Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit Nr. 95

Energy Models for Wireless Communication on Mobile Devices

Jonas Heinisch

Course of Study:	Softwaretechnik
Examiner:	Prof. Dr. rer. nat. Dr. hc. Kurt Rothermel
Supervisor:	Dipl.Inf. Christoph Dibak
Commenced:	March 31, 2016
Completed:	September 30, 2016
CR-Classification:	C.2.0, C.2.m

Abstract

With the evolution of wireless communication networks, the access to the internet became much faster for mobile devices. This enables new distribution strategies like offloading of complex tasks to a remote server. Nevertheless energy resources on mobile devices is limited by the battery capacity. In order to decide if calculations should be offloaded to a server ot executed on a mobile device, a prediction model for energy consumption is needed.
This thesis proposes energy prediction models for wireless communication on mobile battery-powered devices. The prediction models cover the network types LTE and WiFi. The energy consumptions predicted by combining several linear regressions alongside statistical prediction intervals. The regression is based on energy measurements on a mobile device of 288 client-server communications. The prediction results can compared to other data to decide on the usage of the wireless network. Evaluating the prediction model with an additional set of measurements shows a relative error of 1.21% in case of LTE and 26.73% in case of WiFi.

Contents

List of Figures

List of Tables

List of Listings

1 Introduction

In the last ten years mobile devices like smartphones and tablets were enriched with greater and faster hardware resources. This enables developers to program complex resource-intensive mobile applications. As an example we can think of augmented reality applications or solving routing problems with various additional information that needs to be considered. These applications need a lot of CPU-time, which is very energy consuming.

Based on the evolution of smartphones there is one critical point from the user perspective: Energy consumption. By design the battery of modern smartphones only has enough capacity to serve the device for a single day or, when the device is heavily used, sometimes even less.

Wireless communication has been improved in terms of throughput, latency, availability and cost. The two common wireless connection types at the moment are WiFi, which's data rates are still improved, and LTE. Fast wireless communication enables offloading of calculations to remote servers in order to save energy.

Nevertheless it has to be decided very carefully whether to offload calculations or to compute it on the device. This can depend on the status of the network or the communication needed to offload and as [DDR15] and [DK14] propose mixing both approaches might be even more efficient. Based on these possibilities an application needs to decide on a per case basis, which way is less energy consuming.

To show the relevance one can think of the following example (also visualized in figure 1.1): A mobile app has been developed that can take a picture of a person and can do an automatic facial expression analysis like Microsoft's Cognitive Services Emotion API[1], which requires a detailed analysis of the face. Imagining there is a library, which could be used to analyze the photo on the phone as well as on a remote cloud server, there is now the question, which of the solutions is more efficient in terms of time and energy consumption.

This thesis focuses on creating a prediction model, that is able to predict the energy that is consumed by the wireless communication triggered by an application. Therefore the

[1] https://www.microsoft.com/cognitive-services/en-us/emotion-api

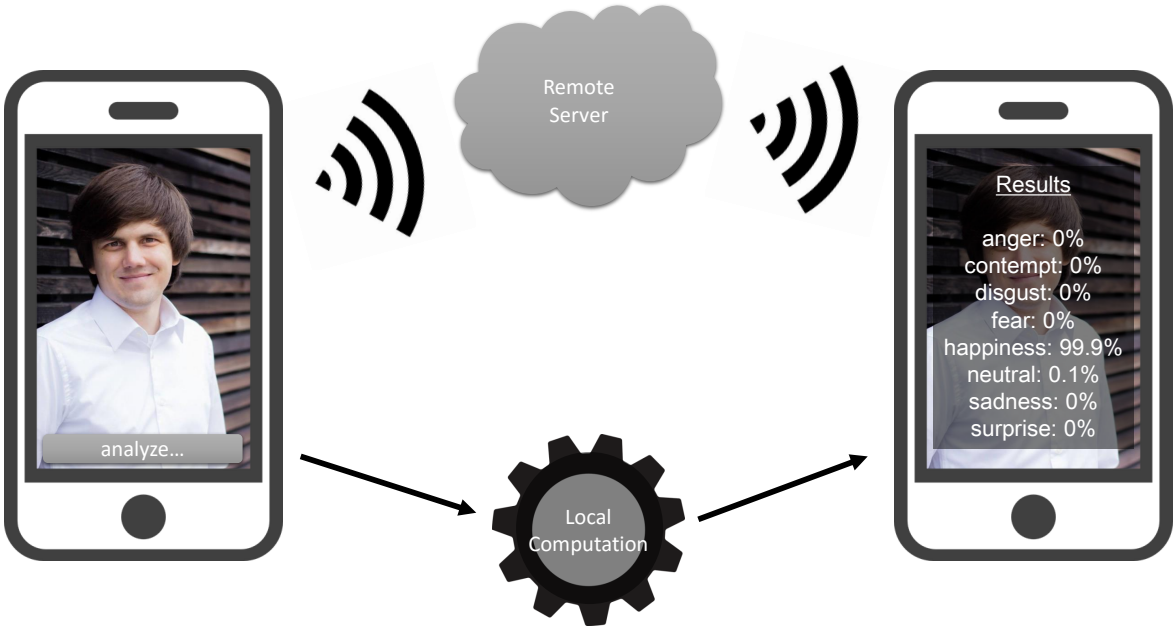

Figure 1.1: Visualization of the example use case

assumption is that the consumed energy is dependent on the current network connection, i.e. its type, signal strength, and the properties of the required data transmissions, i.e. the size of the data and the time the server needs to generate an answer to the request.

Speaking in terms of the example the prediction model needs to know the type of network (e.g. LTE), its signal strength (e.g. -72dBm), the size of request and response (e.g. 1MB request containing a JPEG-file and 2kB response containing various analysis results) and also the time the server needs to perform the analysis (e.g. 1400ms). The result should be a numeric result, to compare it to the energy prediction for local calculation. Another important requirement is that the prediction itself is fast and consumes very little energy, because otherwise the prediction may consume more energy than it may save.

To create such a prediction model it is required to first gather knowledge about the correlations of the different variables involved in a wireless communication. Therefore tests and measurements on signal strength and energy consumption can be performed with the two wireless communication types WiFi and LTE. After extracting the information from the resulting logs, these need to be analyzed. Based on the gathered knowledge it is possible to create a prediction model. To find a good prediction model different approaches to predict the energy consumption can be proposed. To finally evaluate the created models and compare them, additional measurements need to be made and evaluated to find the best model.

1.1 Outline

The aim of this thesis is to accompany the work on the previously described approach. The following list provides an outline on the chapters of the thesis:

Chapter 2 – Background describes the background knowledge required for the proceeding thesis.

Chapter 3 – Related Work shows paper that cover a similar topic and discusses the differences between them and this thesis.

Chapter 4 – System Model & Problem Statement describes the system model and its parameters to which the prediction model applies and points out the problem definition.

Chapter 5 – Measurements outlines the measurements that have been performed to gather the data and presents the data retrieved from the measurements.

Chapter 6 – Prediction Model proposes two different prediction models for predicting the energy consumption of mobile communication.

Chapter 7 – Implementation gives an overview on the implementation that has been done to perform the measurements and analytic tasks. Also the implementation of the prediction models will be presented.

Chapter 8 – Evaluation evaluates the prediction models according to their accurateness and compares both.

Chapter 9 – Conclusions condenses the results of this thesis and collects ideas for future work.

2 Background

This chapter provides background knowledge about several topics, which are important for this thesis. At first the different wireless communication networks, for which prediction models will be created, are introduced. Secondly the two most used transport protocols UDP and TCP will be covered.

2.1 WiFi

WiFi is a synonym for wireless local area networks (WLAN) fulfilling the 802.11 standard. It is typically used to connect several computers with a wired network (LAN or Internet) using an access point (also called base station). The 802.11 standard has evolved since its publishing in 1997 and according to [TW11], it "now defines rates up to [...] 600 Mbps", which is even yet not true anymore as it currently defines a maximum data rate of 6.75 Gbps as stated in [SDEG12]. The standard is still evolving to steadily improve the data rate to 20 Gbps[1].

The 802.11 protocol stack can be seen in an overview in figure 2.1 until the in 2009 released standard 802.11n. The standard covers the data link layer and the physical layer. The data link layer is split into a logical link layer and a MAC (medium access control) sublayer. The data link layer "hide[s] the differences between the different 802 variants and make them indistinguishable as far as the network layer is concerned" [TW11]. The MAC protocol is responsible to decide, when a tenant in the network can access the medium to send data without any other data to collide with it. "802.11 tries to avoid collisions with a protocol called CSMA/CA ([Carrier Sense Multiple Access] with Collision Avoidance)" [TW11]. More details on this protocol can be read in [TW11].

The physical layer protocol of the newer standards (i.e. 802.11g ff.) according to [TW11] is based on OFDM (Orthogonal Frequency Division Multiplexing) and MIMO (Multiple Input Multiple Output). Important to know is that the physical layer adds

[1] Newest WiFi standard 802.11ay is planned to be published in 2017: http://www.ieee802.org/11/Reports/tgay_update.htm

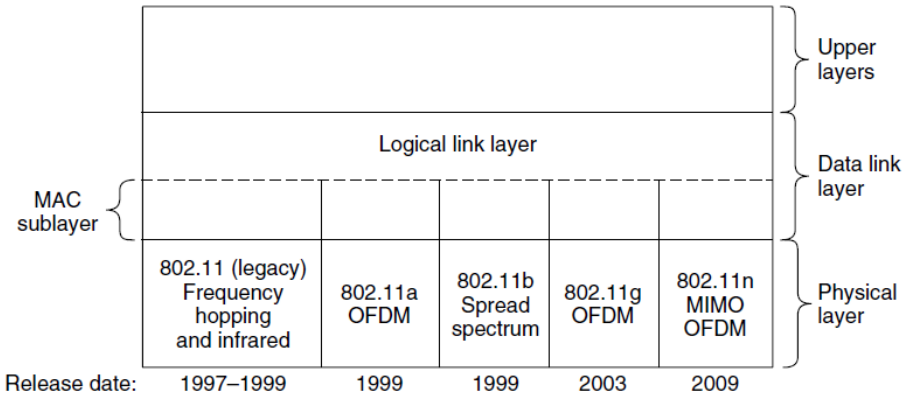

Figure 2.1: 802.11 protocol stack [TW11]

forward error correction (FEC) to correct data loss early by using redundant bits. To avoid interference there are different channels a base station can use.

Regarding the energy management according to [BBV09] the energy "cost of maintaining the association is small" due to the Power Save Mode (PSM). Nevertheless WiFi has a tail time, i.e. time the network device stays in high power model, that is occurring after a transmission for a specific time, before it switches to a power saving mode.

To sum up, WiFi is a wireless LAN standard to connect mobile devices to a local network and/or the Internet. The speed is steadily increasing, while data loss is prevented by applying FEC on the physical layer.

2.2 LTE

In 2004 the 3GPP (3rd Generation Partnership Project) started to develop LTE, which "evolved from an earlier 3GPP system known as the Universal Mobile Telecommunication System (UMTS)" [Cox12]. With the rise of smartphones (Apple's iPhone in 2007[2] followed by the first Android phone in 2008[3]) the usage of mobile data started to increase slowly at the beginning, but around the year 2010 the traffic increased dramatically [Cox12]. Cisco claims that mobile data usage is still steadily increasing (74% in 2015) and forecasts in early 2016 that "[g]lobal mobile data traffic will increase nearly eightfold between 2015 and 2020" to 30.6 exabytes per month (cf figure 2.2) [Cis16]. To handle

[2]Apple Reinvents the Phone with iPhone, January 9, 2007; http://www.apple.com/pr/library/2007/01/09Apple-Reinvents-the-Phone-with-iPhone.html

[3]The first Android-powered phone, September 23, 2008; https://googleblog.blogspot.de/2008/09/first-android-powered-phone.html

Figure 2.2: Cisco forecast for global mobile traffic per month [Cis16]

this amount of data newer and faster mobile data networks are necessary. LTE (as a 4G network) takes an important role in handling it because of its higher data rates.

Higher data rates has been one of the most important requirements for LTE. It was required to have peak data rates of up to 100Mbps for downlink and 50Mbps for uplink, which even has been exceeded to 300Mbps and 75Mbps [Cox12]. This is an increase of more than 21-fold compared to WCDMA. However these peak data rates are typically not achievable as [Mot09] points out.

Another important improvement is the improvement in latency. The requirements to LTE are that the latency from mobile device to wired network should be smaller than five milliseconds. The range of a cell is optimized for a range of 5km, but supports a size of up to 100km with loss of quality. In terms of mobility it is optimized for a speed of 15 km/h, can serve with high performance till 120km/h and works up to 350km/h.

2.2.1 Air Protocol Stack

To have an understanding about the way of communication on the air, the first thing to look at is the protocol stack for this part. An overview on the protocol stack can be found in figure 2.3. The stack is divided into user and control plane at the top. The user plan supports any protocol stack based on IP as network layer protocol. The control plane contains a stack consisting of EMM, ESM and RRC protocols to mainly control the allocation of resources, registering and unregistering as well as controlling the energy states of the connection. RRC (Radio Resource Control) handles the communication of "radio access capabilities, such as the maximum data rate [the mobile device] can handle

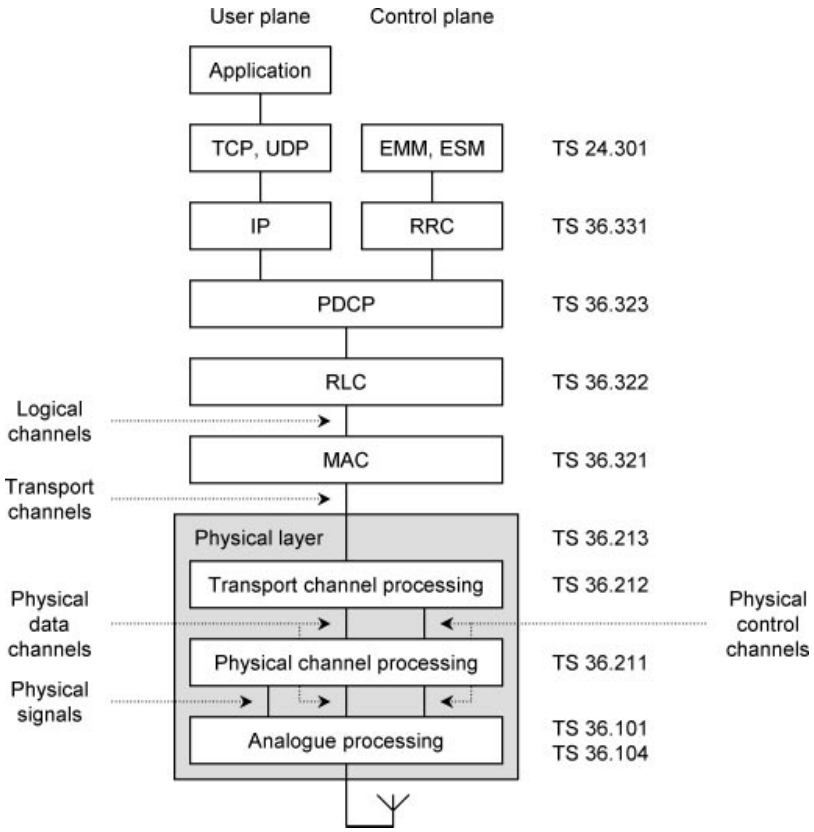

Figure 2.3: LTE protocol stack for air connection [Cox12]

and the specification release that it conforms to" [Cox12]. EMM and ESM are used to communicate with the EPC (evolved packet core), that can authorize the user to use the network.

The lower part stack is common for user and control plane. The packet data convergence protocol (PDCP) is responsible for higher-level transport functions like header compression and security etc. to improve performance and other quality properties [Cox12]. The lower radio link control (RLC) layer "maintains the data link between the two devices" [Cox12] by ensuring reliability.

To handle packet loss according to [Cox12] forward error correction (FEC) with two-stage process for setting an adaptive coding rate is applied: "In the first stage, the information bits are passed through a fixed-rate coder. The main algorithm used by LTE is known as turbo coding and has a fixed coding rate of 1/3. In the second stage, called rate matching, some of the coded bits are selected for transmission, while the others are discarded in a process known as puncturing." Additionally ARQ (Automatic Repeat Request) is applied, which appends a CRC checksum to detect if there is an error that has not been able to be corrected by FEC. In this case the protocol will request the

sender to resend the data and will combine both signals (original and resend) to have a more powerful signal.

The MAC (Medium Access Control) protocol “carries out low-level control of the physical layer” [Cox12], i.e. the “MAC scheduler in the [base station] is in charge of assigning both uplink and downlink radio resources” [LLM+09], which results are then communicated via the MAC protocol. LTE uses a “mixed technique known as orthogonal frequency division multiple access (OFDMA)” [Cox12] for multiplexing transmissions, which is according to [Cox12] also used by some WiFi networks.

2.2.2 Modulation

LTE can use different frequency modulations (BPSK for some control channels, QPSK, 16-QAM and 64-QAM) [Mot09], [Cox12]. As [Mot09] states the modulation can vary by the signal strength and therefore the throughput can vary within a cell and in practice “is very hard to predict and will depend on many factors typical of radio technologies (distance to cell, cell loading, subscriber speed, indoor, outdoor, macro layer, or hotspot)” [Mot09].

Additional to the varying of modulation this is caused by which is called fractional frequency re-use. Fractional frequency re-use is a technique to overcome the problem of interference between base stations at the cell edge. While, according to [Cox12], GSM splits up all carrier frequencies so that neighbors never interfere, which leads to a bandwidth usage of 25% per cell, UMTS uses the full bandwidth and accepts interference at the cell edge.

Fractional frequency re-use, which is used in LTE, distinguishes between nearby mobiles and mobiles at the edge and combines both previous techniques: Within the given bandwidth “each cell transmits to nearby mobiles using the same set of sub-carriers [...]. This works well, because the mobiles are close to their respective base stations, so the received signals are strong enough to overwhelm any interference. Distant mobiles receive much weaker signals, which are easily damaged by interference. To avoid this, neighboring cells can transmit to those mobiles using different sets of sub-carriers” [Cox12]. This results in about 40% of the average data rates at the cell edge.

2.2.3 Energy States

As energy usage is crucial for mobile devices as already stated in the motivation, LTE has to manage the power consumed by the mobile device using LTE. That means it cannot just continuously listen to the wireless channel. These energy states are managed by the

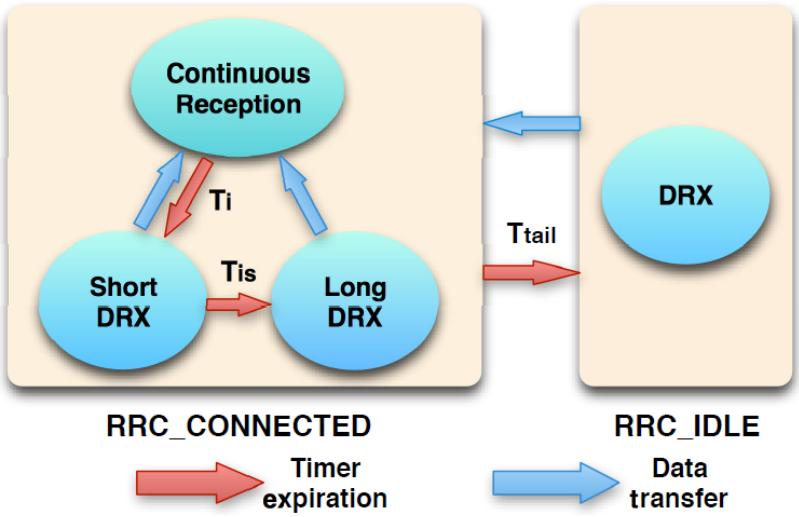

Figure 2.4: RRC state transitions in LTE network [HQG+12]

RRC protocol. The mobile device can stay in two different RRC states, namely `RRC_IDLE` and `RRC_CONNECTED`. "When on standby, a mobile is in RRC_IDLE" [Cox12], which means only a limited communication between the mobile and the base station is possible, i.e. a RRC connection can be initiated from both sides by a paging message.

In both states according to [HQG+12] the device is in a DRX-cycle (discontinuous reception), when it is not transferring data. That means that the device sleeps and awakes at specific time slots to listen for a short time to the medium. If there is a paging request, it awakes and changes into the reception phase. As one can see in the state chart in figure 2.4 the DRX-cycles are of different length, which means the duration between the listening slots changes. A longer time between listening means less energy consumption, but it increases the reaction time at the same time. Also transitioning from `RRC_IDLE` to `RRC_CONNECTED` takes some time, which also increases the reaction time.

However the time the mobile stays in `RRC_CONNECTED`-state is consuming a lot of energy. This is also referred to as *tail time*. According to [Mot09] LTE can improve the time switching the states in comparison to HSPA, where it "can be as much as 1000ms", to give "a user experience close to wired broadband's 'always-on' service".

2.2.4 Signal Strength

As seen before there are some characteristics of LTE that adapt to the distance of base stations and mobile device. The distance is estimated by the power of the received signal, i.e. the signal strength. There are different ways to express the signal strength. In the following lists the major approaches are explained.

Classification	RSRP
Good	$> -95dBm$
Medium	$-95dBm$ to $-108dBm$
Poor	$< -108dBm$

Table 2.1: Rough classification of RSRP according to [AIHF13]

- **RSRP** (Reference Symbol Received Power): According to [XMJC11] "RSRP is defined as the linear average over the power contributions [...] of the resource elements that carry cellspecific reference signals within the considered measurement". That means that the RSRP value also contains the noise that it received. Table 2.1 shows a classification of RSRP values.
- **RSSI** (Received Signal Strength Indicator): "RSSI comprises the linear average of the total received power [...] observed only in OFDM symbols containing reference symbols" [XMJC11]. According to [AIHF13] RSSI considers the interfering with other cells traffic.
- **RSRQ** (Reference Symbol Received Quality): RSRQ uses both indicators (RSRP and RSSI) to "express the relation between signal and noise" [XMJC11] and "is dependent on the load in the own cell as well as the interference from other cells" [AIHF13].

2.3 UDP

UDP (User Datagram Protocol) is a connectionless transport protocol defined in RFC 768. It is closing the gap between network layer, i.e. IPv4 or IPv6, and the application layer by matching packets to applications. This is done with very little overhead. The UDP header (cf. figure 2.5) has a size of only 8 bytes for the source and destination ports, the length of the UDP packet and a checksum, which is optional, to check the correctness of the packet. The ports identify the application that is addressed by the packet.

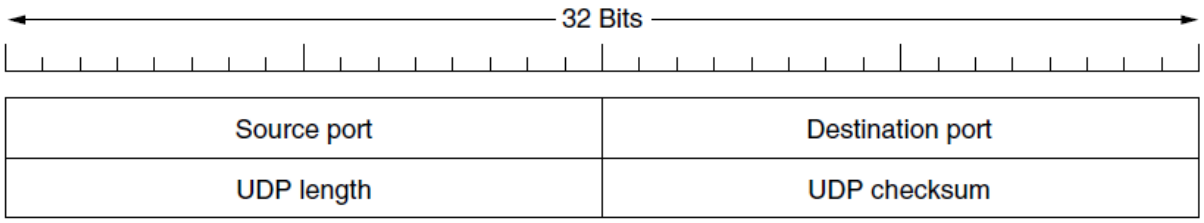

Figure 2.5: UDP header [TW11]

All in all UDP is a lightweight and simple transport protocol. There is not much overhead and no additional communication. That means that the application has “precise control over the packet flow, error control [and] timing” [TW11].

In practice it is good to perform remote procedure calls where request and response are very small, because additional overhead like acknowledgments would increase the relative effort significantly. Other important use cases are live streaming and voice or video telephony, where fast transmission matters and packet loss cannot be fixed by retransmission due to the latency. UDP itself cannot handle packet loss, different throughput limits or order of packets. If one needs some of these properties, they need to be implemented explicitly in the application.

2.4 TCP

TCP (Transmission Control Protocol) is a transport protocol initially “formally defined in RFC 793 in September 1981” [TW11]. Several improvements has been made over the years to make TCP more efficient. The focus of TCP is to provide a reliable end-to-end connection, in which byte streams can be send full duplex (i.e. in both directions). It even can provide reliability within an unreliable internetwork, which “differs from a single network because different parts may have wildly different topologies, bandwidths, delays, packet sizes, and other parameters” [TW11].

As already mentioned TCP handles byte streams in contradiction to UDP, which handles messages. Nevertheless as the underlying network protocol is typically packet oriented TCP splits the byte stream into chunks called *TCP segments*. TCP can decide on the size of the chunks, but “in practice [the size is] often 1460 data bytes in order to fit into a single Ethernet frame [i.e. 1500 bytes] with the IP and TCP headers” [TW11]. Additionally to the fields a UDP packet carries in its header, each TCP segment has a sequence number, a field for acknowledging received segments (piggy-backed) and additional flags and options to control the data flow (cf. figure 2.6).

