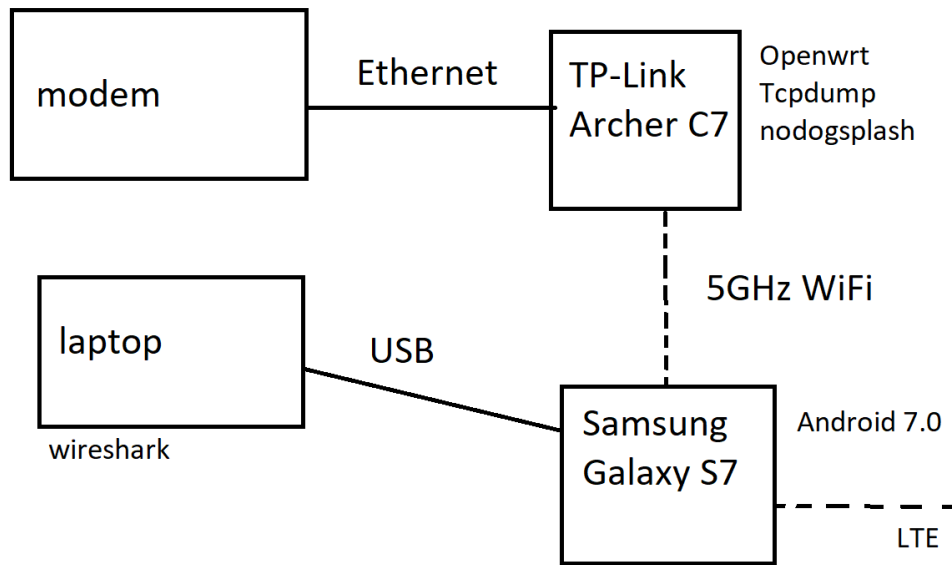


Figure 3.1: Setup

be connected to most of the time in real world. Doing experiments with a setup that is similar to what one could find in a home or office should give some results that are comparable to a real situation. Figure 3.2 on the next page shows what the setup looks like. In the bottom left is the Samsung galaxy S7, and in the top is the Archer C7 router. In the bottom right is a laptop for running scripts, connected to the phone via usb and connected to the router via ethernet.

#### List of hardware

- TP-Link Archer C7 v2 [4]
- Samsung Galaxy S7 model number SM-G930F [34]

For the Android phone we use a Samsung Galaxy S7. According to [35] the galaxy S7 was the best selling smartphone in H1 2016, so this phone is one of the most widely used models and should be a good representation of Android phones in use by people today.

For the router we use a TP-link AC1750[4]. This router is compatible with with the OpenWrt[29] operating system. [18] gives more detailed hardware specifications than those provided by the manufacturer. These can be seen in table 3.1 on the facing page.

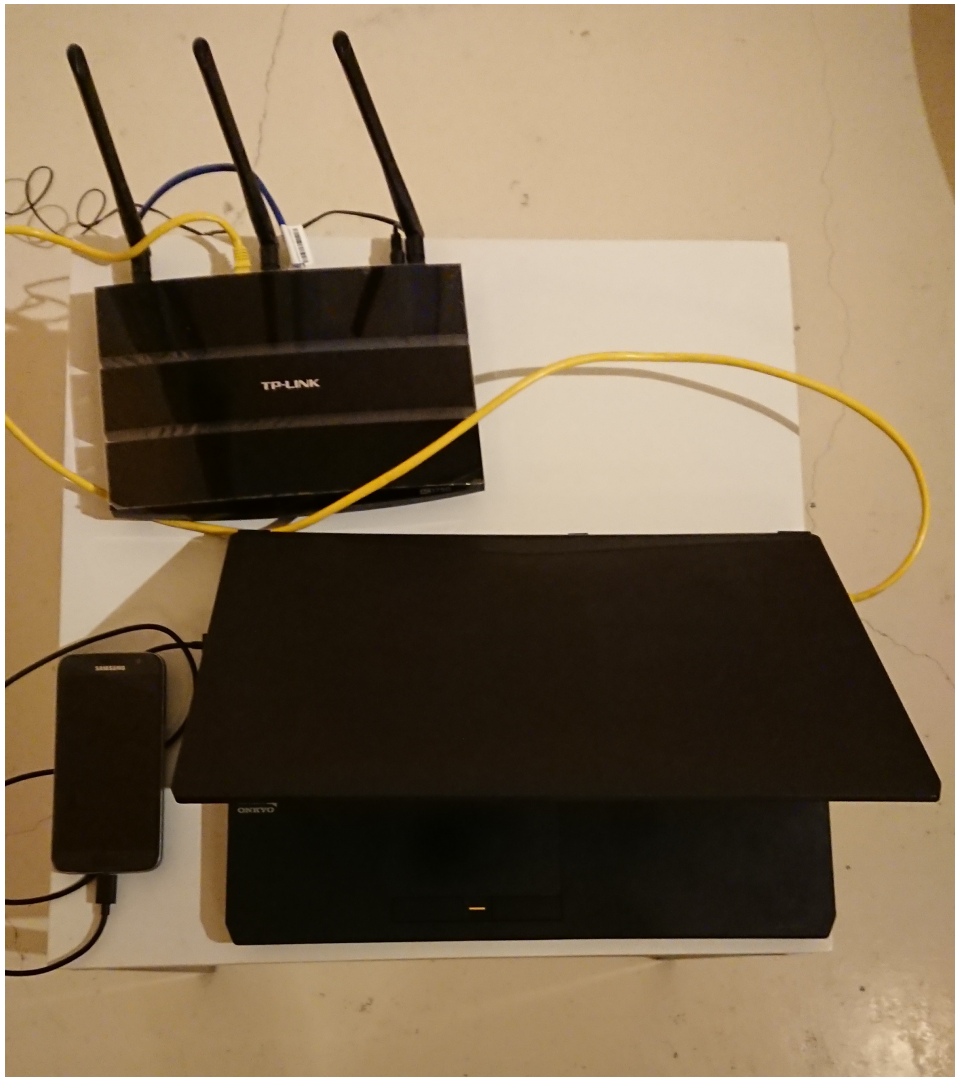


Figure 3.2: hardware setup

CPU	- QCA9558 dual-band - 3-stream 802.11n SoC
Switch	- Atheros AR8327
RAM	- 128 MB Winbond W9751G6JB DDR2 (x2)
Flash	- 16 MB
2.4 GHz radio	- In QCA9558 - Unidentified 2.4 GHz power amp (x3)
5 GHz radio	- QCA9880-BR4A 3x3 802.11ac radio - SiGE 5005L 5 GHz power amp (x3)

Table 3.1: TP-Link Archer C7 V2 specifications

### 3.1.2 choice of operating systems

We use Android 7.0 as the operating system for the phone. Newer versions of Android have been released, however for consistency we have decided to focus on Android 7.0 throughout the work with this thesis. According to [12] the combination of devices running either android 7.0 or 7.1 makes up 28.2 % of all Android devices, making Android 7 the most used major version of Android.

OpenWrt is an open source linux based operating system for emebded devices. It gives us more control over the router than the factory firmware, including the ability to change wireless signal strength and simulating packet loss, which is necessary for experimenting with handover.

The specific version of the operating systems are listed below.

- OpenWrt 18.06.1 r7258-5eb055306f / LuCI openwrt-18.06 branch (git-18.228.31946-f64b152) [21]
- Android 7.0 build number NRD90M.G930FXXS1DQLB

### 3.1.3 Router configuration

The router was connected to a wired network with 120Mbps download and 15 Mbps upload. OpenWrt allows changing the radio transmit power through the configuration web interface. We left transmit power on the default auto setting during all experiments except the ones where we were trying to determine the effect of received signal strength on triggering handover which we will describe in more detail in section 3.2.1 on the next page. Leaving the transmit power on the default setting results in a transmit power of 23dbm(199mW). This way we have a good signal making sure that low signal strength was not causing poor performance. All the experiments were performed using the 5GHz band, specifically channel 36 (5180Mhz), with a channel width of 80Mhz.

### 3.1.4 software tools

- Tcpdump 4.9.2-1
- Wireshark Version 2.6.4 (v2.6.4-0-g29d48ec8)
- Nodogsplash 3.2.1-1

### TCPdump and Wireshark

Throughout the testing it can be useful to be able to inspect the packet flows between the mobile phone and router. To do this we use TCPdump[39] and Wireshark[42]. TCPdump is a command line packet analyser that allows us to capture packets on a network interface. We used tcpdump to capture packtes on the wireless interface of the router. Listing 3.1 on the facing page shows the command to capture packets with TCPdump, where the -v flag gives increased verbosity of the output from TCPdump.

### Listing 3.1: Capturing packets with tcpdump

---

```
$ tcpdump -i INTERFACE -v -w FILENAME
```

---

Wireshark is a network protocol analyser with a powerful GUI for analysing network traffic. After running a test and capturing packets with tcpdump on the router the resulting files were opened with wireshark for further analysis.

### Nodogsplash

Nodogsplash is a captive portal package. Many public WiFi networks require signing in through a splash page in order to access the Internet. Having the ability to set up a captive portal allows us to test if Internet being blocked by the captive portal will cause Android to trigger handover to LTE. We elaborate more on the usage in section nodogsplash in section 3.2.2 on the next page.

## 3.2 WiFi to LTE handover experiments

In this section we describe the procedure for how we tested how handover is currently handled in Android. Based on the results of these tests we can propose propose a simple algorithm for initiating handover, and then use the same procedures to test the proposed algorithm and compare it with how Android currently handles handover.

### 3.2.1 Signal strength

When mobile devices move away from WiFi access points the received signal strength decreases. This has an effect on the effective data rate that the mobile devices are able to send and receive. When the mobile device is far enough away from the access point the performance might have become so low that it would be better to hand over to LTE instead of staying connected. For this reason we are interested in finding out if the phone will hand over before the signal from the access point is completely lost, and what level of signal strength is the threshold for handover.

To test this it is useful to be able to control transmit power of the WiFi router. One way Transmit power can be set is through the web interface of OpenWrt. This is shown in 3.3 on the following page. Unfortunately changing transmit power setting results in disconnecting all the connected stations. Therefore we can not create a script to gradually fade the signal and automate the testing. Instead we must physically move the phone away from the router to decrease the signal.

The command in Listing 3.2 on the next page can be used to obtain a list of client connected to the wlan0 interface and their signal strengths. On


## Wireless Network: Master "OpenWrt" (wlan0)

The *Device Configuration* section covers physical settings of the radio hardware such as channel, transmit power or antenna selection which are shared among all defined wireless networks (if the radio hardware is multi-SSID capable). Per network settings like encryption or operation mode are grouped in the *Interface Configuration*.

### Device Configuration

General Setup   **Advanced Settings**

---

Status  **Mode: Master | SSID: OpenWrt**  
0% **BSSID: A4:2B:B0:A5:AE:74**  
Encryption: None  
Channel: 36 (5.180 GHz)  
Tx-Power: 23 dBm  
Signal: 0 dBm | Noise: -98 dBm  
Bitrate: 0.0 Mbit/s | Country: NO

Wireless network is enabled

Operating frequency

Mode	Channel	Width
AC	36 (5180 MHz)	80 MHz

Transmit Power: auto

dBm

Figure 3.3: Transmit power configuration

Listing 3.2: Getting signal strength example

---

```
$ iwinfo wlan0 assoclist
```

---

the phone we can write the rssi value to the system log when handover is initiated.

By polling this command while the phone disconnects we can get see the last values for signal strength and signal to noise ratio before the phone disconnected.

### 3.2.2 Captive Portal

We used Nodogsplash [16], which is a captive portal package, to setup a captive portal on the WiFi provided by the router. We then used TCPdump to capture packets on the wireless interface in order to learn how captive portals are detected in Android. The results of this is discussed in Section 4.4 on page 42. Nodogsplash comes with a separate application called ndsctl, which can be used to control the running nodogsplash process. When a phone or other device connects to the wireless access point and Nodogsplash is enabled it will get redirected to a splash page where it can be authenticated. Then we can use ndsctl to deauthenticate the phone in order to check how the loss of Internet connection is handled. Listing 3.3 shows the command to deauthenticate a device that already is authenticated.

Listing 3.3: ndsctl example

---

```
$ /usr/bin/ndsctl deauth IP IMAC
```

---

#### Listing 3.4: Throughput limit with tbf

---

```
$ tc qdisc add dev wlan0 root tbf rate LIMIT kbit  
burst 100kbit latency 50ms
```

---

Here we look at the procedure we used to test what happens when the Android phone becomes deauthenticated. When the phone connects to WiFi it is redirected to the splash page of the captive portal for authentication. After authentication is complete and the phone has Internet connectivity we can deauthenticate the phone using the `ndctl` utility like described in section 3.2.2 on the preceding page. After deauthentication Internet connectivity is unavailable and we observe the phone to see if it stays on WiFi.

### 3.2.3 Prerequisites to run to automate experiments

This subsection presents what we need in order to be able to automate the experiments we perform to test Android handover. We decided to automate as much of the testing as possible in order make the testing procedure more consistent and to enable us to do a greater number of repetitions.

We use OpenWrt on the router to control performance with regard to metrics like throughput, delay and packet loss. Therefore, to be able to automate the experiments we need a way to send commands to the router from the computer where the script is running. We also need to send commands to the phone. After the phone has handed over to LTE we need to send commands to reconnect to WiFi in order to do more repetitions. Finally we also need a way to automatically detect that the phone has initiated handover.

The laptop was connected to the router with ethernet and we used we used `ssh` to send commands to the router. Then we used the traffic control capabilities of linux to control performance of the WiFi network.

### 3.2.4 Traffic control in linux

Each network device has an attached queing discipline (qdisc). Qdiscs are algorithms that control the packet queue on the device [26]. By using different qdiscs we can shape the data that is transmitted from the router. The default qdisc for devices is called `pfifo_fast` and the underlying basis for this qdisc is the first in first out algorithm.

`Tc` [38] is a utility for configuring the linux kernel packet scheduler. `Tc` allows us to change which qdiscs are used by the network devices. We can use this utility to set qdiscs that control throughput limits and emulates packet loss and delay.

For limiting data rate we use the token bucket filter (tbf) qdisc. Listing 3.4 shows the command to set a limit to throughput using `tc` and `tbf`. `Netem`

Listing 3.5: Setting delay with netem

---

```
$ tc qdisc add dev INTERFACE root netem delay TIME
```

---

Listing 3.6: Setting loss with netem

---

```
$ tc qdisc add dev wlan0 root netem loss PERCENTAGE
```

---

[27] is another queuing discipline for the linux packet scheduler. It provides the ability to emulate various network properties like delay and packet loss. Listing 3.5 shows an example command for setting delay with netem, and Listing 3.6 shows an example of setting loss.

### 3.2.5 Android automation with ADB

We used the command-line tool Android Debug Bridge (adb) [3] to communicate with the phone. For the automation we need to receive a notification when a handover occurs or state of WiFi changes somehow. We also need to be able to control WiFi on the phone.

Listing 3.7 shows the command to enable or disable WiFi through adb. The problem with this command is that it requires root access to work. For this reason we use a workaround to be able to control WiFi. The way we got around this issue is by using the broadcast mechanism in Android. Broadcasts allows for sending and receiving of messages between apps and the system. We create a dummy app with `android.permission.CHANGE_WIFI_STATE` and `android.permission.ACCESS_WIFI_STATE` permissions, which means the app can change WiFi state. Then we add a broadcast receiver class which will enable or disable WiFi. Listing 3.9 on the facing page displays a broadcast receiver that will enable or disable WiFi depending on whether the extra field of the broadcast is true or false. Now we can enable or disable WiFi through adb using the command in Listing 3.8 on the next page.

The Android API provides the ability to get notified about network changes through the `NetworkCallback` class [10], which provides several callback functions. Applications wanting notifications about network changes can extend this class and then register a `NetworkCallback`.

Listing 3.11 on page 26 shows example code that extends the `NetworkCallback` class by overriding the `onCapabilitiesChanged` and `onLost` meth-

Listing 3.7: WiFi control through adb

---

```
$ adb shell svc wifi enable|disable
```

---



Listing 3.8: Broadcast with adb

---

```
$ adb shell am broadcast -a WifiChange -e wifi true
```

---

Listing 3.9: Android broadcast receiver

---

```
public class ControlWifi extends BroadcastReceiver {
    public void onReceive(Context c, Intent intent) {
        WifiManager wfm = (WifiManager) c.
            getSystemService(Context.WIFI_SERVICE);
        wfm.setWifiEnabled(Boolean.parseBoolean(intent
            .getStringExtra("wifi")));
    }
}
```

---

ods. In the `onCapabilitiesChanged` method we can see an if-statement that checks if the network whose capabilities changes has the `TRANSPORT_WIFI` transport type, if it has `NET_CAPABILITY_INTERNET`, and that it does not have `NET_CAPABILITY_VALIDATED`. This check ensures that the network whose capabilities changes is a wifi network instead of a cellular network, and when a network that has both `NET_CAPABILITY_INTERNET` and `NET_CAPABILITY_VALIDATED`, then that means that Android has successfully detected internet connectivity. Since we want to detect when Android decides that the network no longer has Internet connectivity we instead check that the network does not have `NET_CAPABILITY_VALIDATED`. We have seen that when the user interface on the Android phone displays the message "Ready to connect when network quality improves.", then we also get this notification. However, when the phone first connects to WiFi it will try to detect Internet connectivity, and we could get this notification before Internet connectivity is detected. Because of this we must make sure that the phone is connected to WiFi and has successfully validated that Internet is available before we can monitor for handover. Otherwise we could get false positives. The line `Log.i(TAG, "Handover network " + network);` prints to the Android system log. Now we can read the log in an automated script with the command in listing 3.10 in order to automatically check if handover has been triggered.

Listing 3.10: read system log with adb

---

```
$ adb logcat -d
```

---

Listing 3.11: Detecting WiFi handover in Android

---

```
ConnectivityManager connectivityManager = (
    ConnectivityManager) getSystemService(Context.
    CONNECTIVITY_SERVICE);
NetworkRequest.Builder builder = new NetworkRequest.
    Builder();

connectivityManager.registerNetworkCallback(
    builder.build(),
    new ConnectivityManager.NetworkCallback() {
        public void onLost(Network network) {
            Log.i(TAG, "network_" + network + "_
                lost.");
        }

        public void onCapabilitiesChanged(Network
            network, NetworkCapabilities
            networkCapabilities) {
            if (networkCapabilities.hasTransport(
                NetworkCapabilities.TRANSPORT_WIFI)
                &&
                networkCapabilities.
                    hasCapability(
                        NetworkCapabilities.
                            NET_CAPABILITY_INTERNET) &&
                !networkCapabilities.
                    hasCapability(
                        NetworkCapabilities.
                            NET_CAPABILITY_VALIDATED))
            {
                Log.i(TAG, "Handover_network_" +
                    network);
            }
        }
    });
```

---

### 3.2.6 Testing performance thresholds for triggering handover

Figure 3.4 on the following page shows the algorithm in the testing script. The first step is to initialize the upper and lower limit. These could be percentage of dropped packets or data rates. Then WiFi is enabled on the phone so that it connects to the WiFi network. Next we apply some sort of traffic shaping on the router like explained in section 3.2.4 on page 23. Then we wait for a set period of time while monitoring for handover in the phone. Next upper or lower limit is updated based on whether handover was triggered or not. Then we update the limit variable to be the average of upper and lower limit. Then the process is repeated until upper and lower limit close in on each other and we find the threshold value that will trigger handover.

### 3.2.7 Testing how long it takes before handover is initiated.

We are also interested in how long it takes from a WiFi network becomes unusable until handover is initiated. Figure 3.5 on page 29 shows the basic steps in this procedure. First we connect the phone to the WiFi, then we limit performance somehow, for instance by inducing packet loss or setting a limit on throughput. Then we wait until the phone initiates handover or a timeout in case the phone never initiates handover, and finally we compare timestamps of when we applied the command to limit performance and when handover was initiated. For the timeout we used ten minutes. Ideally handover should be initiated within seconds or even milliseconds, and if it takes several minutes then it is reasonable to assume that if the user would end up disconnecting manually instead.

## 3.3 Battery consumption

Running code on the CPU and using the WiFi radio consumes energy. We use the battery historian [15] tool to analyze overhead in terms of power usage of the proposed new handover algorithm. Getting energy usage with battery historian consist of the following steps:

1. Connect the mobile device to your computer.
2. Shut down the running adb server.  
`$ adb kill -server`
3. Restart adb and check for connected devices  
`$ adb devices`
4. Reset battery data gathering  
`$ adb shell dumpsys batterystats --reset`
5. Disconnect the device and run the app that is to be tested.
6. Reconnect the mobile device to the computer