As TCP is connection-oriented before communicating with each other two endpoints have to set up a connection. This is done by a three-way handshake: The starting host, say A, sends a segment with a SYN flag to the waiting host, say B. If B is accepting the connection it answers with a segment with SYN flag and acknowledges the SYN request. Finally A acknowledges this SYN and a connection is established. Closing a connection optimally is done by each host sending a FIN request and acknowledging each. In practice the connection will also be closed after sending a FIN and an exceeding timeout.

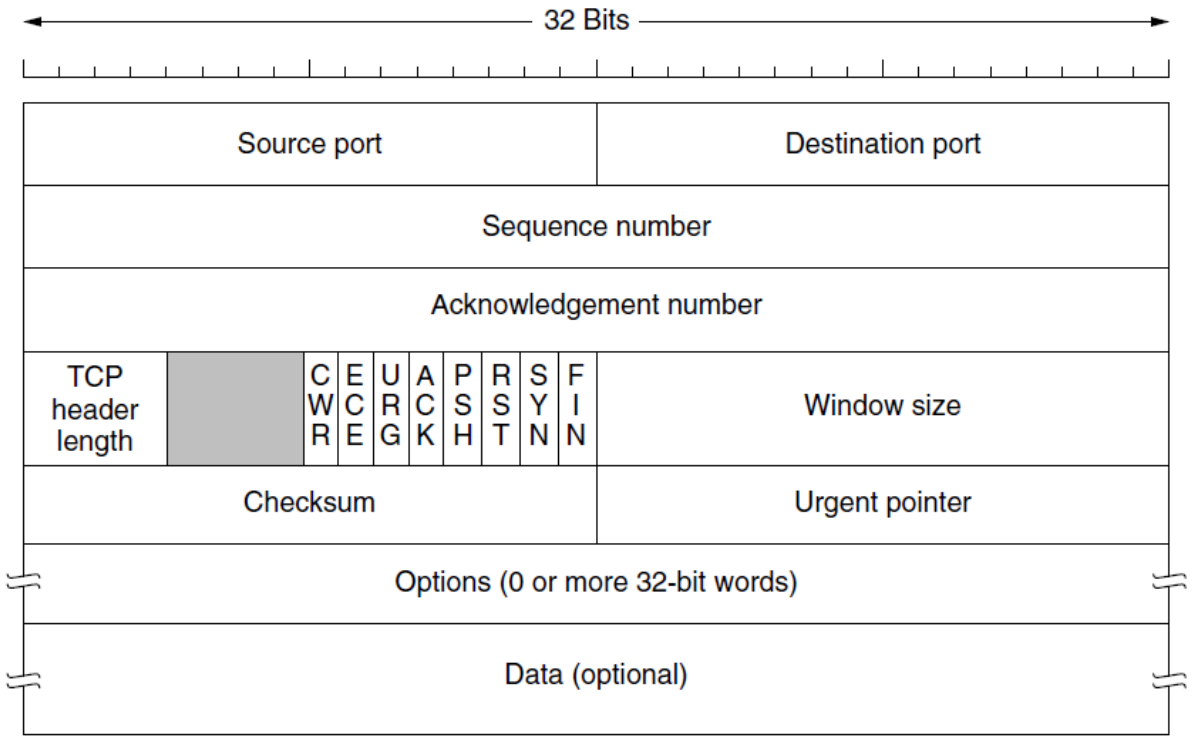

Figure 2.6: TCP header [TW11]

One of the key features of TCP is congestion control. If too many packets are send too quickly into a network, it will become congested. That means the network cannot handle all of the incoming packets and therefore it has to drop them. At this point it is important for the host to slow down sending to avoid that his packets get dropped. As the network does not inform the host about congestion (as additional messages would be counter productive for the network load), the host has to find it out by itself. TCP handles this by maintaining a congestion window. That means the rate of sending packets to the network is dictated by the congestion window which can increase and decrease according to the acknowledgments (and timed out acknowledgments respectively). In TCP this is performed by an algorithm called slow start. The core feature is that the initial window increases exponential to converge fast towards the right window size.

As [TW11] points out “using packet loss a a congestion signal depends on transmission errors being relatively rare[, which] is not normally the case for wireless links”. This is one reason why packet loss on wireless networks is handled separately and masked by the network layer as explained in the earlier sections covering the connection types.

3 Related Work

This chapter presents related work. First of all major work on LTE power characteristics are presented, followed by the description of various energy optimization approaches for wireless communication. At the end a novel approach for energy prediction on mobile devices is presented.

3.1 Performance and Power Characteristics 4G

A study containing many data sets (one of their experiments has been done with more than 3000 users) has been made in [HQG+12], which is also part of Junxian Huang's dissertation [Hua13]. The aim of the study is to examine performance and power characteristics of LTE networks. The study is build on different experiments:

1. Collection of 4G network performance via 4GTest app
2. Collection of user traces and performance test with 20 students equipped with a smartphone each
3. Simulations on UMICH data set
4. Various measurements on energy consumption

First of all they build an Android app called 4GTest to track network performance and compare it with other types of mobile networks. Regarding to [HQG+12] the app has been "attracting more than 3000 users within 2 month". The user can use the app to do a performance test for their connection, while the study can use these results to have a common view on LTE performance. Additionally they equipped 20 students with smartphones and measured their LTE usage and performed different remote performance test on these smartphones.

Also they ran simulations on a data set they collected for 3G and WiFi usage, which is called UMICH data set. It includes traces of 20 smartphone users over a time of 5 month. As all these measurements could not give information on the energy consumption and about the performance in some special scenarios, they did single experiments to measure

the energy consumption and for example measure the influence of mobility at different speeds.

The results also contain an application case study, in which different scenarios' energy consumption are estimated and measured. The estimations seem to be quite accurate. Unfortunately only the concrete results of case study estimations have been published, which include HTML rendering and JavaScript execution. The average error for those is promoted as 6%. Nevertheless their model does not consider the signal strength and it is not clear of the signal strength varied in this evaluation or is tried to be kept constant. Regarding the mobility measurements that have been performed, the signal strength varies across locations and has an influence on the performance. To avoid this they only select sample points with the similar signal strength for fair comparison. As mentioned in chapter 2.2 the signal has various effects on throughput and energy consumptions, which is not covered in that study.

3.2 Optimizing Energy Consumption

Besides studies on network performance and energy characteristics, there are many publications and approaches on optimizing the energy consumption on communication based on heuristics. Typically these approaches do not try to predict a concrete value, but only use energy measurements to validate the optimization's impact.

Bartendr [SNR+10] considers the relation of signal strength and energy consumption and wants to achieve energy efficiency by rescheduling communication to periods with strong signal strength. They focus on downloading information (mainly sync and streaming) and predicting the best point in time it should be performed to save as much energy as possible. Therefore signals are predicted based on previously measured signal tracks. That means it works best for well-known routes, in which the future signal is known.

Focusing on mobility and signal prediction is a good way, as signal changes might be crucial for the energy consumption. Also the approach of using the signal to predict throughput and power per time and therefore the overall energy for downloading data is well thought. Nevertheless this approach differs in the focus compared to this thesis. While Bartendr is focusing on scheduling the download of data, we focus on a precise prediction of RPC calls to compare it to predictions for local calculations later.

[DWC+13] worked on the relation between signal strength and energy consumptions in WiFi and 3G networks. They has made a field study with 3785 smartphones tracking the overall signal qualities to find out that low signal strengths appear quite often. Additionally they have measured and quantified the energy consumption resulted from

a low signal and showed that considering the signal is important for the modeling of energy consumptions for smartphones.

Finally they recommend delaying background traffic until a better signal is available to save energy and reason it with a simulation based on user traces as the measured a reduction of energy consumption of up to 23.7% for WiFi and 21.5% for 3G. Also they claim that their analysis reasons diagnostic cellular network services to improve the user experience.

Also they propose that this study allows *what-if analysis*. A similar approach is used by *TEMPUS* [NRCG15], which proposes Java annotations to wait for a more energy-efficient state before processing the next network communication calls. They also focus on timeliness to ensure that remote calls are not been made too late. As an example evaluation they claim that their approach can reduce the latency of downloading images by up to 10 times, while keeping the energy consumption constant, by only setting 5 annotations to the code.

[BBV09] measured the energy of communication in 2G, 3G and WiFi and designed a protocol called *TailEnder* to reduce energy consumption in applications. This is done by re-scheduling the traffic and performing pre-fetching of content to be more energy-efficient by exploiting tail times. Their evaluation shows that the approach can reduce energy consumption by up to 52% for a news feed use case.

Similar approaches like *TailTheft* [LZZ11] or [BDR13] also focus on exploiting the tail time as the tail time consumes the major part of the energy. Typically these approaches merge contextual information about a specific use case with the knowledge about tail times, to fulfill application specific requirements while sharing as much tail time as possible.

3.3 Energy Prediction Models for Mobile Devices

PowerTutor [ZTQ+10] created regression models for different hardware components of mobile devices and uses it together to make an overall energy prediction. The main aim is to support mobile developers in optimizing their apps to be more energy efficient. The innovative part of this approach it that the measurements are only using the internal battery voltage readings of a phone. Using these values the application can perform tests on different components to see how the voltage reacts and can create regression models based on these observations. Nevertheless the deviation of the model are quite high and the error rates per component vary up to 60%.

4 System Model & Problem Statement

In the following the system model is described including its system components, followed by the problem statement and requirements for the solution to the problem. Also system parameter are defined.

4.1 System Components

The system is composed of a mobile device with an application accessing a remote server over a wireless network.

4.1.1 Mobile Device

The mobile device runs applications that can do calculations, connect with other devices or the internet and interact with the user. These actions consume different amounts of energy, which is provided by a battery. The battery is limited. If the limit is reached, the device will of course stop working. Therefore the overall actions performed by applications are strictly limited.

4.1.2 Application

The application is running on the mobile device and can connect to a server on the internet using a wireless network to perform remote procedure calls, i.e. request data or any kind of calculation results from the server. Therefore it needs to create a request, transmit it to the server and wait for a reply by the server.

It is assumed that the application can estimate the size of request and response and the time it has to wait for the reply.

4.1.3 Remote Server

The remote server needs to be able to understand the request sent by the application, compute the result and transmit the result back to the application. In this model it is assumed that the server is always available over the internet and never fails.

4.1.4 Wireless Network

There are many types of different wireless networks, to which a mobile device can connect. The system model assumes that the device can connect to LTE or WiFi respectively and that both of the networks have a wired connection to the internet. Also it is assumed that the wireless network is the bottleneck of the communication in terms of throughput.

4.2 Problem Statement

The application is interested in the amount of energy a potential remote procedure call would consume. This value can be used by the application to make decisions about the request (e.g. whether it is more energy efficient to execute calculations itself or request a remote server to calculate the result).

To predict the energy consumption the application may only need information about the network, that can be retrieved from the operating system of the mobile device (i.e. network type and signal strength), and about the remote procedure call (i.e. sizes of request and reply, time it has to wait for an answer).

4.3 Prediction Model Requirements

Based on the system components and the problem statement the prediction model has to fulfill some requirements. First of all it should not require more parameter than those that can be estimated by the operating system of the mobile device and the application. The parameters considered are covered more detailed in section 4.4.

Furthermore the calculation of the prediction should consume very little energy and therefore has to be able to be computed fast and not use any kind of network communication or similar. The result of the prediction model should be a numeric value

estimating the consumed energy with little error. Additionally statistic intervals like a confidence interval would support improved decisions based on the prediction.

4.4 System Parameters

There are two kinds of parameter that can be retrieved and may influence the energy consumption. First there are parameters describing current properties of the network. These can typically be retrieved from the operating system of the device. Secondly parameters defining the planned communication need to be considered. These need to be estimated by the application.

The network is described by these parameters:

- **Type** defines to which type of cellular network the device is connected. We require it to be LTE or WiFi.
- **Signal Strength** defines the strength of the signal, which is typically defined as the power of signals retrieved by the base station.

The communication is described by the following parameters:

- **Size of request** is the amount of payload data, that needs to be send to the server.
- **Size of response** is the amount of payload data, that will be send from server to client.
- **Waiting time** denotes the time the client has to wait for the response of the server after submitting a request.

5 Measurements

Before creating a prediction model, data has to be collected to gather knowledge about the system. This chapter first presents parameters and quantities defining the measurements. Secondly the measurement setup is described regarding its components and their connections. Also the sample will be described to show the range of measurements that has been done followed by confounding variables. These show which kind of variables that cannot be controlled may have an influence on the measurements and how their influence can be reduced. Finally first observations will be presented.

5.1 Parameters & Quantities

To describe the different scenarios and track influences on the energy consumption, different parameter describing the in- and outputs for the later predictions need to be defined. These quantities are chosen as they may influence energy consumption and the time needed to transmit data.

5.1.1 Transport Protocol

An important question is, which transport protocol is to choose. The two typical transport protocols used for remote procedure calls are UDP and TCP. As described in chapter 2 UDP is more basic and only closes the gap between network layer and application layer with very little overhead. While TCP offers reliability and ordering, it adds additional overhead.

Although UDP has no additional transmissions like TCP, which could influence the energy measurements, TCP first of all offers congestion control and secondly is more often used in practice, when it comes to network communication with data sizes in the order of MB. This is the reason, why all measurements use TCP as transport layer protocol.

5.1.2 Communication Pattern

The combination of request, response and the waiting time in between is denoted as *communication pattern* in this thesis. As different amount of data will lead to different sending times, the number of packets sent and their length is important to know. It will force the network device to stay in the sending mode for a longer or smaller time with respect to the size. The same applies for responses to the sent package. Due to energy savings by exploiting the tail-time, the time between request and response is also very important.

The values for the size of data used for the measurements will cover some small amounts of data as well as some bigger amounts for both – requests and responses equally: 256 Byte, 512 Byte, 1024 Byte, 0.5MB, 1MB, 2MB.

The waiting time between request and response is $10s$ for WiFi and $40s$ for LTE. This will ensure that the energy consumption of the different phases can be measured separately without an interception or shorting of the tail time. Requests and responses contain random byte-Arrays of the specified length streamed via TCP.

It is important to say that the waiting time does neither include the latency L or round trip time ($RTT = 2 \cdot L$) nor it includes the time needed to receive all data sent ($size \cdot bandwidth$), but is the time the server waits from receiving the byte array completely to sending the byte array for the response. This is because the estimation of the latency would require another network communication, which would increase the overall energy consumption to much.

To measure the communication of patterns, a pattern protocol has been developed. This protocol can be used to trigger nearly any kind of communication between a client and server. The only exceptions are that a single message has a minimum length of at least 12 Bytes (depending on the pattern) and the client has to start the communication (as it fully controls timing and sizing of packets).

The communication from client to server is simple to trigger: The client can just send an arbitrary long random byte array, but the server should also answer these packages. Depending on the use case it may send more than one message back to the client (maybe the server already wants to inform the client about partial results before finally sending the result). This means the server should know when to send a packet of a certain size. This is solved by the pattern protocol.

Each pattern has a distinct pattern ID. This has two reasons: First it helps the client to match the server responses to the initial request and second it allows the server to distinguish different request. A request (identified by the ID) is only processed once by

first byte	length [Byte]	datatype	content
0	8	number	pattern ID
8	4	number	number n of responses
$12 + i \cdot 8$	4	number	response i waiting time
$16 + i \cdot 8$	4	number	response i size in byte
$12 + n \cdot 8$	$l - (12 + n \cdot 8)$		random stuff bytes

Table 5.1: Structure of pattern request (l: size of request in byte)

the server. If multiple requests with the same ID arrive at the server all except the first one will be just dropped.

In table 5.1 there is an overview of the structure of a pattern request as it is typically send by the client. Besides the pattern ID the client sends the number n of responses. Size and timing of each pattern are described by the following bytes: Each response i has a specific size in Bytes the packet should have and a waiting time in ms that should be waited by the server before answering with the response. Followed by this the request can contain arbitrary random stuff Bytes to fill up to the size the request should have.

Figure 5.1 shows an example for a communication with the pattern protocol. The initial request by the client has the ID `420815` and requests two responses. The first should be send after 1000ms with a size of 1024 Byte and the second after additional 500ms with a size of 2048 Byte. After the inital request the client sends two additional request with the same ID. That means the requests will be received but not processed by the server. 1000ms after the first initial request the server answers with a response of size 1024 Byte and the identical ID. 500ms after sending the previous response the second response is send with a size of 2048 Byte and again the same ID.

The pattern protocol can be used with UDP or TCP. In case of UDP the packets should be split up into smaller packets (e.g. to a size of 512 Bytes each). The only important thing is that each packet should contain the ID and the information about the responses as UDP does not ensure any order. In case of TCP the whole packet is packed into a byte array. Additionally the size of the array is transmitted before sending the array to allow the server to detect the end of the array. This means a transmission of 4 additional Bytes, but this small overhead is acceptable for the use case of this thesis.

When using the pattern protocol one also has to be aware of the fact that the sizes are always the sizes for pure application layer data. The underlying protocols typically add additional information. If there is the need for the exact size of data transmitted, this has be considered and calculated. Also in case of TCP there are additional messages like acknowledges to ensure reliability and possibly retransmissions.

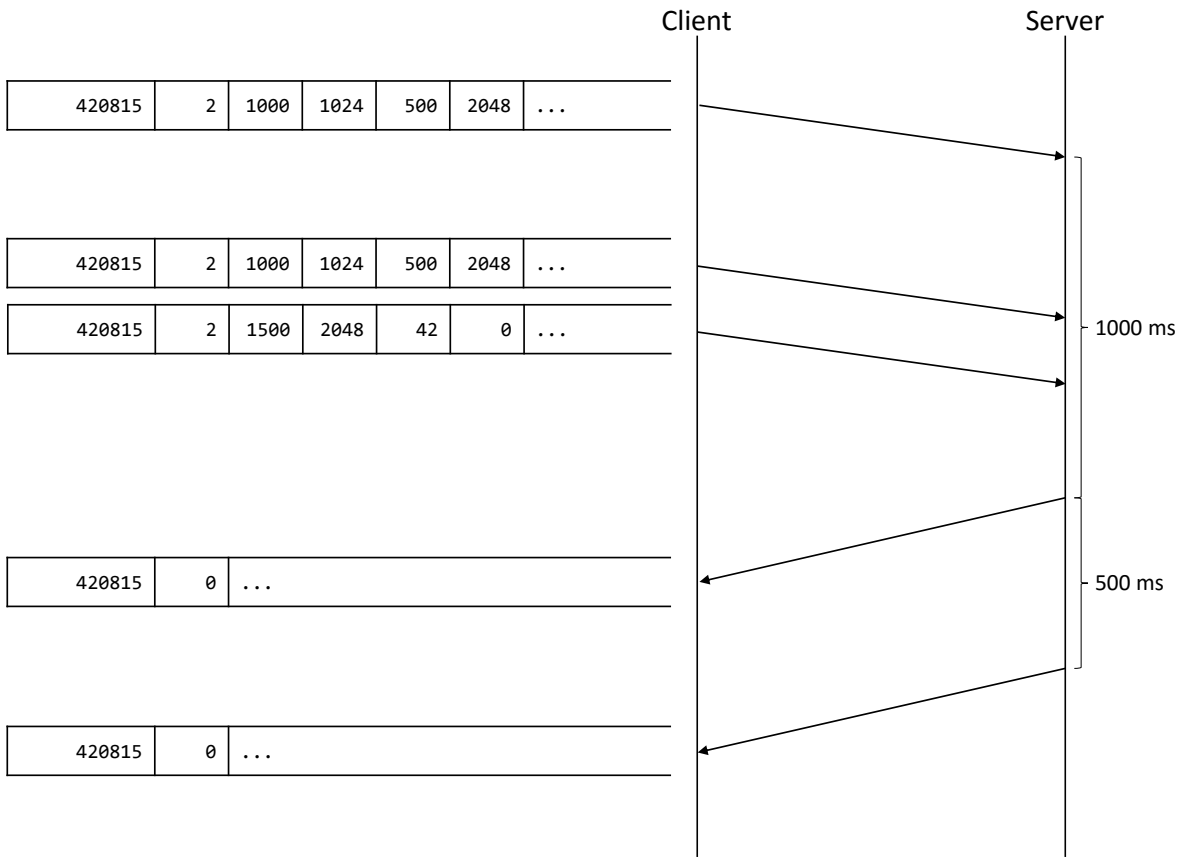

Figure 5.1: Example of pattern

To sum it up the pattern protocol is an application layer protocol that allows realistic communication between a client and a server initiated by the client. The communication pattern is an important parameter for the behavior of a network.

5.1.3 Connection Type

As described in chapter 2, different connection types have different energy states with their own characteristic transitions and energy consumptions. Therefore it is of course necessary to have different connection types as a parameter. Also the connection type includes different data rates, which means that transmissions of requests and responses take more time if the rates are lower, which will affect the overall energy consumption. The measurements are performed using LTE or WiFi connection respectively.

5.1.4 Signal Strength

As described in chapter 2 the signal strength has an influence at least on throughput as the network adapts it to the signal strength. Therefore it is necessary to track the signal strength and vary it, when taking measurements.

The signal strength can also be an indicator for packet loss resulting in TCP retransmissions and therefore a smaller throughput of the payload traffic.

5.1.5 Energy Consumption

As the energy consumption is the value to be predicted later, it is the dependent variable. The energy consumption is measured using a raspberry pi with rpi-powermeter as shown in section 5.2 Measurement Setup.

5.1.6 Time

Additionally to the energy consumption the time needed for the transmissions and the tail times can be extracted with the combination of timestamps in the log and the analysis of energy consumption. By having the energy and time besides the size of packets other variables like throughput can easily be calculated.

5.2 Measurement Setup

The setup for the measurements consists of the mobile device, the measurement equipment and the network infrastructure. An overview over the components can be seen in figure 5.2.

5.2.1 Mobile Device

The mobile device used for the measurements is a Samsung Galaxy Note 4. It has a removable battery, which enables easier measurements of the energy consumption. Also the smartphone supports LTE. When performing the measurements the device was running with the Samsung Version of Android 6.0.1.

The only apps used on the mobile device are NetGuard (cf. section 5.4.2) and NetworkStatusLogger. The implementation of the NetworkStatusLogger, which communicates

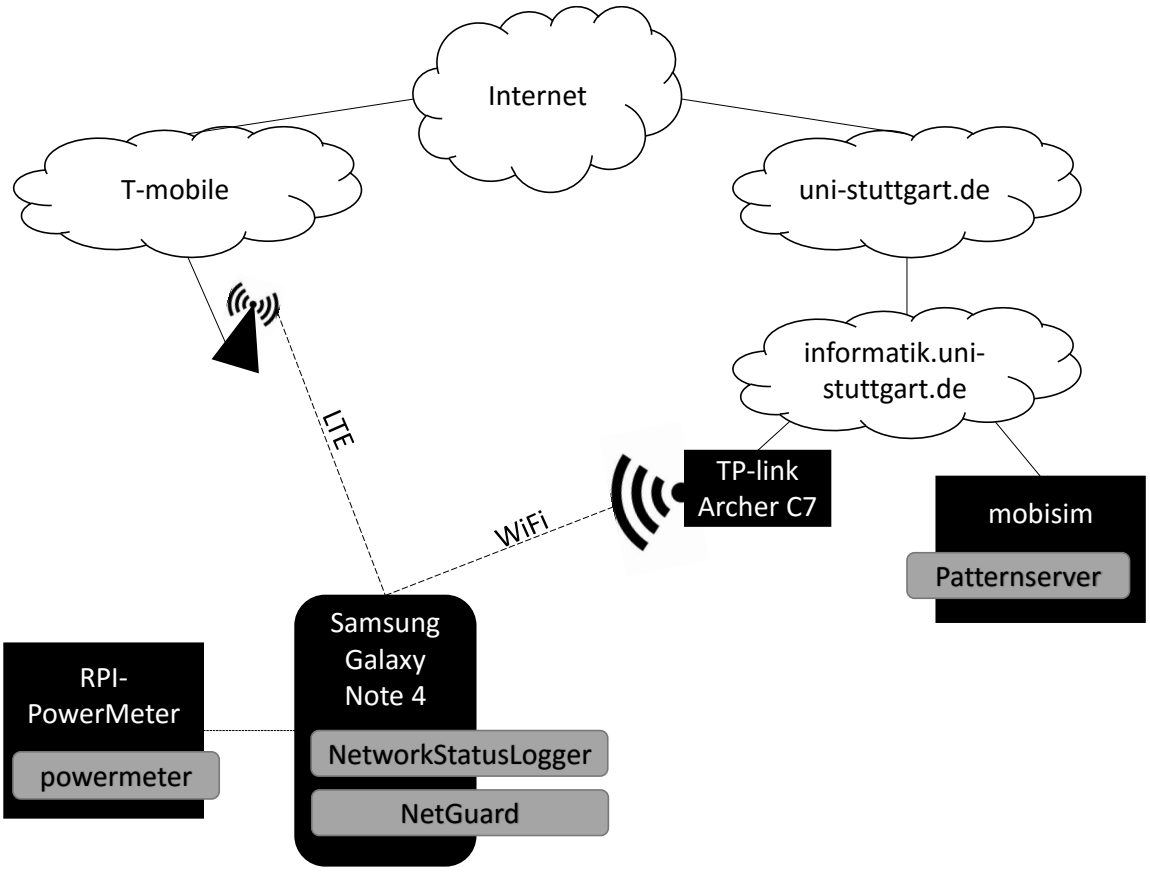

Figure 5.2: Overview of measurement setup

with the server and logs any action performed by it as well as the network information, can be found in detail in chapter 7. One set of measurements consists of six patterns with a different size communicating with a server and repeating the whole measurement four times, resulting in 24 pattern measurements.