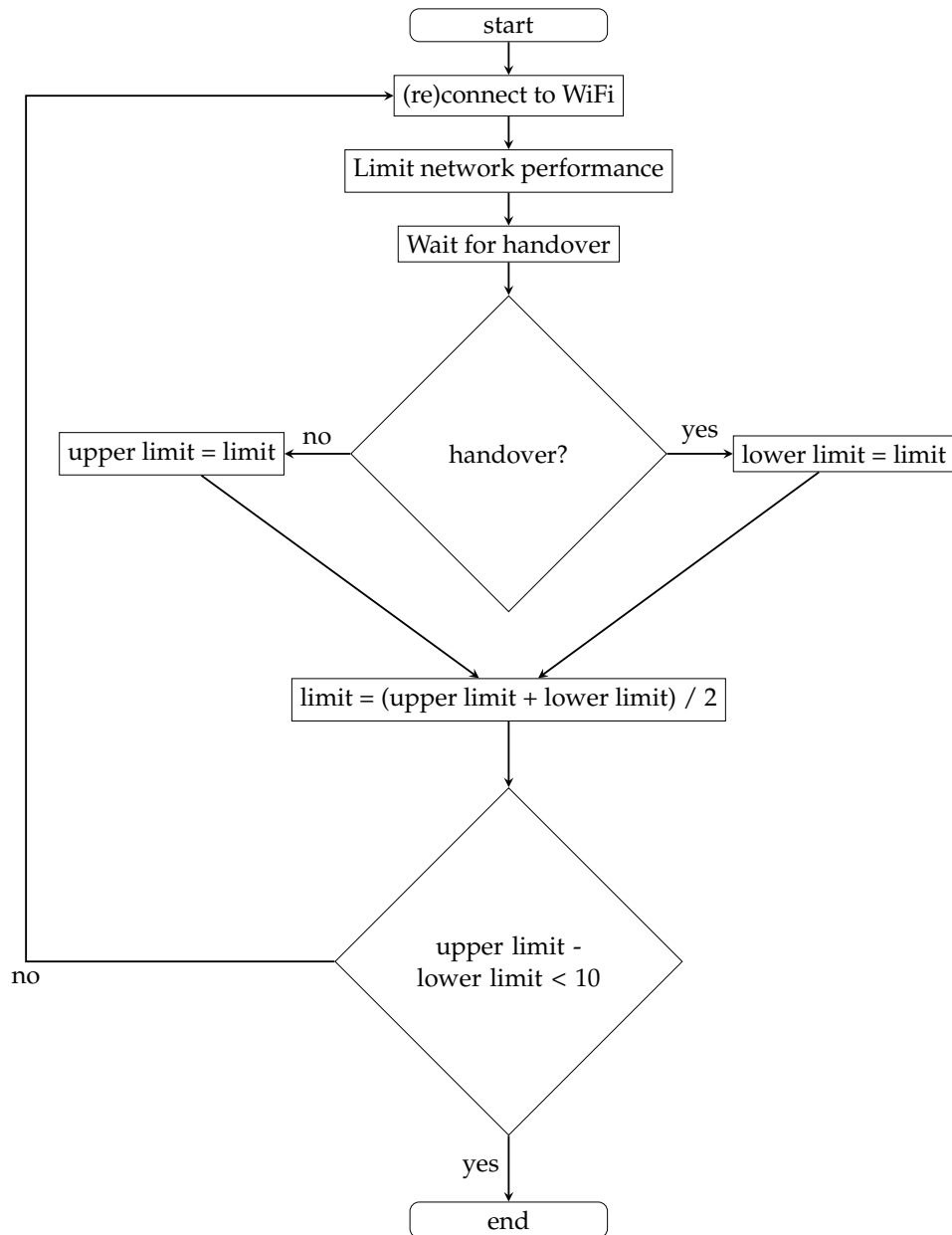


Figure 3.4: This figure shows the steps we used to find out if handover is triggered at a boundary value for various network performance metrics. First the phone connects to WiFi, then we apply a command on the router to create packet loss, delay or other network conditions. We keep three variables, limit, upper\_limit and lower\_limit. Limit is used as a parameter to determine for example how many kbps of throughput to allow. Then based on if handover is detected or not, we update the variable\_upper limit or lower\_limit. Then we update limit and check if upper\_limit and lower\_limit have converged.

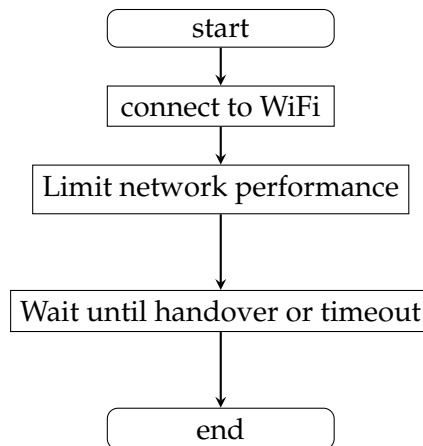


Figure 3.5: Procedure for measuring time until handover initiation.

7. Make sure the device is recognized  
\$ adb devices
8. Dump all battery data  
\$ adb shell dumpsys batterystats [path/]batterystats.txt
9. Create a bugreport from raw data.  
\$ adb bugreport > [path/]bugreport.zip

Now we can open bugreport.zip in the battery historian tool and inspect device power usage of the running apps.

### 3.4 Summary

In this chapter we have covered the setup used in the experiments in this thesis. This includes the devices used and the software packages installed. In addition we covered the procedures used in the various tests, which includes using logging and callback functions to read changes in the WiFi state in the Android phone and using adb to control the phone and ssh to control the openwrt router from scrips running on computer. Finally, we covered how to generate an Android debug log which contains detailed information about power usage. In the next chapter we will look at the results of running the experiments described in this chapter on the existing handover mechanisms in Android.



## Chapter 4

# Handover from WiFi to LTE in Android

In this chapter we present the results of experimenting with handover in Android. We have looked at the threshold to trigger handover with regard to delay packet loss and limited throughput. We cover how long it takes to reach the handover decision, and we analyse a TCPdump of packets between the router and the phone to see if we can find some indications of how the handover decision is made. During the testing we made sure that the phone was not downloading updates or otherwise using the WiFi connection. Later we repeated some of the tests while streaming a video to the phone to generate some network traffic. According to Cisco, video traffic accounted for 59% of mobile data usage in 2017 [7]. Therefore, testing the scenarios that could trigger handover while the phone is receiving video traffic might be more representative of real world use than just testing the scenarios while the phone is receiving on additional traffic. Finally we looked at a scenario where Internet access is blocked by a captive portal. Table 4.1 on the next page shows an overview of all the experiment scenarios.

### 4.1 Handover threshold results

This section presents the results of the experiments we performed to find the threshold for various metrics that would trigger handover in Android. The packet loss, delay and throughput tests were performed with the procedure shown in Figure 3.4 on page 28. A script connects the phone to WiFi, then it sends a command to the router to for example add delay to the packets, then it waits while checking if the phone stays connected. Then the performance value of the metric being tested is updated based on whether handover was detected or not, and this is repeated until we find the threshold value where handover is triggered.

We recall that we saw in Section 2.4 on page 8 that Android includes some settings which, when enabled, are supposed to make the phone automatically handover from WiFi to LTE when the Internet connection on the WiFi network becomes unstable. The experiments were performed

<b>Boundary for triggering handover.</b>
gradually adjusting throughput
gradually adjusting delay
gradually adjusting packet loss
physically moving phone away from router to adjust signal strength
<b>How long does it take to trigger handover?</b>
Time from limiting throughput until handover is initiated
Time from adding delay until handover is initiated
Time from adding packet loss until handover is initiated
<b>Captive portal</b>
Internet is no longer reachable because of deauthentication from captive portal.

Table 4.1: Overview of experiments performed when testing the existing handover in a Android.

with the handover setting in Android called "smart network switch" turned off, with the setting set to normal and with the setting set to aggressive. As expected we do not get handover when "smart network switch" is turned off.

In the remainder of this section we look at the results of how the existing handover mechanism in Android behaves when we simulate various network conditions with the router.

#### 4.1.1 Handover caused by delay

Table 4.2 on the facing page shows the results of the experiments where we provoked handover by adding delay to the network. The leftmost column contains the settings we used on the mobile phone, the middle column shows the mean of 30 repetitions and the rightmost column shows standard deviation. The mean value for all the repetitions with smart network switch to normal is 970ms, and the mean for the aggressive setting is 377ms. Standard deviation is 213,3ms, 160ms and 67,7ms for smart network switch set to normal, smart network switch set to aggressive and smart network switch set to aggressive while WiFi is used to download a video, respectively. There is some variation in the results, but even with variation we can see a trend.

Unfortunately, after having performed the experiments, we found that when we set delay with netem we sometimes get "spikes" in the delay. We tested adding delay between the router and the laptop which is connected with an ethernet cable and then the delay added by netem was completely stable. But when we are connected with WiFi and add delay with netem rtt reported by ping is unstable and sometimes more than 100ms higher than the delay we specified in the command line. The values in Table 4.2 on the next page are calculated using the values used as parameter to the netem command. Because of the results are likely skewed so that the actual delay was higher than the one we specified in the netem command.



<b>setup</b>	<b>mean</b>	<b>standard deviation</b>
Smart network switch normal	970ms	213,3ms
Smart network switch aggressive	377.3ms	160ms
Smart network switch aggressive and streaming video	580ms	67,7ms

Table 4.2: Delay boundary values for Android phone

Even with the high variation indicated by the standard deviation and skewed results we can still see a trend. It is clear that enabling the handover setting actually causes handover when the delay becomes higher, and that the threshold for taking the handover decision is lower with the aggressive setting. When we had smart network switch set to aggressive and a video steaming to the phone the mean threshold for added delay needed to trigger handover actually increased to 580ms. One possible reason for this might be that the algorithm for smart network switch keeps track of how many bytes are sent and received by the phone and that because the video stream is generating traffic that indicates that there is some connectivity despite the high delay.

[14] recommends that one-way end-to-end delay should be kept below 150ms for high quality voice communication, but it also notes that delays between 150ms to 400ms are still acceptable. The mean value for the aggressive setting does fall within this 400ms acceptable value, but it might also be advantageous to have an even more aggressive option that to assure meeting the 150ms recommendation.

#### 4.1.2 Handover caused by Packet loss

Table 4.3 on the following page shows the results of the experiments where we provoked handover by adding packet loss to the network. The leftmost column contains the settings we used on the mobile phone, the middle column shows the mean of 30 repetitions and the rightmost column shows standard deviation. With the normal handover setting, 38,7% is the mean value and 10,% is standard deviation, for the threshold where handover was triggered. With the aggressive setting the threshold is 8,2% with a standard deviation of 3,2%. Again the threshold for triggering handover is clearly lower with the aggressive setting. With smart network switch set to aggressive and a video stream generating network traffic the mean threshold was 15,8% with a standard deviation of 5,2%. Like in the results with delay the phone stays connected up on higher levels of packet loss when the phone is actively receiving data.

According to Cisco [33], the G.729 codec for audio data compression often used in VoIP requires far less than 1 percent packet loss to avoid audible errors. On the other hand WiFi can also be used for many other

<b>setup</b>	<b>mean</b>	<b>standard deviation</b>
Smart network switch normal	38,7%	10,4%
Smart network switch aggressive	8,2%	3,2%
Smart network switch aggressive and streaming video	15,8%	5,2%

Table 4.3: packet loss values for Android phone

purposes than VoIP, and reliable protocols like TCP ensures retransmits of lost packets, so instating handover in networks with a mere one percent packet loss is probably not desirable.

### 4.1.3 Handover caused by low throughput

Table 4.4 on the next page shows the results of the experiments where we set limits on throughput to provoke handover. The leftmost column contains the settings we used on the mobile phone. We remember from section 2.4 on page 8 in the background chapter, that Android includes a setting called smart network switch, which can be set to off, aggressive or normal, and is supposed to automatically switch to using LTE when the Internet connection on WiFi becomes unstable. The middle column shows the mean of 30 repetitions and the rightmost column shows standard deviation.