During the measurements the mobile device has not been moved to keep the signal as constant as possible. The screen has been turned off due to high and fluctuating energy consumptions of the display.

5.2.2 RPi-Powermeter

To measure the energy consumption a tool is needed that can provide high sampling frequencies while being mobile. The RPi-Powermeter [Dür16] is an open source tool (hardware and software parts) that can serve these requirements.

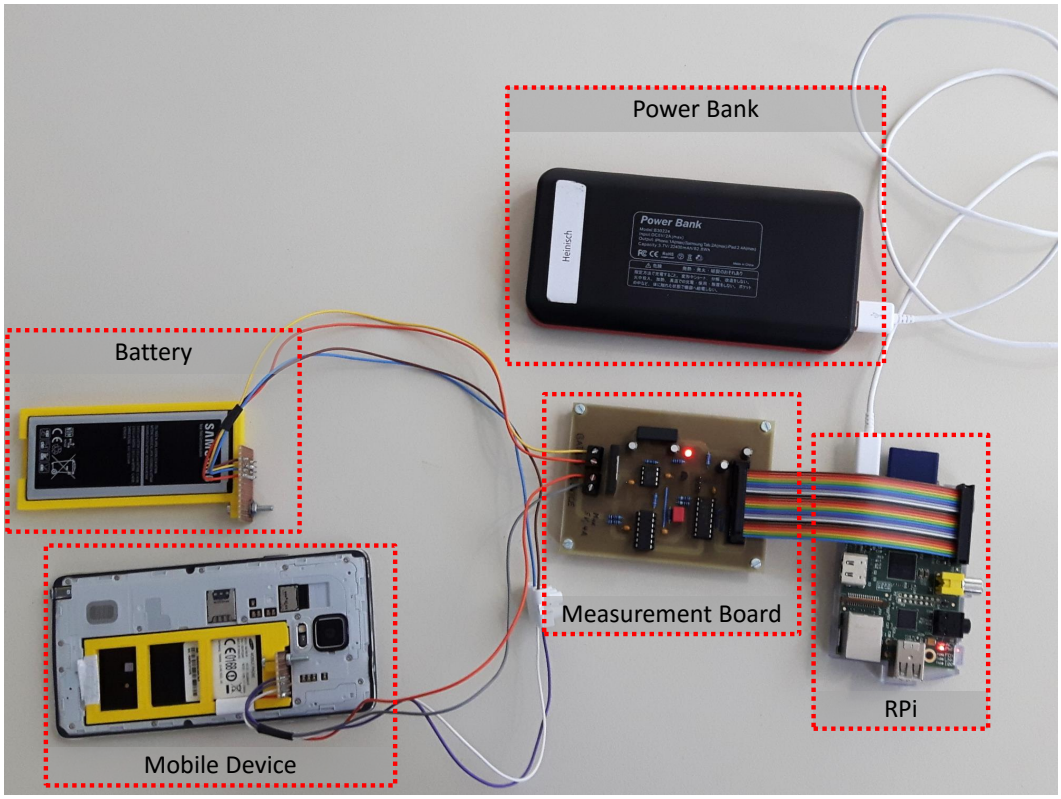

Figure 5.3: RPi-Powermeter setup

The hardware part consists of a measurement board that can be connected to a Raspberry Pi. The measurement board uses a Shunt resistor with $4.7ohm$ resistance. In the setup of this thesis a *Power Bank* with an output of $2.4A$ has been used to power the Raspberry Pi to enable mobile measurements (cf. figure 5.3).

The software of the RPi-Powermeter is running on the Raspberry Pi, which has a realtime extension (RT preempt patch)[1] installed to enable high sampling frequencies ($1kHz$) while writing the log to the SD-card. The software calculating the actual current has been calibrated to be used with the Samsung Galaxy Note 4.

5.2.3 Network Infrastructure

The network infrastructure can be divided in two parts - the wired part and the wireless part. While the wired part is needed to connect different networks with the server

[1] https://rt.wiki.kernel.org/index.php/Main_Page

responsible for the responds, the wireless part is much more interesting for the energy measurement as this is the part in which the mobile device communicates directly.

The server is connected to the computer science network of the University of Stuttgart. The computer science network is connected to the Internet via the network of the university. The firewall is configured such that the server application can be accessed from the internet.

For the wireless connection of the mobile device two ways have been chosen, which are applied distinct:

1 - WiFi: For the WiFi connection an own access point (TP-link Archer C7) has been used for connecting the device to the computer science network to avoid additional traffic on the wireless link. The access point is sending with 2.5GHz frequency and on channel 11.

2 - LTE: For the LTE connection a T-mobile SIM is used to enable a connection to the internet.

5.3 Sample

The sample consists of 288 measurements in sum. 168 measurements have been taken for WiFi and 120 measurements have been taken for LTE. The distribution of the size and signal values of the measurements is visualized in figure 5.4. In the visualization the measurements for 256B, 512B and 1024B are counted together as they are too close to visualize each.

In WiFi seven experiments each consisting of 24 measurements (i.e. four measurements per size) have been run. The detected signal stayed constant during each experiment. The overall range of measured signal strengths is from -86dBm to -13dBm.

In LTE five experiments also consisting of 24 measurements each have been run. The signal strength per experiment had a maximum range of 3.9dBm. The overall range of measured signal strengths is from 107.8dBm to -76dBm. According to section 2.2 this covers a good to medium classification of signal strengths. Nevertheless the connection is typically getting lost, when the signal gets worse, which means signal strengths classified as poor cannot be measured.

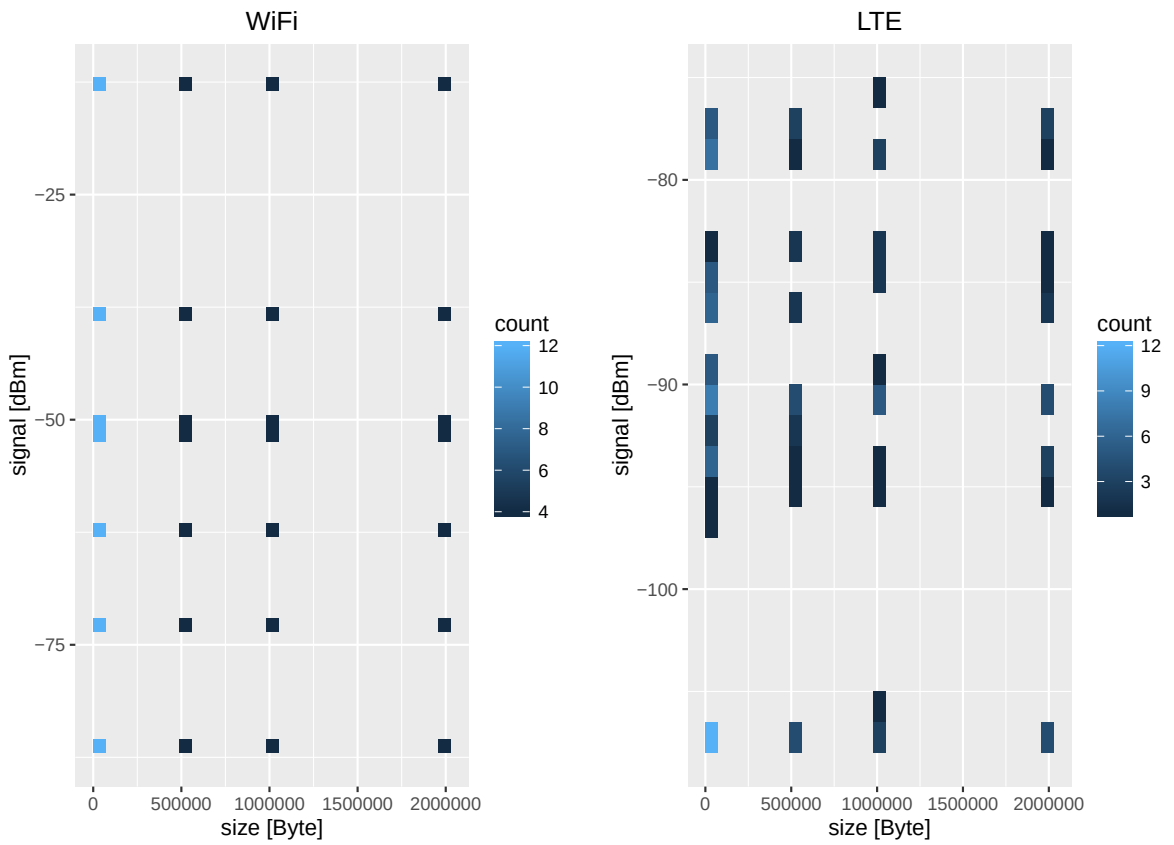

Figure 5.4: Distribution of measurement samples

5.4 Confounding Variables

While some important parameters can be controlled, there are some variables that may influence the measurements. These are called confounding variables and are explained in this section. Despite their explanation and the explanation of their impact on the measurements, also solutions to avoid or reduce them as used for the measurements are explained.

5.4.1 Resource Usages by other Applications

To have a measurement that measures the energy consumption of network communication as exact as possible, it is important to prevent all other resource usages that consume energy. This is only acceptable, if the consumption produces a regular offset, like the android system does in sleep mode.

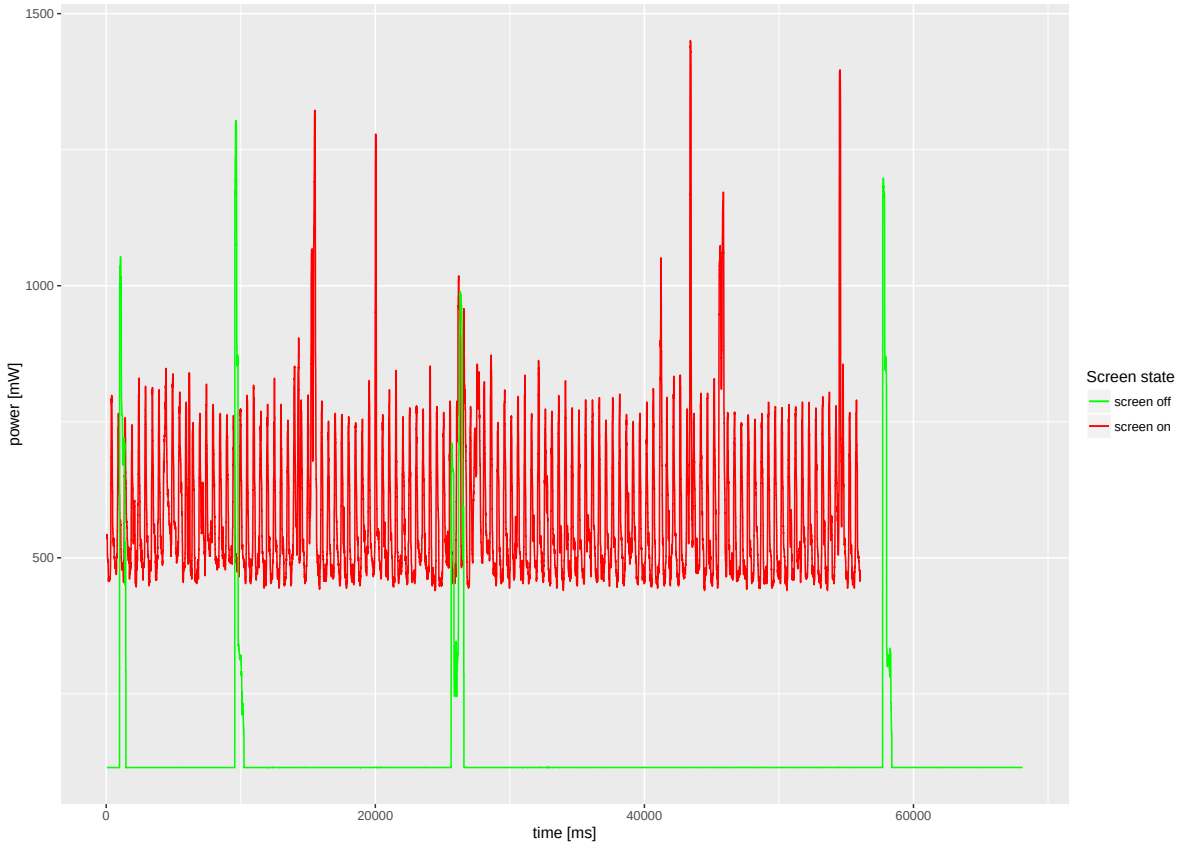

Figure 5.5: Screen off vs. screen on (median-filter with window size 100 applied)

To prevent other apps (installed by the manufacturer) from using any resources, the sleep mode is used, which is activated after locking the screen and waiting for 30 seconds. To be able to log events with our app there is the need to acquire a `WakeLock`[2] for the Service of the NetworkStatusLogger application. Using the sleep mode also prevents the display from consuming energy and results in much less energy consumption with low variance as one can see in figure 5.5. With the screen turned off the median energy is $114.69mW$, while the median energy with screen turned on is $536.66mW$.

Additionally to exploiting Android's sleep-mode other resources, like GPS, Bluetooth and also cellular networks (for measurements in WiFi) or WiFi (for measurements in LTE) have been disabled.

[2]https://developer.android.com/training/scheduling/wakelock.html

5.4.2 Network Device Usage by other Applications

When other applications use the network at the same time, they influence the behavior of the energy states and can lead to wrong measurements, e.g. a tail-time that seems to be much longer than usual caused by a small packet (for example a ping) transmitted during the usual tail-time. Another point is, that while calculating the energy consumption of sending data, it is hard to distinguish between parallel or sequential sending-phases performed by other applications.

To avoid such effects other network traffic except for those caused by the system and the measurement application is dropped using NetGuard [Bok16]. NetGuard is a firewall for android devices. It is implemented as a local VPN server running inside the NetGuard app. It can filter the outgoing traffic for each installed app. That means we are able to stop any outgoing traffic. Though it is necessary to at least allow the Android system to use the network, as otherwise it will assume the network is broken and might eventually close the connection to it completely to save energy.

However in Android's sleep mode the communication of the system is only very infrequent. The TP-Link Router's UI can be used to check the traffic of connected devices. To check how many packets are still transmitted by the system, the device has been locked and the packets tracked for 5 minutes. This resulted in 20 packets with a complete size of $3258Bytes$ (on average $162.9Bytes/packet$). Most of the packets have been transmitted during the first 30 seconds (before the system enters the sleep mode) and appeared in groups of packets (often 4 packets at a time). This behavior can also be observed in the screen-off scenario in figure 5.5. That means that there will be only very few disturbances while measuring in sleep mode with NetGuard blocking all traffic except for the system and the NetworkLogger-App as we can at most expect one small transmission each minute.

Even though NetGuard requires a WakeLock-permission, it appears to be very energy efficient. The median energy-backoff with NetGuard running is has been measured as $111.95mW$ ($sd = 15.87$), while the median energy-backoff without NetGuard running is $110.93mW$ ($sd = 16.04$). So in sum there is only a small offset produced by NetGuard constantly running in the background.

5.4.3 Network Usage by other Clients

To prevent network interferences with other network-clients when measuring on the wifi network, a separate access-point with a dedicated SSID has been set up. Also a non-overlapping channel to other access-points (channels 1 and 6) in the area has been chosen for the separate access-point (channel 11). "Non-overlapping" means that there

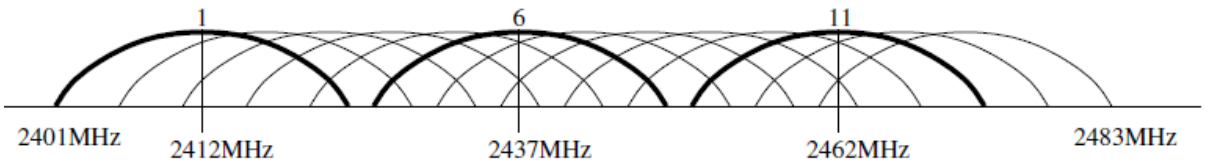

Figure 5.6: IEEE 802.11b/g channelization scheme [FVR07]

is no overlap in the frequencies of the channels (cf. channelization scheme in figure 5.6). Even though [FVR07] shows that also wireless networks with non-overlapping channels may interfere due to the “near-far problem”, we can assume that there is very little interference as the mobile device has been placed in a safe distance to other active wireless network devices.

For the LTE measurements the interference with other users can not be controlled as the network is not under control of the author. Though the measurements have been repeated many times to reduce the overall influence of the traffic generated by other users.

5.4.4 Latency

The latency between client and server has influence on the time between sending and receiving data. The latency is not measured explicitly in the setup, but can be calculated by calculating the additional time between sending and receiving (RTT). Nevertheless it is not necessary to control it as the time between sending and receiving are chosen such that it does not influence each others phases in terms of tail time thefts.

Also for simplification an explicit measure of the latency is avoided as this measure itself would be an additional network communication and would lead to energy consumptions and tail times.

5.4.5 Loss of Data

As the network layer handles most of the appearing loss of data by FEC and retransmissions and losses due to congestions are covered by TCP, loss of data is not measurable in the application layer. Typically data loss occurs due to a low signal. In this cases the retransmissions will influence the time and energy that is needed to send or receive the data. This means the influence of the signal on the loss of data has a transitional relation on time and energy consumption.

In sum the loss of data is not measurable explicit, but influences the dependent variables. As these effects will also appear in the later submissions, for which the energy consumption should be estimated, theses influences might even improve the quality of measurements.

5.4.6 Inexact Signal Strength

Another risk is that measurements of signal strengths are not exact enough. Here we need to rely on the mobile device's operating system to return exact values. In case of LTE the values retrieved have one decimal digit and are changing fast on movement. For WiFi only values could be retrieved that do not have any decimal digits and the values typically only change when the difference is high. This means that inexact signal strengths can influence the exactness of data tuples and influence the quality of models built on the basis of it.

5.5 Observations

In this section the measured data is presented and first observations are made.

5.5.1 Pattern

As each measurement needs to be analyzed on its own, we first have a look at how the typical energy consumption of a pattern looks like. Figure 5.7 shows an example of a single pattern that is extracted by the algorithm described in section 7.3. The red lines symbolize the points in time where the network device is switching its state, while the gray vertical lines show events in the log (end of receive, start of send, start of pattern, signal change, ...). Details on the logging events can be found in section 7.1.

The typical pattern can be divided in four phases:

1. **Sending:** The phase in which the client is actively sending a request to the server.
2. **Tail Time after Sending:** The phase following the sending phase, where the mobile device is still in high power state (cf. chapters 2.1 and 2.2).
3. **Receiving:** The phase in which the client actively receives the response of the server.

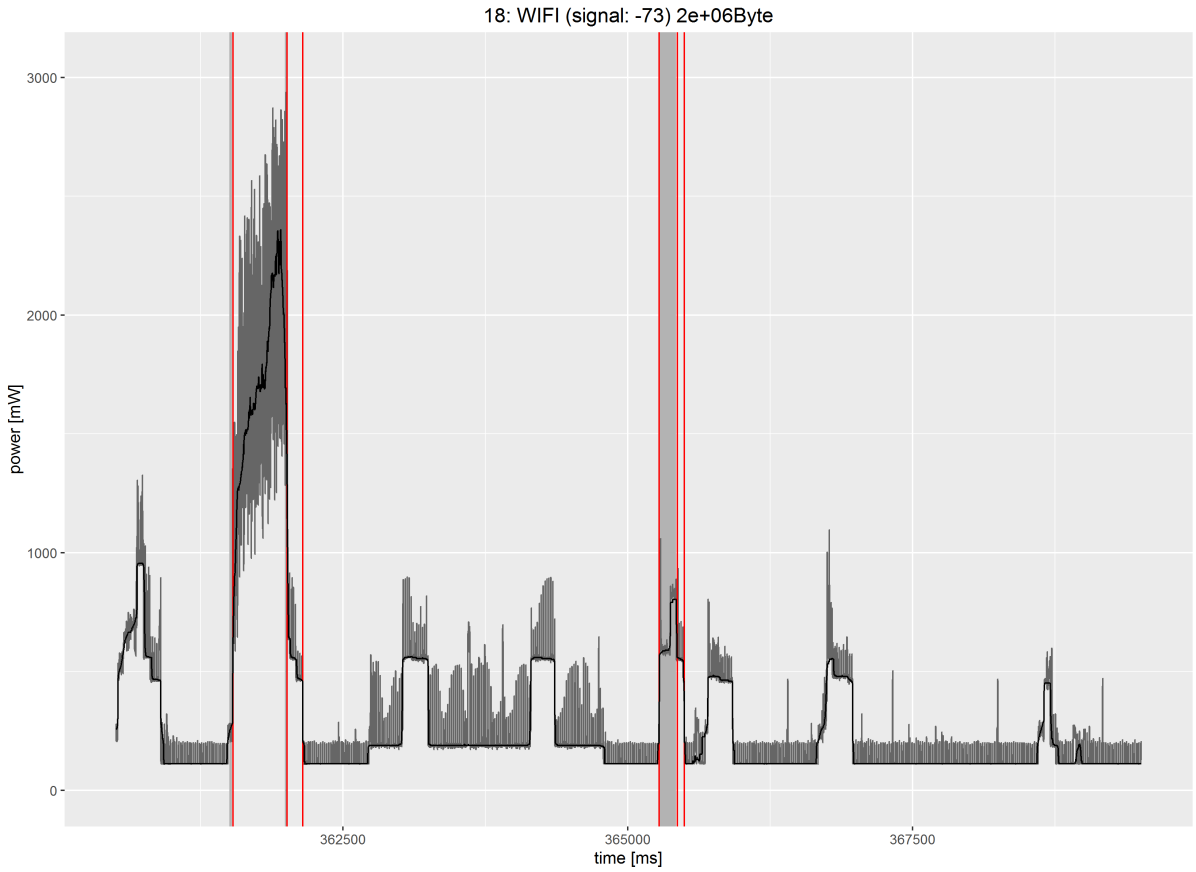

Figure 5.7: Example of a typical automatically extracted pattern

4. **Tail Time after Receiving:** The phase following the receiving phase, where the mobile device is still in high power state.

Though the automated extraction works good in most cases, one still has to sort out some of the calculated data, as there are sometimes distractions to the usual pattern (for example because of another app sending or receiving data over the network).

With the help of the automated extraction it is possible to extract the following data per pattern:

- **size**: Size of request and response in bytes

- **type**: Connection type of the mobile device (MOBILELTE, WIFI)

- **signal**: Signal strength of the connection

- **time**: Time is calculated for each phase (in ms)

- **energy**: Energy is integrated in each phase (in J)

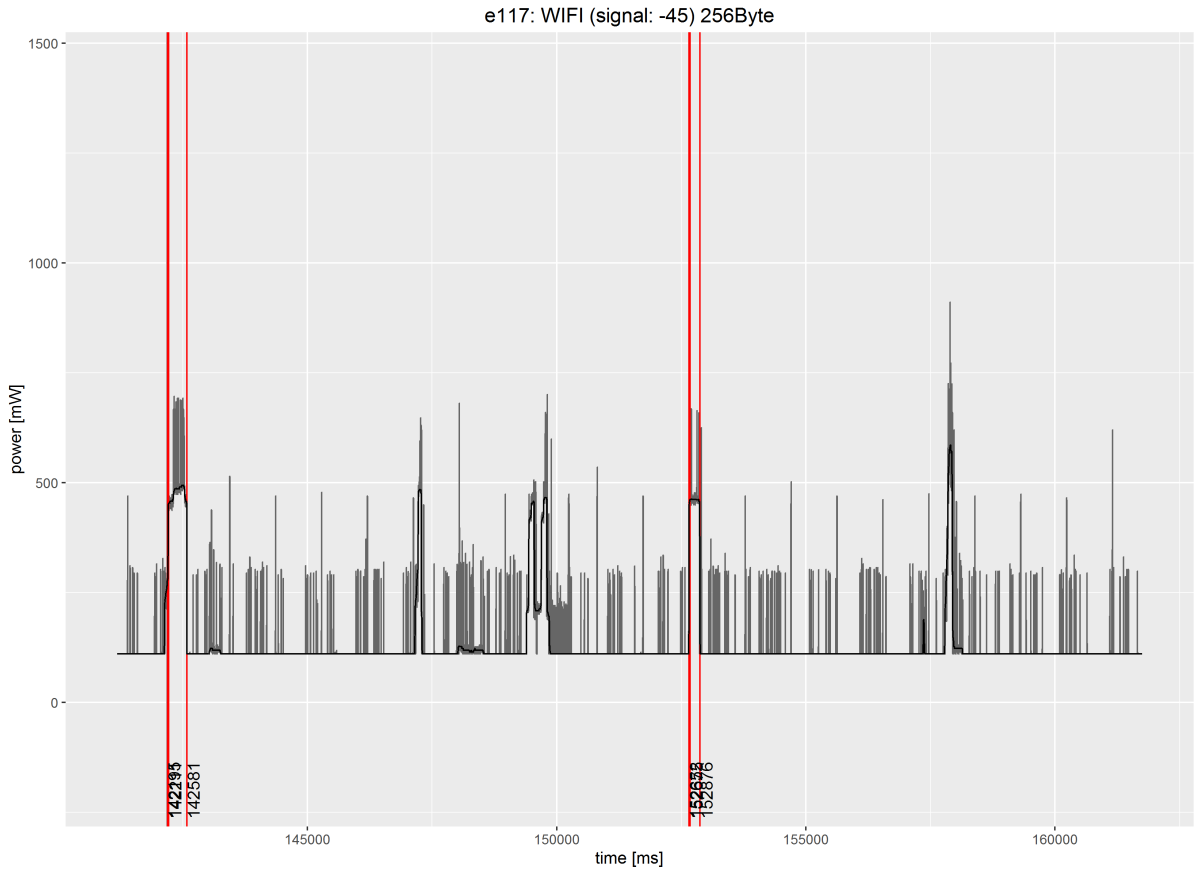

Figure 5.8: Example of a small-sized extracted pattern

- **rtt**: RTT (Round-Trip Time) is the time a packet needs to go from the sender to the receiver and back again ($2 \cdot latency$). It is calculated at the client by calculating the difference between the first send and the first receive subtracted by the waiting time.

Extracting the information from pattern with a small size (i.e. $\leq 1024B$) is very challenging as the sending can hardly be distinguished from the tail time as depicted in figure 5.8.

Size, type and signal are expected to correlate to time and energy of the sending and receiving phases. Nevertheless to handle correlations in a more logical way, time and energy have been recalculated to fractions, that are more connected to the characteristics of the network. For sending and receiving it is calculated to energy per time and throughput (size per time). For the tail times only energy is converted to energy per time as throughput would not make any sense. In that way no information are lost as time and energy can be recalculated with the help of the size and at the same time more relevant information in terms of the network characteristics are provided.

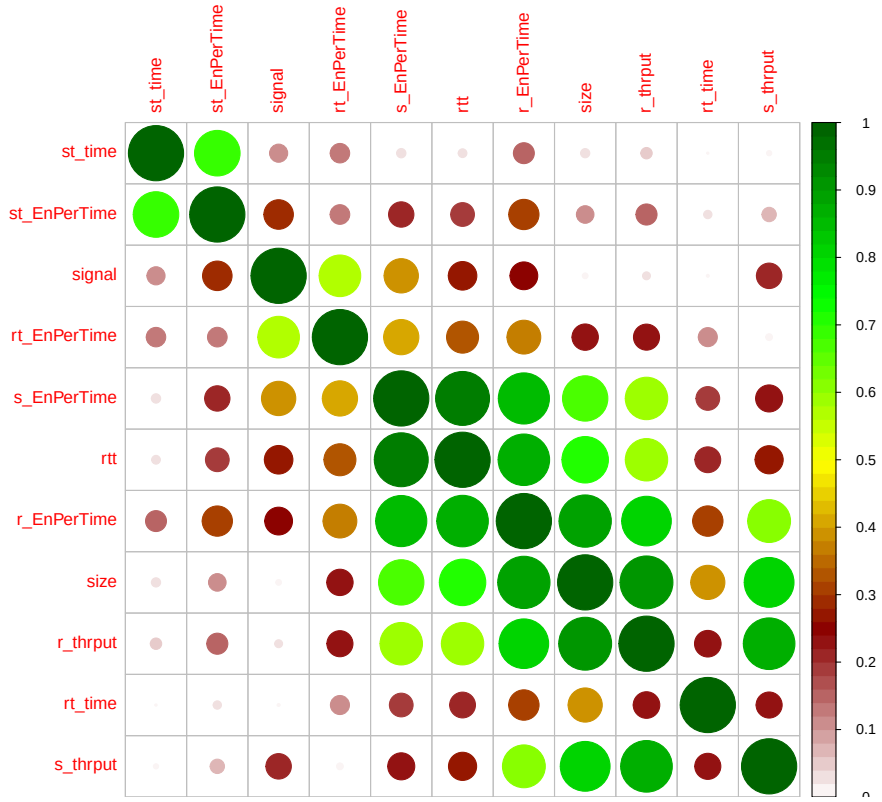

Figure 5.9: LTE correlations

The recalculations are done like in these equations:

$$energypertime = \frac{energy}{time}$$

$$througput = \frac{size}{time}$$

5.5.2 Correlations LTE

Based on the recalculated variables correlations can be analyzed. First a correlation matrix will be calculated and evaluated. Second each variable will be sighted in the context of the two independent variables size and signal and the previously observed correlations.

Figure 5.9 depicts the correlation matrix for the variables from the LTE measurements. For the visualization the absolute value of the correlation has been taken. Though one can already observe several correlations, some of the values are misleading. Due to the

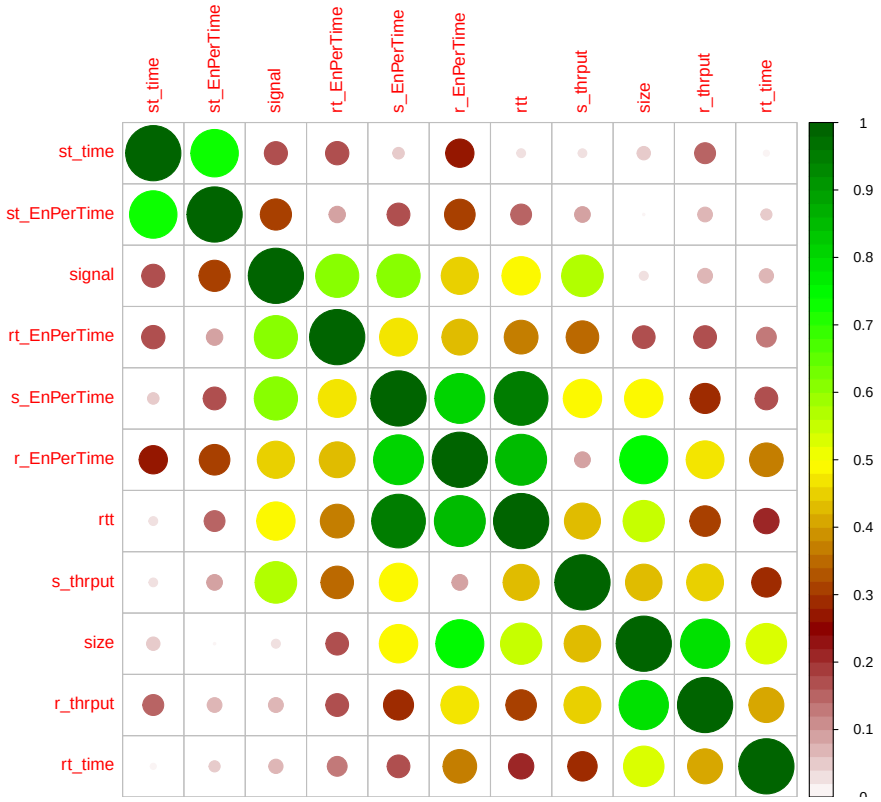

Figure 5.10: LTE correlations for bigger packet sizes

very small differences of the packet size in the lower segment the dependent variables are similar and the pattern extraction is sometimes not accurate enough to measure all differences exactly as shown in section 5.4.6. Also the time of communication might be too small to observe the influence of the signal.

Therefore another correlation matrix only containing the measurements with packet sizes of the upper segment (0.5MB, 1MB, 2MB) can be seen in figure 5.10. Looking at the second matrix we can see more correlations especially for the signal. For every energy per time variable there is a correlation (send: -0.62; send tail: -0.32; receive: -0.46; receive tail: -0.61; each $p < 0.02$) with the signal. Also there is a strong correlation of the signal to the send throughput ($0.55; p < 0.01$) as well as a medium correlation to the RTT ($-0.49; p < 0.01$).

Also for the size there is a correlation to the send throughput ($0.44; p < 0.01$) and the RTT ($0.56; p < 0.01$) as well as correlations to the send and receive energy per time (send: $0.49p < 0.01$; receive: $0.76p < 0.01$). Additionally there are strong correlations to the receive tail time ($0.54; p < 0.01$) and the receive throughput ($0.80; p < 0.01$).

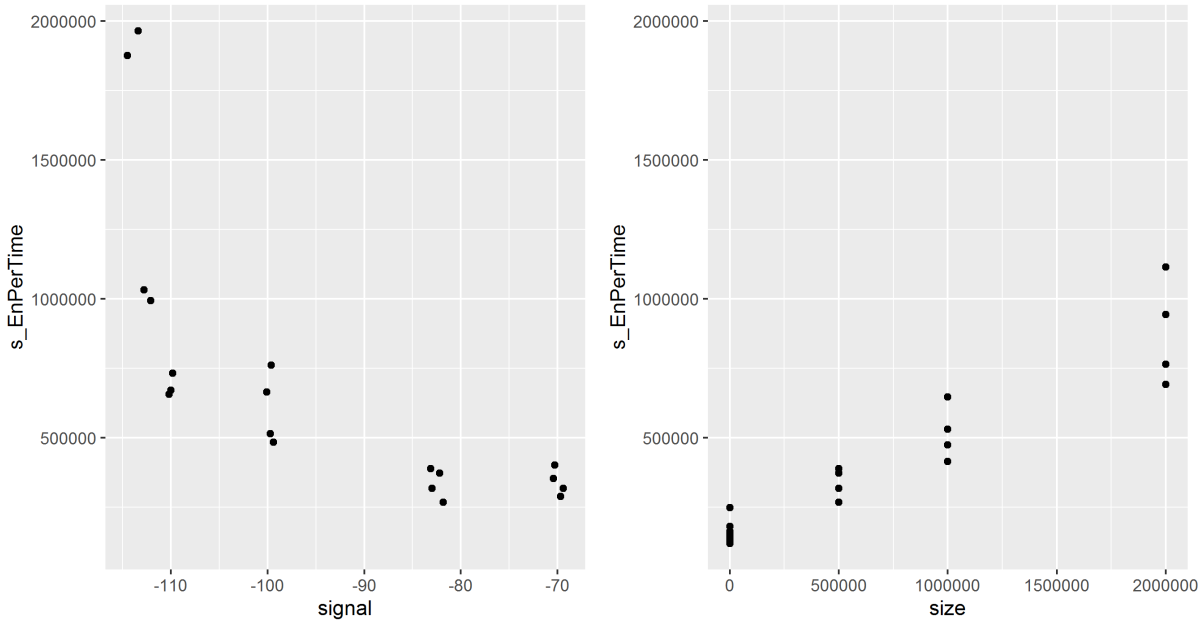

Figure 5.11: Energy per time [mJ/s] while sending
(left: $size = 0.5MB$; right: $-90 < signal \leq -80$)

After having a quick and broader view on the correlations now the focus is on each variable's correlation to size and signal.

Energy Per Time

Energy per time is the overall energy consumed by one part of the communication divided by the time. This shows the relation of energy consumption without considering the time the communication takes place or the tail time lasts. The correlation matrix already shows that there are correlations to the signal, but also to the packet size for the send and receive time. Having a closer look these correlations can be clearly seen in the data.

The figures 5.11 and 5.12 depict the energy per time for the sending and the receiving part. In both there is a correlation observable. The signal has a negative correlation, while the size shows a positive linear correlation bursting with increasing packet size. The correlations to the signal is exponential, which is not surprising as the signal is given in dBm. Comparing both phases, send and receive, one can observe that the energy consumption for sending data is on average much higher than for receiving data.

Figure 5.13 shows the energy per time during both tail times. The energy consumption in this case is higher for the receive tail time than for the send tail time. In both cases the

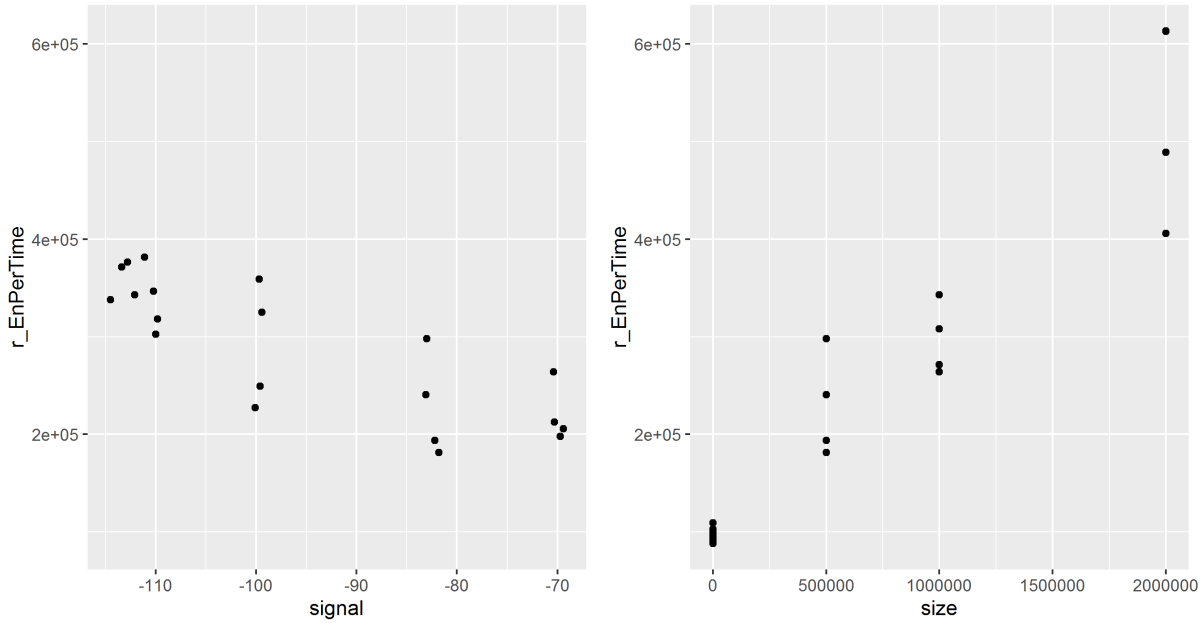

Figure 5.12: Energy per time [mJ/s] while receiving
(left: $size = 0.5MB$; right: $-90 < signal \leq -80$)

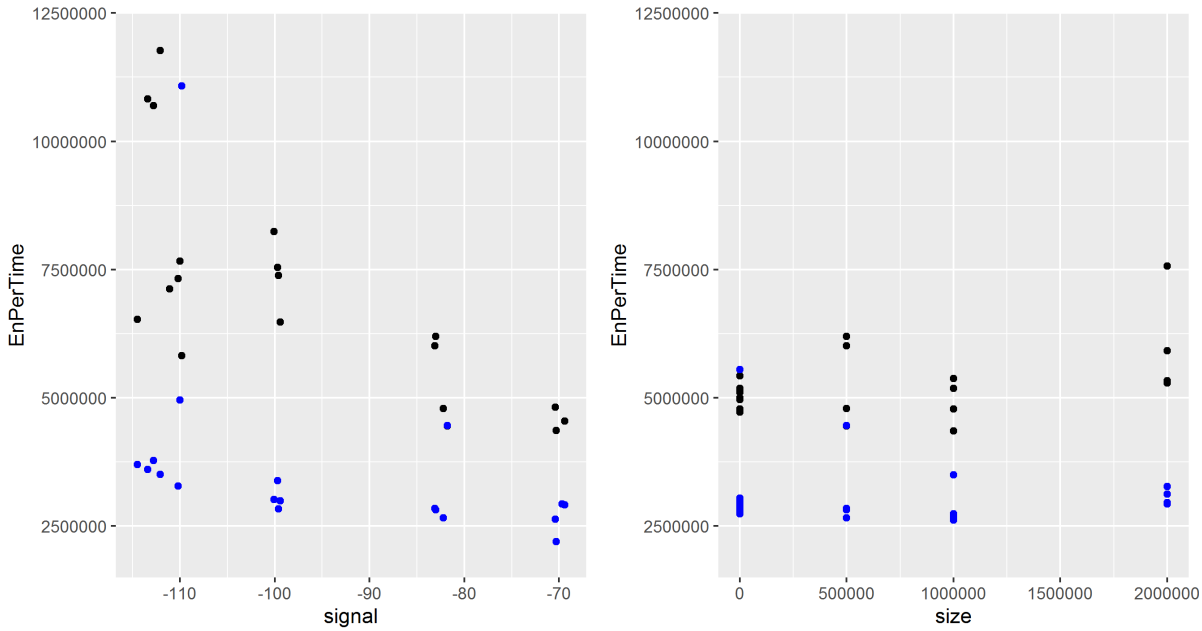

Figure 5.13: Energy per time [mJ/s] during tail time
(left: $size = 0.5MB$; right: $-90 < signal \leq -80$; black: receive; blue: send)

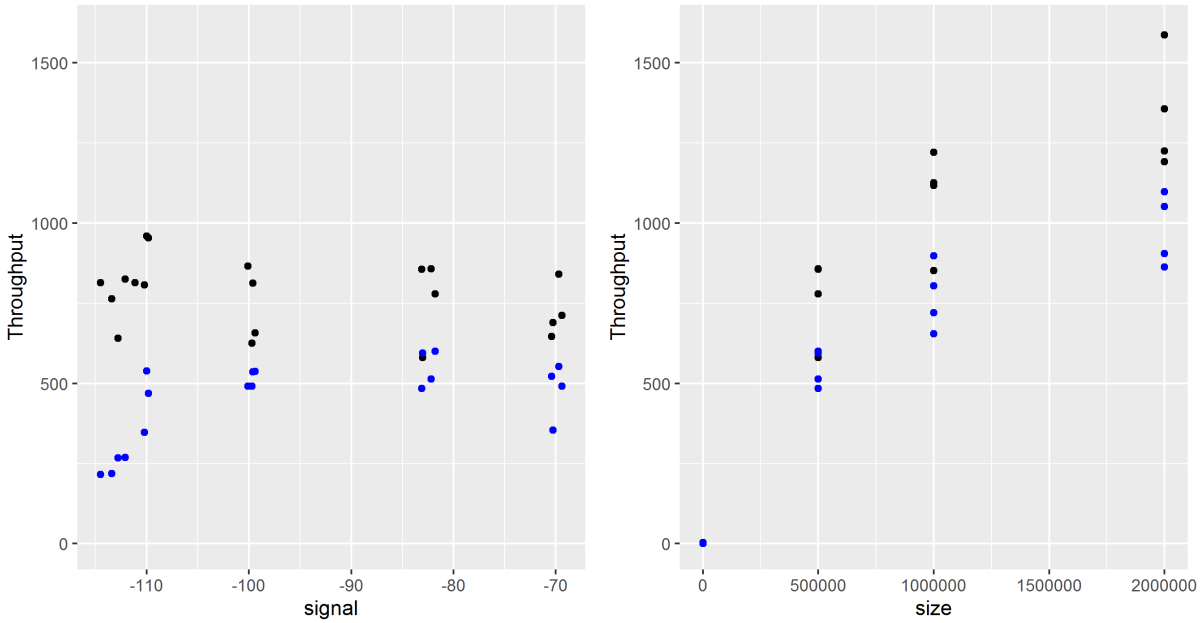

Figure 5.14: Throughput [kB/s]
(left: $size = 0.5MB$; right: $-90 < signal \leq -80$; black: receive; blue: send)

energy per time is constant when varying the size. The plot shows a negative correlation to the signal even though it is much more obvious for the receiving phase, which also has been expressed by the correlation matrix before.

Throughput

Throughput is the amount of traffic, that can be transfered in the network per time unit. Goal of taking throughput into account is to be able to calculate the time based on the packet size. However in the observed data as depicted in figure 5.14 the throughput increases with increasing size.

Observing the throughput in comparison to the signal, it is constant with an exception for a very low signal where it drops in the sending case.

5.5.3 Correlations WiFi

Analogously to the previous section in this section the correlations for the measurements in the WiFi network are observed.

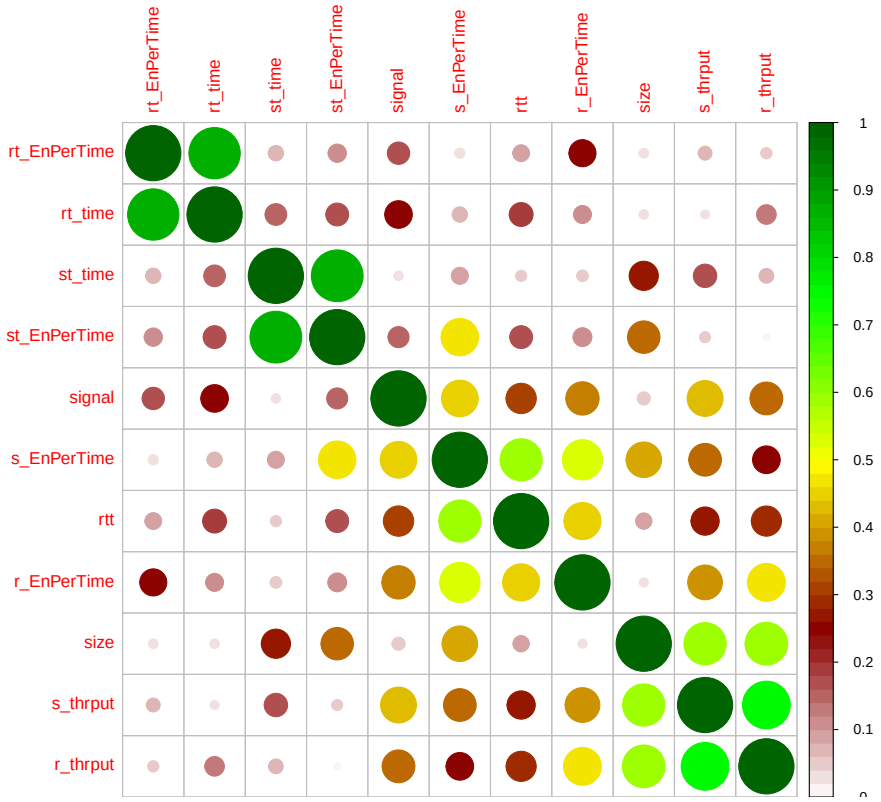

Figure 5.15: WiFi correlations for bigger packet sizes

Figure 5.15 depicts the correlations for the variables extracted from measurements with bigger packet sizes ($\geq 0.5MB$). One notices less strong correlations than for LTE. Nevertheless there are some interesting correlations for the variables.

The signal shows a medium positive correlation to the throughput (send: 0.48; receive: 0.41; $p < 0.01$) as well as a medium negative correlation to energy per time for sending and receiving (send: -0.48; receive: -0.33; $p < 0.01$) and RTT ($-0.36; p < 0.01$).

Also the packet size shows a correlation to the throughput (send: 0.57; receive: 0.58; $p < 0.01$). Additionally the packet size has some smaller correlations energy per time for sending and its tail time (send: $0.29p = 0.02$; tail: $0.34p < 0.01$).