The mean throughput value were handover was triggered with smart network switch set to normal was only 11,3 kbps, and with the aggressive setting it was 198kbps. streaming a video to the phone while throughput is limited gives a mean threshold of 173kbps. For these the values for standard deviation are 9,7kbps, 40,2kbps and 53,1kbps respectively. Unlike the experiments with packet loss and delay the test with streaming video does not give very different results from the tests without streaming video. Because throughput is limited here streaming video results in buffering and lower values for total bytes received by the phone is not a good indication of good performance of the WiFi network. This can be a possible explanation for why the result look the way they do.

The threshold for the normal setting is so low that it is not useful in many situations. For example, trying to load most webpages will result in a timeout if we only have 11kbps available. 200kbps is also quite low, however it is possible to load webpages with this value. The requirements for streaming videos on YouTube is 500kbps and the requirements for steaming live movies, TV shows and live broadcasts is 1+ Mbps[37]. So for the purposes of video streaming an even more aggressive handover algorithm could be beneficial.

setup	mean	standard deviation
Smart network switch normal	11,3kbps	9,7kbps
Smart network switch aggressive	198kbps	40,2kbps
Smart network switch aggressive and streaming video	173kbps	53,1kbps

Table 4.4: throughput boundary values for Android phone

#### 4.1.4 Handover caused by low signal strength

Table 4.5 on the next page Shows the results of triggering handover by moving away from the router so that signal strength becomes weaker. All experiments presented in this table were repeated 30 times. Setting the smart network switch setting to normal or aggressive leads to a soft handover. The phone is still connected to WiFi after it has displayed the message "ready to connect when network quality improves" and has started using LTE for Internet connectivity. For these settings the table includes the RSSI values reported by the phone when doing the handover. With the Aggressive WiFi to mobile handover setting from the Android developer options this message never appears and the phone disconnects from the access point when switching over to the LTE connection. In this case we have a hard handover. Because the phone is no longer connected to the WiFi access point we can not read the rssi value programatically in the callback function that gets called on network changes, so here the table does not include these values.

The signal strength in dbm reported by the router just before handover is the highest at -76 when using the smart network switch aggressive setting. For smart network switch normal an aggressive wifi to mobile handover the results are -83 and - 86.2. From this it appears that smart switch aggressive actually is a bit more aggressive than smart switch normal, but on the other hand the RSSI values, which is what is actually being read by the phone, are basically the same at -84,8 and -85,33. The setting do not appear to make much difference with regard to handover caused by signal strength. During these experiments the phone was never stuck on the WiFi network when signal strength was low. Unlike the scenario with limited throughput the phone always successfully handed over to LTE.

## 4.2 How long does it take to make the handover decision?

Table 4.6 on page 37 shows how long it took from simulating loss, delay or low throughput until handover was initiated. During these tests,

setup	RSSI read by phone		DBM read by router	
	mean	standard deviation	mean	standard deviation
Smart network switch normal	-84,8	3,4	-83	3,85
Smart network switch aggressive	-85,33	3,1	-76	4,4
Aggressive WiFi to mobile handover			-86,2	1,3

Table 4.5: Signal strength boundary values

the "aggressive WiFi to mobile handover" in the developer settings of Android was turned off and "smart network switch" in the WiFi section of settings was set to aggressive. In the different scenarios we set the values for throughput, delay and packet loss such that network performance is well below the thresholds we found in 4.1 on page 31. The tests were repeated 30 times and in each repetition the script monitored the phone for handover up to ten minutes from applying the command to reduce network performance. The table presents the mean time and standard deviation.

For the experiments with limited throughput handover was not initiated after 10 minutes in 4 of the 30 repetitions, and the maximum time it took was 120 seconds. For the tests with increased delay 2 of the repetitions failed to register a handover, and the maximum time it took for handover to occur is 183 seconds. In the scenario with 50% packet loss the phone initiated handover for each repetition and 122 seconds is the maximum time it took before making the handover decision. The calculation of mean and standard deviation in table 4.6 on the facing page was done using the values from the repetitions where we registered a handover.

The mean time to detect poor network performance and start handover was 75 seconds for the setup with limited throughput and 95,9 seconds for the setup with added delay, however it was only 35.1 seconds for the setup with packet loss. Additionally, the setup with packet loss only had 23.6 seconds standard deviation versus 28,5 seconds and 71.7 seconds for limited throughput and added delay. Here there is also quite a lot of variation.

Later we repeated the experiments while streaming a video to the phone. Apart from using a video to generate network traffic the setup and procedure was identical to the previous timing testing. The results of this is shown in table 4.7 on page 38. In the table we can see that the mean time for

setup	mean time before handover	standard deviation
Throughput limited to 50kbps	75s	28,5s
700ms delay	95.9s	71.7s
50% packet loss	35.1s	23.6s

Table 4.6: Time taken before making the handover decision.

handover was 12.5s for the test with limited throughput, 145.5s for the tests with 700ms added delay and 52.8 seconds for the tests with 50% packet loss. Compared to the tests without streaming a video to generate traffic handover was much faster for the test with limited throughput. On the other hand it was somewhat slower, though not by a meaningful amount, in the other tests. The result with limited throughput seems to suggest that bytes received by the phone is used as part of the smart network switch handover mechanism.

In most of the scenarios the phone takes more than a minute to handover, and there is quite a high variation in how long it takes from the network performance becomes poor and until handover is initiated. The aggressive handover setting does help with switching from low performance WiFi to LTE, however it is quite inconsistent. Even though the WiFi is practically useless it sometimes fails completely or takes several minutes to initiate handover. In situations where the a user actively is using the phone's Internet connection, for example streaming a video or using voice over IP, it most likely becomes necessary for the user to disconnect from WiFi manually in order to restore Internet connectivity.

One possible reason for taking a long time to initiate handover is to avoid handover in cases where an issue arises in the network but resolves it self after a few seconds. In such cases it might not be desirable to hand over to LTE in order to avoid the higher cost which is often associated with LTE.

In the background chapter one of the vertical handover decision criteria we presented is network connection time. According to Yan et al. [44], it is important to keep track of network connection time in order to choose the right moment to trigger handover. In the survey they say that initiating handover from WLAN to a cellular network too early would waste network resources, and that being too late would result in a handover failure. Therefore, another possible reason for the slow handover initiation would be avoiding waste of network resources.

### 4.3 TCPdump analysis

In this section we present the results of looking at the packets sent between the phone and router to find out if the phone is sending any packets to aid in the decision to initiate handover.

setup	mean time before handover	standard deviation
Throughput limited to 50kbps	12.5s	6.5s
700ms delay	145.5	97.9
50% packet loss	52.8s	46.25s

Table 4.7: Time taken before making the handover decision while streaming video.

Figure 4.2 on page 40 and 4.1 on the next page shows the frequency of ping packets sent from the Android phone to the router with time in seconds on the X axis and packets per second on the Y axis. These were generated by capturing packets from the phone using TCPdump and filtering the packets using Wireshark. Figure 4.2 on page 40 shows the frequency of sending ping packets when the smart network switch setting was set to aggressive. In the figure we can see that nine ping packets were sent at first and then it sends pings in groups of four packets with about 10 second intervals. After two minutes groups of only two packets are sent instead of four and the interval stays about the same. It is possible that the reason for sending more packets in the beginning is part of validating the connection upon connecting to a WiFi network, and then the number of packets is reduced to reduce overhead.

Figure 4.1 on the next page shows the frequency of sending ping packets when smart network switch was disabled. In this figure ping packets are sent in the phone first sends four ping packets and then after about 30 seconds it sends four packets again, and after this four packets are sent with 60 second intervals.

By comparing the graphs we can see that the frequency of sending ping packets is higher when we set smart network switch to aggressive vs when smart network switch is disabled, with roughly 10 second intervals between sending pings with smart network switch in comparison to roughly one minute intervals with smart network switch disabled.

Figure 4.3 on page 41 shows part of a TCPdump where the Android phone was connected to the OpenWrt router with the "smart network switch" setting set to aggressive. In this figure the first column shows the time of day instead of seconds since last displayed packet, and it also includes a column with info about the packets. It shows that the Android phone is sending ping requests to the wireless router and getting reply packets back. When the time was 13:42:00 we applied a 50% packet loss on the router. By looking at the timestamps in the figure we can see that not all the ping requests are getting replies any more. At 13:42:44 the phone handed over to LTE. After this the phone was still connected to the WiFi router but it stopped sending pings. This suggests that the pinging is used as part of the decision to hand over from WiFi.

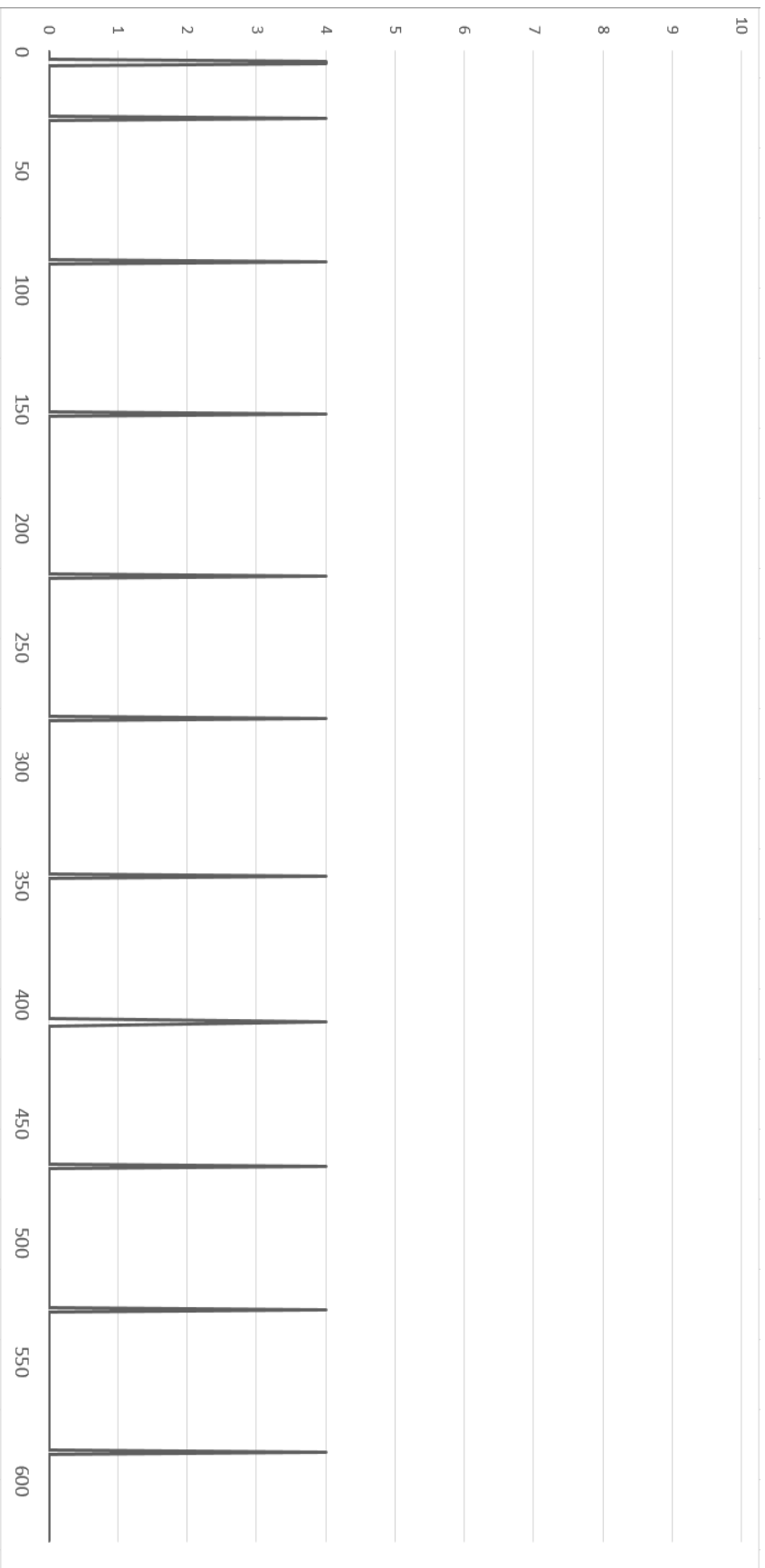


Figure 4.1: Frequency of ping packets sent from the mobile phone when Smart network switch disabled. The X axis is time in seconds and Y axis is packets per second.