Energy per Time

As well as for LTE there is a closer look at the energy per time variable. In the correlation matrix we have seen that the energy per time has weak or no correlations to the size, while having a medium correlation to the signal. Observing the plot in figure 5.16 the

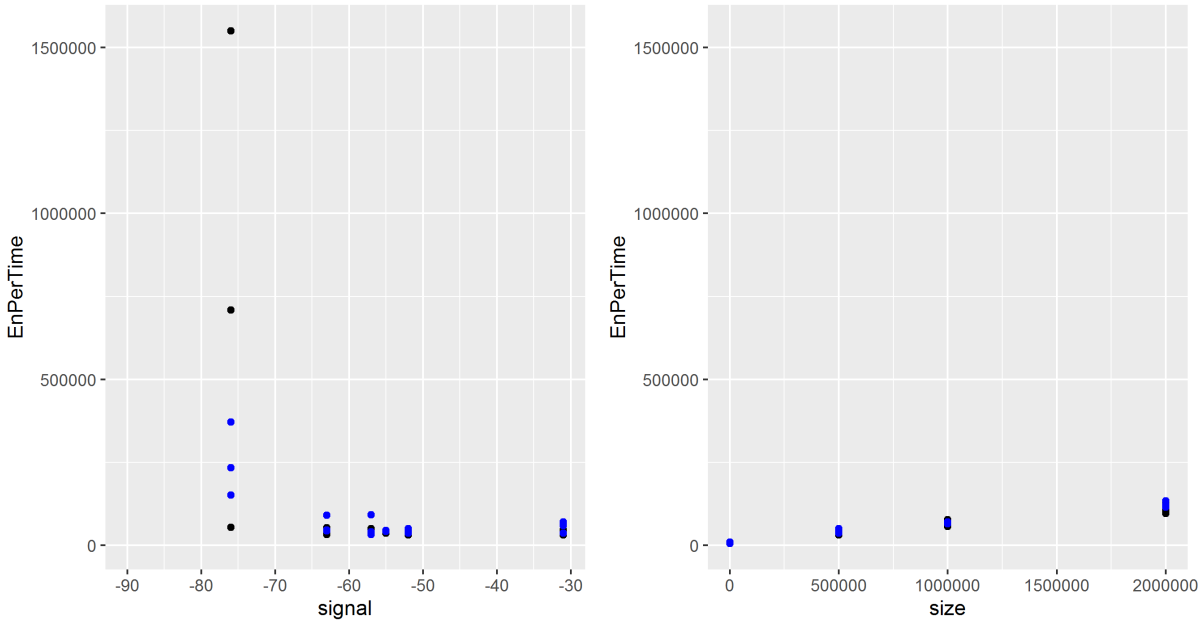

Figure 5.16: Energy per time [mJ/s]
(left: $size = 0.5MB$; right: $signal = -52$; black: receive; blue: send)

energy per time has a small correlation to the size, but is nearly constant in comparison to the effect of the signal on it. There we can see nearly no effect until the signal gets worse. Similar to the results in LTE the signal shows an exponential correlation.

In contradiction to the behavior of the energy per time during tail times in the LTE network, there seems to be no correlation in the WiFi network (cf. figure 5.17). Even though there is a small correlation of size and send tail energy per time, this seems to be caused by fluctuation. It would not make much sense in a technical way that the energy consumption of the tail time is dependent of the packet size (or time) of the previous sending phase.

Throughput

For the throughput in the WiFi network (cf. figure 5.18) the correlation to packet size is similar to the correlation in LTE. Even though the correlation to signal is a bit noisier than for LTE, there is again a drop in the throughput for low signal strength.

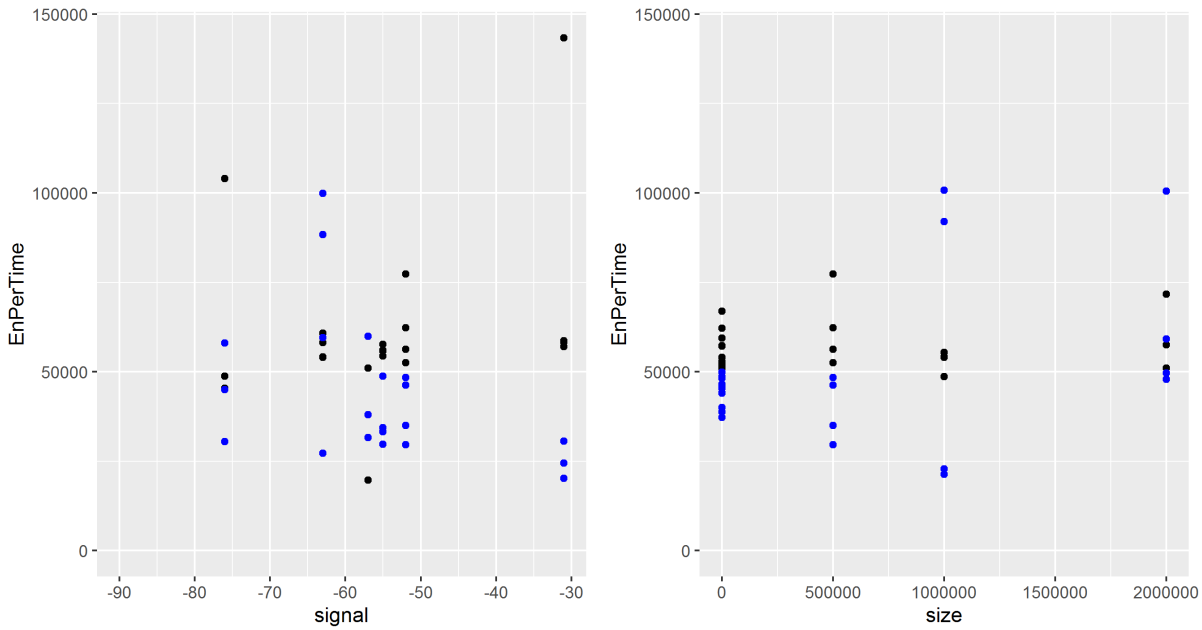

Figure 5.17: Energy per time [mJ/s] during tail time
(left: $size = 0.5MB$; right: $signal = -52$; black: receive tail; blue: send tail)

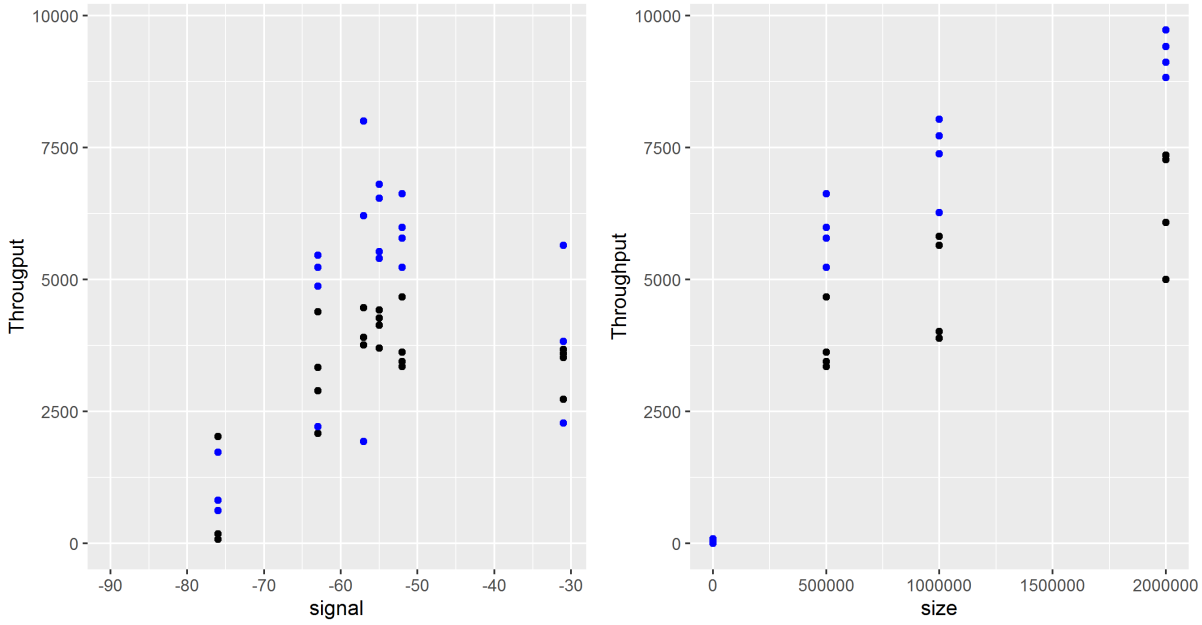

Figure 5.18: Throughput [kB/s]
(left: $size = 0.5MB$; right: $signal = -52$; black: receive; blue: send)

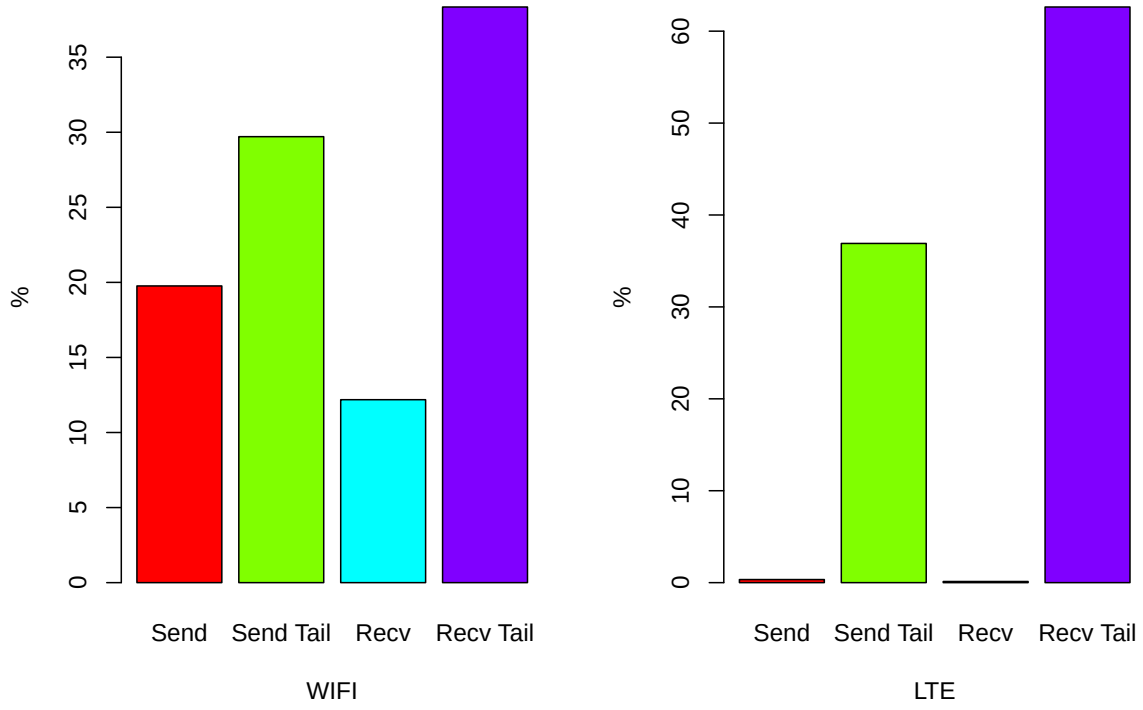

Figure 5.19: Proportions of phases in energy consumption

5.5.4 Distribution of Energy Consumptions in Phases

Another interesting observation is that in both network types the majority of energy is consumed by the tail times as can be seen in figure 5.19. In case of LTE both tail times consume about 99% of the whole energy, while in case of WiFi about two thirds of the whole energy is consumed in the tail times.

Also the receiving tail time consumes more energy than the sending tail time in average, while receiving itself usually consumes less energy than sending as also seen in subsection 5.5.2.

6 Prediction Model

This chapter presents interpretations of the data observed in chapter 5. Based on this, two prediction models for the energy consumption are presented.

6.1 Data Interpretations

Based on the observations from section 5.5, these can be evaluated and interpreted.

6.1.1 Energy per Time

The energy per time during sending and receiving correlates to signal and size. The correlation to the signal was as expected as the device has to increase the effort in listening and transmitting to the medium. Correlations to the size has not been expected. The behavior relates to the increase of throughput, because higher throughput means more energy consumption per time for tasks like checking packets on their correctness, performing error corrections, ordering packets etc. as the number of those operations per time increases.

6.1.2 Throughput

The throughput correlates as well to signal and size. As the data rates adapt to the signal the correlation to the signal was expected. The correlation to size was unexpected, but there are some possible explanations to this observation.

One reason could be that the TCP slow start prevents a high throughput for small transfer sizes as the window size of the algorithm might not be converged yet. Another reason could be an imprecise pattern extraction especially for small sizes leading to this effect. Also it may be possible that the network allocates more bandwidth to the client, if it sends more data. However this cannot be approved as the service providers can set arbitrary rules for this.

6.2 Basic Regression Model

The idea of a basic regression model is to only consider the extracted values, i.e. the energy consumption per phase, of the energy measurements and create a simple regression model. As, while analyzing the correlations, it has been found that the signal has an exponential correlation, the signal is converted to a value ls (linearized signal) that has a linear correlation:

$$ls = exp(\frac{signal}{-10})$$

Using the ls value together with the size as input, a basic linear regression model can be created. The result of such a linear regression is an equation of the form

$$energy = a + b \cdot ls + c \cdot size + d \cdot ls \cdot size$$

for each connection type (i.e. WiFi and LTE) and phase (i.e. sending, tail time after sending, receiving, tail time after receiving). Therefore the parameters a, b, c and d have to be estimated. As the measurements resulted in multiple data tuples $t_i = [energy_i, ls_i, size_i]$ a set of equations

$$energy_i = a + b \cdot ls_i + c \cdot size_i + d \cdot ls_i \cdot size_i + \epsilon_i$$

can be used to minimize the sum of quadratic errors $\sum_{i=1}^{n} \epsilon_i^2$. As the tail is independent of the size, for these energy consumptions the linear regression is calculated without considering the packet size:

$$energy = a + b \cdot ls$$

For the calculation of the prediction value and statistic intervals four different parameters are needed:

- α: The value of Student's t distribution for 95% and the appropriate degree of freedom of the samples.
- V: The variance-covariance matrix is needed to calculate standard error for a specific set of values.
- $coef$: Vector of the coefficients to make a prediction.
- σ^2: Uncertainty of noise to calculate the prediction interval.

These values are everything that is needed to compute the result. However α could also be calculated at runtime, but to save computational time (and therefore energy) this calculations has been performed previously.

To calculate the standard error for a specific value the variance-covariance matrix V and a vector for the concrete values v_c is needed:

$$se^2 = v_c * V * v_c^T$$

Having the standard error the confidence interval CI and the prediction interval PI can be calculated as well as the prediction value p:

$$p = v_c * coef$$

$$CI = p \pm \alpha \cdot \sqrt{se^2}$$

$$PI = p \pm \alpha \cdot \sqrt{se^2 + \sigma^2}$$

The overall prediction model adds the results of the four predictions to retrieve a value for the complete energy used. Details on the implementation of the prediction model and its architecture can be found in section 7.4.

6.3 Dividing into sub-problems

The second idea is to first split the calculations of the energy consumption into the 4 parts that have already been used for the analysis of the measurements and secondly use a formula to split the problem into sub-problems. As a third steps the sub-problems are tried to solve with the help of linear regression on the measured data and using the information extracted from the correlations.

The basic formula for retrieving the send and receive power is based on [SNR+10], where they use the following formula to calculate the energy used for streaming a chunk of a stream of size S with N chunks:

$$energy = \frac{Signal\_to\_Power(signal_l) * \frac{S}{N}}{Signal\_to\_Throughput(signal_l)}$$

This formula uses the size of a single chunk ($\frac{S}{N}$) and the throughput to calculate the time the reception of the download takes and multiplies it with the power, while power and throughput are both estimated by using the signal. This formula has to be taken carefully as [SNR+10] takes only 3G into account. Especially the dependency on the

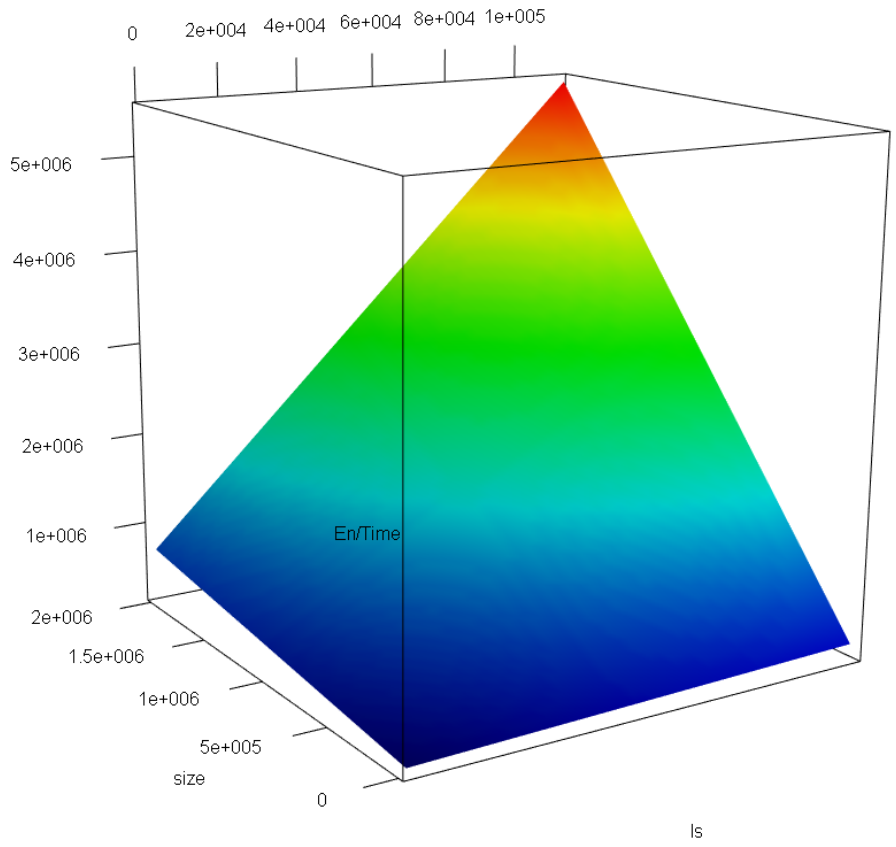

Figure 6.1: Example of a linear regression model

signal as only paramter is doubtfully based on the observations previously made. We can assume that energy per time in cases of sending and receiving and throughput in case of sending is depending on both signal and size.

As this thesis is focusing on Request-Response patterns, the formula is adapted by removing the dividing by N to get an equation for receiving and sending data each. Additionally the tail time needs to be considered. Details about the tail time can be found in the following section. As the tail time is independent of the size of request or response the power needed for a millisecond of tail time during a specific signal can be multiplied with the time it lasts.

After dividing the formulas into sub-problem there are six different equations that need to be solved: Four functions, defining energy consumption per millisecond during each of the phases (sending, receiving and two tail times), and two functions, defining the throughput for sending and receiving.

Having the six functions the method for extracting a prediction model for each of the values is the same as for the basic regression model described in section 6.2. To make the relations to the signal linear the ls values are calculated and the linear regression models for each variable are generated. A 3D visualization of one of the regression models can be seen in figure 6.1. The overall prediction model first calculates the prediction for each phase and then sums them up as the overall consumed energy as shown in the following equations:

$$sendPower = \frac{size \cdot SendPower(size, signal)}{SendThroughput(signal, size)}$$

$$sendTail = min(waitingtime, regularTailtime) \cdot SendTailPower(signal)$$

$$recvPower = \frac{size \cdot RecvPower(size, signal)}{RecvThroughput(signal)}$$

$$recvTail = regularTailtime \cdot RecvTailPower(signal)$$

$$completePower = sendPower + sendTail + recvPower + recvTail$$

Besides the concrete values for each variable and phase, intervals are calculated to enhance the prediction with statistical intervals. An example for such intervals are visualized in figure 6.2.

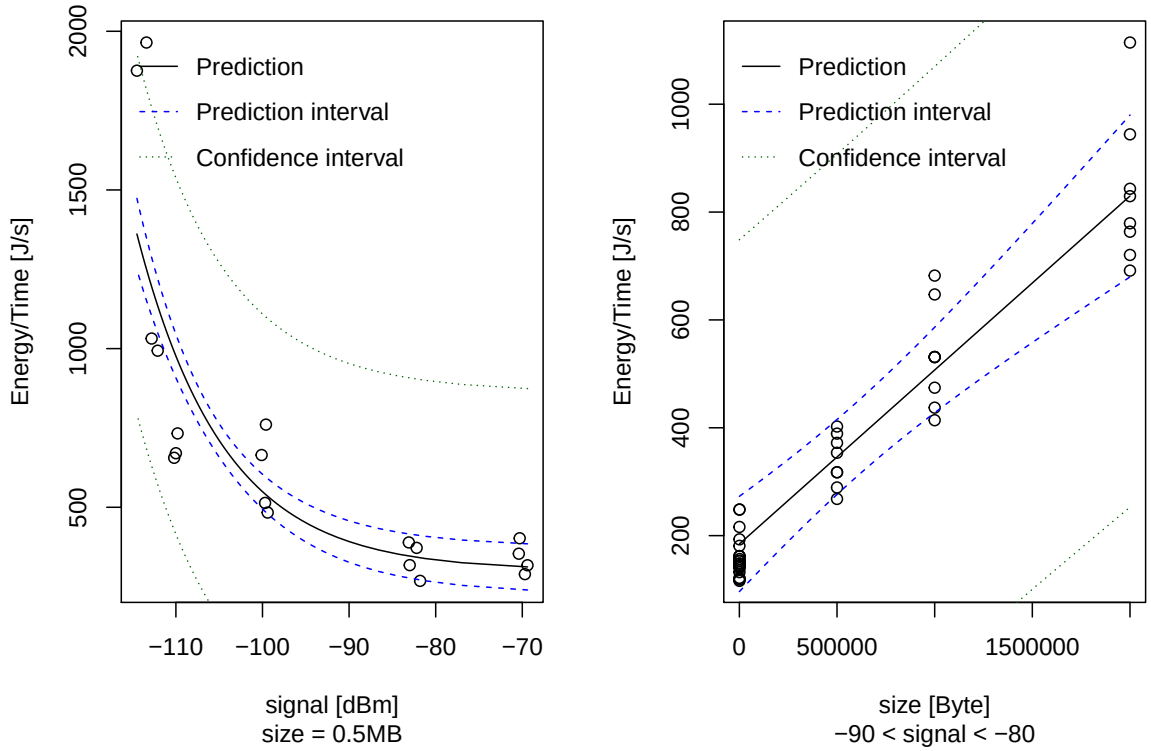

Figure 6.2: Example of prediction and confidence intervals

6.4 Tail Time

As shown in section 5.5.4 tail time consumes the major part of the energy, which is why the tail time needs to be considered. Typically the duration of tail time is a fixed value independent of the time, size or signal during the reception phase. So it can also be handled as a static value in the prediction model. Therefore the mean values has been calculated and by applying a t-test a confidence interval has been calculated. This means the time of the tail times can be used as a concrete value as well as a statistical interval.

In case it is known that the tail time is reduced due to an incoming response before the tail time ends, the sending tail time needs to be shortened to the estimated time the reception starts. In case of the basic regression model this is done by multiplying the predicted energy of the full tail time by the percentage of the real tail time:

$$sendTail = predictedSendTail \cdot \frac{waitingtime}{regularTailtime}$$

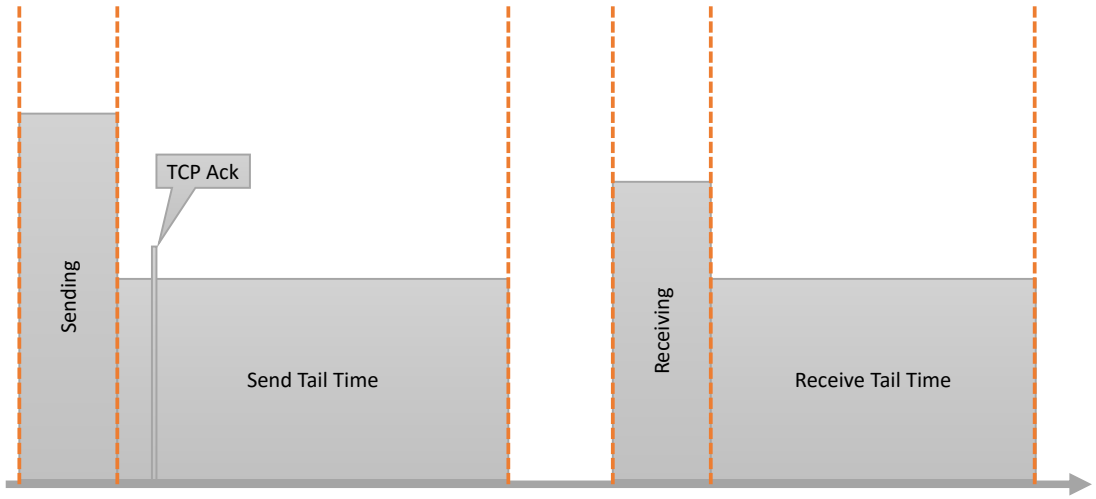

Figure 6.3: TCP acknowledgment increasing send tail time

In case of the calculation model the regular tail time is considered as a time value anyway, so it only needs to be replaced by the waiting time in case that it is smaller than the regular tail time.