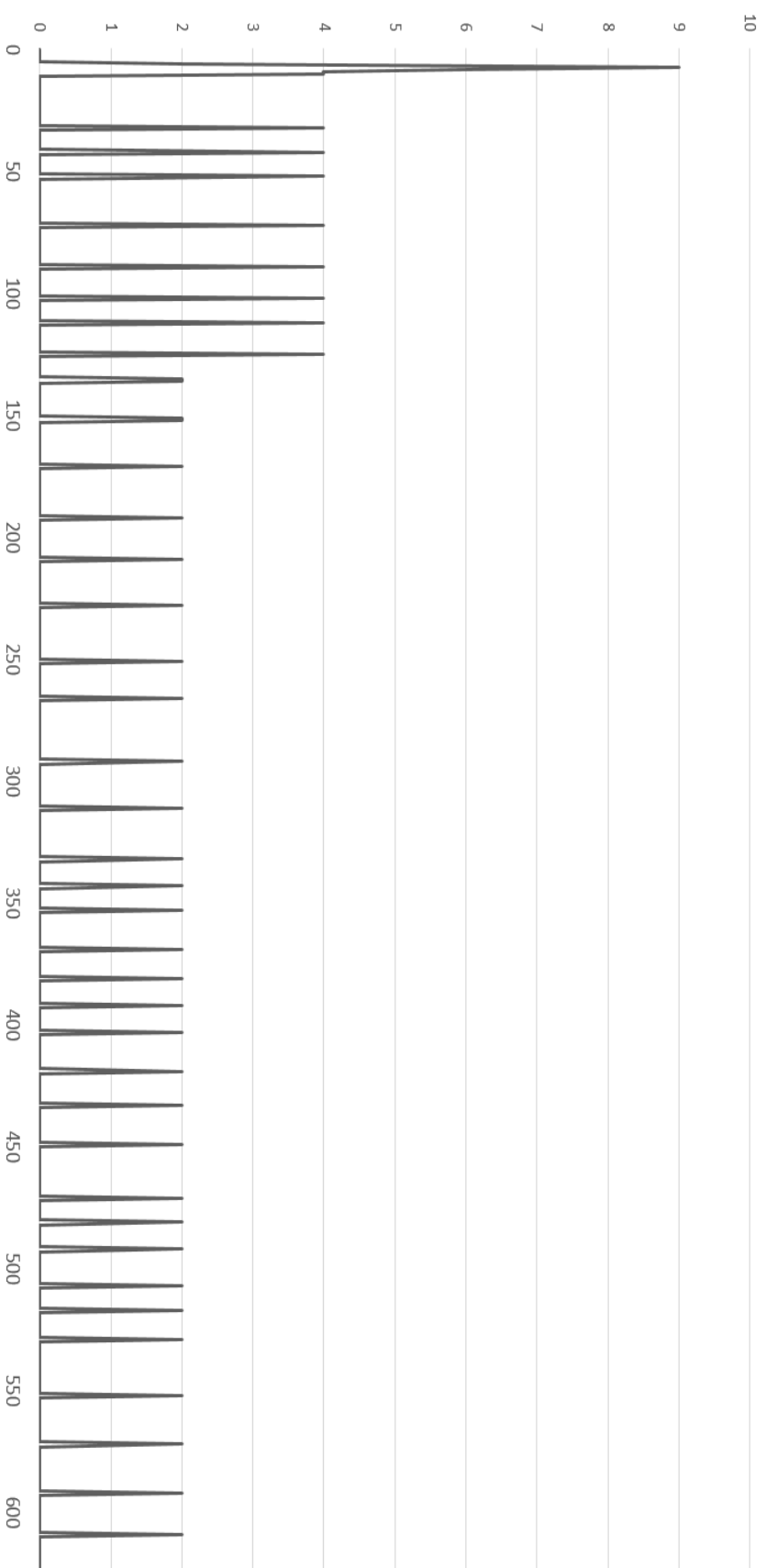


Figure 4.2: Frequency of ping packets sent from the mobile phone when Smart network switch is set to aggressive. The X axis is time in seconds and Y axis is packets per second.



2019-03-11 13:41:14...	192.168.1.1	192.168.1.140	Echo (ping) reply
2019-03-11 13:41:25...	192.168.1.140	192.168.1.1	Echo (ping) request
2019-03-11 13:41:25...	192.168.1.1	192.168.1.140	Echo (ping) reply
2019-03-11 13:41:25...	192.168.1.140	192.168.1.1	Echo (ping) request
2019-03-11 13:41:25...	192.168.1.1	192.168.1.140	Echo (ping) reply
2019-03-11 13:41:37...	192.168.1.140	192.168.1.1	Echo (ping) request
2019-03-11 13:41:37...	192.168.1.1	192.168.1.140	Echo (ping) reply
2019-03-11 13:41:37...	192.168.1.140	192.168.1.1	Echo (ping) request
2019-03-11 13:41:37...	192.168.1.1	192.168.1.140	Echo (ping) reply
2019-03-11 13:41:43...	192.168.1.140	192.168.1.1	Echo (ping) request
2019-03-11 13:41:43...	192.168.1.1	192.168.1.140	Echo (ping) reply
2019-03-11 13:41:43...	192.168.1.140	192.168.1.1	Echo (ping) request
2019-03-11 13:41:43...	192.168.1.1	192.168.1.140	Echo (ping) reply
2019-03-11 13:41:44...	192.168.1.140	192.168.1.1	Echo (ping) request
2019-03-11 13:41:44...	192.168.1.1	192.168.1.140	Echo (ping) reply
2019-03-11 13:41:44...	192.168.1.140	192.168.1.1	Echo (ping) request
2019-03-11 13:41:44...	192.168.1.1	192.168.1.140	Echo (ping) reply
2019-03-11 13:41:44...	192.168.1.140	192.168.1.1	Echo (ping) request
2019-03-11 13:41:44...	192.168.1.1	192.168.1.140	Echo (ping) reply
2019-03-11 13:41:45...	192.168.1.140	192.168.1.1	Echo (ping) request
2019-03-11 13:41:45...	192.168.1.1	192.168.1.140	Echo (ping) reply
2019-03-11 13:42:02...	192.168.1.140	192.168.1.1	Echo (ping) request
2019-03-11 13:42:03...	192.168.1.140	192.168.1.1	Echo (ping) request
2019-03-11 13:42:22...	192.168.1.140	192.168.1.1	Echo (ping) request
2019-03-11 13:42:22...	192.168.1.1	192.168.1.140	Echo (ping) reply
2019-03-11 13:42:22...	192.168.1.140	192.168.1.1	Echo (ping) request
2019-03-11 13:42:32...	192.168.1.140	192.168.1.1	Echo (ping) request
2019-03-11 13:42:32...	192.168.1.1	192.168.1.140	Echo (ping) reply
2019-03-11 13:42:33...	192.168.1.140	192.168.1.1	Echo (ping) request
2019-03-11 13:42:35...	192.168.1.140	192.168.1.1	Echo (ping) request
2019-03-11 13:42:35...	192.168.1.140	192.168.1.1	Echo (ping) request
2019-03-11 13:42:36...	192.168.1.140	192.168.1.1	Echo (ping) request
2019-03-11 13:42:36...	192.168.1.140	192.168.1.1	Echo (ping) request
2019-03-11 13:42:42...	192.168.1.140	192.168.1.1	Echo (ping) request
2019-03-11 13:42:42...	192.168.1.140	192.168.1.1	Echo (ping) request
2019-03-11 13:42:42...	192.168.1.140	192.168.1.1	Echo (ping) request
2019-03-11 13:42:43...	192.168.1.140	192.168.1.1	Echo (ping) request
2019-03-11 13:42:44...	192.168.1.140	192.168.1.1	Echo (ping) request

Figure 4.3: Ping packets between phone and router captured by TCPdump with packet loss created by the router.

92	8.176590	192.168.1.140	216.58.211.99	HTTP	398 GET /generate_204 HTTP/1.1
94	8.210828	216.58.211.99	192.168.1.140	HTTP	168 HTTP/1.1 204 No Content
112	8.656835	192.168.1.140	216.58.211.99	HTTP	398 GET /generate_204 HTTP/1.1
114	8.692247	216.58.211.99	192.168.1.140	HTTP	168 HTTP/1.1 204 No Content

Figure 4.4: captive portal check in Android

## 4.4 Captive portal

### Captive portal detection in Android

When an Android device connects to a WiFi network it will make a DNS request to get the address of `connectivitycheck.gstatic.com`. Then the Android device makes an HTTP get request to `connectivitycheck.gstatic.com`. If 204 No Content is received as a response then there is no captive portal on the device and it can connect normally. This can be seen in Figure 4.4. Otherwise, if a 307 temporary redirect is received then there is a captive portal on the WiFi network.

#### 4.4.1 Deauthentication from captive portal

Table 4.8 on the next page shows an overview of the observations when the Android phone loses Internet connectivity because it becomes deauthenticated by the captive portal. The android phone detects that Internet is no longer available and starts using LTE instead not long after Internet was blocked in the cases where "smart network switch" is disabled. When "smart network switch is enabled" the phone does not seem to detect that Internet is unavailable, and no handover takes place. Strangely the phone only does handover when smart network switch is disabled, which is the opposite of what one might expect to happen. In the previous section we saw that the phone is pinging the router as a way to check if the connection is still good, and since the phone stays connected to the router these pings will still get replies even though the phone can not reach the Internet anymore.

#### 4.4.2 Summary of handover in Android

In this chapter we have seen that the handover settings actually do switch from WiFi to LTE automatically when WiFi becomes unstable, and that the aggressive variant of the smart network switch is significantly more aggressive than the normal variant. We have however also seen that as we suspected the phone can sometimes get stuck on WiFi network with poor performance without initiating handover. In addition to that we have found that it sometimes takes a long time before handover is initiated and that the time to initiate handover is quite inconsistent.