TCP may have an influence on the tail time. The acknowledgment received after sending some data may not be detected as part of the communication as its size is typically small and one RTT later than the last reception. This means that the TCP acknowledgment can increase the tail time as sketched in figure 6.3. This would also happen in a environment using TCP. Nevertheless this only applies for the tail time after sending as after receiving TCP acknowledgments are send, but not received, which means there should not be a stealth arrival of an acknowledgment.

For LTE the median tail time in TCP is 14768ms (mean: 15516ms) for sending and 14990ms (mean: 15273ms) for receiving. In measurements that has been taken with UDP the mean for both is 15007ms. The distribution of the tail time measurements in TCP can be seen in the violin plot in figure 6.4. It is also important to know that these values are specific for the vendor as the tail time may vary per vendor. In [HQG+12] for example they measured a tail time of 11576ms.

For WiFi the median tail time in TCP is 202ms (mean: 206ms) for sending and 200ms (mean: 211ms) for receiving. In the UDP measurements the mean for both is 181ms.

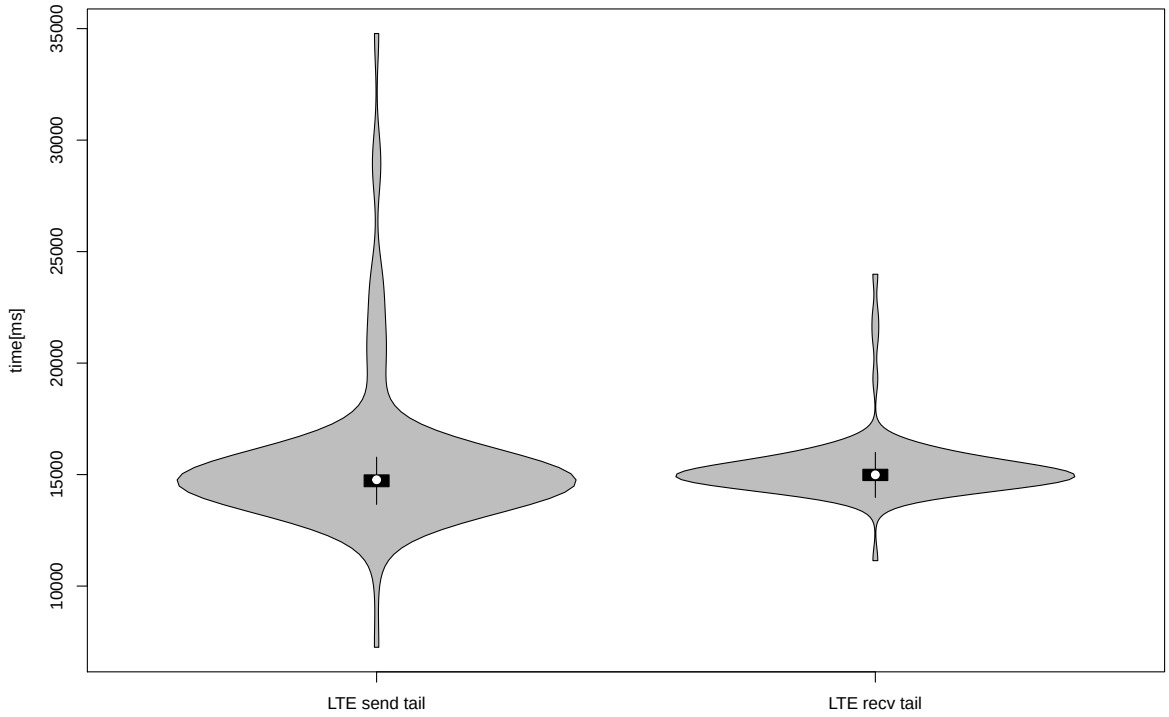

Figure 6.4: LTE tail time violin plot

The distribution of the tail time measurements for WiFi in TCP can be seen in the violin plot in figure 6.5.

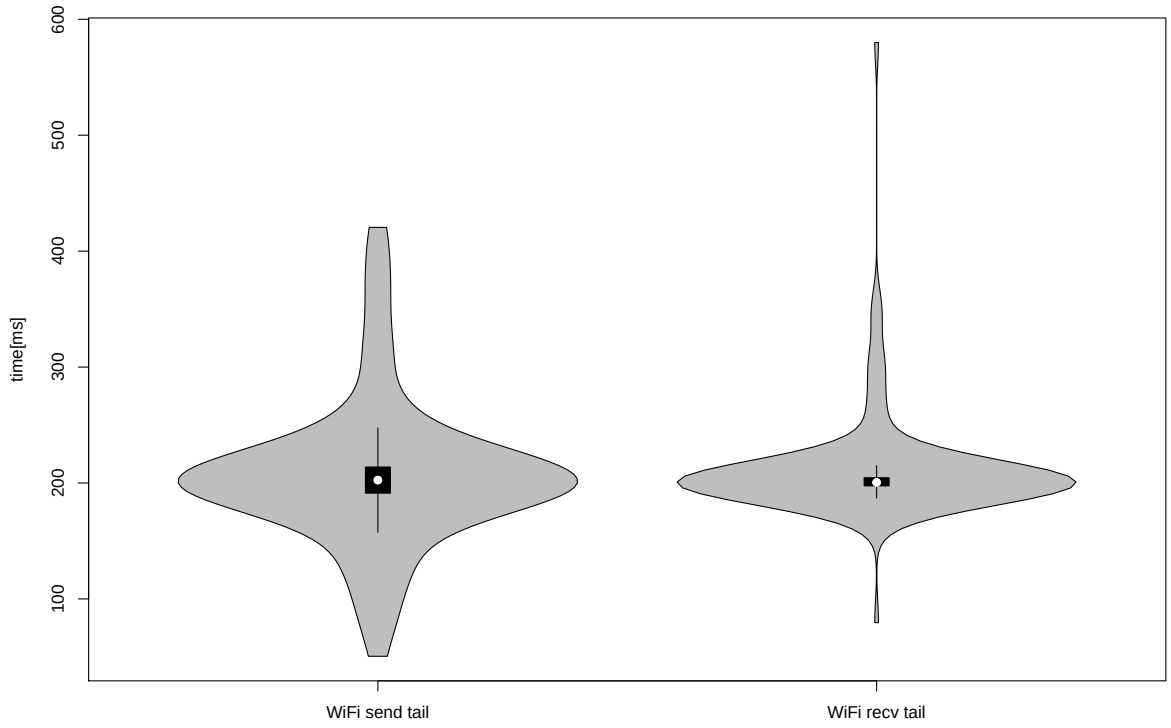

Figure 6.5: WiFi tail time violin plot

7 Implementation

This chapter focuses on the implementation that has been necessary for different phases of the thesis. First of all the Android app used for the measurements and the evaluation is presented. Part of the app is the powerpeak generation, which is covered separately in the section after. The pattern extraction helps to extract concrete values from the power measurements in combination with the logging performed by the app. Finally the implementation of the proposed prediction models will be presented.

7.1 App

For the mobile device an Android app called NetworkLogger has been implemented. This app provides logging of network properties and can act as a client for the pattern protocol described in section 5.1.2.

The user interface is implemented as a single activity, which is shown in figure 7.1. The purpose of the user interface is starting and stopping the logging service, showing some of the network parameter the logging service logs, defining server address and port of the pattern protocol server and starting two example pattern (mainly for testing purpose) and the automatic experiment. The activity itself does not perform anything besides controlling the service by broadcasts and receiving broadcasts from the service (for displaying logging values).

Most of the work is done by the service as the service is even running when the activity is not shown to the user. To allow the device to go in sleep mode, i.e. apps normally are not able to process something, and still be able to do its work, a `WakeLock` is acquired as mentioned in section 5.4.1:

```
this.wakeLock = powerManager.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK,
   "NetworkStatusLoggerWakelock");
this.wakeLock.acquire();
```

When the service is started it starts listening to information about the WiFi network and the phone's cellular network. Every time the parameter of any network changes a `LogEntry`, which is consisting of the timestamp (in ms), an event (i.e. the parameter that

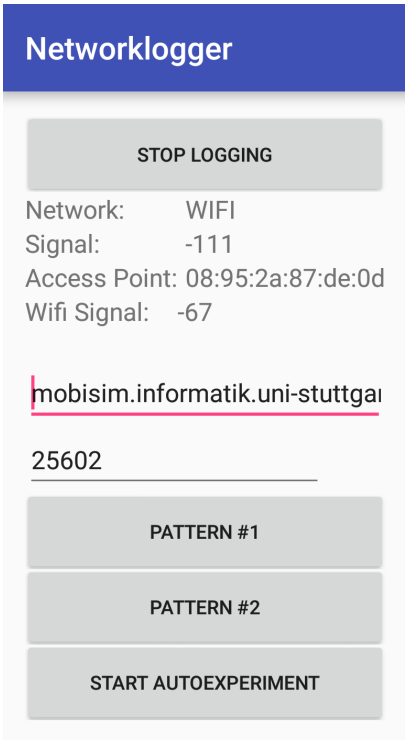

Figure 7.1: Screenshot of the Networklogger app

has changed) and a value, is created and appended to a `LinkedList`. This log is serialized to a file on the SD-card when the service is stopped. If the values would directly be written to the SD-card, it would consume to much energy as accessing to the SD-card is very energy intense.

Additionally to the logging of the network the service has to manage and log the communication with the server. The whole life cycle of an auto experiment as described in the following is depicted as an UML sequence diagram in figure 7.2. When the user is pushing the *start autoexperiment* button, the service receives a broadcast containing information about the server address from the activity. Before starting the communication the service first waits for 10 seconds so the user can lock the screen to allow the phone to go into sleep model and generates a powerpeak, which is covered more detailed in the following section. After this a new thread is started with the `AutorunExperiment`, which is responsible for triggering a predefined set of communication patterns. Therefore a TCP connection is opened and the different patterns are called with a waiting time in between to allow them to finish the communication.

A single pattern execution consists of generating a byte array (cf. listing 7.1) and transmitting the byte array. The connection is then left open for at least the waiting time plus additional 10 seconds to receive the response.

Listing 7.1 Generating byte array for request

```
public byte[] serializeRequest(int length) {
      int minlength = 8 + 4 + 8 * packets.size();
      if (minlength > length) {
            length = minlength;
      }
      ByteBuffer bb = ByteBuffer.allocate(length);
      bb = bb.putLong(this.id).putInt(this.patternlength);
      for (Packet p : this.packets) {
            bb = bb.put(p.serializeRequest());
      }
      byte[] random = new byte[length - minlength];
      r.nextBytes(random);
      bb.put(random);
      return bb.array();
}

public class Packet{
      ...

      public byte[] serializeRequest() {
            ByteBuffer bb =
                ByteBuffer.allocate(8).putInt(sleepbefore).putInt(length);
            return bb.array();
      }
}
```

The whole set of patterns can be repeated a specific number of times. In the experiments performed for this thesis the set of patterns (six patterns) has been performed four times. This means that each `AutorunExperiment` is triggering 24 patterns.

During communication several events are logged, which are listed and explained in the following list:

- `PATTERNSTART_EVT`: This event announces the start of a pattern. That means this event is called directly after the timer expiration before generating any request.
- `REQEST_START_EVT`: This event is triggered after generating a request, but before calling the send method. So this event really announces the start of communication.
- `REQEST_FIN_EVT`: This event is triggered after sending the request. Nevertheless this event does not mean that the data has been really submitted as the send method is non-blocking.
- `ANSWER_EVT`: This event is triggered, when the length of the following byte array (containing the pattern protocol data) has been received. This is typically a sign

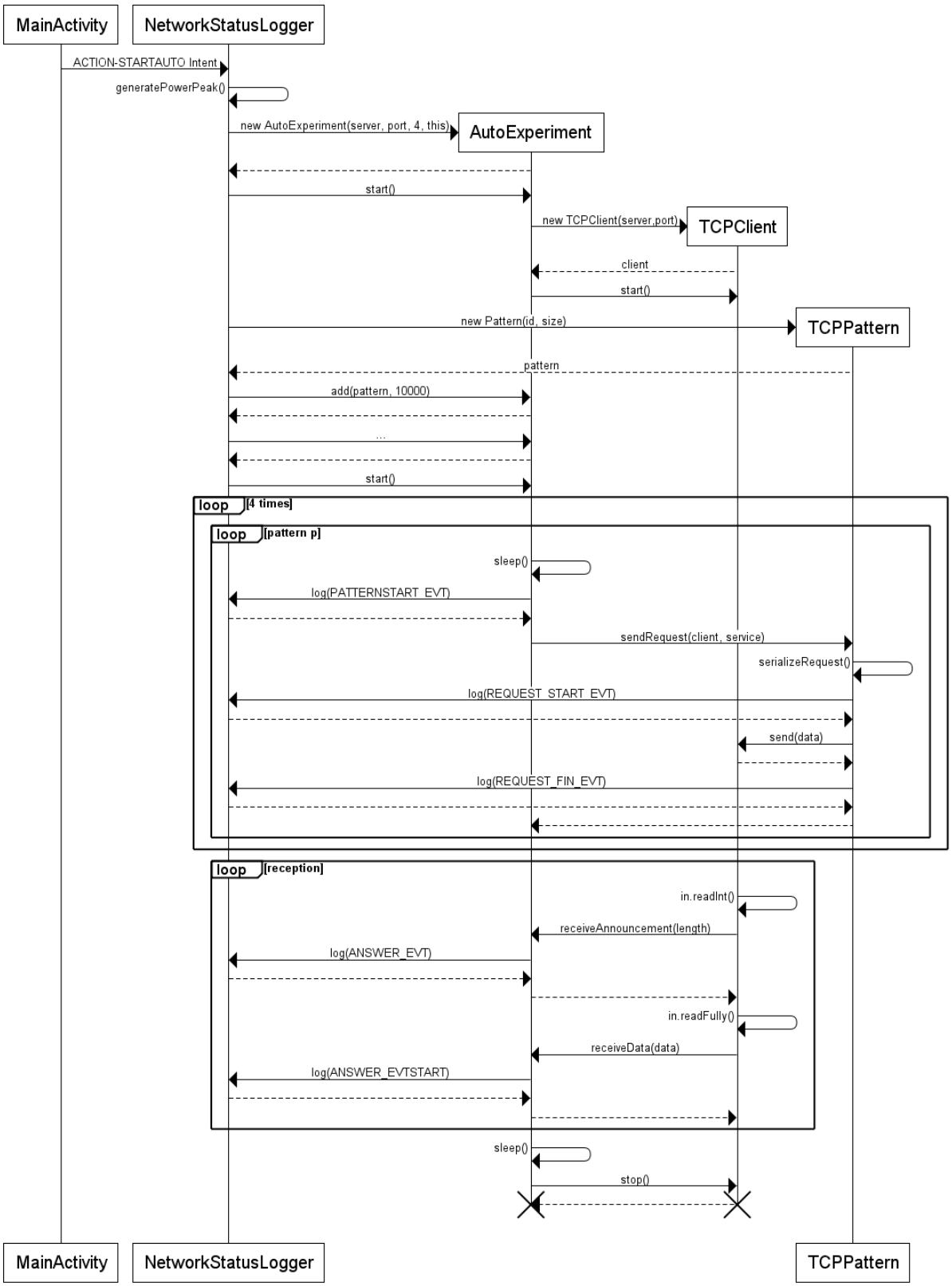

Figure 7.2: UML sequence diagram for the life cycle of `AutorunExperiment`

that the receiving of the data recently started as the length should be received in one of the first packets (assuming the order is roughly preserved during transfer).

- `ANSWER_EVTSTART`: This event finally announces that the complete byte array has been received.

All in all these events mark significant points in the lifetime of a pattern. These log entries are really important, when it comes to the automatic pattern extraction as explained in section 5.5.1.

7.2 Powerpeak & Synchronization of Logs

The measurements include two different logging files, namely the energy measurement from RPi-Powermeter and the event log from NetworkStatusLogger. Synchronizing these two logs is not trivial, but it is necessary to do to extract the information needed for the calculations. In an optimal case there would be an entry in both logs logging the same event at the same time.

To achieve this the basic idea is to consume CPU resources independent of the communication measurements within the app and log the start and the end. This is done by the source code in listing 7.2. It just does simple additions for a slot of about one seconds and logs the start and the end for at least one second. This means there is at least one entry in the event log that can be synchronized. To track the event in the energy measurement log, the one second lasting slot needs to be detected. It creates a typical pattern as plotted in figure 7.3. It looks like an increasing step function.

As the number of logs needed for the measurements is manageable, the actual synchronization of the two logs has then been done manually. Nevertheless it could be done automatically by defining thresholds for the typical pattern and perform an automatic search for those. This should be done in case that there are more logs that need to be synchronized.

Listing 7.2 Powerpeak generation code

```
log(POWERPEAK_EVT, "ON");
Thread work = new Thread(new Runnable() {
      @Override
      public void run() {
            int duration = 1000;
            powerpeakcount = 0;
            long finish = System.currentTimeMillis() + duration;
            while(System.currentTimeMillis() < finish){
                  powerpeakcount = powerpeakcount + 1;
            }
      }
});
work.start();
try {
      work.join();
} catch(InterruptedException e) {
      Log.e(LOGTAG, "waiting for powerpeak got interrupted", e);
}
log(POWERPEAK_EVT, "OFF");
```

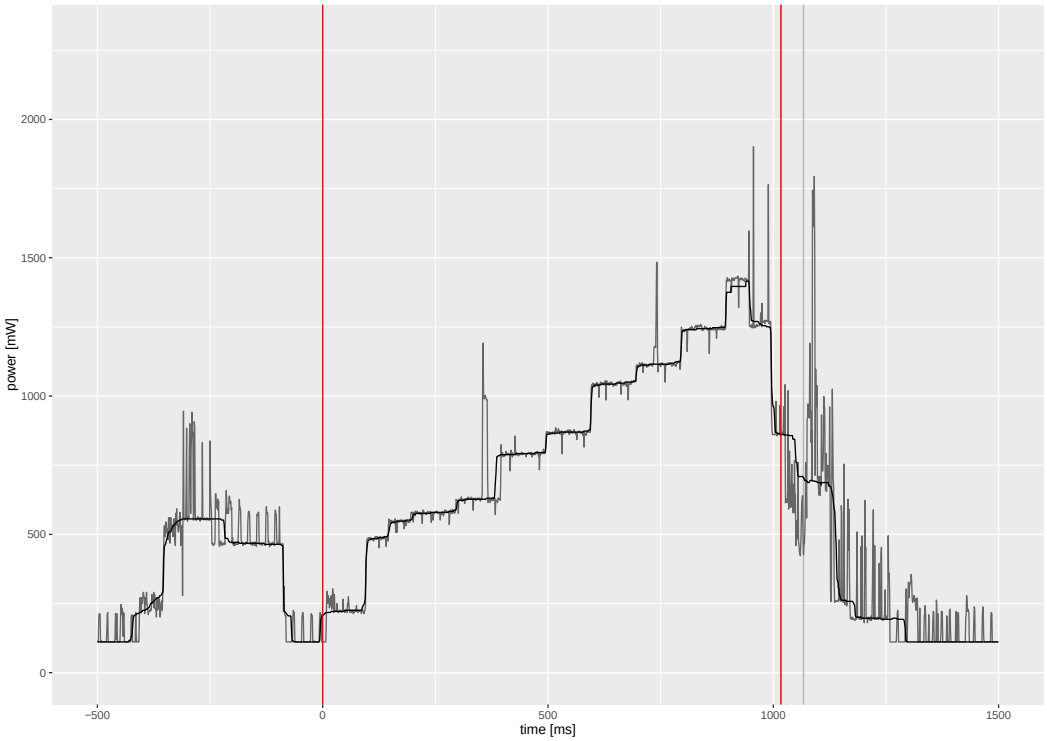

Figure 7.3: Powerpeak (gray: measurement; black: measurement with median-filter w=100; red: logged powerpeak)

7.3 Pattern Extraction

After synchronizing the two logs single request-response patterns can be extracted from the automated experiment sets. The log contains the start of the pattern, the time of send-calls and the time of the receive-callbacks as described in section 7.1. Nevertheless there are some points in time that are important for extracting times and energy consumption, but cannot be logged such as the end of sending a packet and the end of both (send and receive) tail times.

However with the help of the existing point in time and by observing the measurement-data it is possible to detect these times. To efficiently perform these extractions automatically a simple algorithm has been implemented that uses existing log entries and thresholds as shown in listing 7.3. The thresholds has been found by observing the patterns. The minimum energy needed for the tail time is 250mW for WiFi and 150mW for LTE (`t_tail`). The minimum energy needed for communication (i.e. actively sending or receiving data) is 500mW for WiFi and 800mW for LTE (`t_comm`). Additionally a threshold, setting the maximum time the device starts to receive data before announcing the start of the receive (length of byte array), is defined (`t_maxbeforerecv`).

Based on the values from the log first estimations only set a rough point that acts as fallback solution in case there is no value within the given range that satisfies the conditions. The thresholds are used to define the ranges and the conditions.

Listing 7.3 Pseudo code for extracting information

```
//defining thresholds
if(r_type=="WIFI"){
      t_tail <- 250
      t_comm <- 500
      t_maxbeforerecv <- 25
      tailtime <- 200
}else{
      t_tail <- 150
      t_comm <- 800
      t_maxbeforerecv <- 1000
      tailtime <- 15000
}

//first estimations
lastsend <- send command + 10
lastrecv <- receive callback
startsend <- send command
firstrecv <- receive start callback
endsendtail <- lastsend + tailtime
endrecvtail <- lastrecv + tailtime

//improvements using thresholds
lastsend <- first < t_comm in [lastsend,lastsend+1000]
firstrecv <- last < t_comm in [firstrecv-t_maxbeforerecv,firstrecv]
endsendtail <- first < t_tail in [lastsend + 1/4 * tailtime, endsendtail + tailtime]
endrecvtail <- first < t_tail in [lastrecv + 1/4 * tailtime, endrecvtail + tailtime]
```

7.4 Prediction Model

The final prediction model needs to be implemented in a way that it can be executed as fast as possible with all information needed. At the same time it should be extensible to support future work to reuse the prediction model and extend it with other types of networks. The implementation is done in Java to make it easy to implement it in any Android app.