We have seen that handover will be initiated before we are completely out of range of the access point, which suggests that signal strength is used as part of the decision to initiate handover. Based on analysis of the traffic captured by `tcpdump` it also appears that ping to the router is used as part

<b>Setup</b>	<b>observation</b>
Smart network switch off and aggressive WiFi to mobile handover off	A few seconds after deauthentication “Internet may not be available” is displayed in the WiFi settings menu, and the phone is still connected to the WiFi access point. The phone is now using LTE when opening a webpage etc
Smart network switch normal	“connected” is still displayed 2 min after deauthentication, and the phone is still connected to the WiFi access point. Loading web pages or using the Internet is not possible before manually disconnecting from WiFi.
Smart network switch aggressive	Same behavior as Smart network switch normal
Aggressive WiFi to mobile handover	Same behavior as smart network switch off and aggressive WiFi to mobile handover off

Table 4.8: Internet blocked by captive portal

of the decision to initiate handover. Based on the results of sending data in the form of streaming a video while limiting available throughput, the amount of bytes received by the phone is also likely used as part of the handover decision.

Requirements for throughput and other metrics depend on the use case. In our opinion the aggressive handover settings are not aggressive enough for many typical use cases of mobile phones, like real time voice communication, video streaming or online games. A possible drawback of more aggressive handover decision algorithm is the risk of being too aggressive, making it impossible to stay connected to a network with sufficient performance.

On the other hand using LTE is often more expensive than WiFi. While LTE often has a data cap or charges based on how much data is downloaded public WiFi is often provided for free in many places. Avoiding unexpected costs related to LTE usage is one possible motivation for not being more aggressive with initiating handover.

We think the results in this chapter shows that there is some room for improvement through an algorithm that is more aggressive and more consistent in taking the handover decision. In the next chapter we suggest such an algorithm that can run on the phone in the form of an app. The algorithm is monitors WiFi and disconnects from a network if performance becomes low.



## Chapter 5

# A new algorithm for initiating WiFi to LTE handover

In this chapter we propose a simple algorithm to initiate handover more aggressively than how it is done currently with smart network switch. Then we run the same experiments on the proposed algorithm as we did on handover with Smart network switch in Android, and compare the results.

### 5.1 New handover algorithm

In the previous chapter we saw that initiating handover was slow and inconsistent even when the delay, packet loss were high or the available throughput was low. We propose to send ping packets at regular intervals to estimate packet loss and round trip time. RX is a count of the total number of bytes received by the Android phone. If RX is above the desired minimum throughput then there is no need to initiate handover. On the other hand, if RX is below the minimum desired threshold then we can do an active test by doing a small download. If the download finishes in a reasonable amount of time then we have sufficient throughput.

Figures 5.1 on the following page and 5.2 on page 47 shows the basic steps of the proposed algorithm as a pseudo code and as a flow chart. The algorithm starts of with executing ping. Ping [30] works by sending ICMP ECHO\_REQUEST packets and tracking how much time passes before a ECHO\_RESPONSE is received. How many ping packets to send and the interval between them are parameters that we can adjust. If the results of executing ping is that packet loss or round trip time is higher than some boundary then we initiate handover. Otherwise we monitor RX for a while and if RX is too low, for example if the phone is idle, then we do a test download to make sure connectivity is good. IF the download takes too long we initiate handover. Finally we sleep for a while before repeating the loop. Probing the network with pings and test downloads causes overhead in the form of generating traffic and in using energy, which is a limited resource in mobile devices. On the other hand probing the network more frequently leads to shorter time to initiate handover when a performance problem occurs.

---

```

while true do
  rtt, loss ← PING()
  if loss > threshold or rtt > threshold then
    INITIATEHANDOVER()
  end if
  rx ← CHECKRX()
  if rx < threshold then
    HTTPDOWNLOAD
    if timeout then
      INITIATEHANDOVER()
    end if
  end if
  SLEEP
end while

```

---

Figure 5.1: Handover Initiation algorithm

In the next chapter we will look at the effect of adjusting the parameters of the algorithm and how it compares with current handover in Android.

## 5.2 File transfer for bandwidth estimation

We are using file download over the HTTP protocol with a timeout to ensure that the WiFi network provides some level of throughput. The main benefit of using this method is that it is done at the user level and gives a close idea of the actual user experience.

There are two big drawbacks of using file transfer as an active test of the WiFi network. First is the overhead. When transferring a file we are generating traffic on the network and using the WiFi radio consumes energy. The second drawback is the high influence of cross-traffic. The HTTP protocol uses TCP as its transport protocol. TCP limits the amount of data that can be sent without receiving an acknowledgement with a congestion window mechanism.

### 5.2.1 tuning parameters for the new algorithm

The goals for the handover algorithm is to be more aggressive and consistent than the current android implementation in initiating handover. Acceptable values for packet loss, delay and throughput comes down to personal preferences and also what tasks the network is to carry out. We suggest that 500kbps might be a reasonable minimum accepted value because this is the requirement for streaming video from youtube.com [37], and according to CISCO nearly four-fifths of mobile data traffic in the world will be video by 2022[7]. Through some trial and error we arrived at setting the timeout for the download to 800ms + 2\*rtt, where rtt is measured by ping. In the results we present in this chapter we will see

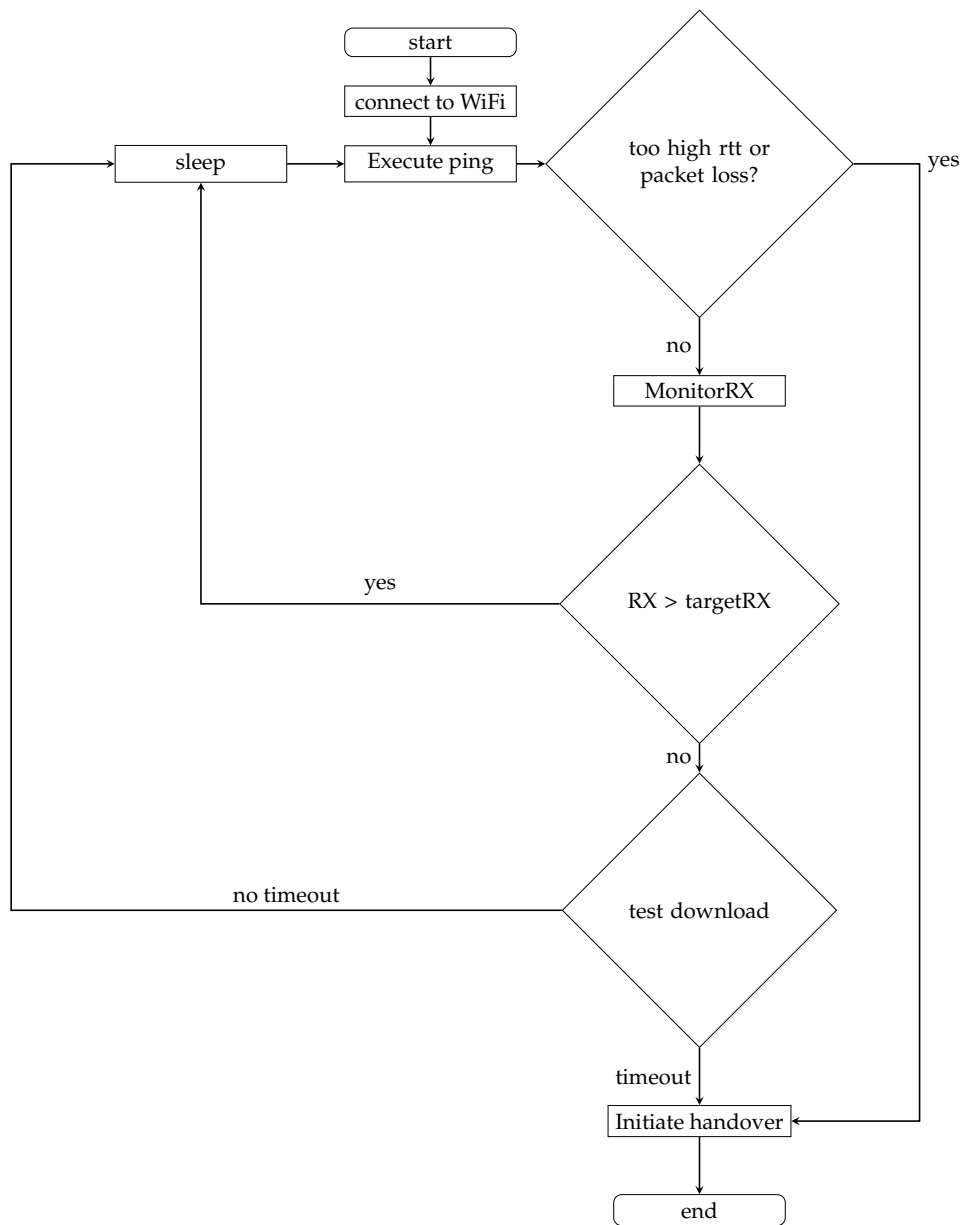


Figure 5.2: Handover initiation algorithm

setup	mean	standard deviation
Smart network switch normal	970ms	213,3ms
Smart network switch aggressive	377.3ms	160ms
Smart network switch aggressive and streaming video	580ms	67.7ms
New handover Algorithm	205ms	15.2ms

Table 5.1: Delay boundary values for Android phone

that the throughput threshold for initiating handover ends up quite close to 500kbps when using this timeout. Timeout and rtt are compared directly with the results from ping so these can easily be adjusted. We propose for instance using 300ms as the upper boundary for delay which means that we achieve the 150ms one-way delay which is recommended by [14] and for instance not accepting more than 10 % packet loss which is close what we saw when testing the aggressive setting of smart network switch in Android.

### 5.2.2 Results of testing the new algorithm in comparison with existing handover in Android

Table 5.1 shows the previous results for the mean delay value where handover was triggered and also the observed value where the new algorithm triggered handover. Using ping we found that average rtt to google.com was about 20 milliseconds. We recall from the methodology chapter that when we add delay using the router the delay only applies in one direction. So if we add 50ms of delay then rtt should increase by 50 instead of 100. Since we set the delay cutoff to 300 we would expect that handover would be triggered when we added around 280ms, but what actually happens is that it is triggered around 205 milliseconds. We investigated this further and found that when we set delay with netem we sometimes get "spikes" in the delay. Before doing any experiments we tested adding delay between the router and the laptop which is connected with an ethernet cable and then the delay added by netem was completely stable. But when we are connected with WiFi and add delay with netem rtt reported by ping is unstable and sometimes more than 100ms higher than the delay we specified in the command line. This explains why we see the handover with much less added delay than expected.

When we added around 200ms of delay with netem we will sometimes see delay above 300ms with ping, which explains why we was handover at a mean value of 205ms. The other delay boundary results all have quite high variation. Even with all this variation it is at least clear that handover is not reliably triggered by the existing mechanism in Android before we



<b>setup</b>	<b>mean</b>	<b>standard deviation</b>
Smart network switch normal	38,7%	10,4%
Smart network switch aggressive	8,2%	3,2%
Smart network switch aggressive and streaming video	15,8%	5,2%
New handover Algorithm	7,6%	3,8%

Table 5.2: packet loss values for Android phone

<b>setup</b>	<b>mean</b>	<b>standard deviation</b>
Smart network switch normal	11,3kbps	9,7kbps
Smart network switch aggressive	198kbps	40,2kbps
Smart network switch aggressive and streaming video	173kbps	53,1kbps
New handover Algorithm	540	118,6

Table 5.3: throughput boundary values for Android phone

have several hundred milliseconds of delay. Variation is a lot lower in the results for the new algorithm at 15.2ms.

Even with some variation in the results we can clearly see that the proposed algorithm will initiate handover at a lower delay and the lower standard deviation indicates that it is more consistent in initiating handover even though the mean for added delay is misleading because of the "ping spikes".

Figures 5.2 and 5.3 shows the previous results of testing boundary for packet loss and throughput in the new algorithm in addition to the previous results of testing the existing handover mechanism. The mean boundary value for packet loss is 7,6% with a standard deviation of 3,8%. Since the router is randomly dropping packets such that on average 10% of all packets are dropped it seems reasonable that we would get handover when the loss value is somewhat lower than 10 % since the packets to drop are chosen randomly sometimes more than one of the ping packets end up being dropped.

Table 5.4 on the next page shows the times it took to initiate handover in Android which we discussed in chapter 4 on page 31. It also shows the time it took to initiate handover at the same conditions with the new algorithm. The delay between low performance of the WiFi network and

setup	mean time before handover	standard deviation
<b>Android smart network switch aggressive</b>		
Throughput limited to 50kbps	75s	28,5s
700ms delay	95,9s	71,7s
50% packet loss	35,1s	23,6s
<b>New handover algorithm with sleep set to 0 seconds</b>		
Throughput limited to 50kbps	6,6s	3,1
700ms delay	7,3s	3,6s
50% packet loss	6,6s	3,3s
<b>New handover algorithm with sleep set to 20 seconds</b>		
Throughput limited to 50kbps	18,2s	7,2s
700ms delay	17,3s	9,3s
50% packet loss	14,9s	10s

Table 5.4: Time taken before making the handover decision.

handover depends on how often we actively test with ping or a download. The parameters for number of seconds spent monitoring RX or sleeping will affect this delay.

Sending ten ping packets with an interval of half a second takes about five seconds depending on round trip time for the replies. Timeout for the download is set to  $800\text{ms} + \text{rtt} * 2$ , and because we are sleeping for 1 second and repeating the download in the case of timeout this can around 2,6 seconds depending on rtt. Most of the time if the download is successfully it will take much less time at most a few hundred milliseconds.

We let the algorithm monitor RX for five seconds, meaning that testing ping, testing download and monitoring RX takes roughly ten seconds. we Tested the scenarios of low throughput, high delay and high packet loss with the sleep parameter set to zero seconds and again with the sleep parameter set to 20 seconds. Therefore the interval between sending pings or between doing a download would be roughly to ten seconds in the first set of tests and 30 seconds in the second set of tests. If we are running the the test that triggers handover every ten seconds then delay from limiting performance until handover is triggered could be anything between 0 and ten seconds and the average will be five seconds. Because we sleep for one second and repeat the download before initiating handover we take an additional one seconds plus the timeout of the downloads before initiating handover, which means that we the maximum time it could take is closer to 12,6 seconds than 10 and that we should expect a value around 6.3 seconds to be the mean of the observations. This matches quite well with the observed values, which are 6,6 seconds for limited throughput 7,3 seconds for tests with added delay and 6,6 seconds for the tests with packet loss.

The higher mean for the tests with delay could be because the increased delay causes it to take a longer time before the ping response packets arrive at the phone.

For the set of tests with a 20 second sleep the mean times before handover are 18,2 seconds with a standard deviation of 7.2s for limited throughput 17,3 seconds with a standard deviation of 9,3 seconds for the tests with added delay and finally 14,9 seconds with 10 seconds standard deviation for the packet loss tests.

The results in this section have shows that When we run tests more frequently we can initiate handover more quickly in the event of loss of Internet connectivity or poor performance. In the next section we look at the overhead of running this algorithm.

## 5.3 Overhead

### 5.3.1 Data usage

To estimate data usage we started by using tcpdump to capture all traffic generated when executing ping and when downloading a webpage. The rest of the time the process is either sleeping or monitoring RX which does not send any data over WiFi.

The size of each ping packet is 98 bytes and we are sending 10 ping packets and receiving 10 in reply if there is no loss. In the download test we are downloading <https://www.google.com>. The get request to start the download as captured in tcpdump is 255 bytes. At this time the total size of the received packets containing the google webpage is 47398 bytes. The total download for one round of the active tests is  $98 * 10 + 47398$  bytes, and total upload is  $98 * 10 + 255$ . The total data usage in day depends on the frequency of the active tests of the algorithm. The goal of the algorithm is to automatically disconnect from WiFi when performance is insufficient to remove the need for users to manually disconnect from the WiFi access point. Therefore we don't need to run the algorithm when WiFi is disabled or when the phone is not in use like when the screen is off.

We also used TCPdump to capture packets from the phone to compare data usage of the existing handover method vs the algorithm proposed in this thesis. As we saw in section 4.3 on page 37, Android is sending ping packets to the default gateway and when we activate the aggressive smart network switch it sends pings more frequently.

The participants in a study by Andrews et al. [2] used their mobile phones for a total of 5.05 hours each day. Assuming that the phone is connected to WiFi and used for 5 hours a day we can estimate the total data usage of running the proposed handover initiation algorithm. The download test will only be executed when the measured RX is low, but to get a worst case estimate we assume that the measured RX always is so low that the download test will be executed.

In Figure 5.3 on the following page we see a comparison of the effect of adjusting parameters of the algorithm. How much data is used depends on

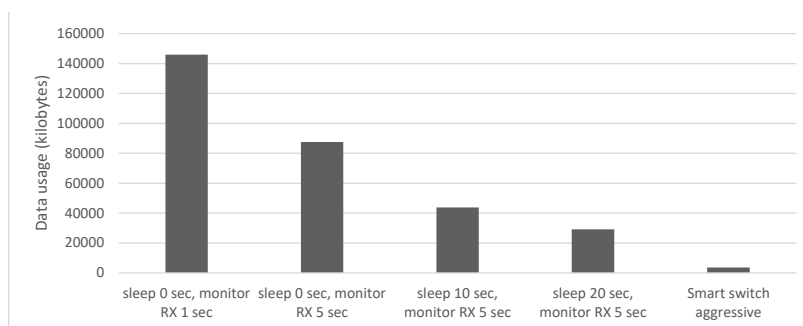


Figure 5.3: Data usage while adjusting parameters

the frequency of the active tests. Active tests, in the form of a file download or ping, are not done during the sleep period, and they are not done during the period of monitoring RX on the phone. When we decrease the values of these parameters we see that the data usage is increased.

### 5.3.2 Energy usage

We measured energy usage with Google’s battery historian tool [32]. Battery historian gives details about each apps power usage and Samsung Galaxy S7 has a 3000 mAh battery [35]. We let the algorithm run for 30 minutes and then used battery historian to inspect the energy usage. The standard deviation for all these measurements is 0,01 or less. Then to get an estimate of battery usage for five ours we multiply the power usage values for 30 minutes with 10. According to [2] five hours was how much time the participants spent using their phones per day. Figure 5.4 on the next page shows the percentage of total battery capacity expended when running the algorithm for five ours with different adjusting the values for some of the parameters. For the first test we set sleep to zero and only monitored RX for one second. With these values the energy usage was 1,1% of total battery capacity. In the rest of the tests we monitored RX over five seconds and varied the sleep time. In the second measurements sleep was set to 0 seconds and energy usage was 0.9 %. In the third measurements sleep was set to ten seconds and the measured battery usage was 0,7%, and in the last measurements we set the sleep parameter to 20 seconds and measured energy usage was 0.6%.

[6] is an analysis of power consumption in a smartphone. In this study Carroll et al. perform multiple benchmarks and analyse the power consumption of various components in a smartphone. In the network intensive benchmarks like downloading files, they found that WiFi used much more power than components like CPU or RAM. This could explain why we see that battery usage increases with the data overhead.

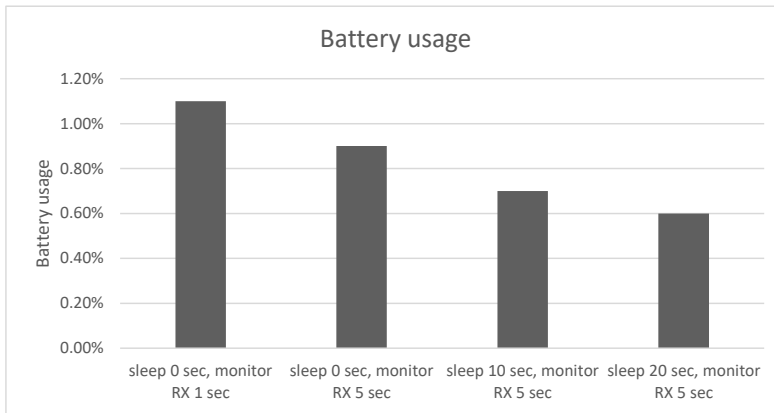


Figure 5.4: Battery usage while adjusting parameters

## 5.4 Summary and conclusions

In this chapter we have suggested some performance threshold values that might be reasonable for initiating handover and we have experimented with some values for the parameters of active tests of the algorithm and seen lead to initiating handover when performance of the WiFi network is relatively close to the suggested thresholds. We have also seen that running the active tests in the algorithm more frequently leads to a lower delay before initiating handover but also lead to higher overhead in terms of data usage and power usage. The results show that the new algorithm, depending on the sleep parameter, can initiate handover faster than the existing smart network switch in Android. However this comes at the cost of a much higher data usage overhead. Higher data usage might not be a problem on WiFi where you typically don't have data caps or pay per megabyte. On the other hand, on mobile networks like LTE networks this way of initiating handover is not suitable since data costs are typically higher in mobile networks. In other words, the proposed algorithm, can potentially be suitable for initiating handover from WiFi to LTE, but not the other way around. We have used an Android phone in this thesis, however the proposed algorithm could also be used on other kinds of devices that have access to multiple different wireless networks.



# Chapter 6

## Conclusion

### 6.1 Summary of contributions

In this thesis we investigated how handover from WiFi to LTE is initiated in a mobile phone with the Android operating system. More specifically we conducted experiments where we looked at how poor network performance in terms of low throughput, high delay or high packet loss would trigger the initiation of handover.

Based on the experiments we saw that activating the setting to automatically hand over from WiFi to LTE when WiFi becomes unstable indeed does cause handover, and that the aggressive settings actually will be more aggressive in instating handover. On the other hand it often takes several minutes before handover is triggered automatically, meaning that a user would likely disable WiFi manually to attempt using LTE instead. The experiments also showed that the performance in terms of throughput, delay or packet loss become so poor that the network was practically useless for many tasks before handover would initiate.

Based on this we proposed a simple algorithm to automatically initiate handover, and created a prototype implementation which we tested in comparison to the existing handover mechanism in Android. Our tests showed that this prototype implementation would initiate handover faster than the current implementation in Android but at the cost of overhead in terms of data and energy usage.

### 6.2 Future work

In this thesis we have focused on handover in a Samsung Galaxy S7 mobile phone with Android 7.0. Many other kinds of devices also have access to both LTE and WiFi. One way to extend the work in this thesis is to look at how handover is handled in other devices.

Another way to extend the work in this thesis is to implement and run experiments on more handover algorithms. For example some of the handover algorithms covered in related studies, like [22], have only been evaluated by simulation. Running experiments would allow us to evaluate

the real world performance of these algorithms and to see if this is in line with the results from running simulations.

In the proposed algorithm we, used a file transfer as an active test of the throughput of the WiFi network. If we had developed a server in addition to the code running on the Android phone itself then we could have tried other bandwidth estimation techniques, which for example could result in lower overhead in data usage.



# Bibliography

- [1] *3GPP Declares First 5G NR Spec Complete*. URL: <https://www.fiercewireless.com/wireless/3gpp-declares-first-5g-nr-spec-complete> (visited on 05/07/2019).
- [2] Sally Andrews et al. "Beyond Self-Report: Tools to Compare Estimated and Real-World Smartphone Use." In: *PLOS ONE* 10.10 (Oct. 28, 2015). Ed. by Jakob Pietschnig, e0139004. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0139004. URL: <https://dx.plos.org/10.1371/journal.pone.0139004> (visited on 04/11/2019).
- [3] *Android Debug Bridge (Adb)*. URL: <https://developer.android.com/studio/command-line/adb> (visited on 02/23/2019).
- [4] *Archer C7 | AC1750 Wireless Dual Band Gigabit Router | TP-Link*. URL: [https://www.tp-link.com/en/products/details/cat-9\\_Archer-C7.html#overview](https://www.tp-link.com/en/products/details/cat-9_Archer-C7.html#overview) (visited on 11/15/2018).
- [5] Stefano Busanelli et al. "Vertical Handover between WiFi and UMTS Networks: Experimental Performance Analysis." In: (Jan. 2011).
- [6] Aaron Carroll and Gernot Heiser. "An Analysis of Power Consumption in a Smartphone." In: (), p. 14.
- [7] *Cisco Visual Networking Index: Forecast and Methodology, 2016–2021*. URL: <https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html> (visited on 11/15/2018).
- [8] D. E. Comer et al. "Computing As a Discipline." In: *Commun. ACM* 32.1 (Jan. 1989). Ed. by Peter J. Denning, pp. 9–23. ISSN: 0001-0782. DOI: 10.1145/63238.63239. URL: <http://doi.acm.org/10.1145/63238.63239> (visited on 04/23/2019).
- [9] Douglas Comer. *Computer Networks and Internets*. 6., ed., global ed. Always Learning. OCLC: 900493613. Boston: Pearson, 2015. 667 pp. ISBN: 978-1-292-06117-7.
- [10] *ConnectivityManager.NetworkCallback*. URL: <https://developer.android.com/reference/kotlin/android/net/ConnectivityManager.NetworkCallback> (visited on 02/23/2019).
- [11] *Different Wi-Fi Protocols and Data Rates*. URL: <https://www.intel.com/content/www/us/en/support/articles/000005725/network-and-i-o/wireless-networking.html> (visited on 03/04/2019).

- [12] *Distribution Dashboard*. URL: <https://developer.android.com/about/dashboards> (visited on 03/03/2019).
- [13] Paweł Foremski and Krzysztof Grochla. "LTE or WiFi? Client-Side Internet Link Selection for Smartphones." In: *Computer Networks*. Ed. by Piotr Gaj, Andrzej Kwiecień, and Piotr Stera. Springer International Publishing, 2015, pp. 43–53. ISBN: 978-3-319-19419-6.
- [14] *G.114 : One-Way Transmission Time*. URL: <https://www.itu.int/rec/T-REC-G.114-200305-1/en> (visited on 03/11/2019).
- [15] *GitHub - Google/Battery-Historian: Battery Historian Is a Tool to Analyze Battery Consumers Using Android "Bugreport" Files*. URL: <https://github.com/google/battery-historian> (visited on 04/24/2019).
- [16] *GitHub - Nodogsplash/Nodogsplash: Nodogsplash Offers a Simple Way to Provide Restricted Access to an Internet Connection Using a Captive Portal. Pull Requests Are Welcome!* URL: <https://github.com/nodogsplash/nodogsplash> (visited on 11/10/2018).
- [17] Andreas Hanemann et al. "A Study on Network Performance Metrics and Their Composition." In: *Campus-Wide Information Systems* 23.4 (Aug. 2006). Ed. by Ingrid Melve, pp. 268–282. ISSN: 1065-0741. DOI: 10.1108/10650740610704135. URL: <http://www.emeraldinsight.com/doi/10.1108/10650740610704135> (visited on 11/15/2018).
- [18] Tim Higgins. *TP-LINK Archer C7 V2 Reviewed*. URL: <https://www.smallnetbuilder.com/wireless/wireless-reviews/32498-tp-link-archer-c7-v2-reviewed> (visited on 02/24/2019).
- [19] *How Do I Turn on the Developer Options Menu on My Samsung Galaxy Device? | Samsung Support UK*. URL: <http://www.samsung.com/uk/support/mobile-devices/how-do-i-turn-on-the-developer-options-menu-on-my-samsung-galaxy-device/> (visited on 03/13/2019).
- [20] "IEEE Standard for Information Technology– Telecommunications and Information Exchange between systemsLocal and Metropolitan Area Networks– Specific Requirements–Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications–Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6 GHz." In: *IEEE Std 802.11ac-2013 (Amendment to IEEE Std 802.11-2012, as amended by IEEE Std 802.11ae-2012, IEEE Std 802.11aa-2012, and IEEE Std 802.11ad-2012)* (Dec. 2013), pp. 1–425. DOI: 10.1109/IEEESTD.2013.6687187.
- [21] *Index of /Releases/18.06.1/Targets/Ar71xx/Generic/*. URL: <http://downloads.openwrt.org/releases/18.06.1/targets/ar71xx/generic/> (visited on 03/21/2019).
- [22] T. Inzerilli et al. "A Location-Based Vertical Handover Algorithm for Limitation of the Ping-Pong Effect." In: *2008 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*. 2008 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WIMOB). Avignon, France: IEEE, Oct. 2008, pp. 385–389. DOI: 10.1109/WiMob.2008.

64. URL: <http://ieeexplore.ieee.org/document/4654269/> (visited on 02/25/2019).
- [23] Heikki Kaaranen et al. *UMTS Networks: Architecture, Mobility and Services*. 2 edition. Chichester, West Sussex, England ; Hoboken, NJ: Wiley, Mar. 4, 2005. 422 pp. ISBN: 978-0-470-01103-4.
- [24] Markku Kojo and Jukka Manner. *Mobility Related Terminology*. URL: <https://tools.ietf.org/html/rfc3753> (visited on 02/13/2019).
- [25] G. Lampropoulos et al. "Handover Management Architectures in Integrated WLAN/Cellular Networks." In: *IEEE Communications Surveys Tutorials* 7.4 (Fourth 2005), pp. 30–44. ISSN: 1553-877X. DOI: 10.1109/COMST.2005.1593278.
- [26] *Linux Advanced Routing & Traffic Control HOWTO*. URL: <https://lartc.org/howto/> (visited on 03/01/2019).
- [27] *Networking:Netem [Wiki]*. URL: <https://wiki.linuxfoundation.org/networking/netem> (visited on 02/10/2019).
- [28] Shahriar Nirjon et al. "MultiNets: A System for Real-Time Switching between Multiple Network Interfaces on Mobile Devices." In: *ACM Transactions on Embedded Computing Systems* 13 (4s Apr. 1, 2014), pp. 1–25. ISSN: 15399087. DOI: 10.1145/2489788. URL: <http://dl.acm.org/citation.cfm?doid=2601432.2489788> (visited on 12/08/2018).
- [29] *OpenWrt Project: Welcome to the OpenWrt Project*. URL: <https://openwrt.org/> (visited on 11/07/2018).
- [30] *Ping(8) - Linux Man Page*. URL: <https://linux.die.net/man/8/ping> (visited on 02/03/2019).
- [31] G. P. Pollini. "Trends in Handover Design." In: *IEEE Communications Magazine* 34.3 (Mar. 1996), pp. 82–90. ISSN: 0163-6804. DOI: 10.1109/35.486807.
- [32] *Profile Battery Usage with Batterystats and Battery Historian | Android Developers*. URL: <https://developer.android.com/studio/profile/battery-historian> (visited on 04/11/2019).
- [33] "Quality of Service for Voice over IP." In: (), p. 44.
- [34] *Samsung Galaxy S7 and S7 Edge*. URL: <http://www.samsung.com/global/galaxy/galaxy-s7/> (visited on 03/21/2019).
- [35] *Samsung Galaxy S7 Edge Was World's Top-Selling Android Smartphone in H1 2016, Says Strategy Analytics*. URL: <https://www.strategyanalytics.com/strategy-analytics/news/strategy-analytics-press-releases/2016/08/01/strategy-analytics-samsung-galaxy-s7-edge-was-world's-top-selling-android-smartphone-in-h1-2016> (visited on 02/14/2019).
- [36] *Specifications Home*. URL: <http://www.3gpp.org/specifications/specifications> (visited on 12/10/2018).
- [37] *System Requirements - YouTube Help*. URL: <https://support.google.com/youtube/answer/78358?hl=en> (visited on 03/14/2019).

- [38] *Tc(8) - Linux Manual Page*. URL: <http://man7.org/linux/man-pages/man8/tc.8.html> (visited on 02/10/2019).
- [39] *tcpdump. Tcpdump/Libpcap Public Repository*. URL: <https://www.tcpdump.org> (visited on 02/08/2019).
- [40] *The Evolution of WiFi Standards: A Look at 802.11a/b/g/n/Ac*. June 22, 2017. URL: <https://www.actiontec.com/wifihelp/evolution-wi-fi-standards-look-802-11abgnac/> (visited on 03/04/2019).
- [41] *What Is Adaptive Wi-Fi? | Samsung Support Australia*. URL: <http://www.samsung.com/au/support/mobile-devices/galaxy-note-8-what-is-adaptive-wifi/> (visited on 11/14/2018).
- [42] *Wireshark · Go Deep*. URL: <https://www.wireshark.org/> (visited on 02/08/2019).
- [43] *Xkcd: Wifi vs Cellular*. URL: <https://xkcd.com/1865/> (visited on 12/06/2018).
- [44] Xiaohuan Yan, Y. Ahmet Şekercioğlu, and Sathya Narayanan. "A Survey of Vertical Handover Decision Algorithms in Fourth Generation Heterogeneous Wireless Networks." In: *Computer Networks* 54.11 (Aug. 2010), pp. 1848–1863. ISSN: 13891286. DOI: 10.1016/j.comnet.2010.02.006. URL: <https://linkinghub.elsevier.com/retrieve/pii/S1389128610000502> (visited on 02/25/2019).