Figure 7.4 depicts an UML class diagram modeling the current implementation of the prediction models. The first step was to create common classes for linear regressions with one or two parameter. This led to implementing `BasicPredictionModel` and `BasicPredictionModel2D` and the common interface `PredictionModel`. The interface can be used for any kind of prediction model that needs at most two parameter. The result of a `predict(...)` call is always a `Prediction`, which includes a concrete prediction value as well as two intervals (prediction interval and confidence interval – the calculation of

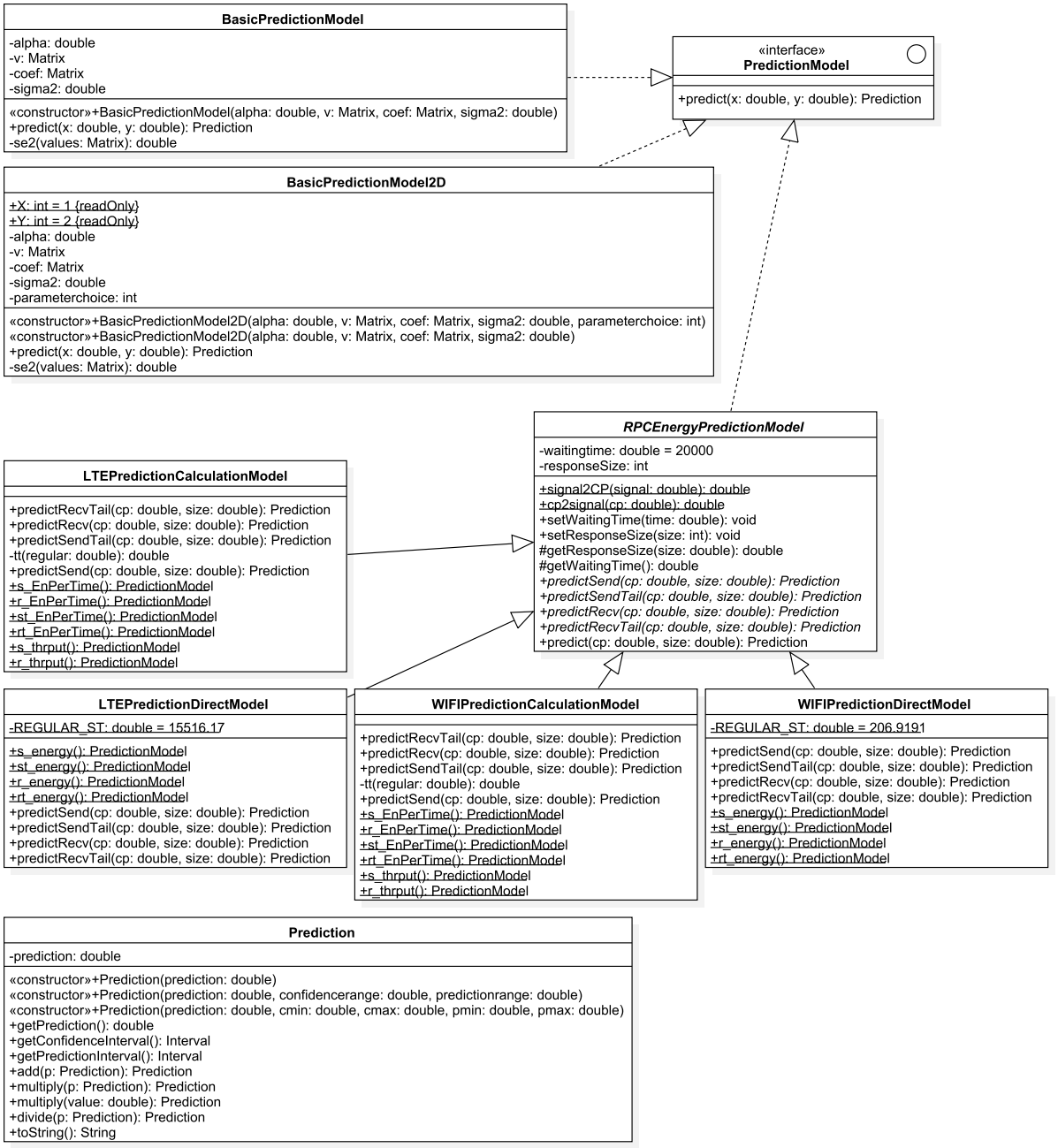

Figure 7.4: Class diagram of prediction model implementation

Listing 7.4 Predicting energy consumption for the sending phase

```
public Prediction predictSend(double ls, double size) {
      Prediction psize = new Prediction(size);
      Prediction s_thrput = s_thrput().predict(ls, size);
      Prediction s_EnPerTime = s_EnPerTime().predict(ls, size);
      return psize.divide(s_thrput).multiply(s_EnPerTime);
}
```

these intervals is covered in section 6.2). The `Prediction` class can also be used to do operations like addition, multiplication and division between predictions.

As in the measurements each part of the transmission has been handled separately, it is now possible to predict the phases each. This can be very helpful in case there is only a send operation without a following reception or when the tail time has to be reduced. The abstract class called `RPCEnergyPredictionModel` requires its concrete implementations to implement respective methods predicting the different phases of the communication. It then uses the results of each prediction to make an overall prediction by simply adding up the predictions.

In the current implementation there are four different classes implementing the `RPCEnergyPredictionModel`: Two for each connection type, one per model. The basic models (or direct models as they are called in the code) are making prediction for each phase directly using a linear regression, while the calculation models use linear regression for throughput and energy per time (for each phase) to calculate the respective phase energy consumption. In case of the energy consumption of the sending phase this is done as shown in listing 7.4.

For the basic prediction models each class has static methods that instantiate the `BasicPredictionModel`s. Four parameters are needed to instantiate it:

1. `alpha`: value for student's t-distribution
2. `v`: variance-covariance matrix (as two-dimensional **`double`** array)
3. `coef`: coefficient vector
4. `sigma2`: uncertainty of noise

These static methods instantiating the models are generated using a method implemented in R (cf. listing 7.5), where all statistical evaluations of this thesis has been performed. As input it only needs a linear regression instantiated with **`lm`**`(z ~x*y)` and optionally a name. In case the linear regression has only one input parameter one has to set `zweid="2d"`. The result of a single generation is what can be seen in listing 7.6 (except for indentation).

Listing 7.5 Method to generate Java prediction model in R

```
generateModelFromRegression <- function(l, name="UNKNOWN", zweid=""){
      v <- vcov(l)
      sigma2 <- sum(l$residuals ^ 2) / l$df.residual
      coef <- l$coefficients
      alpha <- qt((1-0.95)/2, df = l$df.residual)
      writeLines(paste("public static PredictionModel ",name,"(){\n",sep = ""))
      writeLines("double[][] v = {{")
      write.table(v,col.names = FALSE, row.names = FALSE,eol = "},\n{",sep = ", ")
      writeLines("}};\ndouble[][] coef = {{")
      write.table(coef,col.names = FALSE, row.names = FALSE,eol = "},\n{",sep = ", ")
      writeLines(paste("}};\nreturn new BasicPredictionModel",zweid,"(",alpha,",new
          Matrix(v),new Matrix(coef),",sigma2,");",sep = ""))
      writeLines("}\n")
}
```

Listing 7.6 Generated code instantiating a `BasicPredictionModel`

```
public static PredictionModel s_energy() {
      double[][] v = { { 262696740771973216.0, -4068732262497.2, -174961977757.179,
          2692875.01837794 },
        { -4068732262497.2, 125393789.05992, 2693451.58030932, -82.6088917049568 },
        { -174961977757.179, 2693451.58030932, 298296.995406732, -4.5297154534251 },
        { 2692875.01837794, -82.6088917049568, -4.5297154534251, 0.000138970555590831 } };
      double[][] coef = { { 163917441.139577 }, { -17442.686473998 }, { -382.426640056172
          }, { 0.165091691101672 } };
      return new BasicPredictionModel(-1.9815667570749, new Matrix(v), new Matrix(coef),
          9228286347809357824.0);
}
```

In case the waiting time the user estimates is below the tail time, the tail time in the calculation models will be replaced. In the direct models there is no time that can be replaced. Therefore the Prediction of the send tail time is then multiplied with a factor calculated by dividing the expected tail time by the regular tail time as explained in section 6.4.

8 Evaluation

In this chapter the prediction model proposed in chapter 6 is evaluated. The goal is to check how accurate the model can predict the actual energy consumption.

8.1 Evaluation Measurements

Separate measurements has been made, to avoid that the model performs well in the evaluation, but is actually not representative. They follow the same method described in chapter 5.

The measurements for WiFi contain 66 patterns with 3 different measured signal strength (-45dBm, -65dBm and -76dBm). For LTE the measurement has a size of 40 patterns with 2 measured signal strength ranges (-71.5dBm to -79.6dBm and -97.5 to -105.6dBm). Also 48 additional patterns have been measured in LTE with a waiting time of one second, which means the tailtime is reduced by about 14 seconds. These measurements are made to evaluate the predictions with reduced tail time.

Additionally to measuring energy consumption and signal strength etc. the app also uses the prediction model implemented in Java (cf. section 7.4) to make a prediction before starting with the transmission of a pattern request. This enables to measure the time and the energy consumption for the prediction. To reduce the logging effort only the calculation prediction model has been used in the app. Nevertheless it can be assumed that the basic regression prediction model performs similar or even slightly better than the calculation prediction model as it needs to perform less calculations of the same kind. For the further evaluation of prediction results the values has been recalculated afterwards due to an easier data extraction.

8.2 Overhead for Prediction

One requirement for the prediction model formulated in section 4.3 is that the model should have very little overhead in terms of time and energy consumption. In figure 8.1

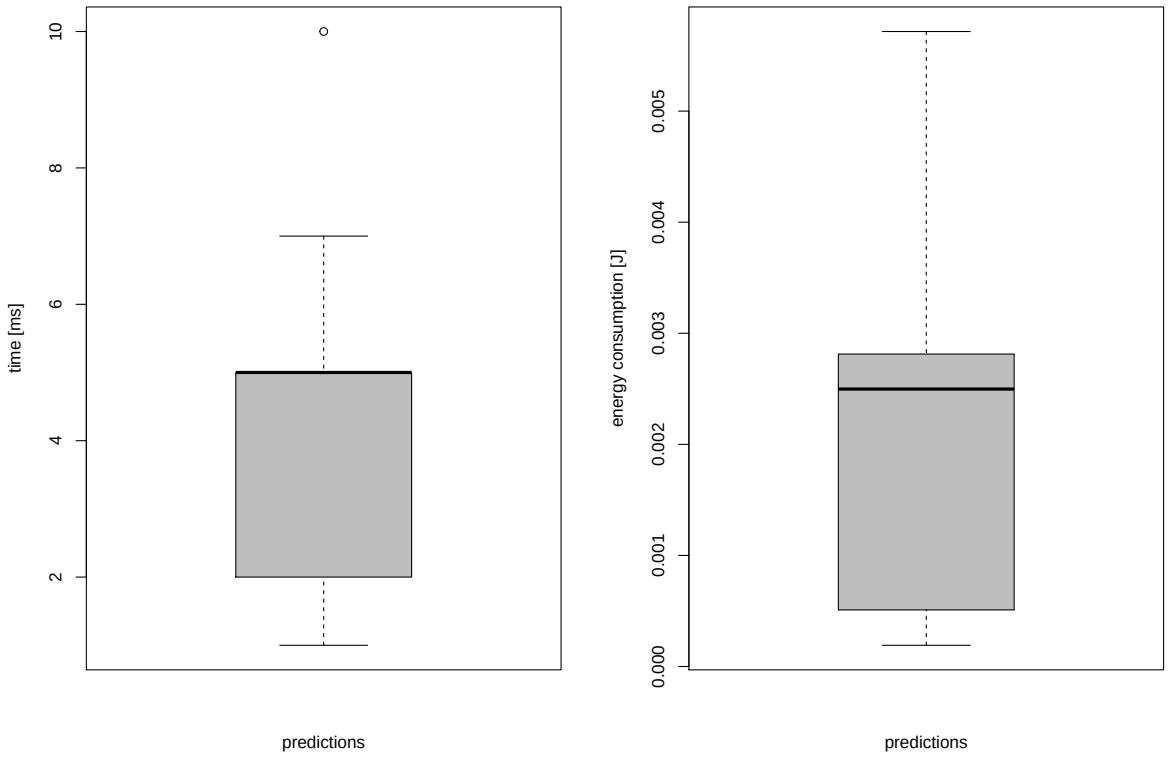

Figure 8.1: Time and energy consumption for predictions

the time and the energy consumption is visualized in a boxplot. There are only few outliers for the energy consumption with more than 0.5J caused by other activities. These are not visualized in the boxplot to increase readability. The median time for a calculation is 5ms and the median energy consumption is 0.0025J. These results show that the model can be calculated very fast and consume only little energy. In comparison the energy consumption without any computations for 5ms is 0,0006J based on the values presented in section 5.4.2.

8.3 Model Comparison

Another requirement for the prediction model is a low error rate. Therefore the predicted values of each model can be compared to the measured value of the evaluation measurements. Finally the error rates of both models will be compared. As the energy consumption is very different between WiFi and LTE the results of the models for both network types are handled separately.

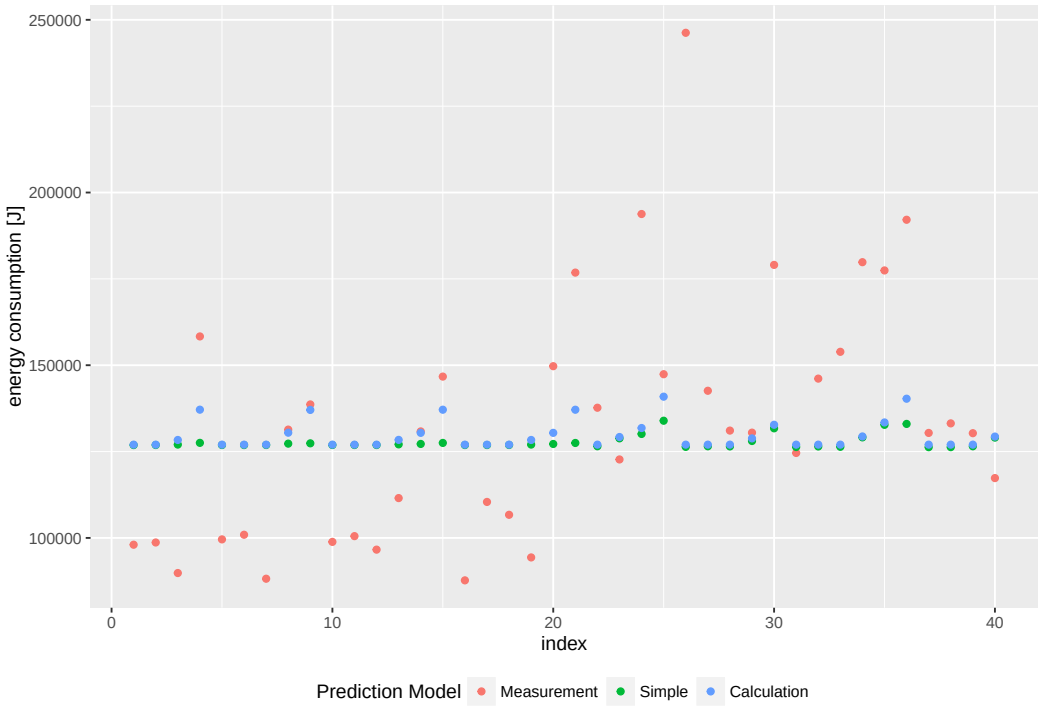

Figure 8.2: Measurement and prediction values for LTE

8.3.1 LTE

Figure 8.2 shows the measured values and the predictions of the two prediction models. To associate the indexes in the figure, one has to know that there are small rows with a size of four to six points (depending on how many measurements has been removed due to interference with other apps) that are of rising size and there are two big rows changing at index 22 that show the two different signal strength ranges (low on the left, high on the right).

The model is overshooting regularly at least for the small sizes as can be seen at low signal strength. For sizes greater than a half MB the model is more accurate (cf. indexes 8, 9, 14) or even too low (cf. indexes 4, 15, 20, 21). At high signal ranges the measurements are regularly higher than the prediction. Nevertheless the indexes 37 to 40 the prediction is accurate although the signal is similar to the previous measurements.

These variations show that LTE is depending on more than just the signal strength. It is hardly controllable how many users are in the network and how much traffic they produce. This can be one reason for such deviations.

Comparing the two prediction models the prediction values are quite similar. In figure 8.3 the relative error ($err/measure$) of the two models is compared in a boxplot. Again you

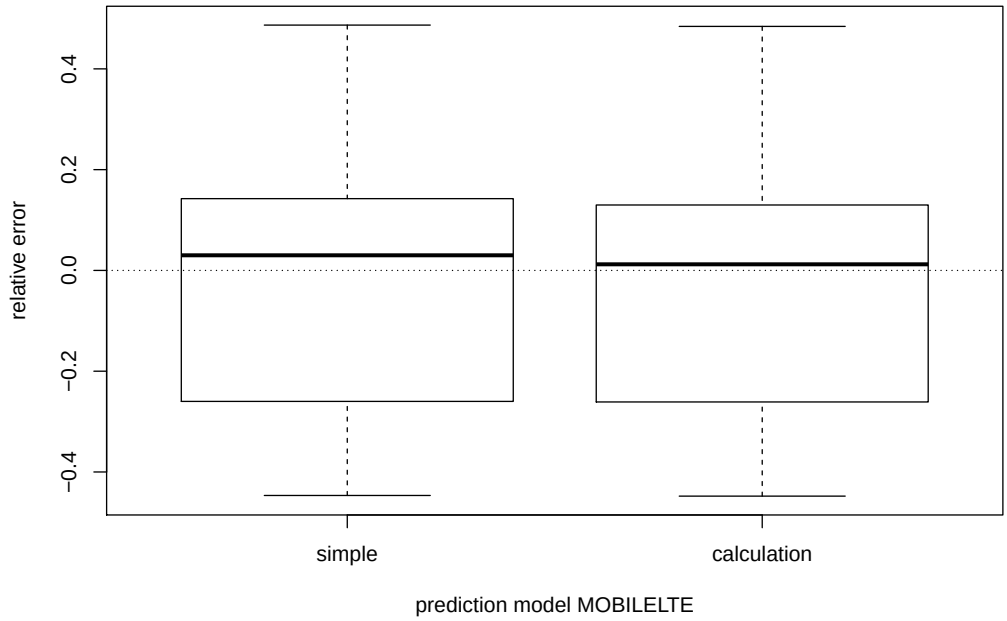

Figure 8.3: Relative error of prediction models for LTE

can see that both models perform similar, but the calculation model is slightly better, as the median relative error for the simple model is 3.00% (absolute: $3926J$), while for the calculation model it is 1.21% (absolute: $1629J$). The optimal case would be that the relative error is at 0%, which means the error would be distributed regularly. Additionally the relative absolute error shows the difference between the measured value and the predicted value: For the simple model the median is 18.41% (absolute: $24244J$), while the calculation model is slightly better with 16.29% (absolute: $20756J$).

For both models the prediction intervals include the measured value to 100%.

Overall the calculation model performs better than the basic model. Even though the difference is quite small. Nevertheless the models seem to make good predictions, but it still has some influencing factors that are not covered by size and signal. These weaknesses can be further improved, which will be discussed in section 9.1.

8.3.2 WiFi

The WiFi predictions are different compared to the LTE predictions. Figure 8.4 depicts the measured values of energy consumption in direct comparison to the basic and the

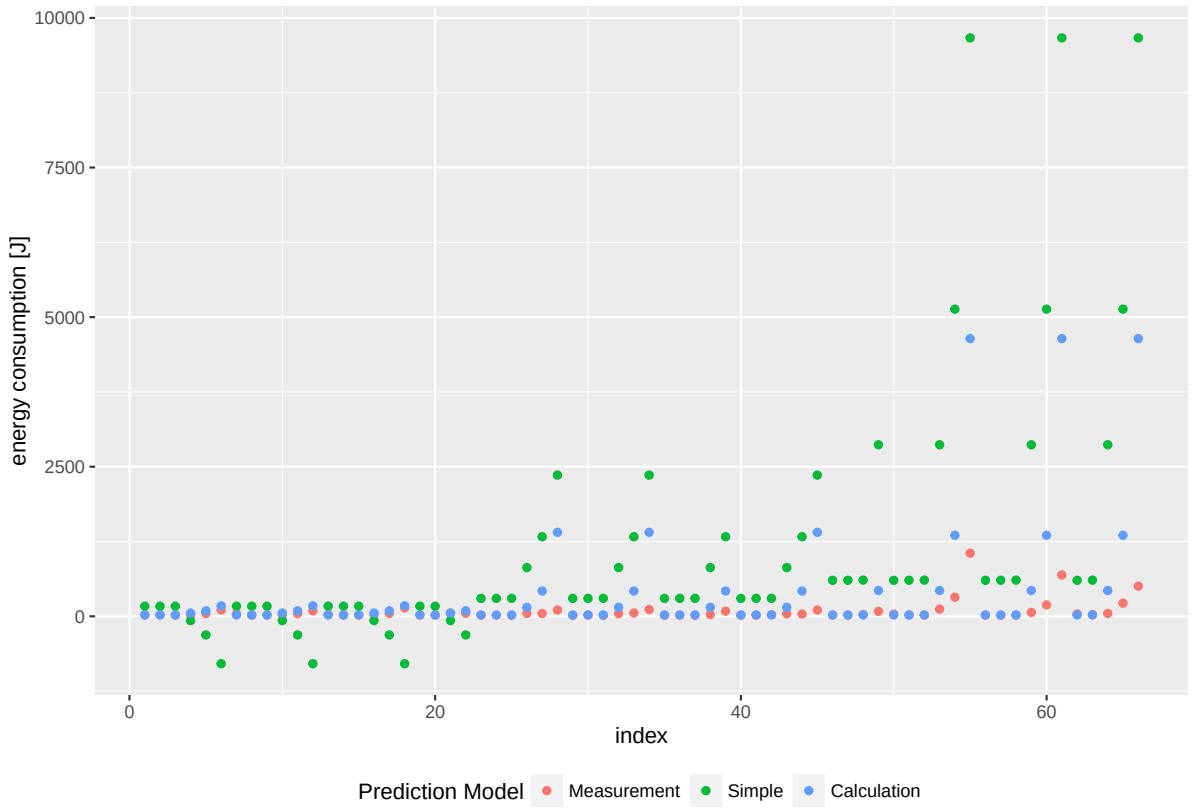

Figure 8.4: Measurement and prediction values for WiFi

calculation model. The different sizes appear in groups of four to six data points, while the different signal strengths are varying from high (-45dBm; indexes 1 to 22) over medium (-65dBm; indexes 23 to 49) to low (-76dBm; indexes 50 to 66).

For the high signal the simple model fails completely as it even predicts negative values. The calculation model is exact most of the time (the points for measurement are mostly covered by the points for the calculations model). The other signals show that both prediction models seem to exaggerate the behavior of the energy consumption with increasing packet sizes. Comparing the two models the calculation model is better in any case even though it overshoots in many cases for low signal and big sizes.

Comparing the relative error, visualized in figure 8.5, the simple model overshoots most of the time with up to 4000%. The median relative error for the simple model is at -1518.68% (absolute: $-283J$), while for the calculation model the median relative error is -26.73% (absolute: $-5J$). For the simple model the median absolute error is $570J$, while the calculation model performs much better with $15J$.

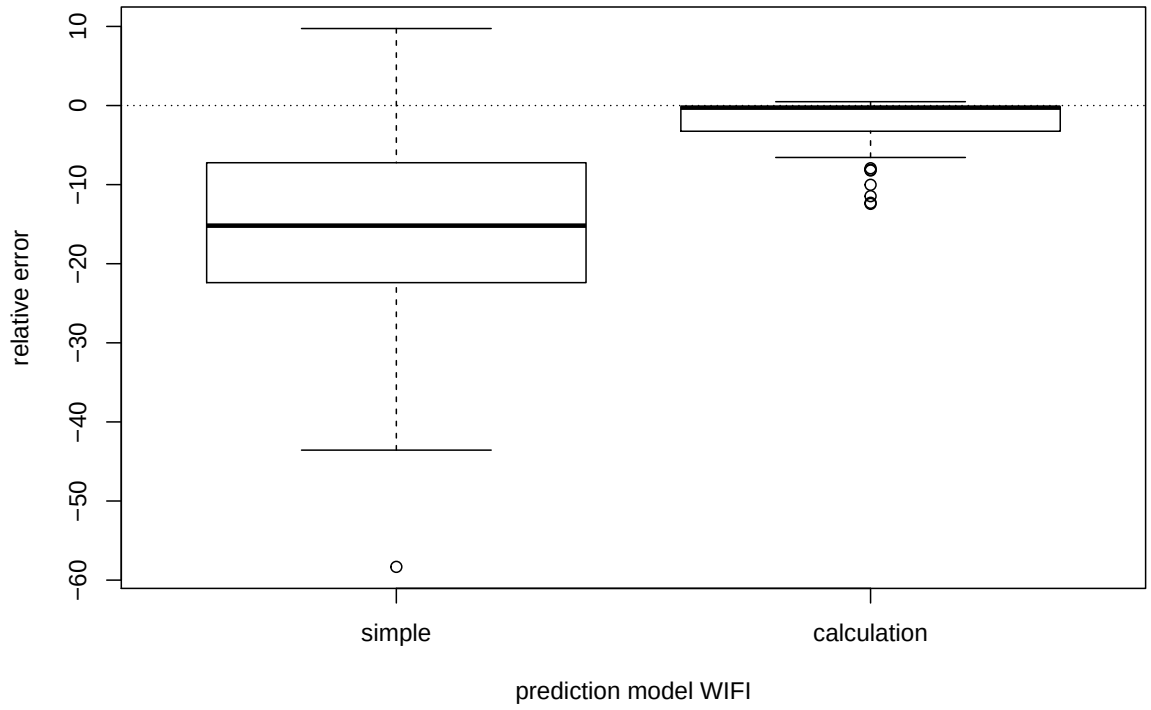

Figure 8.5: Relative error of prediction models for WiFi

The relative error may show that the model is really bad, but in comparison to LTE the absolute measured values are much smaller and thus a relative deviation is much higher very fast. Anyway the calculation model performs way better than the simple model. Especially it has not been predicting negative values for any measurements.

8.3.3 Reduced Tail Time

While previous measurements and predictions always used a waiting time between the sending and receiving parts that allows the tail time to end before the new transmission starts, additional measurements with a reduced tail time (due to earlier responses) has been made to check whether the models are able to make proper predictions. As the WiFi tail times are very small this has only been evaluated in LTE to better show the resulting influence.

Figure 8.6 shows the measurement and prediction values in direct comparison. On the left side (indexes 1 to 24) there are 24 measurements with a signal strength in the range

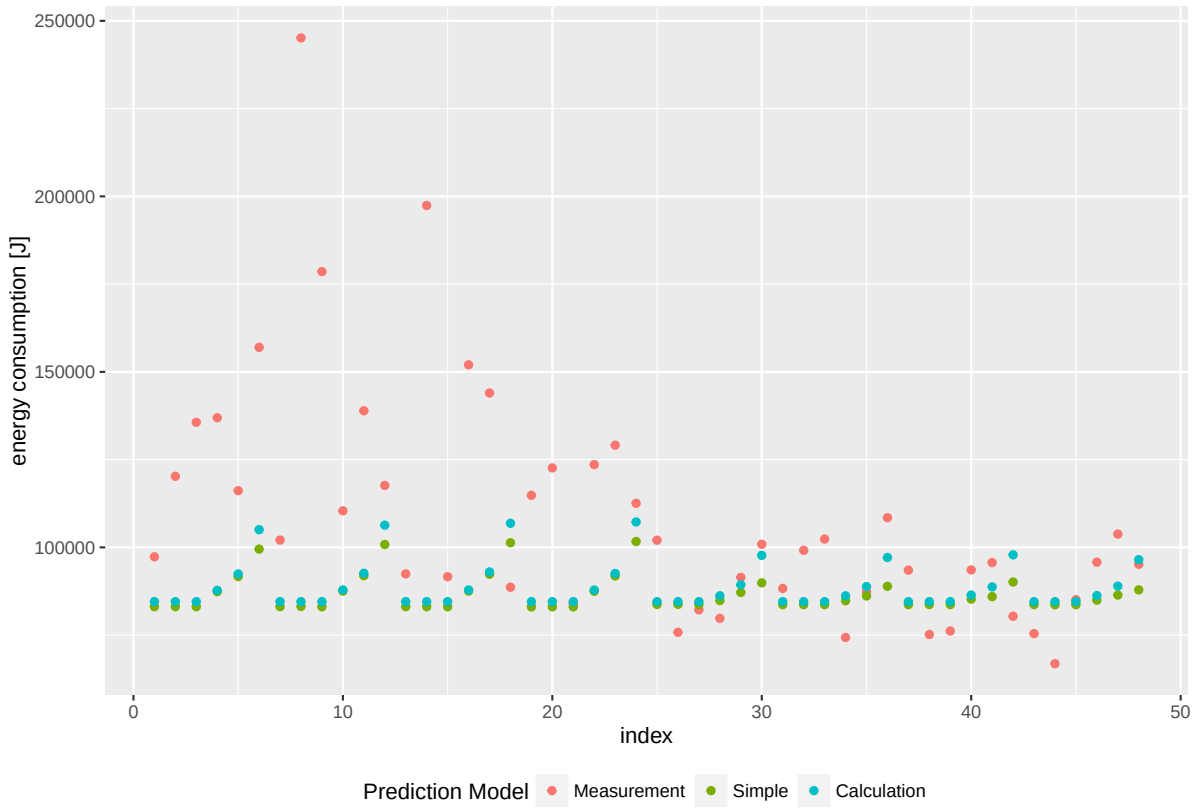

Figure 8.6: Measurement and prediction values for LTE with short tail time

between -107.6dBm and -109.2dBm and on the right side (indexes 25 to 48) between -94.1dBm and -98.7dBm. The overall behavior is very similar to the behavior described in section 8.3.1. Even though it seems like the predictions for the right side are much better, also the left side does not perform too bad. It confirms the impression from the LTE evaluation that the fluctuation are depending on a weaker signal and they are quite hard to predict.

To check if the tail time reduction has a negative influence, figure 8.7 shows the relative error. The median is not as near to 0% as before, but is 12.79% for the simple model and 9.77% for the calculation model. Again the calculation model slightly performs better. In this case the predictions do have a way less absolute error: For the simple model it is $14853J$ (in comparison to $24244J$ for the previous evaluation) and for the calculation mode it is $13704J$ (in comparison to $20756J$). That means in absolute values the prediction even seems to be improved by reducing the tail time. This can have two reasons: Either the tail time is a weak point in the prediction or the sample is just better than the previous.

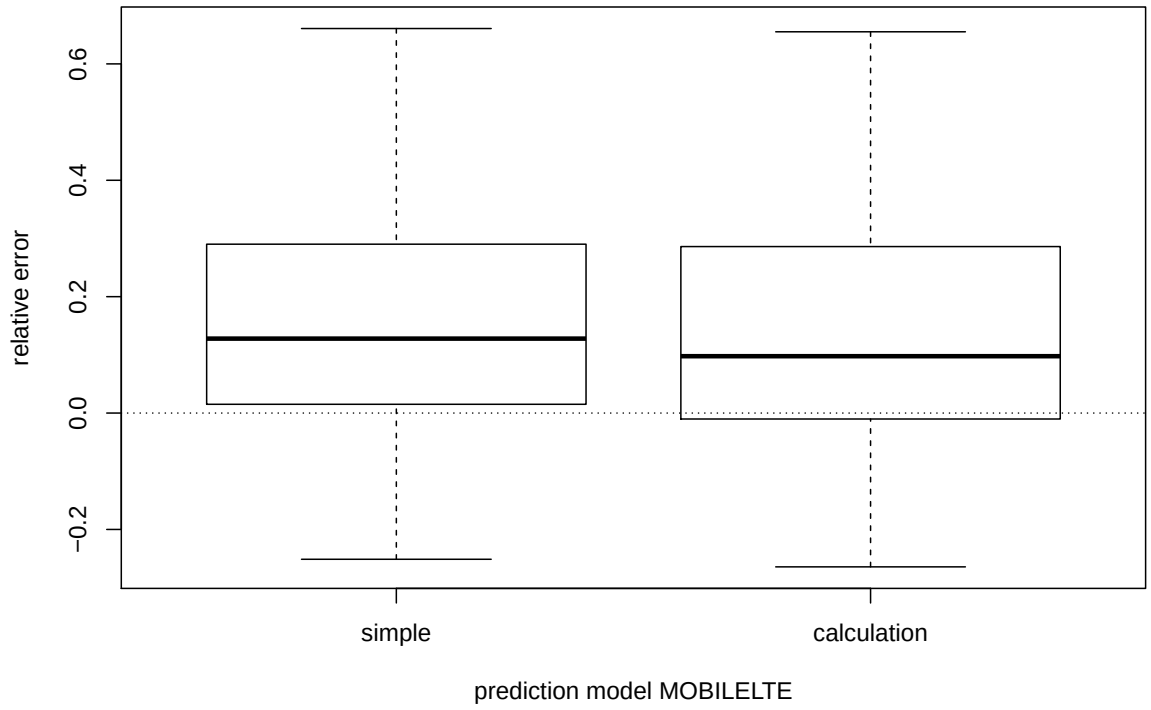

Figure 8.7: Relative error of prediction models for LTE with short tail time

To find an answer, which of the reasons apply, it is possible to have a deeper look at the tail time prediction of the previous example. As the prediction model can predict each phase separately these predictions can be compared to the measurements for this phase. Figure 8.8 shows the relative error for the send tail time energy consumption predictions. While the relative median is near 0% in both cases (-4.40% simple model; -3.28% calculation model), the distribution is slightly screwed to the negative, which might be an explanation for the error with shorter tail drifting into the positive part. In sum the skew of the tail time predictions might point out a weakness of the model, but the results of the additional measurements also show that the absolute error might even be lower than originally evaluated.

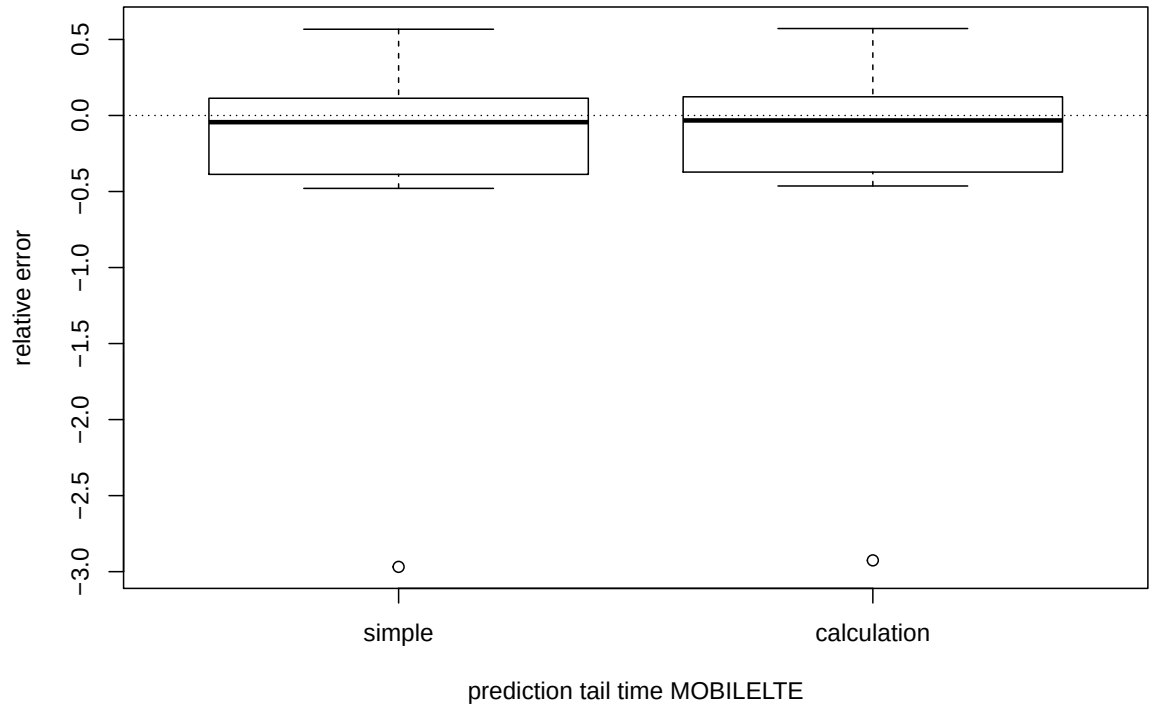

Figure 8.8: Relative error of prediction models for LTE send tail time

8.4 Restrictions

Even though the prediction model shows quite good results, there are some restrictions to the models especially when it comes to generalization. These are described in this section.

8.4.1 Mobility

The classical definition of mobility means the switching between network cells, while preserving connections on IP layer and above. That means even though the physical connection has changed (i.e. the mobile device is communicating with a new cell) the upper layers are preserved (e.g. the device can keep the previous IP, TCP streams are rerouted automatically etc.). Even though in theory the change is transparent to the mobile device's upper layers, this may have an influence on the network performance and the energy consumption. Additionally to the classical definition of mobility, this

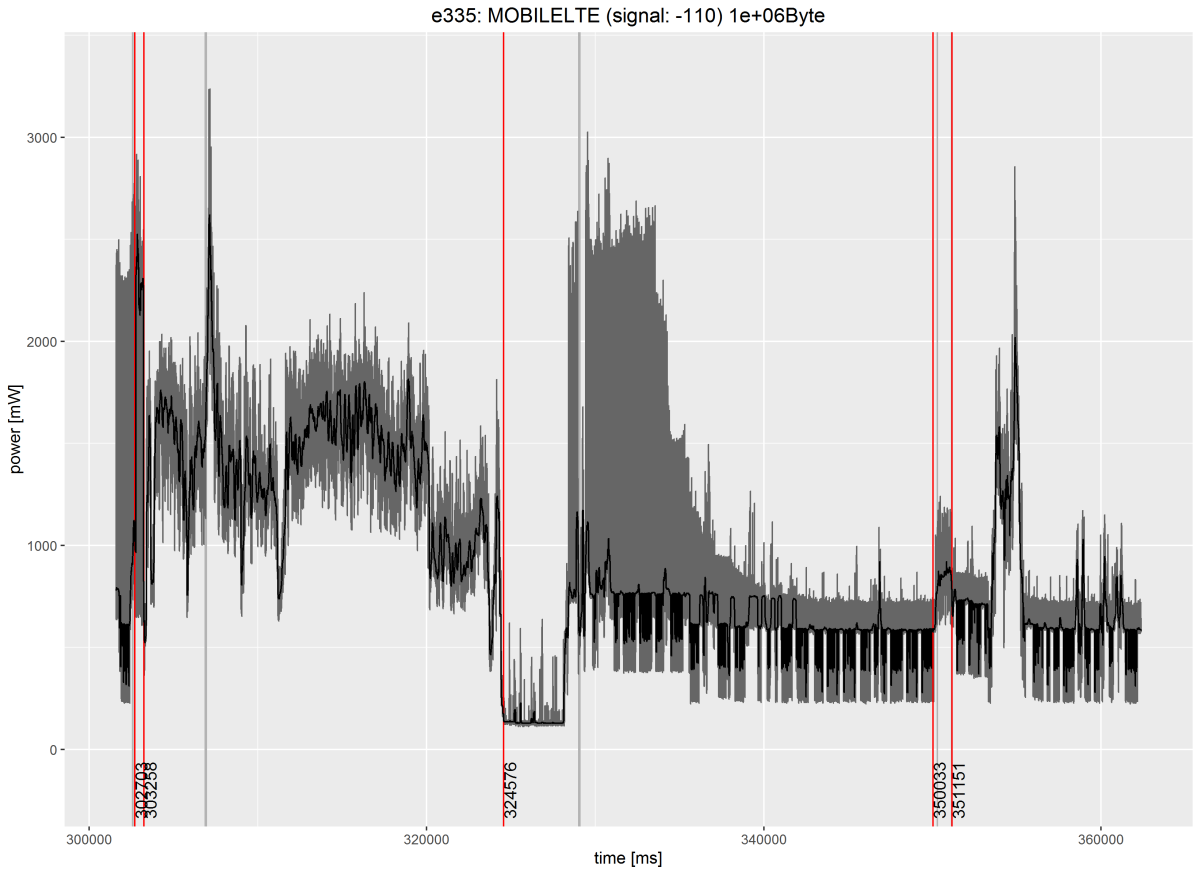

Figure 8.9: Example for mobility restrictions

section also considers signal changes and loss of connection to a type of signal (i.e. switching between different types or loosing any signal) as mobility.

The presented prediction model completely relies on the signal right before making the transmission and the assumption is that it stays more or less static. Of course especially in fast moving vehicles like cars and trains mobility is an important aspect as signal and cell changes can happen very fast.

In figure 8.9 the influence of signal loss (falling back to UMTS) can be observed. The data has been measured by entering an elevator during the transmission. At the beginning (visualized by the first red vertical line) the client starts to send a request in LTE with a signal strength of -110dBm. Shortly after the device lost the signal to the LTE cell it connects to an UMTS cell. Approximately 24s after loosing the signal the device can reconnect to the LTE cell (at time 329000, marked by a gray vertical line). Finally it can receive the response (between the two red lines at the right). Looking at the energy consumption the change of the network clearly produced much overhead.

The most difficult thing about mobility in a prediction model is, that the change of signal and/or networks has to be known or predicted beforehand. Considering the software architecture of the prediction model it should easily be possible to extend it based on a model predicting changes in signal as discussed in section 9.1.

8.4.2 Vendor Dependencies

Another restriction is that this concrete implementation of the model might only work properly with the used network device as specific behavior, like the tail time, can depend on the vendors specifications as these parameters are not specified in the standard as pointed out in [Cox12]. Additionally to the device side there is also a dependency on the service providers specific implementation. As the cell is responsible for assigning channels to the mobile device, it can of course define the possible rates. This is also typically done when a device exceeds its data limit defined in the service contract with the provider.

The best way to resolve these restrictions would be if vendors and providers provide detailed information about their parameters. Unfortunately this is hard to achieve as the providers are typically not publishing such data.

8.4.3 External Influence

As already seen in section 8.3.1 there are more influences on the energy consumption than signal and size. Especially a cell typically adapts the data rates to the number of mobile devices and the traffic they produce. The measurements have all been done on a university campus at similar times as earlier described. This leads to a more or less constant amount of devices in the cell. Still the traffic is not possible to control.

In real situations the traffic can hardly be considered to be constant: Imagining a sports event or a music festival there are many devices connected to a cell at the same time. Also the devices might use traffic-extensive applications like live-streaming the event or posting photos and videos to social networks. In this cases it would be ideal to have information about the number of users connected or information about the networks available capacity.

Also the model assumes that the wireless connection is the bottleneck of the connection in terms of throughput. This is often the case, but can be different. For example WiFi connections offered in public transport vehicles typically rely on mobile connections like LTE. In these cases often the WiFi connection is very good as the mobile device is near the base station, but the (shared) LTE connection may be very bad (which is typically

transparent to the user). That means the model may assume that a good WiFi connection means a high throughput, which is definitely not the case as the LTE connection is the bottleneck and reduces the throughput drastically.

9 Conclusions

The goal of this thesis was to find a prediction model predicting the energy consumption of wireless network connections specifically WiFi and LTE. The model should be as exact as possible and fast to calculate.

To elaborate deeper knowledge and basic data, an Android app has been implemented that can log information about the current network and communicate with a server in arbitrary patterns. The energy consumption of the mobile device performing the communication has been measured using an external device. The retrieved data has been analyzed on correlations between values and variables derived from the values. The results of this analysis lead to a prediction model that combines linear regressions of variables to a prediction of phases of the communication, which can be added up to gain a numeric prediction of the overall consumed energy. The implementation of the model has been considered to be extensible to allow future work to reuse the model or add further models for other network types.

The evaluation show that the calculated model performs better than basic linear regression on the data: For LTE the median absolute error has been evaluated as 14.39% lower, while for WiFi the median absolute error is 97.37% lower. Nevertheless there are some restrictions that fluctuation in the measured values and are not covered by the prediction models yet.

9.1 Future Work

This section sketches ideas and points out aspects that are still open.

As stated in the restrictions (cf. section 8.4) the currently used input parameters to make a prediction (size and signal) both need to be statically defined at the beginning. Usually the size can be estimated quite accurate before sending a request. Predicting the signal for at least the next 15 seconds in a mobile environment is a much bigger challenge. There are approaches like Bartendr as described in the related work section. If a working approach could be integrated with the proposed prediction model, it would be possible to use signal predictions as input for the prediction model and split up the resulting

energy in different phases with different signal strength or even different connection types.

Furthermore the restrictions (cf. section 8.4) of the model include problems with fluctuation in the energy consumption. To overcome these problems additional information on the status of the network are required. These information can hardly be predicted by a mobile device. If these information would be given by a service provider, which of course has knowledge about the load factor of its cells, it would be much easier to do reliable and more exact predictions.

Additionally the model is based on measurements with a specific network provider and on a specific device. Other network provider and other device may use different parameter and therefore differ in the energy consumption. There are three possible ways to overcome this restrictions: The first solution is that service providers and device vendors publish the values, so constants like the tail time can be used vendor-specific. The second solution would be to make additional measurements per vendor and service provider. This of course would mean a lot more measurements to collect reliable data, which would be quite a heavy overhead. Considering the idea of *PowerTutor* [ZTQ+10] it may thirdly be possible to use their approach in combination with the knowledge gathered from this work to improve their regression model for wireless communications (as they basically depend on a simpler model than the basic regression model presented in this thesis) and make it usable in any combination of phones and service provider.

Bibliography

[AIHF13] M. M. Ahamed, Z. Islam, S. Hossain, S. Faruque. “LTE network coverage prediction for multi-traffic users in an urban area.” In: *Electro/Information Technology (EIT), 2013 IEEE International Conference on*. May 2013, pp. 1–6 (cit. on p. 11).

[BBV09] N. Balasubramanian, A. Balasubramanian, A. Venkataramani. “Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications.” In: *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*. IMC ’09. Chicago, Illinois, USA: ACM, 2009, pp. 280–293 (cit. on pp. 6, 17).

[BDR13] P. Baier, F. Dürr, K. Rothermel. “Opportunistic Position Update Protocols for Mobile Devices.” In: *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp ’13. Zurich, Switzerland: ACM, 2013, pp. 787–796 (cit. on p. 17).

[Bok16] M. Bokhorst. *NetGuard on GitHub*. 2016. URL: https://github.com/M66B/NetGuard (cit. on p. 33).

[Cis16] Cisco. *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2015–2020*. Whitepaper. Cisco, 2016 (cit. on pp. 6, 7).

[Cox12] C. Cox. *An introduction to LTE: LTE, LTE-advanced, SAE and 4G mobile communications*. John Wiley & Sons, 2012 (cit. on pp. 6–10, 79).

[DDR15] C. Dibak, F. Dürr, K. Rothermel. “Numerical Analysis of Complex Physical Systems on Networked Mobile Devices.” In: *Mobile Ad Hoc and Sensor Systems (MASS), 2015 IEEE 12th International Conference on*. Oct. 2015, pp. 280–288 (cit. on p. 1).

[DK14] C. Dibak, B. Koldehofe. “Towards Quality-aware Simulations on Mobile Devices.” In: *GI-Jahrestagung*. 2014, pp. 89–100 (cit. on p. 1).

[Dür16] F. Dürr. *RPi-Powermeter*. 2016. URL: https://github.com/duerrfk/rpi-powermeter (cit. on p. 28).

[DWC+13] N. Ding, D. Wagner, X. Chen, A. Pathak, Y. C. Hu, A. Rice. “Characterizing and Modeling the Impact of Wireless Signal Strength on Smartphone Battery Drain.” In: *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems*. SIGMETRICS ’13. Pittsburgh, PA, USA: ACM, 2013, pp. 29–40 (cit. on p. 16).

[FVR07] P. Fuxjäger, D. Valerio, F. Ricciato. “The myth of non-overlapping channels: interference measurements in IEEE 802.11.” In: *2007 Fourth Annual Conference on Wireless on Demand Network Systems and Services*. Jan. 2007, pp. 1–8 (cit. on p. 34).

[HQG+12] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, O. Spatscheck. “A Close Examination of Performance and Power Characteristics of 4G LTE Networks.” In: *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*. MobiSys ’12. Low Wood Bay, Lake District, UK: ACM, 2012, pp. 225–238 (cit. on pp. 10, 15, 53).

[Hua13] J. Huang. “Performance and Power Characterization of Cellular Networks and Mobile Application Optimizations.” PhD thesis. University of Michigan, 2013 (cit. on p. 15).

[LLM+09] A. Larmo, M. Lindström, M. Meyer, G. Pelletier, J. Torsner, H. Wiemann. “The LTE link-layer design.” In: *IEEE Communications Magazine* 47.4 (Apr. 2009), pp. 52–59 (cit. on p. 9).

[LZZ11] H. Liu, Y. Zhang, Y. Zhou. “TailTheft: Leveraging the Wasted Time for Saving Energy in Cellular Communications.” In: *Proceedings of the Sixth International Workshop on MobiArch*. MobiArch ’11. Bethesda, Maryland, USA: ACM, 2011, pp. 31–36 (cit. on p. 17).

[Mot09] Motorola. *Realistic LTE Performance. From Peak Rate to Subscriber Experience*. Whitepaper. Motorola, 2009 (cit. on pp. 7, 9, 10).

[NRCG15] N. Nikzad, M. Radi, O. Chipara, W. G. Griswold. “Managing the Energy-Delay Tradeoff in Mobile Applications with Tempus.” In: *Proceedings of the 16th Annual Middleware Conference*. Middleware ’15. Vancouver, BC, Canada: ACM, 2015, pp. 259–270 (cit. on p. 17).

[SDEG12] S. Shankar N., D. Dash, H. El Madi, G. Gopalakrishnan. “WiGig and IEEE 802.11ad - For multi-gigabyte-per-second WPAN and WLAN.” In: *ArXiv e-prints* (Nov. 2012). arXiv: 1211.7356 [cs.NI] (cit. on p. 5).

[SNR+10] A. Schulman, V. Navda, R. Ramjee, N. Spring, P. Deshpande, C. Grunewald, K. Jain, V. N. Padmanabhan. “Bartendr: A Practical Approach to Energy-aware Cellular Data Scheduling.” In: *Proceedings of the Sixteenth Annual International Conference on Mobile Computing and Networking*. MobiCom ’10. Chicago, Illinois, USA: ACM, 2010, pp. 85–96 (cit. on pp. 16, 49).

[TW11] A. S. Tanenbaum, D. Wetherall. *Computer networks*. 5. ed., international ed. Boston [u.a.]: Pearson, 2011, 951 S. (Cit. on pp. 5, 6, 11–13).

[XMJC11] H. Xian, W. Muqing, M. Jiansong, Z. Cunyi. “The impact of channel environment on the RSRP and RSRQ measurement of handover performance.” In: *Electronics, Communications and Control (ICECC), 2011 International Conference on*. Sept. 2011, pp. 540–543 (cit. on p. 11).

[ZTQ+10] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, L. Yang. “Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones.” In: *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. CODES/ISSS ’10. Scottsdale, Arizona, USA: ACM, 2010, pp. 105–114 (cit. on pp. 17, 82).

All links were last followed on September 23, 2016.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature