TU WIEN Informatics

Agenten-basiertes und ressourcenschonendes Benchmarking in zellularen Mobilfunknetzen

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Wolfgang Hofer, BSc
Matrikelnummer 1126742

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Mitwirkung: Senior Scientist Dipl.-Ing. Dr.techn. Philipp Svoboda

Wien, 30. Jänner 2020

Wolfgang Hofer	Wolfgang Kastner

TU WIEN Informatics

Agent-based non-intrusive methodology for characterizing reactive mobile communication networks

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Wolfgang Hofer, BSc
Registration Number 1126742

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Assistance: Senior Scientist Dipl.-Ing. Dr.techn. Philipp Svoboda

Vienna, 30th January, 2020

Wolfgang Hofer	Wolfgang Kastner

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Wolfgang Hofer, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 30. Jänner 2020

Wolfgang Hofer

Acknowledgements

Firstly, I want to thank my supervisors Wolfgang Kastner and Philipp Svoboda for giving me the opportunity to work on this diploma thesis and for their continuous support, patience, and immense knowledge.

Furthermore, I am indebted to my many colleagues at the Institute of Telecommunications who helped me and made the time of writing the thesis enjoyable.

Finally, I owe my deepest gratitude to my family and friends for their encouragement and patience. This thesis would not have been possible without them.

This research was supported by A1 and the SEMONE FFG project. I have greatly benefited from accessing A1's anechoic chamber with the reference cell.

Kurzfassung

Die fünfte Generation an Mobilkommunikationsnetzen (5G) steht vor der Tür. Ihre Flexibilität macht auch ihre Charakterisierung für Benutzer höchst interessant. Ein fairer Leistungsvergleich von Kommunikationsnetzen ist schon seit Jahrzehnten ein Forschungsthema. Im Gegensatz zu drahtgebundenen Netzwerken fehlt jedoch in Mobilfunknetzwerken ein schnelles, genaues und datensparendes Programm, um die Eigenschaften des Netzwerks auf einem Handy zu beschreiben. Deshalb ist es das Ziel dieser Arbeit, eine Softwareanwendung zu entwickeln, die Informationen über die aktuellen Eigenschaften des Netzwerkes anzeigen und an einen Schwarm ausgelagert werden kann. Um dieses Ziel zu erreichen, wird in dieser Arbeit nach einer schnellen und datenschonenden Methode gesucht, die reaktive Mobilfunknetze auf einem Handy charakterisiert. Reaktiv bedeutet in diesem Zusammenhang, dass das Netzwerk seine Eigenschaften mit eingespeisten Daten ändert.

Basierend auf einer Literaturrecherche über geeigneten Methoden zur Charakterisierung von Mobilfunknetzen wurde die schnelle, datensparende und exakte Methode *CRUSP* ausgewählt. Anschließend wurde CRUSP in einem Softwaresystem realisiert, das die Durchführung, Erfassung und Analyse von Messungen ermöglicht. Dazu zählt eine Android-Anwendung, die schnellen und einfachen Zugriff auf Ergebnisse liefert. Die Evaluierung der Software wurde in einer konfigurierbaren und abgeschirmten Referenzzelle durchgeführt und liefert einen durchschnittlichen Fehler von 2,44% zu iPerf3, einem beliebten Referenzprogramm. Die Ergebnisse zeigten, dass die entwickelte Software in Long-Term Evolution-Netzwerken (LTE) exakt misst. Jedoch zeigte ein Versuch mit aktivierter Carrier Aggregation (mit Datenraten von rund 420 MBit/s) große Differenzen zu Referenzmessungen. Weitere Forschungsarbeiten sind im Bereich der Validierung für aktive 5G-Netzwerke sowie in der Ermittlung optimaler CRUSP-Konfigurationen für verschiedenste Bedingungen erforderlich.

Abstract

The fifth generation of mobile communication networks (5G) is just around the corner, and its flexibility makes the characterization of it interesting for end users. Fair benchmarking of communication networks has been a research topic for decades. In contrast to wired networks, mobile networks lack a quick, accurate, and data-saving tool to describe the properties of the network on a user device. Therefore, this thesis aims to create a crowdsourcing-ready software application capable of showing information about the current network conditions. In order to reach this goal, this thesis searches for a quick and data-saving methodology on user devices that characterizes reactive mobile communication networks. In this context, reactive states that the network changes its properties on injected traffic.

Based on a literature research on the methods to characterize mobile communication networks, the quick, data-saving, and accurate method *CRUSP* was selected. Afterwards, CRUSP was realized in a software system for performing, gathering, and analyzing measurements. This included an Android application for fast and easy access to results. The evaluation of the software was performed in a configurable and shielded reference cell and delivered an average error of 2.44% to iPerf3, a popular reference tool. However, an experiment with activated carrier aggregation (with data rates of about 420 MBit/s) showed substantial differences to reference measurements. In conclusion, the results indicated that the developed software operates accurately in Long-Term Evolution (LTE) networks. Further research is needed for validation in a 5G network and to identify optimal CRUSP configurations for different conditions.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Internet access is an omnipresent part of today's life. We use it to read news, watch videos, or send messages. With the emergence of the fifth generation of mobile communication networks (5G) in 2020, billions of new devices will connect to the Internet. Devices like smartphones, drones, or sensors will produce, consume, and process a massive amount of data. Thereby, a large amount of data will be transferred via mobile radio networks.

All the applications on these devices come with distinct Quality of Service (QoS) requirements. For example, health care applications frequently decide about life and death. Subsequently, such applications need ultra-reliable communication channels with low latency in sub-milliseconds range. In contrast, video streaming applications consume a vast amount of data. Therefore, communication channels need to achieve high data rates.

That is why the Radiocommunication Sector of the International Telecommunication Union (ITU-R) has defined [1] three usage scenarios for 5G networks:

- *Enhanced Mobile Broadband* (eMBB): Provides data rates up to 20 Gbit/s [2, p. 6].
- *Ultra-Reliable Low-Latency Communications* (URLLC): Ensures low latencies (less than 1ms) and reliable communication [2, p. 6].
- *Massive Machine Type Communications* (mMTC): Allows millions of connected devices ($1.000.000/km^2$) [2, p. 6].

However, there is no one-size-fits-all configuration of the network. As we can see in Figure 1.1, each application needs to make tradeoffs in order to meet its QoS requirements. For example, performance-oriented applications may have a high bit rate and low latency, but they pay for those properties with a low battery life and inefficient energy management. In short, 5G networks need to be flexible to support a wide variety of usage scenarios.

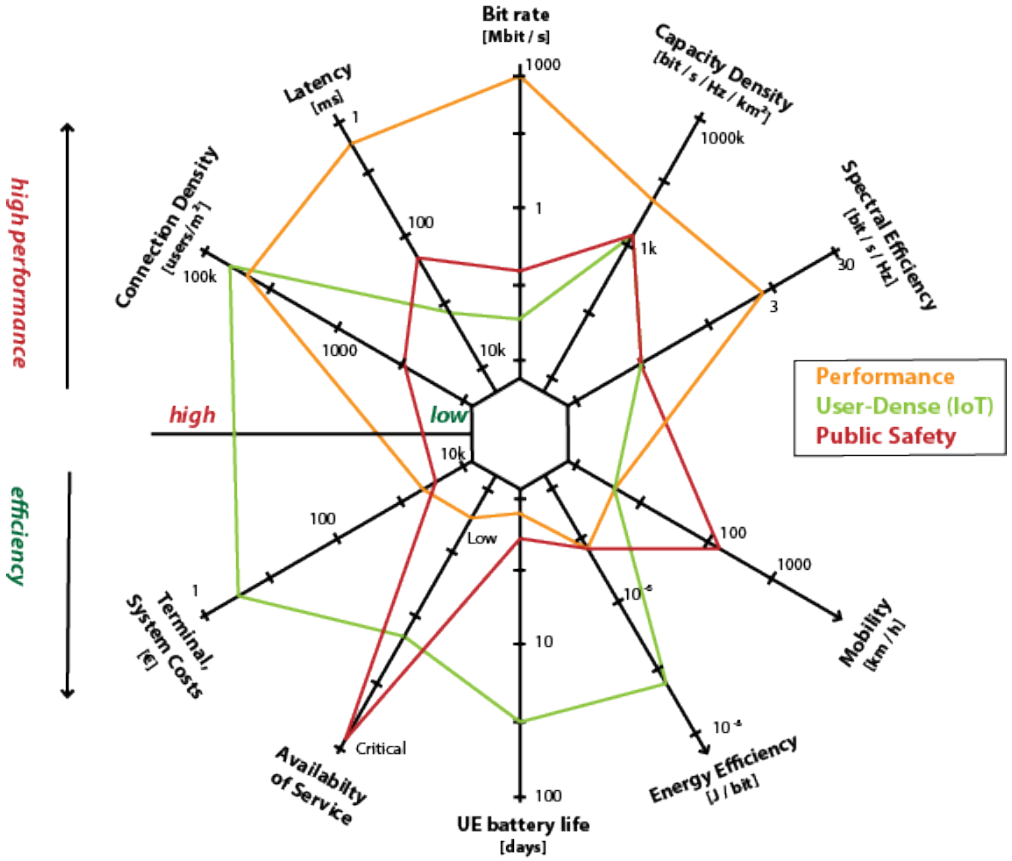

Figure 1.1: Service profiles for 5G, from [3]

Traditional mobile communication networks like Long-Term Evolution (LTE) use a universal architecture to service all connected devices. However, this architecture is not suitable for 5G; it needs a more flexible approach to support multiple usage scenarios. Indeed, technologies like *Software Defined Networking* (SDN), *Network Functions Virtualization* (NFV), and *Fog Computing* can provide an adaptable core network. Further, smaller cells can support the frequency spectrum expansion to the millimeter-wave range. In conjunction with more and more intelligent devices, these disruptive changes enable a shift to a device-centered architecture that enables device-to-device (D2D), machine-to-machine (M2M), and vehicle-to-vehicle (V2V) communication [4].

In a device-centered architecture, the device wants to know the currently available network performance characteristics. This information enables the device, which is acting as an *agent* in the network, to assess if the QoS level is sufficient. However, current tools to assess these characteristics are often slow, data-hungry, or inaccurate when deployed in mobile networks. Consequently, it is necessary to find agent-based methods for a fair benchmarking of current (4G) and future (5G) reactive mobile networks.

1.2 Aim

This thesis aims to *find an agent-based non-intrusive methodology for characterizing reactive mobile communications networks*. Thereby *reactive* states that the network changes its properties when a user injects traffic into it. This behavior is typical in traditional mobile communication networks like LTE. Furthermore, the term *non-intrusive* means fast and data-saving.

To be prepared for the challenges of the future, the focus will be on the upcoming 5G mobile communication networks. In order to answer the research question, the thesis breaks it down into several subquestions:

- Which non-intrusive methods exist to characterize reactive mobile communication networks?
- Which method is most suitable for implementation on an agent?
- How can someone build an agent-based solution using the most suitable method?
- Are the results produced by the developed solution accurate?
- For which kind of applications in 5G networks could this tool be useful?
- What are the preconditions to use this solution in 5G networks?

1.3 Contributions and Methodology

Scientists have been characterizing communication networks for decades [5, 6]. However, the characterization of reactive mobile networks poses multiple challenges.

In order to master these challenges and answer the research question, the following stepwise approach is taken:

- Analysis of the state of the art of agent-based non-intrusive methods for characterizing reactive mobile communication networks. This part of the work is carried out using literature research.
- Selection of the most suitable method for agents based on quantitative criteria. The state of the art serves as a basis for this choice.
- Implementation of the chosen method in an Android application called *CRUSP for Mobile*. Furthermore, a scalable software system to collect and analyze the results of the Android application was developed.
- Evaluation of the created Android application in terms of accuracy in a configurable shielded reference cell.
- Elaboration of requirements and use cases for deployment in 5G.

1.4 Structure

The diploma thesis is structured as follows. At first, Chapter 2 deals with the characterization of wired and WiFi networks. Afterwards, Chapter 3 analyzes methods for characterizing reactive mobile communications networks. Additionally, it selects the most suitable method for implementation.

Chapter 4 describes the software development process of the selected method. After that, Chapter 5 verifies the accuracy of the constructed software. In Chapter 6, the thesis elaborates on requirements for a successful deployment in a 5G network. Further, it describes possible use cases in a 5G network.

Finally, Chapter 7 summarizes the findings of this thesis. Furthermore, it takes a look at possible limitations and future work.

CHAPTER 2

Related Work

For decades scientists have been studying the characterization of communication networks [5, 6]. However, for a long time, the focus has been on wired networks and wireless networks like Wi-Fi. Therefore, a solid knowledge of those networks in static and nomadic usage exists. This chapter will summarize the relevant basics to characterize networks. The first section looks at the characteristics when describing communication networks. The second section focuses on the characteristic *bandwidth* and metrics representing it. Afterwards, the thesis presents an overview of different bandwidth estimation techniques. The last section lists an excerpt from the existing tools that describe the performance of wired and wireless communication networks.

2.1 Characterization of Networks

As a first step, the thesis identifies relevant characteristics to describe communication networks in general.

2.1.1 Performance Characteristics

Scientists describe the performance characteristics of communication networks from different points of view. In 1997, Paxson et al. [5] defined four important characteristics for them: *latency*, *packet loss*, *congestion level*, and *bandwidth*. Thirteen years later, Brockmeyer et al. [7] identified slightly different network characteristics. They presented the same ones as Paxson et al. but dropped *congestion level* and added *path detection* instead. In contrast, Tanenbaum et al. [6, p. 405] choose a different perspective. They were looking for QoS parameters, which a flow needs in order to describe its performance. Thereby, a flow is a stream of packets between a source and destination host. Nevertheless, they also found the three primary characteristics: *bandwidth*, *delay* (latency), and *loss* (packet loss). Additionally, they identified *jitter* as an essential characteristic. In summary, scientists identified three core characteristics and one alternating one.

The following paragraphs describe the mentioned characteristics above based on Brockmeyer et al. [7] definitions.

Latency When a message is traveling through a network, it passes multiple routers. Thereby, each packet can take a different route through the network. In short, latency is the measured time a message needs to travel from a sending host to a destination host.

When talking about latency, researchers distinguish between *One-Way Delay* (OWD) and *Round Trip Time* (RTT). OWD is the time a message takes from a sending host to a destination host. The Internet Engineering Task Force (IETF) defines OWD more precisely. They measure the time from the first bit leaving the sending host till the last bit is received at the destination host [8].

In contrast, the RTT measures the time a message travels from the sending host to the destination host and then right back to the sending host in one go [9]. Quick and lightweight methods already exist for both key figures. For example, when measuring OWD, a client can send an Internet Control Message Protocol (ICMP) packet, the Transmission Control Protocol (TCP), or the User Datagram Protocol (UDP) to a server. Afterwards, the hosts record the timestamps when sending and receiving the packet. The difference in time represents the measured OWD. Nevertheless, OWD requires synchronized clocks of the sender and receiver.

When measuring the RTT, a simple HTTP request and HTTP response is enough. Thereby, the sending host records the timestamps of the request leaving and the response arriving back at the host. The elapsed time between the two timestamps represents the RTT. In contrast to OWD, the clocks do not need to be synchronized. However, an ICMP ping is different from a TCP ping; they use different packet-sizes. Therefore the latency can be slightly different.

A low latency is useful when selecting the nearest server to minimize transmission time. Furthermore, it is a crucial factor when performing video conferencing [6, p. 405].

Packet Loss This characteristic describes the rate of dropped packets along a path while sending messages from a source host to a destination host. However, this characteristic has some pitfalls. Congestion control algorithms often interpret packet loss as a sign of congestion. Admittedly, wireless connections loose packets all the time [6, p. 539]. This phenomenon is also a common problem in cellular networks [10]. For example, a hardware failure can block, or an overloaded buffer can drop packets. Consequently, congestion control algorithms can unnecessarily slow down connections with packet loss.

Furthermore, detecting packet loss can be challenging. The naive approach waits for the packet arrival confirmation (e.g., acknowledge signal in TCP). If the packet arrives in a predefined time, it counts as delivered. However, if the confirmation arrives too late or does not arrive at all, it counts as not received.

Unfortunately, it is hard to predefine a duration after which a packet counts as not received. If the point is too early, the sender produces additional network traffic by

unnecessarily resending the packets. In contrast, if the point is too late, application performance suffers from long waiting times for the resent packets. Taken together, lost packets degrade application performance (e.g., pixelated artifacts in videoconferencing via UDP).

Path Detection Each message takes a specific *network path* from a source host to a destination host. Thereby the path through a network represents the actual sequence of routers the message traverses. Detecting and reporting all those routers with their Internet Protocol (IP) addresses and names is called *path detection*.

A common strategy for path detection uses the time-to-live (TTL) information in the IP header. Thereby the TTL represents the amount of remaining routers a message can pass before it gets dropped. Each time the message passes a router, the TTL is decreased by one. If the current router on the path detects a TTL of 1, it drops the message. Furthermore, it reports a *time exceeded* message, including its IP address to the sender.

So how is TTL used to determine the message path through the network? At first, the source host sends multiple messages to the destination host. Thereby, the source host increases the TTL for each sent message. As a result, the source host gets a series of *time exceeded* messages along the messages's path. Based on these response messages, the source host can reproduce the path the packets must have taken to reach the destination host.

Path detection is useful when searching for network failures or determining the topology of a network.

Jitter Tanenbaum et al. [6, p. 406] defines *jitter* as "(standard) variation in the delay". For example, consecutive packets can have different delays, even if they take the same path. Furthermore, if single packets take different paths through the network, they can arrive in a different order.

Low jitter is essential for a good video quality in video streaming. When the packets arrive at irregular intervals, they can not be put together in the right order. Thus, jitter can cause packet loss, and the video quality decreases rapidly. Luckily, buffering can smooth this problem.

Bandwidth The term *bandwidth* refers to the maximum data transfer rate from a source host to a destination host. Unfortunately, this description is ambiguous. Is it the maximum data transfer rate currently available to the user? Or is it the maximum data transfer rate achievable in perfect conditions (e.g., without cross-traffic along the path)? The next section will take a closer look at the metrics to describe the bandwidth.

Nevertheless, bandwidth is an essential factor for fast file sharing. Another particularly important use case is video streaming. According to the *Cisco Visual Networking Index: Forecast and Trends, 2017–2022* [11], video streaming will be responsible for 85% of the

Internet traffic in 2022. Thus, determining the best possible video quality while taking the available bandwidth into account can improve the streaming experience significantly.

2.2 Bandwidth Estimation Metrics

The term *bandwidth* originates from the field of telecommunication. There, it describes the width of a frequency band. However, the meaning of the term changed in digital communications. Digital bandwidth refers to the amount of data transmitted over a unit of time. Common units for digital bandwidth are for example bits per second (b/s) or megabits per second (Mb/s, Mbit/s, or Mbps). In this thesis, the term *bandwidth* will always refer to the concept of *digital bandwidth*.

Various metrics are associated with bandwidth. Salcedo et al. [12] identified four of them: *Capacity*, *Available Bandwidth* (ABW), *Bulk Transfer Capacity*, and *OWD*. Other papers [13, 14, 15] put the focus on just three metrics: *Capacity*, *ABW*, and *Bulk Transfer Capacity / Achievable Throughput*. Additionally, Dovrolis et al. [16] defined a metric called *Asymptotic Dispersion Rate* (ADR). However, this metric sees little use.

The following paragraphs present a short description of the most exciting metrics.

Capacity When talking about capacity, most papers distinguish between end-to-end capacity and per-hop (or per-link) capacity. Thereby, the term *end-to-end* always represents the complete path between two hosts. In contrast, the term *per-hop* identifies the metric for each router along the path [17]. In short, capacity is the maximum amount of data a link/path can carry. Table 2.1 shows a detailed definition of *capacity*.

Term	Definition
Link	Is a physical link between two nodes in a network. An end-to-end path is made up of n links $l_1, l_2, \cdots, l_n$.
Capacity	The capacity B of a link is defined as the maximum bit rate a single link is able to sustain. The capacities of a end-to-end path are $B_1, B_2, \cdots, B_n$.
Bottleneck link Narrow link	The link with the smallest capacity along a path $B_b = min(B_1, B_2, \cdots, B_n)$.
Tight link	The link $l_t (1 \leq t \leq n)$ with the min. available bandwidth $B_t - C_t = min(B_1 - C_1, B_2 - C_2, \cdots, B_n - C_n)$ along a path where $C_1, C_2, \cdots, C_n$ are the traffic loads on the links.
Available bandwidth	Is the unused bandwidth $B_t - C_t$ available on the tight link for any end-to-end transmission.

Table 2.1: The definition of ABW from [18]

Available Bandwidth (ABW) The previous Section 2.1.1 already introduced the concept of ABW. It is the remaining bandwidth after subtracting the cross-traffic from the capacity. Table 2.1 lists a detailed specification of the metric ABW.

Bulk Transfer Capacity (BTC) / Achievable TCP Throughput Another way to describe the transfer performance of a TCP connection is the bulk transfer capacity (BTC) [14, 19]. Mathis and Allman [20] defined BTC as a "measure of a network's ability to transfer significant quantities of data with a single congestion-aware transport connection". It is important to note that BTC metric focuses on a single connection and a protocol with congestion control. Equation 2.1 shows the exact definition of it. Thereby *data_sent* expresses the unique data bits transferred (excluding the header), and *elapsed_time* the elapsed time in seconds.

$$BTC = \frac{data\_sent}{elapsed\_time} \tag{2.1}$$

In contrast, the *Achievable TCP throughput* is a particular version of BTC; it just focuses on the protocol TCP. However, different implementations of TCP lead to different results in TCP throughput measurements [21]. Therefore, comparisons between TCP-based tools need to consider those differences.

In summary, multiple metrics exist to describe bandwidth. The most suitable one is ABW as we will see in the next Chapter 3.1.

2.3 Bandwidth Estimation Techniques

After identifying various metrics to characterize the bandwidth of a network path, let us take a closer look at the techniques to determine them.

2.3.1 Types of Network Monitoring

In general, Paxson [22] divides network monitoring into two categories: *active* and *passive*.

Active Monitoring In active network monitoring, the client injects self-generated traffic into the network. For example, when measuring the RTT, the client sends an ICMP packet.

Paxson [22] identified two disadvantages for active monitoring. Firstly, they add a new load to the network. This additional traffic can lead to degradation of existing connections. Secondly, additional traffic can manipulate the measurement result. For example, vast amounts of injected data can congest the network. As a consequence, the network may drop packets and then produce additional traffic by resending them (e.g., with TCP). Paxson calles this phenomenon the *Heisenberg effect*.

Passive Monitoring In contrast to active monitoring, passive monitoring analyzes traffic that already exists. For instance, when calculating the packet loss, the client can use the acknowledge signal of TCP [7]. Thereby, the client does not inject additional network traffic.

Another benefit of passive monitoring is the number of devices in the network which need additional equipment. Only one device, for example, a router, needs to execute the monitoring tool. In contrast, active monitoring often needs to equip two devices, a sender and a counterpart.

Unfortunately, passive monitoring needs a continuous flow of network traffic to provide results [7]. Thus, useful and valid results (e.g., for a specific time and space) are often unavailable.

In short, passive monitoring delivers limited results when used on User Equipment (UE). However, it works well on network infrastructure like routers. Unfortunately, network operators usually deny access to critical infrastructure. Consequently, an active method is more suitable for agent-based characterization of reactive mobile communication networks.

2.3.2 General Categorization of Bandwidth Estimation Techniques

Simple techniques estimate the ABW by generating a maximum level of traffic for a specific period of time. For example, the *RTR NetTest* [23] performs a 7-second test with an exponentially increasing data volume[1].

More sophisticated techniques use various approaches to estimate the bandwidth. In general, scientists divide the sophisticated techniques into three types: *model based*, *passive*, and *active* [24].

Model-Based In order to estimate the bandwidth, model based approaches create mathematical models of the traffic. However, this approach is highly dependent on the network topology [25].

Passive Passive techniques observe and analyze existing traffic in the network via wiretapping. Thereby, a significant advantage over active techniques is the possibility to analyze already existing traffic. Thus, the techniques do not need to inject additional traffic. Unfortunately, their dependence on already existing traffic is inconvenient for agent-based characterization. Besides, they are uninformed about any rate limitations from content providers. For example, video streaming services may have introduced a rate limitation for streaming their videos. Nevertheless, researchers already produced detailed reviews of passive bandwidth monitoring [7, 13, 15, 24, 26].

Active These techniques use lightweight probe packets to estimate the ABW. Thereby, probe packets can differ in size, amount, or time in-between packets. Strauss et al. [18] divided the active techniques into two main approaches: *Probe Gap Model* (PGM) and *Probe Rate Model* (PRM). In contrast, Yuan et al. [27] also defined a third category called *Packet Dispersion Measurement (PDM)*. However, other authors [12, 15, 28] took

[1] `https://github.com/rtr-nettest/open-rmbt/blob/master/RMBTClient/src/at/rtr/rmbt/client/RMBTTest.java`

a another approach when organizing the techniques. They identified four different groups: *Variable Packet Size*, *Packet Pair/Train Dispersion*, *Self-Loading Periodic Streams*, and *Trains of Packet Pairs*. A detailed description of the active techniques is given in the Subsection 2.3.4 below.

Moreover, Arsan [28] distinguishes between techniques suited for wired networks and suited for wireless networks. In the case of wired networks, one can further distinguish between *passive estimation* and *active probing* techniques [28, 29]. However, both groups of techniques are already part of Chaudhari et al. [24] general grouping. In contrast, wireless networks use a different categorization.

2.3.3 Categorization of Wireless Bandwidth Estimation Techniques

Wireless networks operate differently than wired networks. They use a shared medium, and therefore, the channel conditions change permanently. Consequently, wireless networks use a different categorization than wired networks. Yuan et al. [30] and Arsan [28] divide the bandwidth estimation techniques for wireless networks into three categories: *Probing-Based*, *Cross Layer-Based*, and *Model-Based*.

Probing-Based These techniques uses *probes* like *packet pairs* or *packet trains* [28]. A packet pair corresponds to two consecutive sent packets with the same size. In contrast, a packet train consists of multiple packet pairs.

In probing-based techniques, the source host sends probes to a destination host. On arrival, the destination host records the corresponding timestamps for each probe. Finally, these techniques can guess the ABW in combination with the sending-timestamps of the probes. A more elaborate explanation of these techniques is in the next Subsection 2.3.4. A few examples are the tools Wbest [31], DietTOPP [32], AdhocProbe [33], and ProbeGap [34].

Cross Layer-Based In order to get an accurate result, the cross-layer technique utilizes multiple layers of the Open Systems Interconnection (OSI) model. In the first step, the client collects information at a lower OSI level. For example, at the *medium access control layer* (MAC-layer). Unfortunately, this often requires changes in MAC protocols. Afterwards, the lower-level layer sends the information to the application layer. There, the tool applies intelligent post-processing to the collected data to determine the relevant metric [30]. IdleGap [35] and iBE [27] are two examples of this technique.

Model-Based This approach renounces probing and manipulation of protocols. Instead, it tries to model the TCP/UDP throughput. A more detailed description presents the paper of Airon and Gupta [15]. Based on this technique, Yuan et al. [30] developed a tool called *MBE*.

Taken together, three different types of techniques exists in wireless bandwidth estimation. However, the *active* and *probing-based* techniques are the most promising for this thesis due to its applicability to agents.

2.3.4 Active Bandwidth Estimation Techniques

The following paragraphs describe the different techniques when actively estimating the bandwidth. However, there are multiple approaches to group them. For example, Yuan et al. [27] defined three categories: *Packet Dispersion Measurement* (PDM), *Probe Gap Model* (PGM), and *Probe Rate Model* (PRM).

Packet Dispersion Measurement (PDM) The technique focuses on the inter-arrival time of packets. Based on this data, it estimates the capacity of the network with a total data volume of just a few kB. In contrast, traditional estimations tools like Ookla's speedtest.net [36] consume up to hundreds of MB. Unfortunately, low estimation accuracy makes PDM-tools unhandy in wireless networks [27]. One example is bprobe [37].

Probe Gap Model (PGM) This model also uses the time gap between two consecutive arrived probes to infer the ABW [18]. In contrast to PDM, PGM additionally takes the dispersion of the packets when sending into account.

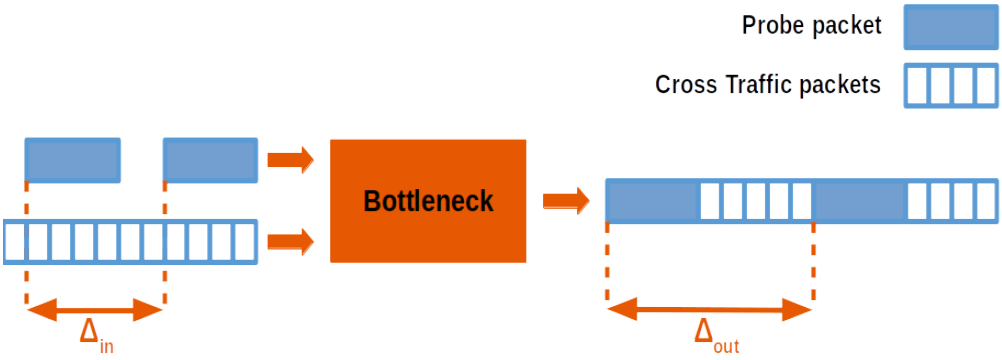

Figure 2.1: PGM from [18]

For example, Figure 2.1 illustrates the PGM with a single bottleneck, probe packets, and some cross traffic. When the two probe packets arrive at the bottleneck, they get interwoven with the cross-traffic. This process creates a filled queue with a new time gap between the two probes. Thereby Δ_{out} denotes the time gap between two probes when sending them and Δ_{in} indicates the time gap between the two probes when arriving. Based on the two distances Δ_{out} and Δ_{in}, PGM-techniques can estimate the ABW with a few hundreds of kB.

Examples are the tools Spruce [18], IGI / PTR [29], Delphi [38], Traceband [39], and Abing [40].

However, the PGM-technique is only useful with single-hop estimations. Furthermore, the accuracy depends on the amount of cross-traffic [15].

Probe Rate Model (PRM) In contrast to the PGM, the PRM uses self-induced congestion and the difference in sending and receiving rate of the probes to conclude the ABW [18]. For this purpose, the sender transmits probes with a predefined rate. If sending and receiving rates are equal, PRM assumes that the tight link is uncongested. Consequently, the path can carry data at an even higher data rate.

On the contrary, if the receiving rate is lower than the sending rate, too much traffic congests the network. This phenomenon is an indicator that the rate is too high. Consequently, the PRM searches for the turning point at which the rates turn from too high to too low. This process can inject up to a few MB of additional traffic into the network. The rate at the turning point represents the ABW. Examples are Pathload [41], Pathchirp [38], TOPP [42], BART [43], and Yaz [44].

Nevertheless, a few other authors [12, 15, 28] chose a different approach when examining the techniques. They identified following four techniques: *Variable Packet Size* (VPS), *Packet Pair/Train Dispersion* (PPTD), *Self-Loading Periodic Streams* (SLoPS), and *Trains of Packet Pairs* (TOPP).

Variable Packet Size (VPS) This technique measures the capacity for each link along a path [12, 15]. Arsan [28] describes the idea behind it by measuring "the Round-Trip Time (RTT) from the source to each hop of the path as a function of the probing packet size". The tools *clink* [45] and *pathchar* [46] are two examples of this technique.

Packet Pair/Train Dispersion (PPTD) PPTC estimates the end-to-end capacity for a path using packets with constant size [12, 15]. In the end, the receiving host utilizes the dispersion of the arrived packets to calculate the capacity. The techniques in this group are also known as *PGM techniques*. However, PPTD tools like cprobe [37] can also estimate the metric ADR. This metric is related to ABW but not the same [16]. Two other examples of PPTD tools are *pathrate* [16] and *Sprobe* [47].

Self-Loading Periodic Streams (SLoPS) This technique produces a self-induced congestion at the tight link in order to find the ABW. At first, the sender sends a periodic packet stream (which are packets of the same size) with a predefined data rate. Afterwards, the receiver collects the OWDs of the packets to calculate the receiving rate. The goal of SLoPS is to find the point where the tight link starts to congest. If the calculated rate is lower than the sending rate, the receiver assumes a congestion. Consequently, the receiver informs the sender to decrease the rate for the next stream. Otherwise, the sender should increase its rate. The data rate at the point at which the congestion starts is the ABW [12]. In short, SLoPS uses a binary search to find the turning point in the data rate difference. Nevertheless, the challenge remains to choose a

suitable starting data rate and suitable adjustments. Examples are Pathload [41], IGI [29] or pathchirp [38].

Trains of Packet Pairs (TOPP) TOPP follows a similar approach than SLoPS. It sends multiple packet pairs (denoted as a train) with a predefined rate to the receiver. In contrast to SLoPS, the sending rate in TOPP increases linearly. At the point of congestion, the sending rate exceeds the receiving rate. Finally, the technique estimates the ABW from the collected receiving timestamps of the packets at the turning point [12, 15]. Here too, it is essential to choose a reasonable start data rate and a suitable function to increase the data rate increase. The tools DietTopp [32] and iperf [48] are examples for this technique.

2.4 Bandwidth Estimation Tools

After getting to know many different techniques for bandwidth estimation, let's take a look at the tools. Over the years, researchers created many tools to estimate the bandwidth. At the beginning, the focus was on wired networks. With the advent of Wi-Fi, research shifted to the area of wireless networks.

2.4.1 Bandwidth Estimation Tools for Wired Networks

The following Table 2.2 shows an incomplete list of existing tools for ABW estimation. Thereby, the focus is on active monitoring. Furthermore, multiple authors [13, 49, 50] carried out a comparison between different tools.

Tool	Year	Author	Metric
pathchar	1997	Jacobson	Per-hop Capacity
clink	1999	Downey	Per-hop Capacity
pchar	1999	Mah	Per-hop Capacity
bprobe	1996	Carter	End-to-End Capacity
pathrate	2001	Dovrolis, Prasad	End-to-End Capacity
Sprobe	2002	Saroi	End-to-End Capacity
Cprobe	1996	Carter	ADR
Pathload	2002	Jain, Dovrolis	End-to-End ABW
IGI / PTR	2002	Hu	End-to-End ABW
pathchirp	2003	Ribeiro	End-to-End ABW
Abing	2003	Navratil	End-to-End ABW
Spruce	2003	Strauss	End-to-End ABW
FEAT	2006	Wang	End-to-End ABW
BART	2006	Ekelin	End-to-End ABW
YAZ	2007	Sommers	End-to-End ABW
ASSOLO	2009	Goldoni	End-to-End ABW
Treno	1996	Mathis	BTC
Cap	2001	Allman	BTC
Ttcp	1984	Muuss	Achievable TCP Throughput
Sting	1999	Savage	Achievable TCP Throughput
iPerf	2003	NLNAR	Achievable TCP Throughput
Netperf	N/A	NLNAR	Achievable TCP Throughput

Table 2.2: Bandwidth estimation tools for wired networks, based on [28] [13]

2.4.2 Bandwidth Estimation Tools for Wireless Networks

Wireless networks are different than wired networks. They use different link-layer protocols [51]. As a result, they face additional challenges. For example, they usually lose more packets than wired ones. Furthermore, the lost packets are often interpreted as congestion along the path [6, p. 539]. Unfortunately, tools, developed for wired networks,

deliver inaccurate results. For instance, PDM-based tools have very low accuracy in wireless networks [27]. Consequently, bandwidth estimation tools for wireless networks need to adapt their measurement techniques. As an overview, Table 2.3 lists many bandwidth estimation tools for wireless networks.

Tool	Year	Author	Measurement Metric
Wbest	2008	Claypool	Per-hop Capacity
DietTOPP	2004	Johnsson	End-to-End ABW
ProbeGap	2004	Lakshminarayanan	End-to-End Capacity
AdhocProbe	2005	Sun	End-to-End Capacity
IdleGap	2007	Lee	End-to-End Capacity
iBE	2009	Yuan	End-to-End Capacity
MBE	2012	Yuan	End-to-End Capacity
BEST-AP	2013	Dely	End-to-End Capacity

Table 2.3: Bandwidth estimation tools for wireless networks, based on [28]

Summary Taken together, many tools for bandwidth estimation in wired and wireless networks exist. Unfortunately, those tools can not be applied for cellular data networks. Wi-Fi uses quite different link layer protocols than UMTS, LTE, or 5G. Hence, the next chapter presents a closer look at bandwidth estimation in cellular networks.

CHAPTER 3

State of the Art

This chapter presents the state of the art when characterizing reactive mobile communication networks. Thereby, the focus is on agent-based, non-intrusive methodologies. Thus, many questions arise.

- What are reactive mobile communication networks?
- What is meant by the term *non-intrusive*?
- What are peculiarities of mobile networks?
- What characteristics describe such networks?
- Which techniques exist to characterize them?
- Which tools already exist to characterize them?
- Which techniques are non-intrusive and most suited?

This chapter will answer all the above questions step by step.

3.1 Characterization

Let us start with the question: What are reactive mobile communication networks? Miao et al. [52] defined mobile (or cellular) communication networks as networks, where the last link is wireless. Furthermore, Laner et al. [53] describe the meaning of the word *reactive* as "the network reacts to the injected traffic". Thereby, they mean that the network changes its properties based on current user traffic.

Next, let us define the term *non-intrusive*. In the best case, non-intrusiveness means no additional injected traffic into the network. However, *active estimation techniques* depend on injection of small amounts of probe traffic. Therefore, in this thesis, non-intrusiveness focuses on a small amount of injected traffic with a data volume $DV < 5MB$. In contrast, traditional estimation tool like Ookla's speedtest.net [36] consume around 100 MB and more.

Peculiarities of Mobile Networks Xu et al. [51] took a closer look at ABW measurements in mobile networks. They made four interesting findings.

Firstly, mobile networks use different link-layer protocols than wired or wireless networks. This difference means that packets often arrive in bursts.

Secondly, the instantaneous throughput and latency [54] is a function of time and space. Factors like received signal power, cell load, cell noise, and inference power influence the result significantly [55]. Consequently, reproducing measurements is a nearly impossible task and only achievable in a controlled environment, like an anechoic chamber.

Thirdly, mobile Internet Service Providers (ISPs) deploy large buffers in their networks to compensate packet loss and to guarantee high throughput. Unfortunately, these large buffers cause a phenomenon called *bufferbloat*. Thereby the huge buffers cause the TCP congestion control algorithm not to work as intended. Consequently, massive delays happen in UMTS and LTE networks [56].

Fourthly, mobile ISPs use "some form of fair queuing policy" [51]. For example, they treat TCP and UDP traffic equally. Therefore, estimation tools based on UDP can also estimate data rates for TCP traffic.

In conclusion, mobile networks behave differently than wireless or wired networks. As we will see, these differences affect the choice of estimation techniques.

What is the most interesting characteristic? In order to determine the QoS of a connection, four characteristics are vital [6, p. 405]:

- Bandwidth
- Latency
- Jitter
- Packet loss

A detailed description of them is given in Chapter 2.1

However, not all of them are equally useful to the user. Indeed, users are hardly interested in jitter and packet loss. Conveniently, applications can fix minor problems with those characteristics. For example, with buffering or retransmissions [6, p. 405].

Unfortunately, applications can not mask problems with latency and bandwidth. Hence, both characteristics are more important to the perceived service quality of a user than jitter and packet loss. Fortunately, quick and useful tools for latency measurement already exist. However, there are some points to consider. For example, mobile devices with Android as an operating system (OS) may deliver inflated results (up to tens of milliseconds) when measuring the RTT [57]. These delays are created primarily when the kernel processes the packet. Furthermore, the packet size of the measurement influences the resulting latency [58]. Last but not least, time synchronization is the major challenge when measuring the OWD.

However, Internet applications still need a non-intrusive and quick way to measure the bandwidth. Indeed, methods to determine the ABW are more complex and wasteful than methods to determine latency, packet loss, or path detection. Considering the importance and complexity of bandwidth measurements, the remainder of the thesis will focus on the characteristic *bandwidth*.

What is the most suitable metric to describe bandwidth? As Section 2.2 illustrated, the term *bandwidth* is associated with multiple metrics. However, not all of these metrics are equally well suited to describe the bandwidth for reactive mobile communication networks.

For instance, the metric *capacity* is volatile in wireless networks due to its dependence on the radio channel. Furthermore, *BTC* is only applicable for protocols using a congestion control algorithm; for example, TCP. The same applies to the metric *TCP throughput*. This restriction makes them not an ideal choice.

In contrast, ABW is a good proxy for throughput in general. Aceto et al. [19] describes ABW as "the ideal choice to characterize the network path". They point out that it is independent of the transport and application layer. It also reflects the performance of other transport layer protocols like UDP or QUIC. Furthermore, it represents the whole network path and is less volume intrusive than BTC.

Consequently, the most suitable metric to describe the bandwidth is the *ABW*.

3.2 Methods to Estimate the Available Bandwidth

After selecting ABW as the ideal metric to characterize the network path, let us take a closer look at the methods to determine the bandwidth in reactive mobile networks.

Measuring the ABW in mobile networks (like 2G, 3G, 4G, and 5G) can get inaccurate and intrusive due to heterogeneous network configurations. Conventional techniques based on *packet pairs* or *packet trains* are inaccurate due to the packet arrival in bursts [51]. The same problem appears in applications using traditional packet dispersion techniques [59]. Consequently, most active methods developed to estimate the ABW in wired and wireless networks are inaccurate in reactive mobile networks.

Additionally, Rindler et al. [59] noted that ABW estimation needs to be quick with a maximum duration of a few hundred milliseconds. Only then a software can determine the geographical position and exact time of a measurement. For example, in a 10-second test, a vehicle moving 36 km/h would cover a distance of 100 meters. Thus it would be difficult to pinpoint the exact location of the test. Further, it reduces the risk of a cell hand-over or distortion through varying signal strength. Moreover, the characterization of the reactive mobile networks should be non-intrusive as defined at the beginning of the Section 3.1. For example, RTR-NetTest (oRMBT) is intrusive because it can inject up to a few hundreds of megabytes on traffic over several seconds.

In summary, methods estimating the ABW in mobile, reactive networks should be *accurate*, *non-intrusive*, quick, and representative for the service quality. The following paragraphs will introduce some of them.

NEXT-FIT Paul et al. [60] developed this method to create self-induced congestion. Afterwards, a parameter-independent curve-fitting technique finds the ABW. In short, they search for the turning point in the one-way queuing delay curve.

A big plus of NEXT-FIT is its non-intrusive approach, which injects only around 12 to 15 KB of additional traffic into the network. However, its long duration of 2 to 4 seconds leads to a high variance in signal strength. For example, when testing NEXT-FIT on an 80 km/h fast train in an LTE network, the *Reference Signals Received Power* (RSRP) varied from -67 dBm (excellent signal strength) to -120 dBm (poor signal strength). Consequently, measurements are too inaccurate for drive tests. Furthermore, the packet size of the probes significantly influences the accuracy; too small packets tend to underestimate the ABW.

In summary, NEXT-FIT is non-intrusive but much slower than the following presented methods. Due to a lack of a prototype, a comparison to a reference tool like oRMBT or iPerf3 is missing. Consequently, the paper can not make an exact statement about the accuracy in LTE networks.

PathQuick3 Oshiba et al. [61] developed *PathQuck3* in order to estimate the ABW in LTE networks. As a basis, they use the *PathQuick* [62] method created in 2010. PathQuick3 uses a probe rate model to generate self-induced congestion. In detail, a server sends a packet train to a client. Thereby, the packet size increases linearly, but the inter-packet gap stays constant, resulting in a linear increase of instantaneous data rate. In the post-processing step, a curve fitting algorithm (non-linear least-squares) estimates the ABW.

For the evaluation, the authors used *RBB SPEED TEST* [63], a famous speed test app in Japan, as ground truth. However, tests in an LTE network led to an estimation error of 8.5% at best. Nevertheless, PathQuick3 only produces 80.4 kB of additional network traffic for each measurement.

Shiobara and Okamawari [64] also tried to improve the *PathQuick* method by introducing *Train of Packet Groups* (TPG). Thereby, a packet group consists of multiple packets, and the size of the packet groups increases linearly. Interestingly, the gap between consecutive packet groups is set to the same value as the Transmission Time Interval (TTI) of the wireless technology, for example, LTE. Thereby the TTI depends on the selected technology (3G, 4G, 5G). This idea should cope with the burstiness of the network traffic in the estimation. The evaluation of TPG in an LTE network confirmed an estimation accuracy of ±5 Mbps.

Fast Lightweight Available Rate Probing (FLARP) Rindler et al. [59] combine probe rate and packet dispersion techniques to determine the ABW. Thereby FLARP calculates the *instantaneous capacity* of the link and uses it as an estimation for the ABW.

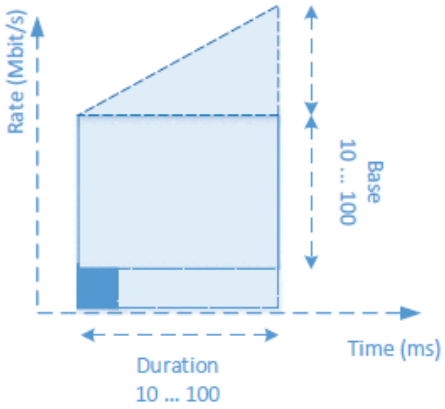

Figure 3.1: Design of probing pulse from [59]

A server selects five different sets of probing patterns. Each set covers a data rate interval. Furthermore, a set of probing patterns consists of ten different pulses. One pulse, also called *chirp*, covers a small range of probing rates. For example, Figure 3.1 shows the design of one probing pulse. In short, the probing rate linearly increases until it exceeds the available bandwidth at the tight link. Thereby, the distribution of the received packet timestamps indicate the point of overshoot.

Moreover, Rindler et al. [59] found that in LTE, the received packet timestamps are forming clusters. Interestingly, the inter-cluster interval is around 3.3 ms, whereas the inter-packet interval in a cluster is in the low microsecond range. In contrast, UMTS delivered a more uniform distribution of timestamps while still forming clusters. The clusters have fewer packets, and the inter-cluster interval is only around 2 ms. Furthermore, the inter-packet interval in a cluster is only around 5 μs. Indeed, the inter-cluster interval is the same as the Transmission Time Interval (TTI) of UMTS. They use this knowledge to create efficient patterns and apply intelligent post-processing.

In conclusion, FLARP has some interesting properties. Its low data volume consumption of smaller than 3.6 MB makes it non-intrusive compared to conventional methods like oRMBT. Furthermore, the short measurement duration of just several hundreds milliseconds allows the bypassing of traffic shaping. Consequently, it can estimate the instantaneous capacity of the link. Another advantage is its flexibility. FLARP is applicable in LTE, UMTS, and wired networks. Nevertheless, it delivers similar accuracy as established measurement methods like oRMBT [59]. In short, this method is excellent for use on agent-based systems. However, the user needs expert knowledge to design efficient patterns and apply intelligent post-processing. Moreover, there is still room for improvement in the design of patterns and data volume consumption.

Constant Rate Ultra Short Probing (CRUSP) Raida et al. [55] developed a simplified version of FLARP. Thereby their method focuses on just one specific pattern. The CRUSP pattern sends datagrams (UDP packets) at a constant rate high enough to exceed the expected available bandwidth, for example, the maximum capacity of the tight link. After recording the timestamps of the packets at the receiver, the post-processing step calculates the available bandwidth.

Figure 3.2 illustrates the CRUSP framework and post-processing. The data used for illustration are real-life measurements from an LTE network. In the beginning, the transmitter sends all datagrams with the same data volume (DV) and at a constant rate. As in FLARP, the receiver obtains the datagrams in clusters (bursts) due to the bursty nature of LTE. Figure 3.2 (a) shows the bursts.

$$r = \frac{c[K] - c[0]}{t[K] - t[0]} \tag{3.1}$$

CRUSP can calculate a *naive rate* and a *sophisticated rate* in the post-processing step. Equation 3.1 describes the formula for the naive rate. Thereby c[k] denote the cumulative data volume (CDV) until (and including) the kth out of $K+1$ received datagrams, where $k \in \{0, 1, \ldots, K\}$. Additionally, $t[k]$ represents the timestamp (in ns) for the kth received datagram. In short, to calculate the naive data rate, one divides the total data volume of all packets by the total elapsed time.

However, multiple issues lead to an inaccurate result with the naive formula. Firstly, CRUSP does not use synchronized clocks. Therefore, the initial burst can not determine the latency to the previous one. As a result, the naive approach overestimates the available bandwidth. Secondly, the LTE resource blocks are at a low level at the beginning of the estimation. This means that not all physical resources are available to the connection at the beginning. Consequently, the sophisticated method discards the initial burst. The naive method has another inaccuracy. It utilizes the last datagrams of the measurement even though they may not represent a valid burst. Thus, the sophisticated formula does not include the unfinished burst at the end of the measurement.

$$\bar{r} = \frac{w'[N] - w'[0]}{t'[N] - t'[0]} \tag{3.2}$$

Additionally, the sophisticated approach divides the chronologically ordered datagrams into different bursts. Thereby, the post-processing separates two consecutive datagrams into different burst, if the duration between them is higher than Δ_{min} (usually around $0.5ms$). In total, CRUSP finds $N+2$ bursts out of the k arrived datagrams. After discarding the first and the last burst due to inaccuracy reasons, the formula in given Equation 3.2 calculates the sophisticated data rate. Thereby, let $w'[n]$ denote the cumulative data volume until (and including) the nth burst, and $t'[n]$ the timestamp for the last datagram in the nth burst. In short, the first and last burst gets cut, and the

remaining total data volume is divided by the time needed for these remaining bursts. Figure 3.2 depicts the post-processing step by step.

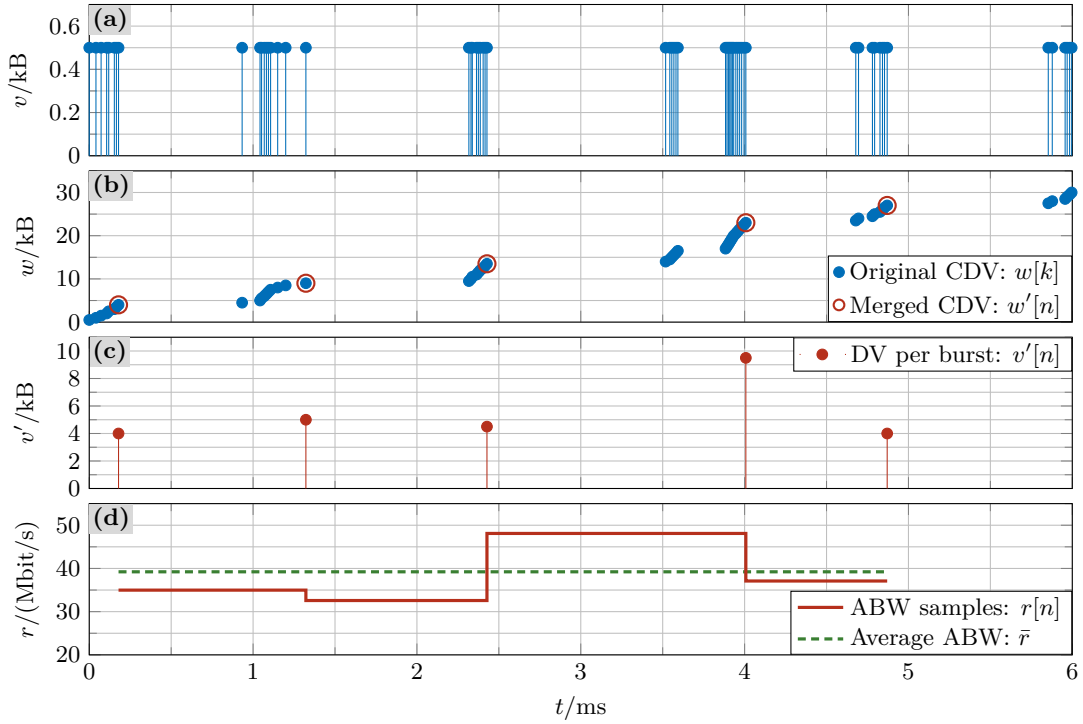

Figure 3.2: CRUSP post-processing algorithm: (a) Data volume samples $v[k]$ arrives at time $t[k]$ to the receiver in bursts. (b) The cumulative data volumes (CDV) samples $w[k]$ are mapped to merged CDV samples $w'[n]$ for $N+2$ bursts where $n \in \{0, 1, \ldots, N\}$. (c) From the merged CDV samples, let us calculate the merged data volume $v'[n]$, i.e., data volume per burst. (d) From the merged DV, let us calculate the data rate samples $r[n] = \frac{v'[n]}{t'[n]-t'[n-1]}$ and average data rate $\bar{r}$, from [65]

There are a few key points to note about CRUSP. A typical CRUSP estimation consumes around 1 MB on data volume to achieve similar results than the reference tool oRMBT with around 100 MB. Furthermore, CRUSP can perform its estimations in around 100 to 200 milliseconds. In contrast, oRMBT needs seven seconds. This quick estimation leads to results that correspond to the unlimited rate (before traffic shaping takes effect). Further, the performance measured by CRUSP relies on UDP. Nevertheless, CRUSP data rates are also applicable for TCP connections [66].

Method Overview Table 3.1 lists an overview of methods developed for ABW estimation in mobile networks. In summary, CRUSP and FLARP have a balanced mix of accuracy, non-intrusiveness, and quick execution time. Additionally, CRUSP is easy to use compared to FLARP. The latter needs expert knowledge about FLARP patterns

to operate efficiently. Consequently, CRUSP is the best candidate for an agent-based implementation.

Name of the Method	Accurate	Quick	Non-Intrusive	User-Friendly	5G Applicable
NextFit	?		✓	✓	✓
PathQuick3		✓	✓	✓	✓
TPG		✓	✓	✓	✓
FLARP	✓	✓	✓		✓
CRUSP	✓	✓	✓	✓	✓

Table 3.1: Overview of methods developed for mobile networks

Below can be found a description of the columns for Table 3.1.

- *Accurate*: Checked off if the average error to a reference tool is less than 5%.
- *Quick*: Checked off if the duration is shorter than one second.
- *Non-Intrusive*: Checked off if true based on the definition in Section 3.1.
- *User-Friendly*: Checked off if the user does not need expert knowledge about the method.
- *5G Applicable*: Checked off if the method is, in principle, applicable in 5G networks.

3.3 Existing Agent-Based Software Applications

Many modern methods lack an implementation available for download and execution on a mobile phone or other UE. However, the Google Play Store and Apple App Store list dozens of apps to estimate the available bandwidth. Therefore, this section will shed light on the current agent-based solutions and how they compare to each other.

At first, let us take a closer look at the term *agent-based*. In software engineering, this term is often used in the context of artificial intelligence (AI). However, in the context of this thesis, the term *agent-based* leans on Nwana's [67] definition of a *mobile collaborative agent*. Thereby such agents can move freely like a mobile phone or vehicle. Furthermore, they use cooperation to acquire new knowledge. For example, multiple agents can perform ABW measurements and then calculate the average ABW for a specific time of the day and location. The sum of all such agents forms a so-called *crowd-sourcing system*.

Table 3.2 lists an overview of currently popular agent-based software applications to estimate the available bandwidth. Thereby the focus is on applications running on Android, on Linux, or in the Web. All of the mentioned solutions support estimations for wired, wireless, and 4G networks.

Name of App	ABW	Latency	Jitter	Packet Loss	Radio Parameters	Android App	iOS App	Downlink Test Duration	Non-Intrusive	5G Ready	Min. Resource Requirements	Open Source
RTR NetTest (oRMBT)	✓	✓			✓	✓	✓	Avg.		✓	IoT	✓
Speedtest (by Ookla)	✓	✓	✓	✓		✓	✓	Long		✓	IoT	
nPerf	✓	✓	✓		~	✓	✓	Avg.		✓	mOS	
FLOQ	✓	✓			~	✓		Long			mOS	
Speedcheck (by Etrality)	✓	✓				✓	✓	Avg.			mOS	
Traffic Monitor	✓	✓			~	✓	✓	Long		✓	mOS	
Meteor (by OpenSignal)	✓	✓				✓	✓	Long			mOS	
LibreSpeed	✓	✓	✓			~		Avg.		~	Web	✓
NDT Test (by M-Lab)	✓	✓						Long		~	Web	✓
fast.com (by Netflix)	✓	✓				✓	✓	Long		~	mOS	
iPerf3 Tool	✓	✓	✓	✓		~	~	Long		✓	IoT	✓
CRUSP for Mobile	✓				✓	✓	~	Instant	✓	✓	IoT	

Table 3.2: State-of-the art Android Apps

The Table 3.2 contains many self-explanatory columns. However, a few of them need further explanation.

- *Radio Parameters*: Checked off if multiple radio parameters during the estimation are tracked — for example, signal strength, timing advance (TA), or cell identity (CI). In contrast, tools with a tilde-symbol only report the signal strength.
- *Downlink Test Duration*: Divides the duration of an applications downlink estimation into four categories:
  - Instant: 0.00 sec - 0.99 sec
  - Fast: 1.00 sec - 4.99 sec
  - Average (Avg.) 5.00 sec - 9.99 sec
  - Long: more than 10.00 sec
- *5G Ready*: Checked off if the tool supports 5G radio parameters. Additionally, it provides enough resources to estimate data rates reachable in the first 5G deployments around the globe.
- *Minimum Resource Requirements*: This column presents, which technical requirements a system must fulfill to execute the tool.
  - IoT: needs a machine running Linux as OS, may also run on an IoT-device
  - mOS: needs a mobile OS (mOS) like Android or iOS
  - Web: needs a web browser like Firefox or Chrome
- *Open Source*: Checked off if the tool provides access to their source code. The problem with closed source tools are their unclear measurement procedures.

Analysis The most downloaded Android application (more than 100.000.000) is *Speedtest* by Ookla [36]. It measures almost all essential characteristics and can run on Linux-powered IoT devices. However, its downlink and uplink tests consume a vast amount of data and take at least 10 seconds each.

This trend manifests itself in all mentioned tools. They take at least five seconds and inject a considerable amount of data into the network. Even leading tech companies like Google or Netflix are not able to offer a quick and non-intrusive tool publicly available. For example, Google collaborates with the Measurement Lab (M-Lab) when offering a speed test. M-Lab also operates an app called MobiPerf [68]. However, MobiPerf's code was last updated five years ago, and an execution of the code throws a Java error. Consequently, the application is not listed in Table 3.2.

The last row in the table shows the capabilities of a new tool developed as part of this thesis. Its name is *CRUSP for Mobile*.

CRUSP for Mobile As we have seen, agent-based, non-intrusive, quick, and accurate tools to estimate the ABW in mobile network do not exist. The first application to break through this barrier is *CRUSP for Mobile*. The application developed by the author of this thesis can estimate the ABW instantaneous without injecting vast amounts of additional traffic. Thereby it uses the CRUSP technique developed by Raida et al. [55]. This method combines non-intrusiveness, high accuracy, and quick response time.

Furthermore, the application is 5G ready and deployable on Android mobile phones and IoT devices. The next Chapter 4 holds detailed information to the software development process.

CHAPTER 4

CRUSP for Mobile

Schatten et al. [69, p. 13] defined a systematic and structured software life-cycle consisting of the following steps:

- Requirements & Specification
- Planning
- Design
- Implementation
- Operation & Maintenance
- Retirement

This chapter focuses on the first four steps, whereby the steps can interleave with other steps. For example, *planning* started with the *requirements & specification* step and lasted into the *implementation* step. The remaining steps, *operation & maintenance*, and *retirement* are out of scope for the thesis.

4.1 Requirements

At the beginning of a software project, the developer team collects all relevant requirements from all stakeholders. Thereby each requirement represents a problem to solve and the incentive it provides for the system to develop [69, p. 17]. A suitable tool to describe requirements are user stories [69, p. 19][70].

4.1.1 User Stories

In short, a user story focus on why and how the user interacts with the software [70]. For example, one story can read as follows: *"As a user, I want to start a bandwidth test on*

the UE so that I know the downlink data rate of my mobile connection". User stories are plain and leave room for discussion; the detailed formal requirements are defined later when implementing the user story. Hence, the flexibility and simplicity of user stories facilitate agile development. In addition, some acceptance criteria (AC) accompany the description of a user story. They work as conditions to fulfill the user story. For instance, an AC can look like this: *"Given the bandwidth measurement was successful when started then the down-link bandwidth should be displayed"*.

Moreover, each user story's narrative is from a specific user's view. Thereby each user story can be assigned to a user role. In the software product exist three distinct user-roles: *users, engineers, and analysts*. A *user* installs the application on the mobile phone to estimate the available bandwidth of the mobile connection. An *engineer* plans the bandwidth tests, either in the mobile application or on the server. An *analyst* manages and analyses the produced measurement data.

The following Table 4.1 lists all the nine gathered user stories for the role *user*. Thereby user story with id 2 constitutes an exception; it represents a non-functional requirement. Table 4.2 shows the user stories for engineers, Table 4.3 shows the user stories for analysts.

ID	User Story	Story Points	Priority
1	**Story:** As a user, *I want to start a bandwidth test on the UE* so that I know the downlink data rate of my mobile connection. **AC:** Given the bandwidth measurement was successful when started then the downlink bandwidth should be displayed.	13	5
2	**Story:** As a user, *I want the app to be scalable* so that I never have to wait more than 1 second for a measurement result. **AC:** Given a test is initiated when the servers are fully loaded then the system should be able to handle my request immediately.	8	5
3	**Story:** As a user, *I want to have continuous measurements* so that I can identify the mean data rate over some period of time. **AC:** Given the continuous measurement delivers a new result when activated then it should be integrated into the displayed mean.	3	5
4	**Story:** As a user, *I want to deactivate continuous bandwidth* measurements so that I can save data and battery. **AC:** Given the continuous measurement is activated when using the app then I can deactivate continuous measurements.	1	4
5	**Story:** As a user, *I want to see previous measurements results* so that I can compare different results. **AC:** Given measurements were taken in the past when using the app then I want to be able to see the results for those measurements.	5	4
6	**Story:** As a user, *I want to see the results of continuous* measurements in a graph so that I can identify patterns or a trend. **AC:** Given the continuous measurement delivers a new result when activated then the measurements-graph should be updated.	3	3
7	**Story:** As a user, *I want to know the amount of data consumed by measurements* so that I don't exceed my data plan. **AC:** Given continuous measurement is activated when using the app then it should display the amount of data already consumed.	2	2
8	**Story:** As a user, *I want to know the next scheduled measurement* so that I know when a new result is available. **AC:** Given continuous measurement is activated when using the app then it should display the remaining time to the next measurement.	1	2

Table 4.1: User stories for the role *user*

ID	User Story	Story Points	Priority
11	**Story:** As an engineer, *I want to change parameters of the downlink measurement on the UE* so that I can configure my own tests. **AC:** Given the parameters on the UE are correct when they are manually changed then test results should be displayed on the UE.	8	5
12	**Story:** As an engineer, *I want to change test parameters on the server* so that each client gets the new test configuration. **AC:** Given the test parameters on the sever are changed when a test should be performed then all clients should apply these parameters.	5	5
13	**Story:** As an engineer, *I want to see extended measurement results on the UE* so that I can have a better understanding of the measurement. **AC:** Given a new measurement result arrives when performed then I want see an overview of the data collected in the process.	5	3

Table 4.2: User stories for the role “engineer”

ID	User Story	Story Points	Priority
50	**Story:** As an analyst, *I want that each measurement is stored on the server permanently* so I can take a look at it again later. **AC:** Given the measurement is sent to the server when it is executed then it is saved in a database.	8	5
51	**Story:** As a analyst, *I want to filter measurements by various available properties* so that I can analyze them. **AC:** Given data-set is not empty when executing a query on the stored data then I can filter measurements from the found data.	13	5
52	**Story:** As a analyst, *I want to export selected measurements* so that I can further process them. **AC:** Given data-set is not empty when extracting data, then I receive a file with the selected data.	5	5
53	**Story:** As a analyst,*I want to sort measurements by various available properties* so that I can organize them. **AC:** Given data-set is not empty when executing a query on the stored data then I receive a sorted data set.	13	4

Table 4.3: User stories for the role “analyst”

4.1.2 Non-functional Requirements

Functional requirements express the behavior of the software product. They are mostly collected through user stories. In contrast, non-functional (or non-behavioral) requirements define software quality attributes [71], for example, performance, security, or privacy. Moreover, non-functional requirements usually run through all software components in the project.

When collecting non-functional requirements, the stakeholders laid the focus on scalability. Scalability describes a behavior where the architecture of a software product can service a growing number of users and work [72]. It plays a crucial role in this software product, allowing the software to grow and shrink dynamically. Thus, applications utilizing crowd-sourcing can immediately react to changes in workload. Consequently, scalability was defined as a user story on its own with the id #2.

Table 4.4 contains all collected non-functional requirements. In there, each requirement has an identification number (ID), the description, a priority, and (if available) a connected user story (US). Other non-functional requirements concerning security, availability, privacy, or open-source did not make it into the scope of the project. The non-functional requirements are kept short in numbers because they are not part of the main focus. Hence, they are a matter of future work.

ID	US	Non-functional requirement	Priority
101	#2	Scalability: The architecture should be scalable to serve a growing number of users.	5
102	-	Performance: No user should wait longer than one second for a measurement request.	4
103	-	Performance: No analyst should wait longer than one second for a filtering or sorting request to the database.	4
104	-	Security: The architecture should be decentralized to avoid a single point-of-failure	3

Table 4.4: Non-functional requirements

4.1.3 Data

The last part of this chapter explores the data requirements in more detail. Software applications create a vast amount of data. For example, each CRUSP measurement uses specific settings, creates UDP-packets, and collects data about the mobile device in use. Furthermore, each component of the software produces log-data. Storing all available data is not productive. The hardware resources are limited; therefore, the software can only save a thorough selection of the data. Indeed, a well-conceived data selection would keep the operational costs low while still offering enough information about the applications state and the CRUSP measurements. Therefore, interviews with the project stakeholders about data requirements were conducted.

In total three classes of data requirements were identified. The *first class covers user story #51*, where an analyst wants to filter the data to perform evaluations. In its detailed elaboration phase, user story #51 revealed the following data to persist: date, time, CRUSP settings, data rate, received data volume, GPS location, signal information, operator, and mobile device used.

The *second class of data-requirements was created when implementing the user story #52.* In there, an analyst wants to extract all the gathered data about a measurement. The interviews revealed that next to the filter-data in user story #51, analysts also want to study the network-traffic during a measurement. In short, analysts want to persist the following data for each UDP-packet: the timestamps in nanoseconds, the amount of data received, and the receiving order.

The third class of data requirements covers the logging data. Although not specified in a user story, the engineer needs to maintain the software and identify bugs. Thus, each component should implement a simple logging functionality. Consequently, the created logging-data needs to be saved for predefined time.

4.2 Planning

When developing a product, proper planning is an integral part of the project's success. This phase describes all necessary tasks to fulfill the requirements, the resources it takes, the chronological order of the tasks and the implementation deadline for the tasks [73]. Thereby periodic progress evaluations, as well as changes in requirements, can trigger revisions of the current project plan. Consequently, the planning process can interleave with multiple other phases [69, p. 25]. For example, project planning already started during the requirements & specification phase and lasted till the implementation phase.

When developing a software product, project management, and appropriate process models play a crucial role in ensuring a project's success. Therefore, it was essential to find a suitable process model and suitable project management tools. For small teams, agile development is a fitting process model. Thereby the software is built incrementally; typically, new features are added step by step while maintaining a working version of the software. In contrast to traditional process models, agile development offers a quick first version of the software as well as flexibility when requirements change [69, p. 62].

The cornerstones for the project plan provided the collected user stories. Additionally, the efforts it takes to implement the user story, as well as the priority of the user story, were estimated.

Story points are a common way to estimate the effort needed to fulfill user stories [74]. Thereby, factors like the complexity of user stories, uncertainty, and amount of work influence the effort. Based on the Fibonacci numbers, a low value indicates a small effort, whereas high values indicate a high effort to solve a user story. For example, a user story with 5 points should involve less effort to fulfill than a user story with 8 points.

Furthermore, numeric values ranging from 1 to 5 express the priority of a user story. High values mark important user stories; low values express low importance.

Both key figures can be found in Tables 4.1, 4.2 and 4.3 next to the user story description.

During the planning phase, multiple project management tools helped to facilitate the development of the software product. They also provided documentation for the project as well as an overview of the projects state. For example, GitLab offers issue-tracking to show the project plan and the project's progress. In addition, GitLab provides a repository for source code management. Conveniently the Institute of Telecommunications offers its own GitLab server. Another example is Maven, a build automation tool for Java. Maven describes the software dependencies and automates the compiling and building of the software component. Similar tools used in the project are Gradle (for Android), Cargo (for Rust programming language), and npm (for React web framework).

4.3 Design

After collecting all requirements and finding a suitable process model, the product's design phase started. Designing the whole software is a quite complicated task. To simplify complex problems, software engineers create models. Those models are abstract images of real-world problems; they build on the gathered functional and non-functional requirements. To summarize, models represent the real-world problem by dropping irrelevant information.

Nevertheless, models can get huge. To further reduce complexity, software engineers use diagrams to illustrate different characteristics of the model. Thereby they provide a specialized view on the model, which focuses only on selected aspects, dropping all other aspects. For example, diagrams can visualize the aspects structure, interaction, or behavior of the model. In software development, the Unified Modeling Language (UML) is the standard modeling language to model systems [75, p. 29]. It is an approved ISO standard with 14 different diagrams [76]. The standardization provides the various stakeholders with a common language to discuss different aspects of the software.

In this project, the following four diagram types provide a solid coverage of the model: *use case diagram, activity diagram, sequence diagram, and package diagram.* The *use case diagram* and the *activity diagram* reflect the behavior of the system. In contrast, the *sequence diagram* focuses on the interaction between different components of the system. Finally, the *package diagram* shows the structure of the systems [69, p. 167].

4.3.1 Use Case Diagram

The use case diagram provides an overview of the gathered requirements. It achieves the same goals as the user stories; however, they are not the same technique. Use cases are more specific in the requirements than user stories [77]. Therefore, user stories often serve as inspiration to derive use cases. Nevertheless, the use case diagram also enables

an overview of all the user stories. Consequently, the use case diagram serves a compact overview of all gathered user stories and their relationship with each other.

In a use case diagram, an oval symbol represents a use case, labeled with the name of the use case. Additionally, each use case connects to an actor, represented by a stick man. Furthermore, dotted arrows visualize dependencies between use cases [69, p. 170].

In summary, Figure 4.1 shows all use cases derived by the user stories. Only user story #2 was left out since it is a non-functional requirement.

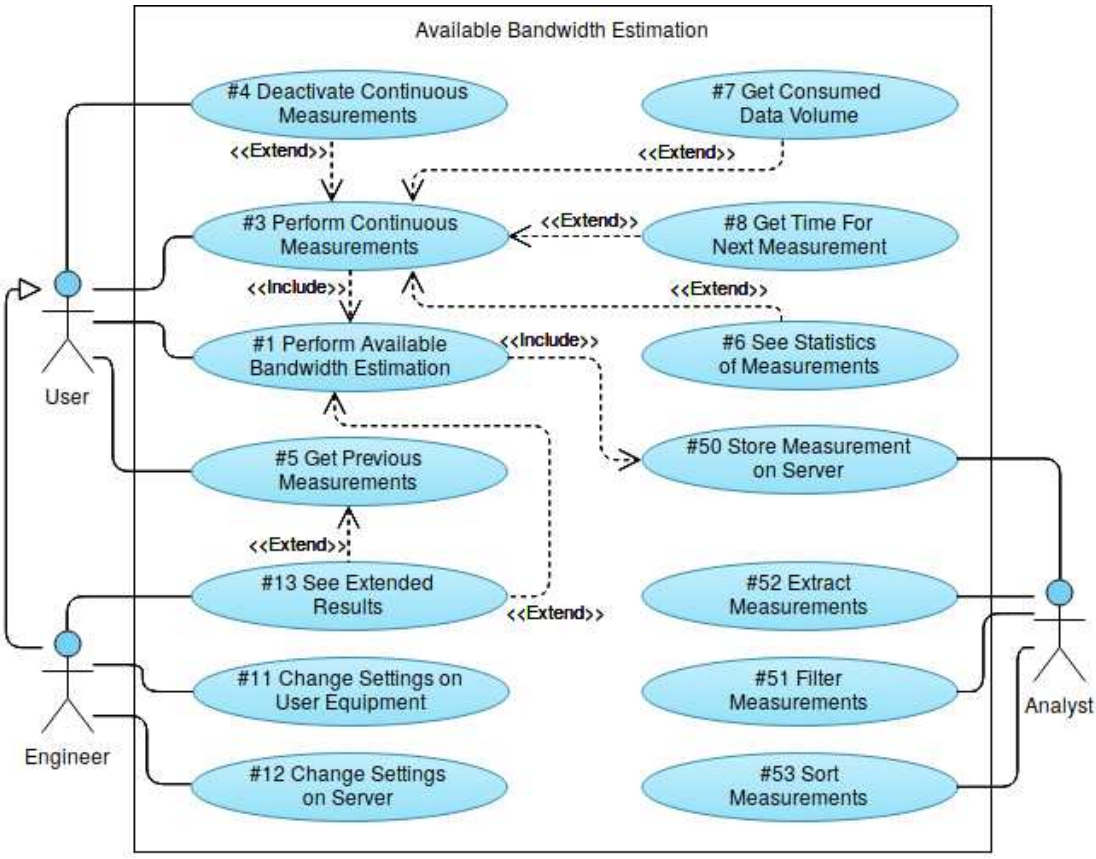

Figure 4.1: Use case diagram

4.3.2 Package Diagram

The package diagram offers an overview of the model on a high abstraction level [69, pp. 167]. A package is drawn as a rectangle and represents classes or a collection of classes. Furthermore, the dotted lines show the dependencies between the packages. In conclusion, the systems consist of 10 packages; the Figure 4.2 shows an overview of all packages.

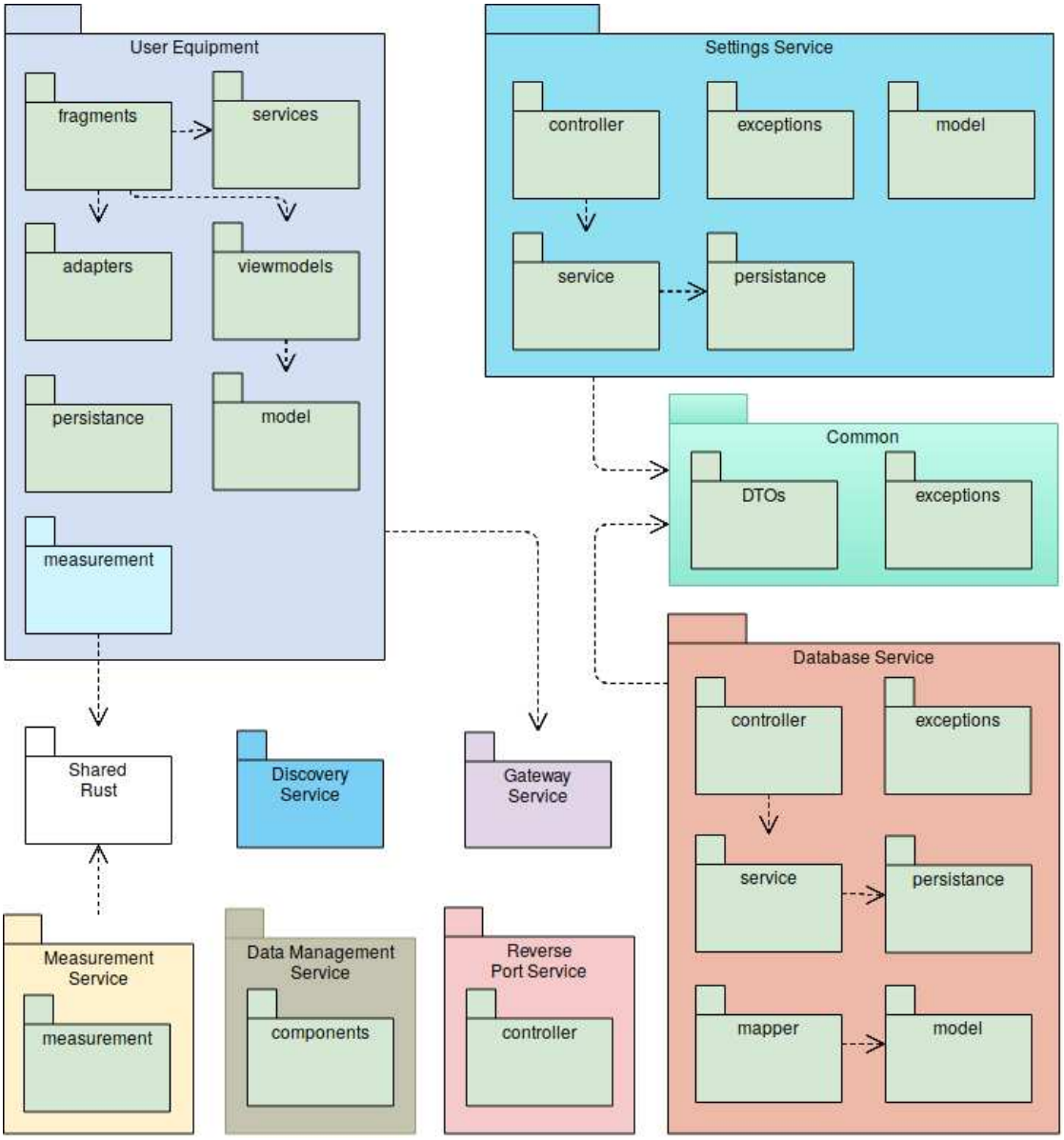

Figure 4.2: Package diagram

4.3.3 Activity and Sequence Diagrams

A more detailed view of each use case provides the activity diagrams and sequence diagrams. An activity diagram visualizes the behavior of a system by breaking down a use case into a flow of atomic or non-atomic activities. Thereby rectangles with round corners visualize those activities. Additionally, arrows signal the flow of control or the flow of objects between the activities. Furthermore, full circles mark the start of an activity, full circles in an empty circle mark the end of the activity. Moreover, so-called

"swimming lanes" represent the components of the system. A swimming lane is a vertical band throughout the diagram, labeled at the top with the component name.

In contrast to activity diagrams, sequence diagrams focus on the interaction between different objects of the system. Therefore, the sequence diagram puts objects like classes, use cases, or actors on the top. Underneath dashed lines extend the class, those so-called lifelines represent the lifespan of an object. Each object can send messages to other objects, visualized with arrows between the lifelines. In short, an activity diagram presents the flow of activities one after the other. On the contrary, a sequence diagram shows the sequence of the messages between objects one after the other [69, p. 181].

The following Figures 4.3 – 4.30 show the activity diagrams and sequence diagrams for all defined use cases.

Use case #1 Perform ABE

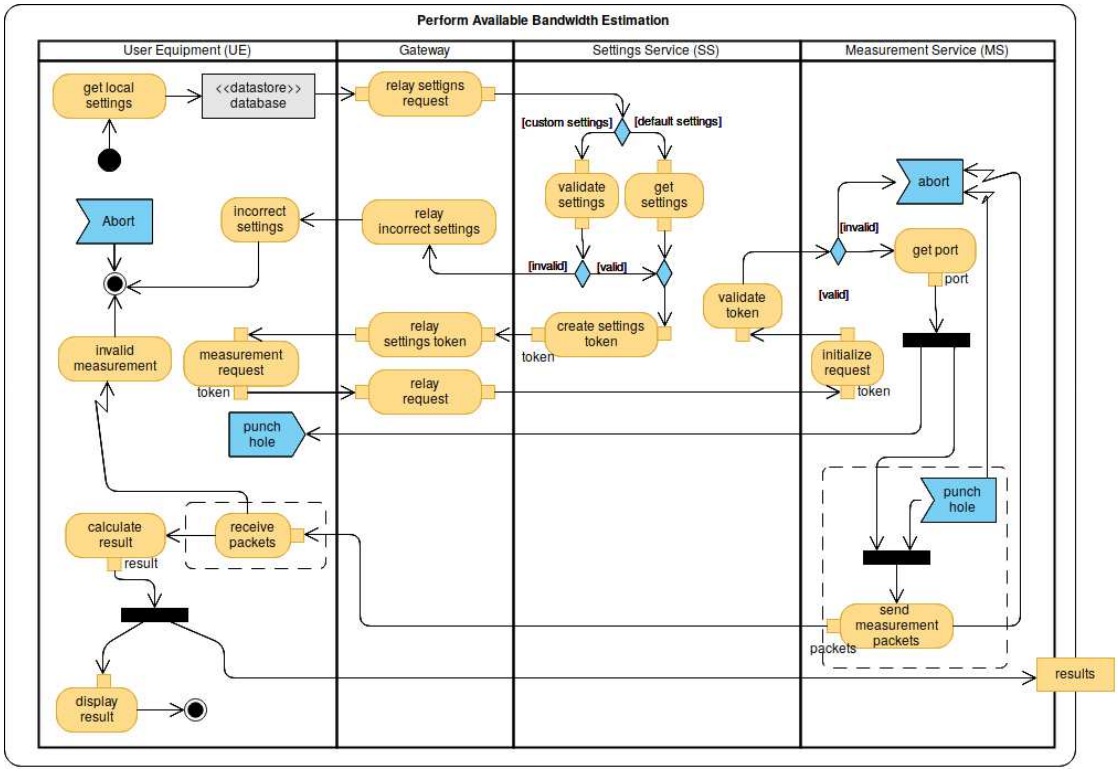

Figure 4.3: Activity diagram for use case "Perform ABE"

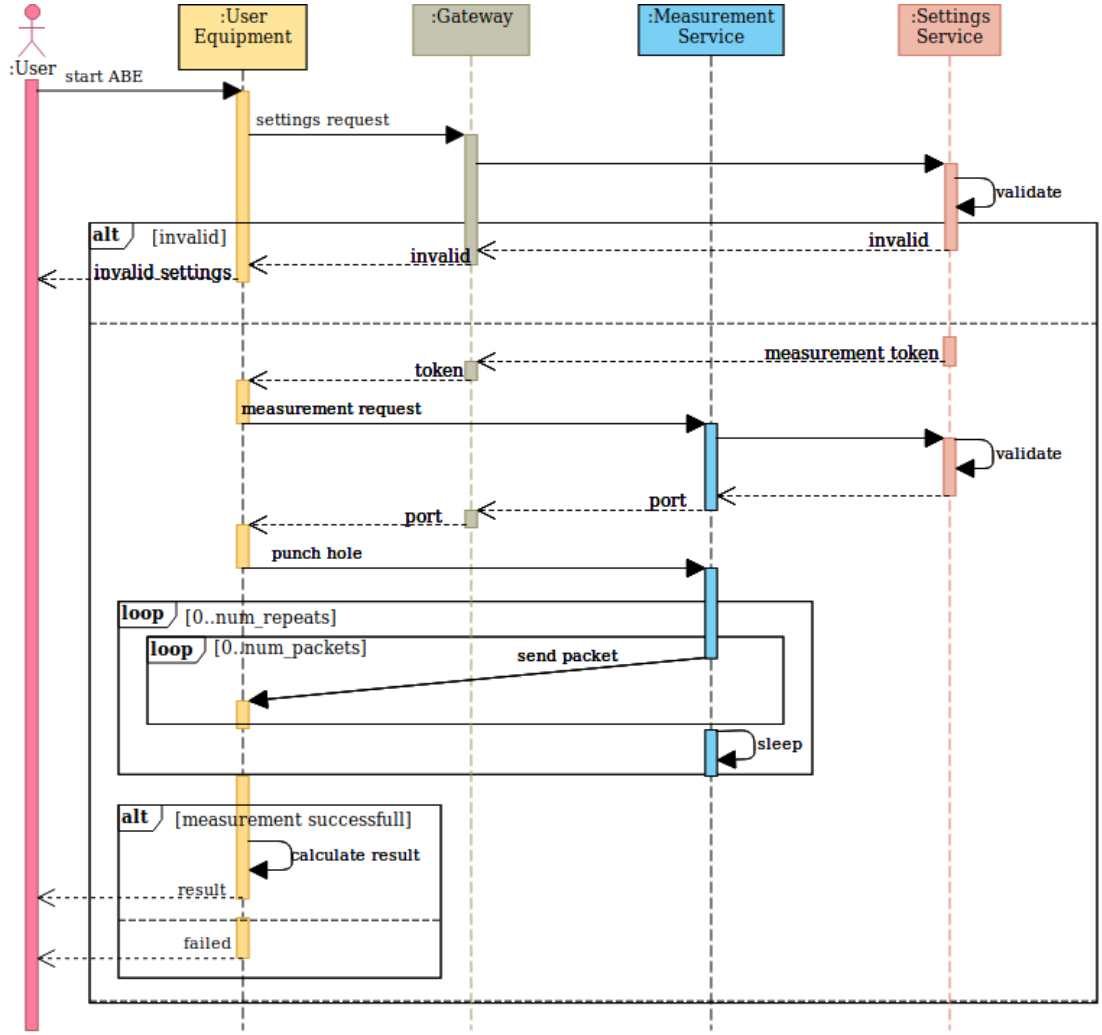

Figure 4.4: Sequence diagram for use case “Perform ABE”

Use case #2 Scalable Infrastructure

This use case is a non-functional requirement. Therefore, it has no activity diagram and sequence diagram.

Use case #3 Perform Continuous Measurements

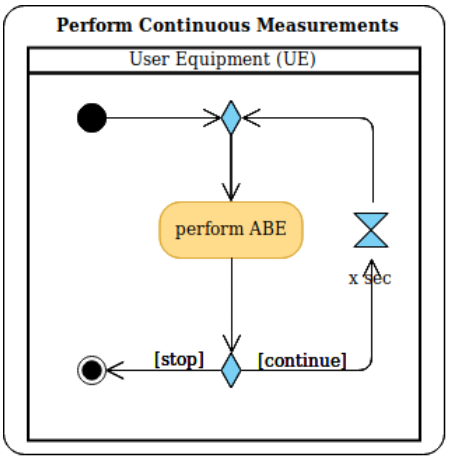

Figure 4.5: Activity diagram for use case "Perform Continuous Measurements"

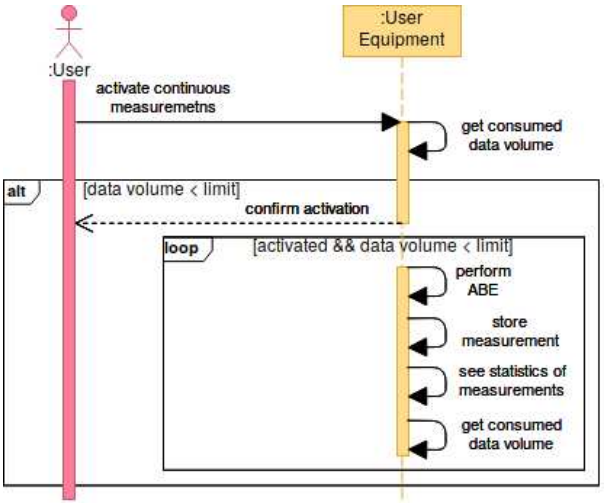

Figure 4.6: Sequence diagram for use case "Perform Continuous Measurements"

Use case #4 Deactivate Continuous Measurements

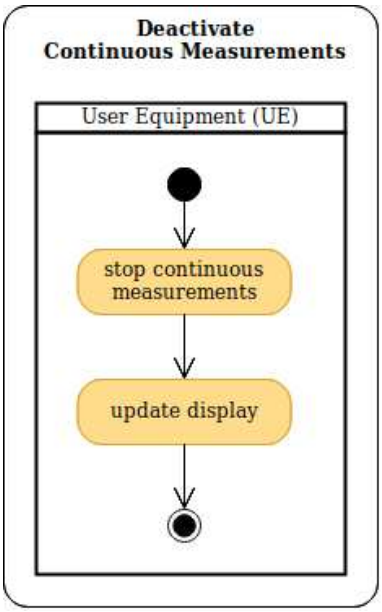

Figure 4.7: Activity diagram for use case "Deactivate Continuous Measurements"

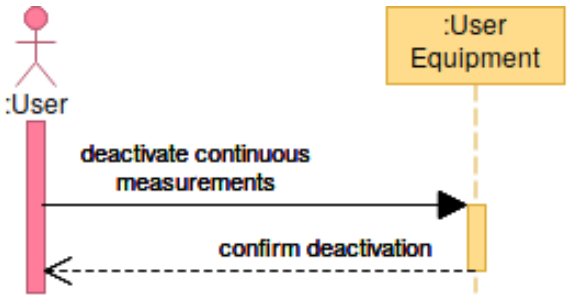

Figure 4.8: Sequence diagram for use case "Deactivate Continuous Measurements"

Use case #5 Get Previous Measurements

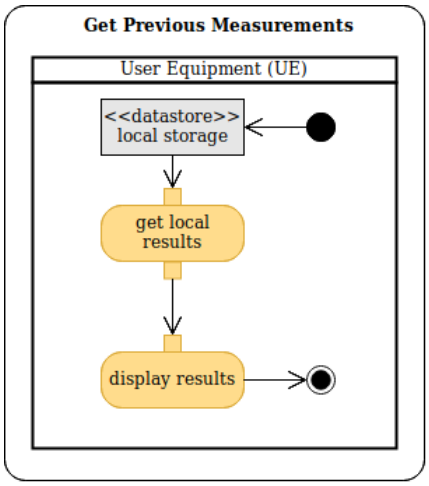

Figure 4.9: Activity diagram for use case "Get Previous Measurements"

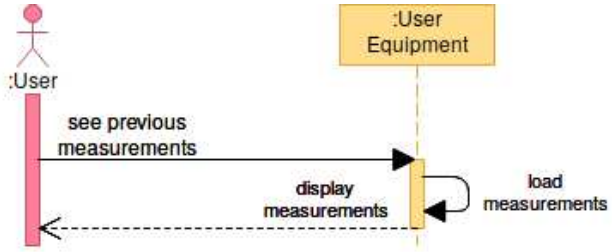

Figure 4.10: Sequence diagram for use case "Get Previous Measurements"

Use case #6 See Statistics of Measurements

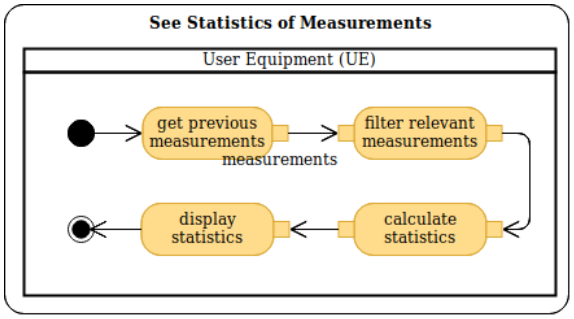

Figure 4.11: Activity diagram for use case "See Statistics of Measurements"

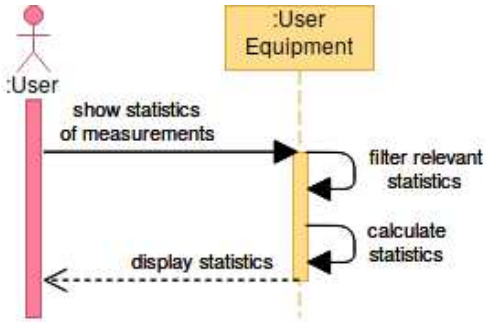

Figure 4.12: Sequence diagram for use case "See Statistics of Measurements"

Use case #7 Get Consumed Data Volume

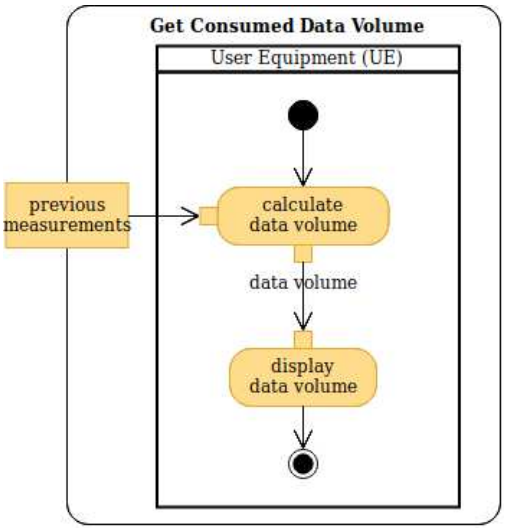

Figure 4.13: Activity diagram for use case "Get Consumed Data Volume"

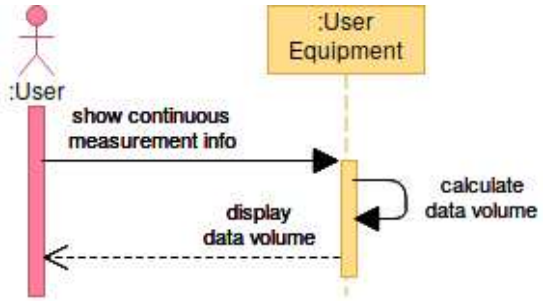

Figure 4.14: Sequence diagram for use case "Get Consumed Data Volume"

Use case #8 Get Time for Next Measurement

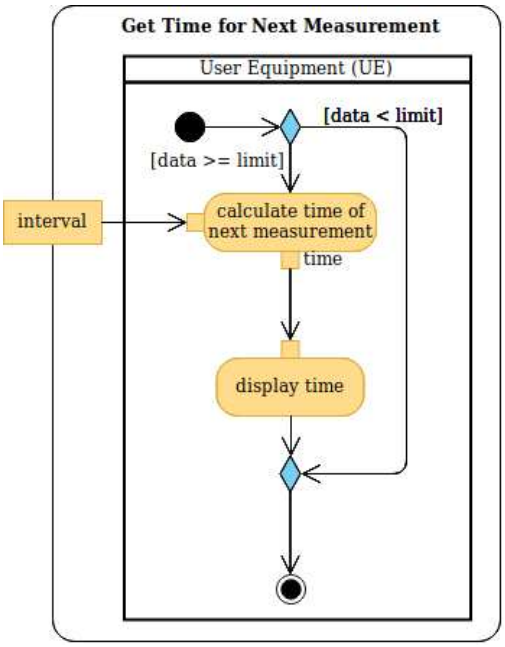

Figure 4.15: Activity diagram for use case "Get Time for Next Measurement"

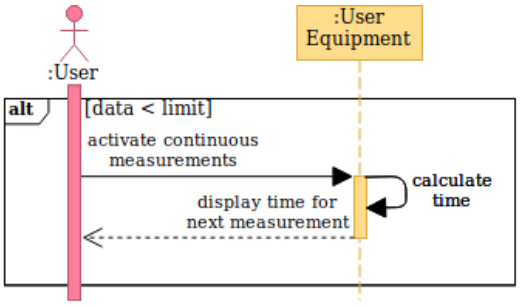

Figure 4.16: Sequence diagram for use case "Get Time for Next Measurement"

Use case #11 Change Settings on UE

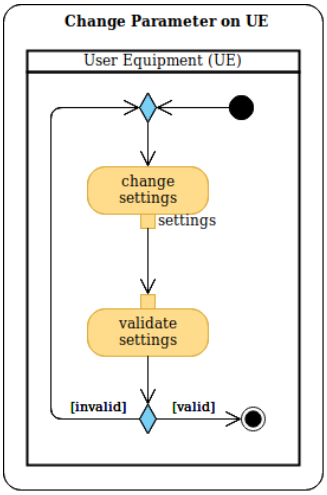

Figure 4.17: Activity diagram for use case "Change Parameter on UE"

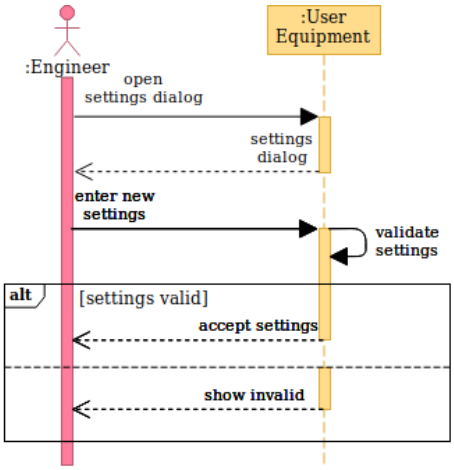

Figure 4.18: Sequence diagram for use case "Change Parameter on UE"

Use case #12 Change Settings on Server

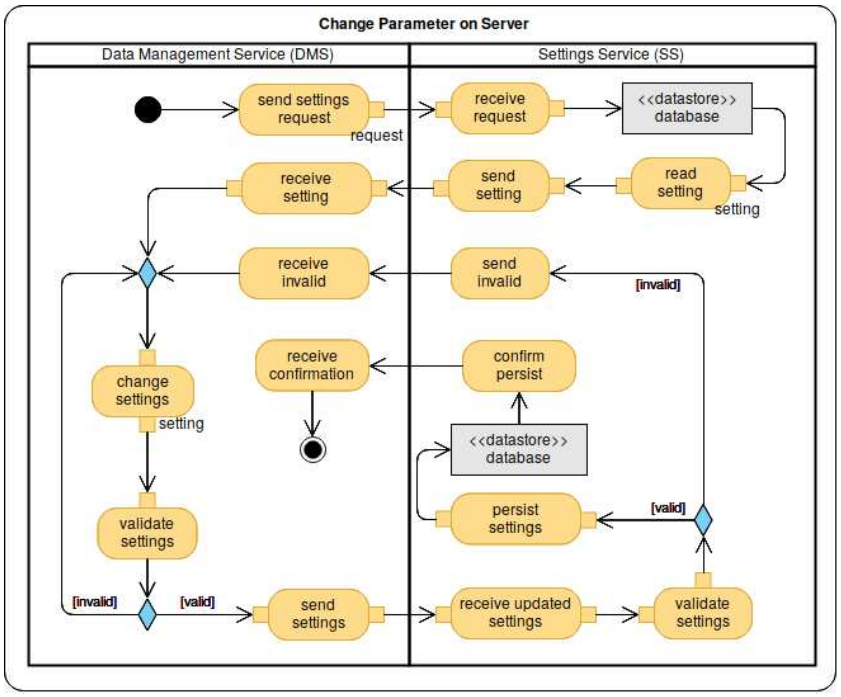

Figure 4.19: Activity diagram for use case "Change Parameter on Server"

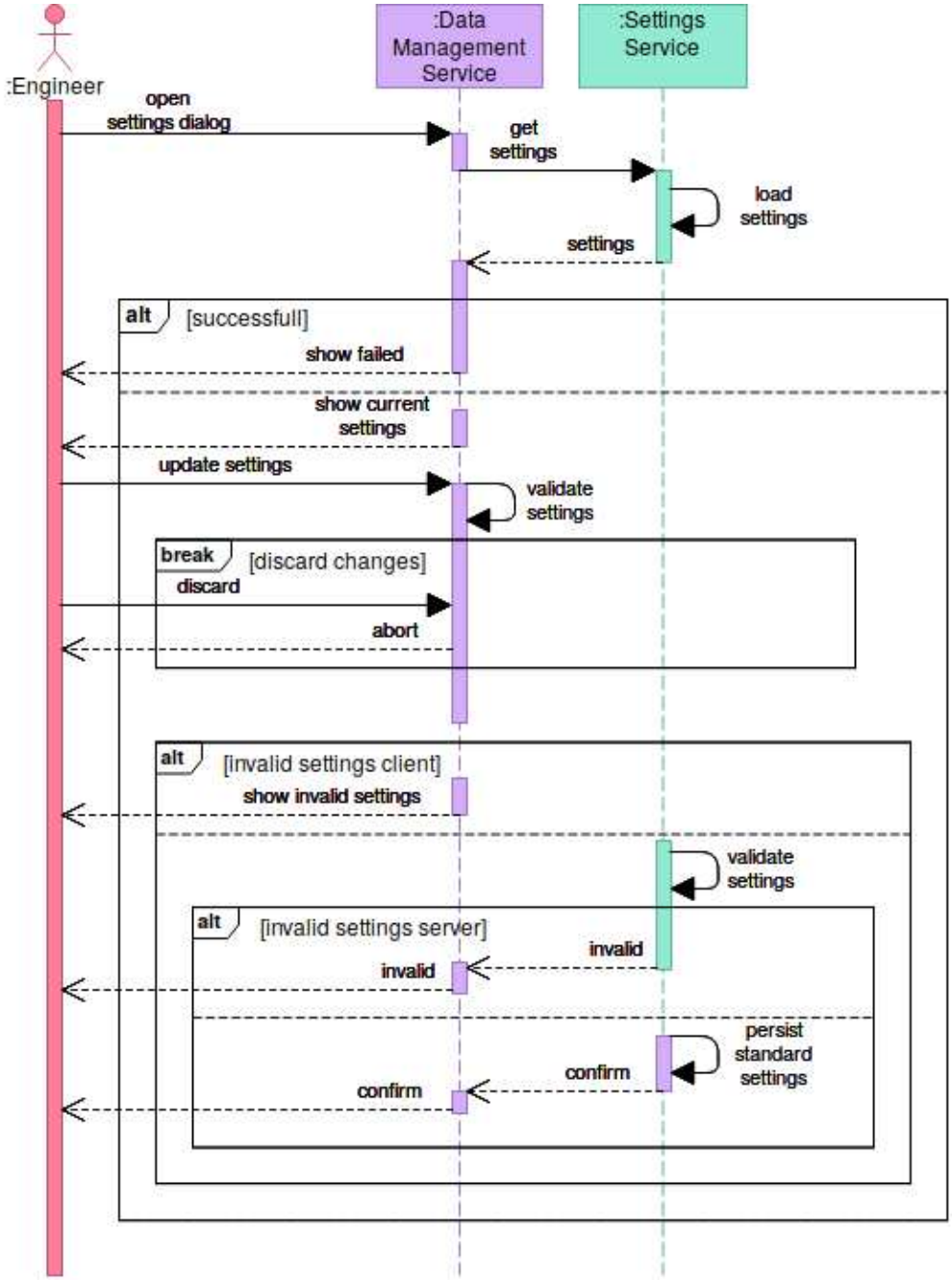

Figure 4.20: Sequence diagram for use case “Change Parameter on Server”

Use case #13 See Extended Measurement

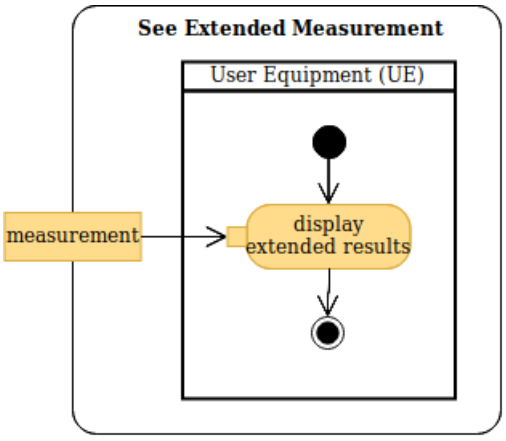

Figure 4.21: Activity diagram for use case "See Extended Measurement"

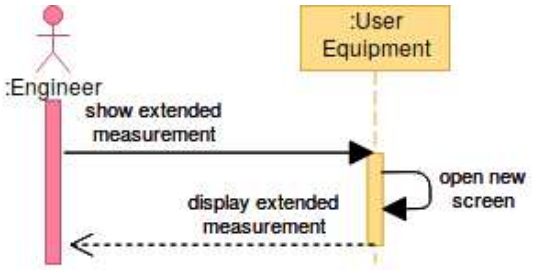

Figure 4.22: Sequence diagram for use case "See Extended Measurement"

Use case #50 Store Measurements

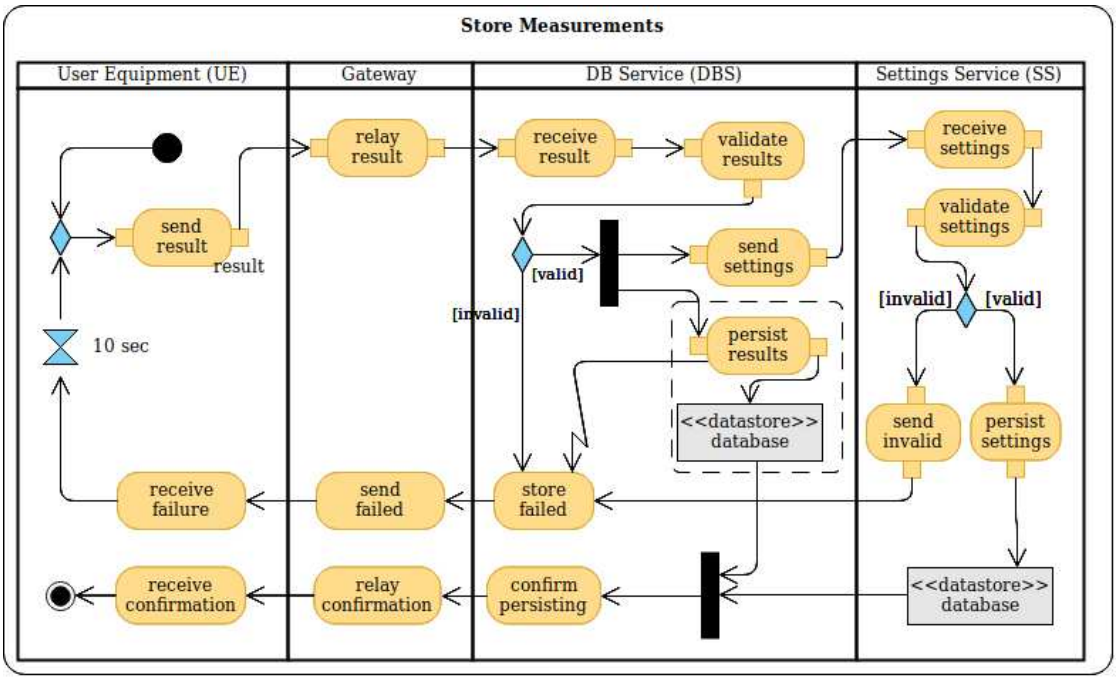

Figure 4.23: Activity diagram for use case “Store Measurements”

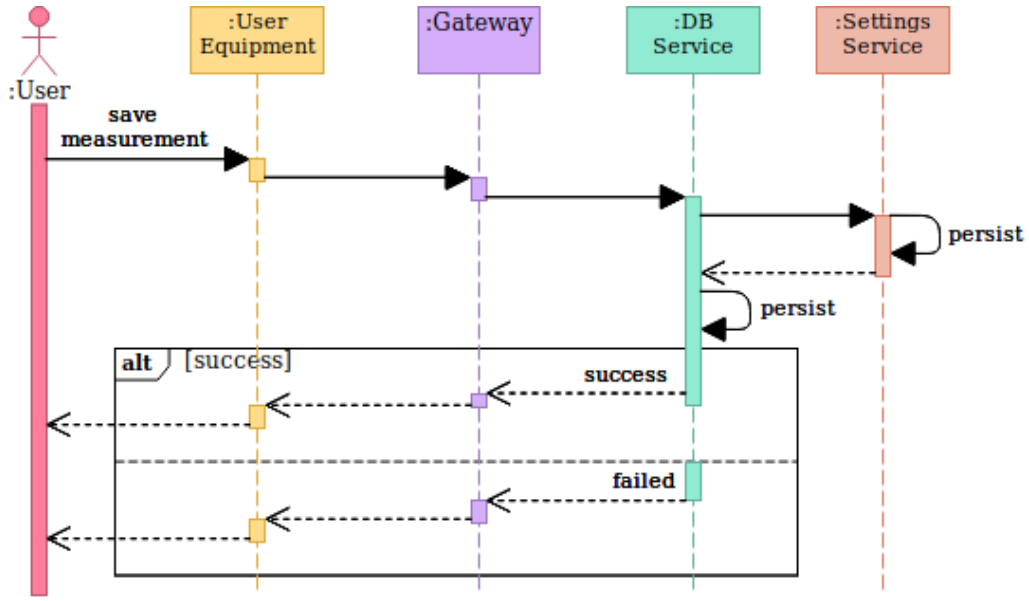

Figure 4.24: Activity diagram for use case “Store Measurements”

Use case #51 Filter Measurements

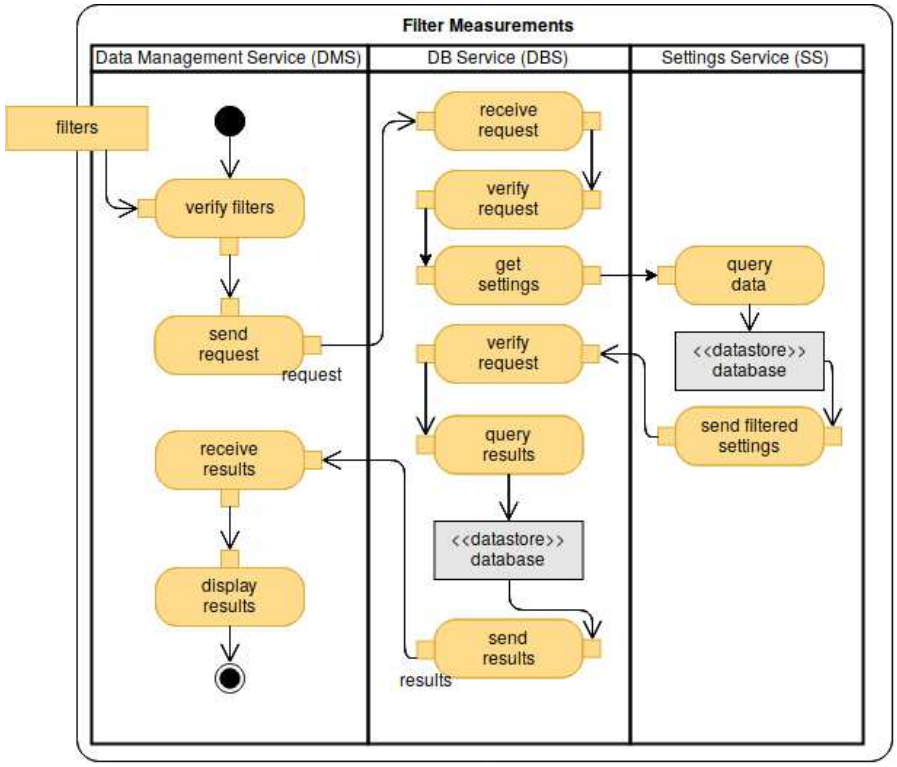

Figure 4.25: Sequence diagram for use case “Filter Measurements”

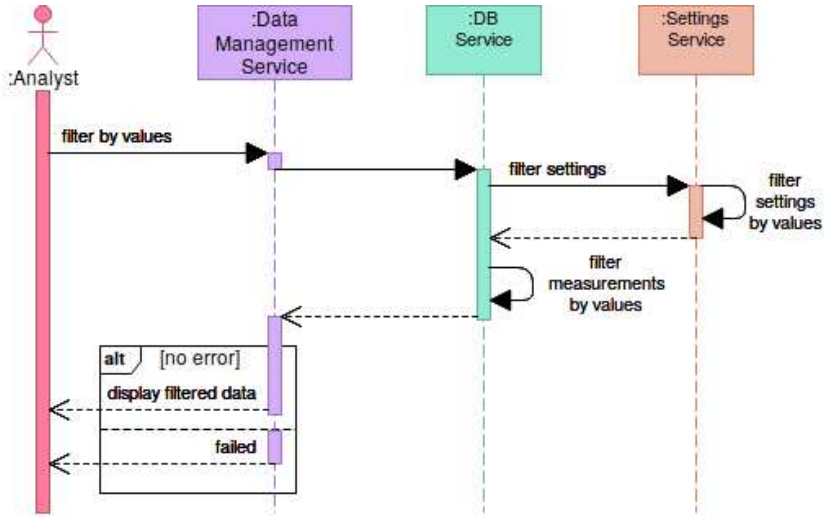

Figure 4.26: Sequence diagram for use case “Filter Measurements”

Use case #52 Extract Measurements

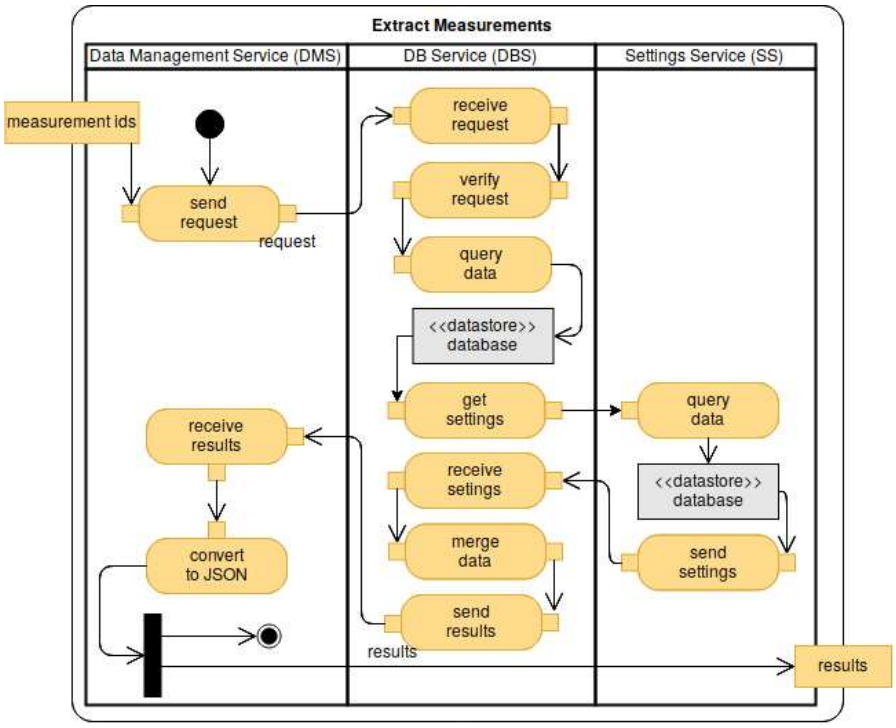

Figure 4.27: Sequence diagram for use case “Extract Measurements”

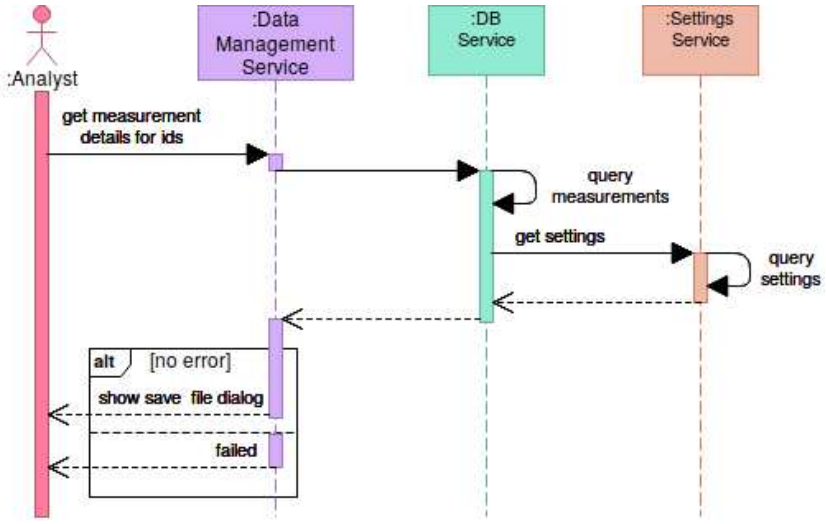

Figure 4.28: Sequence diagram for use case “Extract Measurements”

Use case #53 Sort Measurements

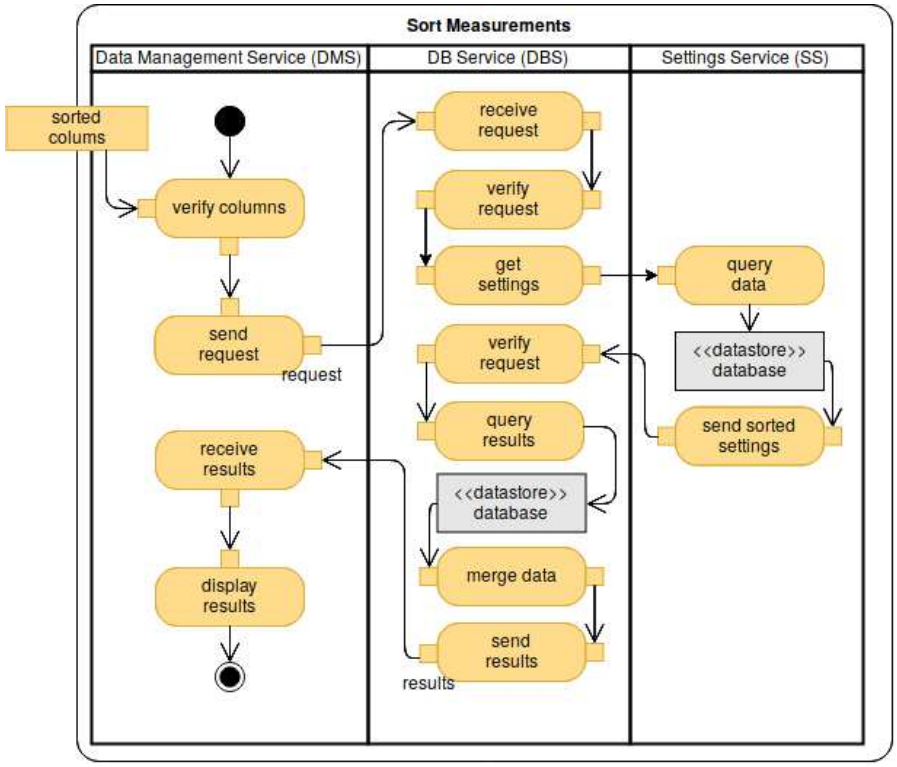

Figure 4.29: Sequence diagram for use case “Sort Measurements”

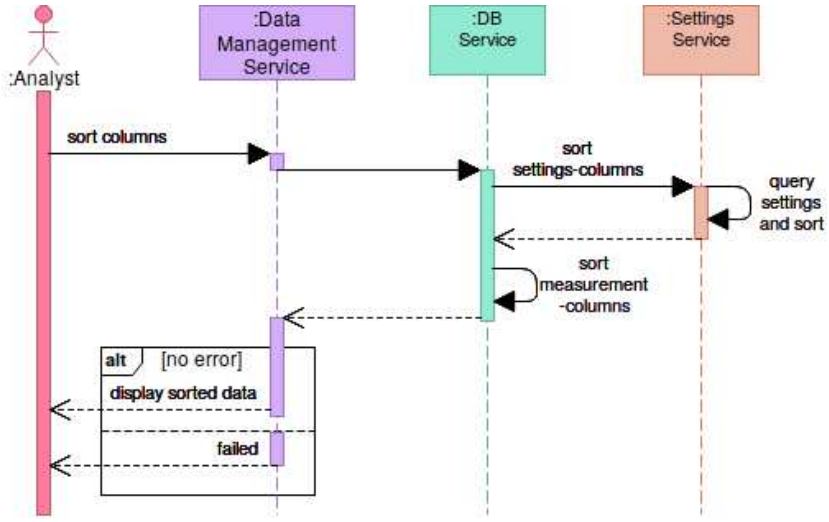

Figure 4.30: Sequence diagram for use case “Sort Measurements”

4.3.4 Architecture

A software architecture defines the structure of a software system. It exposes all components of the software system and handles the interaction between them. In short, it is like an architecture in the construction industry. However, there are some differences; software components are considerably more flexible than construction industry components. Therefore, the software architecture also focuses on non-functional requirements like scalability or performance to describe their dynamic behavior [69, p. 200].

After careful consideration, a service-based architecture was chosen over a monolithic or layered software architecture. In contrast to monolithic and layers architecture, the service-based architecture provides an inherent scalability feature required by user-story #2. In a service-based software architecture, the system separates its components into loosely coupled, coherent services with well-defined interfaces. In conclusion, a service-based architecture allows for a separation of concerns and, therefore, better maintainability. This project uses a particular form of services-based architecture, called microservices. Martin Fowler describes the microservices architectural style on his website as "an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API" [78]. Thereby each service contains a small part of the functionality of the software. For example, small services can handle tasks like a CRUSP measurement, authentication, or monitoring. Furthermore, microservices use an approach called "polyglot persistence". Thus, each service uses a separate database; all data follows the microservice architecture. Additionally, microservices add expandability to the software product. Indeed, expandability facilitates adding new functionality to the software product. In summary, the microservice architecture was chosen due to its excellent scalability and maintainability characteristics.

When looking at the software architecture in a higher granularity, each service itself also requires a suitable architecture. Most services are small and expose an interface, have a business logic, and have a data persistence logic. Therefore, an efficient and easy to understand 3-tier layered architecture with an interface layer, a business layer, and a data layer was chosen. Communication flows from top to bottom. Hence, each layer offers an interface to its super-incumbent layer while hiding its internal complexity. Furthermore, a layered architecture offers separation of concerns and exchangeability with its loosely coupled layers [69, p. 211].

Figure 4.31 shows an overview of all seven services. Rectangles depict services, arrows represent dependencies between two services.

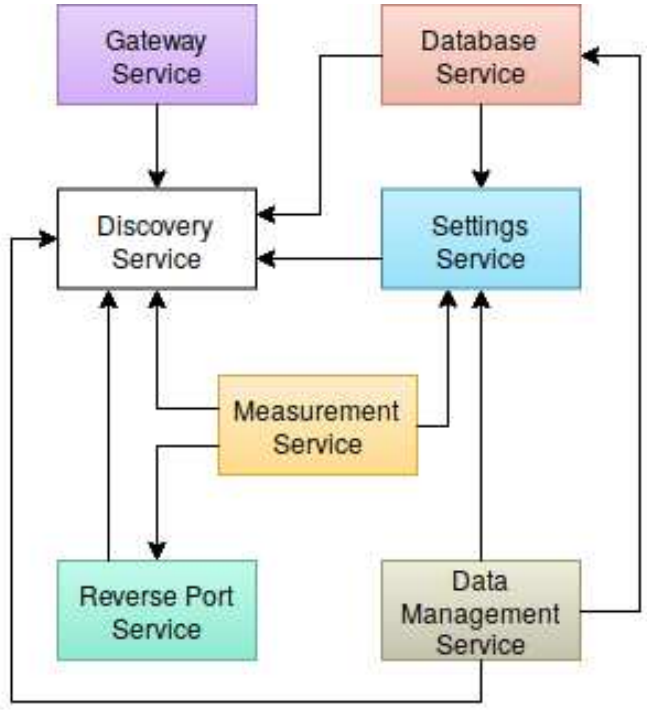

Figure 4.31: Services Overview

Discovery Service When communicating with other services, each service needs to know all other available service instances. However, in a microservice architecture, it is unfeasible for each service to maintain a global view of the system. Therefore, this project uses a dedicated service responsible for discovering other services. Thereby, service instances must register at the discovery service if they want to be findable. Finally, each other service can query the current state of the discovery service to see all available service instances. In short, this service holds and maintains an overview of all current available service instances.

Gateway Service This service is the API gateway for applications outside of the microservice systems boundaries; for example, the Android application or the web browser. It queries the discovery service for all available service instances and then redirect the user-requests to the available instance.

Measurement Service The measurement service handles measurement requests to perform CRUSP downlink and uplink measurements. Thereby, it listens to new requests, verifies the received settings configuration, conducts the measurement, and then calculates the results.

Database Service This service persists the measurement results combined with additional information about the measurements like GPS location or device identification. In a microservices architecture, each service usually handles the data it creates. However, the CRUSP measurement process is time-critical and performance-critical. It heavily relies on timing information at the nanosecond level. Therefore, the database service needs to run as a separate service.

Reverse Port Service The reverse port service provides the measurement service with a publicly available port of the measurement service. This public port helps the CRUSP algorithm to send UDP packets from the measurement service to the user equipment, bypassing restrictions introduced by specific technologies.

Settings Service Each request to the measurement service contains a configuration called *CRUSP settings*. The settings service verifies, persists, and provides those CRUSP settings. Furthermore, it manages the default CRUSP settings if a user equipment does not provide its configuration.

Data Management Service The data management service operates as a frontend for data analysts. Analysts can view all measurement results, filter, sort, and export them. Furthermore, the service offers an interface to change the CRUSP default settings.

4.4 Implementation

After addressing the steps of *planning* and *designing*, the emphasis shifted to the *implementation* phase. This part of the thesis discusses the technologies used for the software project as well as implementation-specific decisions.

4.4.1 Virtualization

Virtualization plays a crucial role in software development. Schatten et al. [69, p. 403] defined it as "emulation of hardware in software". For example, software can virtualize computer hardware, computer networks, and computer storage. As a result, virtualization supports software installation and software configuration in a controlled and reproducible environment. Thus, virtualized environments provide functioning software as intended by the developers and independent of the host operating system.

A special form of virtualization is called *containerization*. Thereby the virtualization focuses on applications rather than complete operating systems. This approach avoids the overhead of hosting a full operating system in each virtual environment. Therefore, virtual containers start and stop much faster than virtual machines. Consequently, developers prefer the faster creation and deployment of containers over virtual machines. To summarize, containers are lightweight, reproduce-able and can run on various hardware and operating systems.

The de facto standard [79] for Linux containers is Docker [80]. Therefore, this technology is a solid choice for virtualization. Each service and each database runs in a separate container.

4.4.2 Service Coordination

When using a microservice architecture, someone needs to coordinate and instantiate the service. Coordination enables agile and process-oriented applications, perfectly suitable for scaling [69, p. 207]. As defined in the previous subsection, each service runs in its own Docker container. Consequently, a tool capable of managing multiple Docker containers was needed. Three prominent representatives are the orchestration tool *Kubernetes* [81], the orchestration tool *Docker Swarm* [82], and the container management tool *Docker Compose* [83]. Kubernetes is the most mature candidate; it is open source, production-ready, and supported by Google [84]. However, it is also the most complex of the three candidates. In contrast, Docker Swarm is more comfortable to set up and works well with the Docker API, but lacks some features Kubernetes has to offer [84]. The last considered option is Docker Compose. It is the most straightforward approach, configured in one file. Furthermore, Docker Compose is well suited for projects in the development phase as well as a production environment for single-host machines. The Institute of Telecommunications at the TU Wien provides one server to host the software. Hence no "Platform as a Service" (PaaS) provider has to be paid; this keeps the operational costs at a low level. Consequently, the decision was taken to use Docker Compose, even

though it does not fit perfectly for production environments. If the need for a more production-friendly environment arises, it is possible to migrate to Docker Swarm through its integrated compatibility with Docker Compose.

Figure 4.32 shows an overview of all docker containers and their dependencies. A rectangle depicts a container used to hold a service. For example, the *settings service* spawns a single container, and the database for the settings service spawns a separate container called "postgres-settings-service". Additionally, the circles show the ports published to the outside for each service.

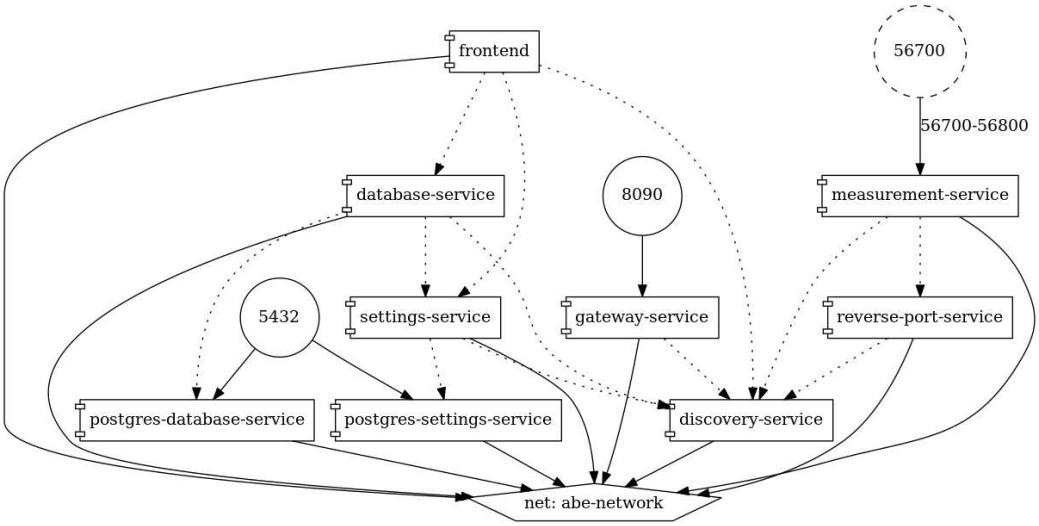

Figure 4.32: Docker-compose diagram

4.4.3 Technology Stack for Services

The microservices architecture contains many different services. Figure 4.33 depicts an overview of the relevant technologies used in the software project.

Basic Setup of a Service In a microservice architecture, a service usually offers a lightweight communication mechanism like the *HTTP resource API* [78]. Furthermore, services usually perform similar tasks like searching for other services, client-side load balancing, or logging. Since all services share these requirements, all of them are solved the same way. Therefore, most services in this architecture build upon the same collection of technologies.

The Java programming language [85] functions as a basis for the services. Java is a well known programming language [86], suited for larger projects due to its extensive support and community. Additionally, Java is an object-oriented programming language; this offers built-in separation of concerns and enables good maintainability. Furthermore, the

developer feels most comfortable with Java. Hence, the primary programming language is Java.

A popular and proven Java-based framework to provide a REST API is the *Spring Boot* [87] framework. It facilitates self-configured and production-ready applications on the Java EE (Enterprise Edition) platform. *Spring Cloud* [88] is a popular framework to enable distributed systems like microservices. Both frameworks, Spring Boot and Spring Cloud, work well with each other and are easily extendable with other technologies supported by the Spring family. Therefore, both frameworks build a superb basis for the microservices written in Java.

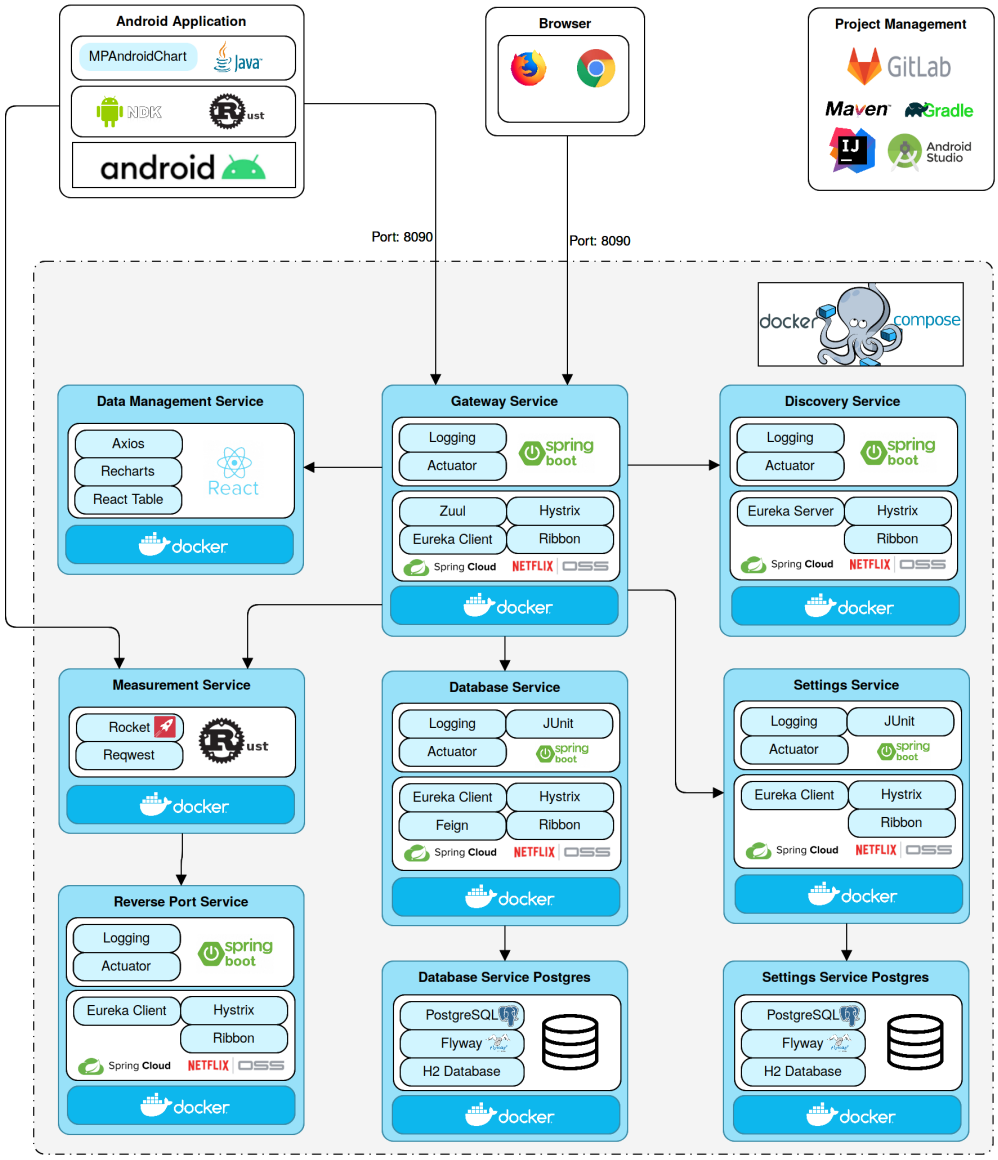

Figure 4.33: Technology stack

Spring Boot is a collection of technologies that work well with each other; for example, the Logging framework and the Actuator API. The Logging framework enables logging via its integrated logging API. The Actuator API makes health information about the service available at runtime. Spring Cloud provides the tools Hystrix [89], Ribbon [90], and Eureka Client [91] for the basic setup of a service. The Hystrix library adds latency and fault tolerance to a microservice. In contrast, Ribbon is responsible for the client-side load balancing. The Eureka Client helps the service to register at the discovery service.

Measurement Service Java has a few shortcomings when it comes to high-performance computing. A Java prototype developed by employees of the Institute of Telecommunications delivered inaccurate measurement results. In contrast, a prototype written in the significant faster programming language C, delivered accurate results. Therefore, the Java language seems to be too slow to provide precise measurements exact to the nanosecond. An advantage of building services in a microservice architecture is the usage of adequate languages and tools tailored for each problem. For example, a high-performance application can be written in a suitable system programming language while still cooperating with other services in the architecture. Subsequently, the measurement services needed a suitable system programming language.

Three prominent system language candidates are *C++* [92], *Rust* [93], and *Go* [94]. C++ is a well-known language, supporting multiple programming paradigms like object-oriented, functional, imperative, and procedural programming. Further advantages are its high speed, excellent support, and big community. Nevertheless, C++ also requires developers to understand low-level aspects of programming languages. Additionally, developers have to be highly cautious to ensure memory safety in their code. In contrast, Rust combines memory safety with similar speed as C++ [95]. However, Rust is a relatively young language; the first stable version was released in 2015 [96]. As a result, the language still lacks libraries and frameworks when compared to C++. Furthermore, its learning curve is steep due to complex programming concepts like ownership or "borrow checker". Nevertheless, Rust is production-ready and already used in big software projects like Mozilla Firefox or Dropbox [97]. According to stackoverflow.com, Rust is also the most renowned language the fourth year in a row [86]. Additionally, it supports various important programming paradigms like object-oriented and functional programming. On the contrary, the Go language only supports the imperative paradigm and procedural paradigm. Furthermore, it does not match the speed of C++ or Rust [98]. Hence, the Go language does not seem perfectly suitable even though Google supports it. In conclusion, the decision was taken to use Rust due to its combination of execution speed and safe code.

Unfortunately, Rust does not support the Spring framework; another solution was needed to provide a REST API. Therefore, the web framework Rocket [99] was chosen. Additionally, the measurement client needed an HTTP client to make calls to the "reverse port service". After careful consideration, the decision was made to use Reqwest [100], a simple high-level HTTP client for Rust.

Discovery Service The discovery service builds upon the basic service architecture mentioned above, a Spring Boot application enhanced with Spring Cloud APIs and libraries. Additionally, it uses the Eureka Server library [91]. This library enables service discovery by providing a global state of all available services. Thereby all services can register themselves with their IP address, port-number and instance name at the discovery service. This library is part of the Spring Cloud family and therefore works well with the other Spring Boot and Spring Cloud technologies used in this project.

Gateway Service The gateway service also builds upon the basic service framework. Furthermore, it uses the Zuul project from Netflix [101], a gateway that provides dynamic routing for the application. The Zuul project is also part of the Spring Cloud family.

Database Service The database service enhanced the basic service setup with multiple technologies. Feign [102] provides a declarative web service client and works well with other Spring Cloud technologies like Eureka or Ribbon. In short, Feign facilitates communication with other services. JUnit [103] is used to ensure high code quality by providing a test framework. Furthermore, the database technologies PostgreSQL, Flyway, and H2 Database were picked. A detailed description of these database technologies can be found in Subsection 4.4.4.

Settings Service The settings service has the same requirements as the database service when it comes to their behavior in the microservice architecture. Therefore, they use the same technologies.

Reverse Port Service The reverse port service only uses the basic setup for a service. It includes already all the necessary technologies.

Frontend The frontend is the applications' data management service. Thus, it exposes the information stored in the database and offers functionality like sorting, filtering, or exporting. A requirement of the frontend is easy access via a platform-independent technology. A Java frontend fulfills this requirement; however, it lacks easy portability and easy application updates. In contrast, web applications are also platform-independent, do not need to be installed, are easily updated, and can visualize highly dynamic content. Therefore, it was chosen to use a web application for the data management service. When developing web applications with dynamic content, JavaScript is the language to go. Three popular web frameworks [86] for single page applications are React [104], Angular [105], and Vue.js [106]. All 3 of them are suitable, whereby React and Vue.js are more lightweight and therefore more appropriate for the project than Angular [107]. Additionally, the author had some experience with React. Consequently, React was picked as the JavaScript frontend framework.

React offers many additional libraries and frameworks created by the community. As a simple HTTP client, Axios [108] handles the HTTP request to the microservices. The Re-

act library Recharts [109] visualized the available information in diagrams. Furthermore, the React Table component displays the stored information in a clear data grid.

As a basis, the Frontend microservice uses a docker container.

4.4.4 Data storage

The data collected by the software has to be organized and needs to be easily accessible. A database is an excellent way to store a vast amount of data.

Database Based on the requirements, the two relational databases PostgreSQL [110] and TimescaleDB [111] and two NoSQL databases InfluxDB [112] and CouchDB [113] were taken into consideration. All four mentioned databases are open source. PostgreSQL is a relational database management system which specializes in storing structured data and providing fast access to it. Furthermore, it is a proven database [110]; however, it does not scale as good as some NoSQL solutions. The TimescaleDB is an extension of the PostgreSQL database with a focus on time-series data. Therefore, retrieving time-series information takes considerably less time than in traditional relational databases. InfluxDB is a database which treats time as first-class citizen. In contrast to the relational databases, InfluxDB works like a NoSQL database with high priority on scalability and focus on time-series data. Subsequently, retrieving time-series out of a massive amount of data is very fast. The other NoSQL database taken into consideration is CouchDB. Its document-oriented data model allows the database to store semi-structured data; they do not have to be thoroughly structured. Additionally, it is highly scalable and would, therefore, fit the requirements of user story #2.

When analyzing the raw data produced by each measurement, they revealed that the measurements follow a structured organization. This structured data fits the relational data model, which enables the use of a relational database like TimescaleDB and PostgreSQL. Furthermore, interviews with the data analysts were conducted to find all use cases when analyzing data. As a result of this, no specific use case was found where access to a fraction of time-series data was needed. Consequently, a highly scalable, relational database would fit perfectly the requirements and a time-series database is not needed. However, none of the four mentioned databases and none other proven database fits all requirements perfectly. In the end, it was decided to use the PostgreSQL database due to its excellent performance and well-known technology to the developer.

Testing the services is an important measure to ensure a high quality of software [69, p. 133]. However, it is infeasible to use a database in production for testing purposes. If automatic tests perform operations on the production data, the database may manipulate or loose information. Therefore, a separate test database was the solution for this task. A fitting database management system for this role is the H2 Database [114]. It offers an in-memory option for high performance and easy startup. Furthermore it is supported by the Spring framework. Consequently, this project uses the lightweight H2 Database for testing.

Migration When using relational databases, each table is bound to a fixed schema. Changes in the schema would break the table; consequently, a tedious migration process from one database version to the next version would start. Nevertheless, changes will happen to the schema; for example, when new data is available, the format of data changes or some data is not available anymore. Subsequently, a migration tool from one database version to the next database version was needed. The migration tool called Flyway [115] works well with the other used technologies Spring Boot, PostgreSQL, and H2 Database. Therefore, Flyway was picket as a migration tool for H2 and PostgreSQL.

4.4.5 Mobile Platform

Next, a suitable mobile platform to carry out the measurements was needed. The most popular mobile platforms worldwide are currently (August 2019) Android and iOS. According to statcounter.com [116], Android holds a market share of 76.23%, whereas iOS holds a market share of 22.17%. Subsequently, Android, with its significantly higher market penetration is a more suitable platform than iOS. A further advantage of Android is the supported programming languages. Android enables development with the languages *Java* and *Kotlin*, whereas iOS supports the languages *Swift* and *Objective-C*. However, Java is already heavily used in the project; therefore, it was convenient to also use it for Android. Furthermore, the Institute of Telecommunications owns many suitable Android test devices. In summary, it was convenient to use Android as a mobile platform and Java as a programming language for the mobile application.

Measurement Client As with the measurement server, the measurement client demands precise results to the nanosecond. Therefore, the measurement client part of the Android application requires a programming language with better runtime performance than Java or Kotlin. When encountering the same problem within the *Measurement Service*, *Rust* [93] was picked to solve it. Hence, it was convenient to use Rust also for the counterpart at the mobile platform. However, Android does not officially support Rust. Therefore, an interface was needed to call Rust functions in Java and return Java objects in Rust. The Java Native Interface (JNI) framework provides this interface. Furthermore, the Rust-binary needed to be cross-compiled for the Android platform. Luckily both, the JNI and the cross-compiler, are part of the Android Native Development Kit (NDK). Hence, the NDK became an important tool-set in the project for enabling the native Rust program at the Android platform.

API Version When developing a mobile application for Android, an API version had to be picked. The API version determines, which Android versions can run the developed application. Thus, it is a challenge to find the sweet spot between an old version supporting many devices and a new version with additional API features. Table 4.5 shows that applications can serve at least 95% of all android devices with API level 19. However, when gathering the requirements, the analysts stated that a large market pervasion is not essential. Additionally, an examination of all available test devices

Android Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.3%
4.1.x	Jelly Bean	16	1.2%
4.2.x		17	1.5%
4.3		18	0.5%
4.4	KitKat	19	6.9%
5.0	Lollipop	21	3.0%
5.1		22	11.5%
6.0	Marshmallow	23	16.9%
7.0	Nougat	24	11.4%
7.1		25	7.8%
8.0	Oreo	26	12.9%
8.1		27	15.4%
9.0	Pie	28	10.4%

Table 4.5: Android platform distribution, collected during a 7-day period ending on May 7th, 2019 [117]

revealed that the highest supported Android version was 8.0. Therefore, API level 26 was picked, the highest API version still supported by some test devices.

CHAPTER 5

Results

The first chapter introduced the research question to find an agent-based non-intrusive methodology to characterize reactive mobile communications networks. After selecting an appropriate method in Section 3.3, an agent-based solution using *CRUSP* was developed in Chapter 4. This chapter answers the question if the developed solution is *accurate* when used in a reactive mobile communication network.

In order to answer this question, this chapter breaks it down into several sub-questions:

- How is it possible to show the accuracy of CRUSP for Mobile?
- What are the ingredients of the test infrastructure?
- What do the results imply?
- Where did the application succeed?
- Where did the application fail?
- What problems were not solved?

5.1 How to show that CRUSP for Mobile is accurate?

In order to show the accuracy of CRUSP for Mobile, a series of measurements were conducted. For reproducibility, they took place in a controlled environment. Additionally, reference measurements were taken with the well-known tool *iPerf3* [48]. Afterwards, CRUSP for Mobile results were compared with the reference measurements. It was assumed that if the average error is lower than 5%, CRUSP for Mobile is accurate.

5.2 Test Setup

A vital challenge in experiments is the reproducibility of results. However, experiments in publicly available mobile communication networks face the problem of multiple uncon-

trollable elements. The always-changing cell load, signal strength, and inference power influence the results fundamentally.

Therefore, the tests happened in a shielded reference cell with a constant cell load. In this cell, a technician can adjust the attenuation level freely. Consequently, the cell can provide a mobile phone with a constant signal strength.

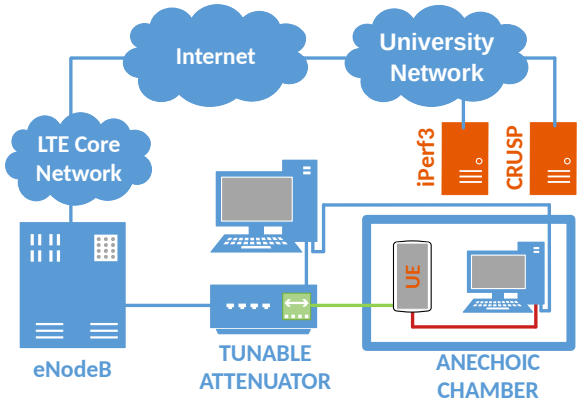

Figure 5.1: Network setup

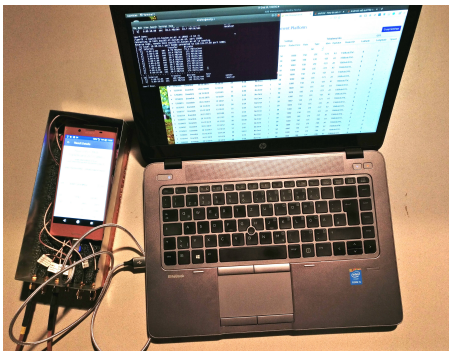

Figure 5.2: Anechoic chamber setup

In short, the test setup consisted of a server in the university network, a reference cell, and an anechoic chamber. Figure 5.1 depicts all vital components of the setup. In order to allow direct connections from a mobile phone, the server sits in front of the university firewall. Further, the measurements assumed that the core network and the Internet are faster than the LTE access network.

The setup for the first LTE measurements used a mobile phone (Sony Xperia XZ Premium) in the anechoic chamber. As Figure 5.2 shows, the mobile phone was connected to the base station to achieve a stable signal strength. The laptop next to the phone triggered iPerf3 reference measurements via USB and changed the attenuation level in the attenuator.

In contrast, the setups for the measurements with the *Xiaomi Mi Mix 3 5G* (LTE with activated carrier aggregation) used a wireless connection.

To summarize, the reference cell, in combination with the anechoic chamber, could provide a constant signal power and a constant cell load.

5.3 Evaluation

Two experiments were conducted to show the accuracy in different situations.

5.3.1 1st Experiment: LTE

In the first experiment, the author examined the accuracy of CRUSP for Mobile in an LTE network. Therefore, a CRUSP configuration was deployed with a data volume of $0.93MB$ and packet size of $1kB$ per estimation. The sophisticated rates use a Δ_{min} of 500 ms. Furthermore, the RSRP-value for each measurement was logged in order to compare the results based on their signal strength. Thereby, RSRP stands for *Reference Signal Received Power* and represents the strength of the LTE signal.

Test Procedure This experiment went through the following steps:

1. SET the CRUSP configuration in CRUSP for Mobile
2. SET the attenuation level at the base station to $52dB$
3. WHILE the attenuation level is $\leq 78dB$
   a) PERFORM one iPerf3 measurement for 10 seconds
   b) PERFORM ten CRUSP measurements with CRUSP for Mobile
   c) INCREASE the attenuation level by $2dB$

Results As Figure 5.3 (a) visualizes, the CRUSP for Mobile estimations are similar to the iPerf3 reference measurements.

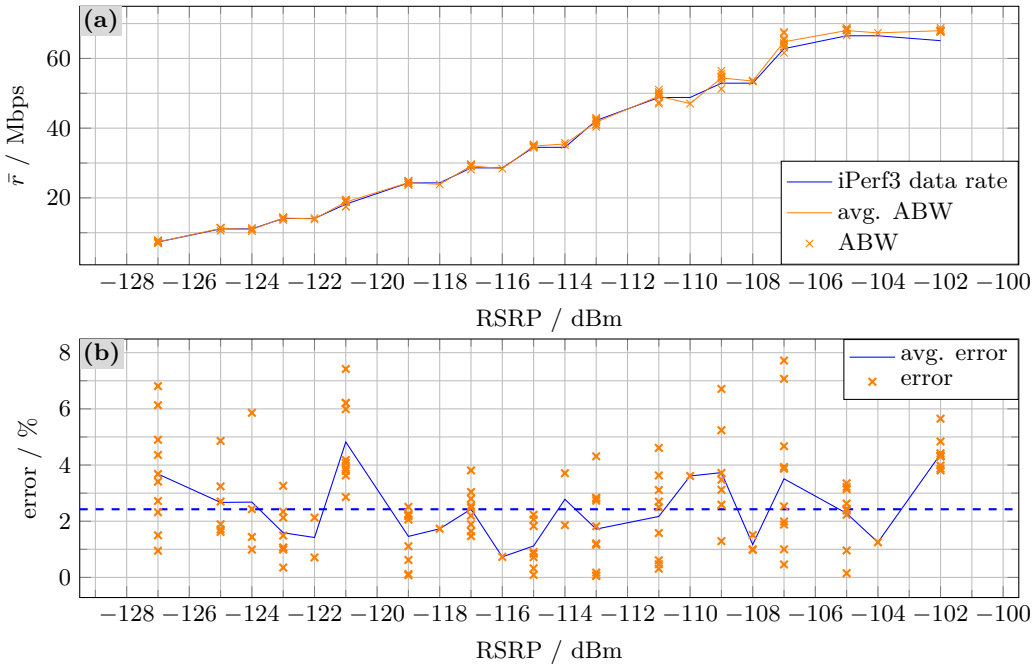

Figure 5.3: (a) CRUSP vs iPerf3 for each RSRP-value (b) error of CRUSP estimations to iPerf3 reference measurement for each RSRP-value

Although the phone was wired to the cell, there was a slight fluctuation (around one dBm) in RSRP. For example, ten measurements with attenuation level 74 resulted in six

measurements with an RSRP of -125 dBm and four measurements with an RSRP of -124 dBm.

Figure 5.3 (b) shows a mean error of 2.44% to the iPerf3 results. Moreover, the error appears to be independent of the RSRP. However, CRUSP for Mobile seems to slightly overestimate at low RSRP-values (RSRP < -108 dBm).

In summary, the CRUSP for Mobile estimations achieve similar results compared to the 10-second iPerf3 measurements in an LTE network. This result coincides with the experiments on CRUSP by Raida et al. [66].

5.3.2 2nd Experiment: LTE with Carrier Aggregation

5G will be able to reach peak data rates up to 20 GBit/s. In order to simulate data rates that exceed standard LTE connections, experiment two used a technique called *carrier aggregation*. Thereby, multiple component carriers get aggregated to increase the data capacity [118]. Subsequently, downlink data rates up to 600 MBit/s were theoretically possible. In contrast, LTE without carrier aggregation could achieve downlink data rates up to 150 MBit/s theoretically.

Table 5.1 shows the configurations used for this experiment. In order to calculate the ABW with the sophisticated approach, the formula needed to define a value when two consecutive datagrams are separated into two bursts (Δ_{min}). A reference measurement with iPerf3 resulted in an average downlink rate of 423.4 MBit/s. Therefore, a sending rate of 450 MBit/s was picked for CRUSP.

This experiment used a normal *Xiaomi Mi Mix 3 5G* mobile phone wirelessly connected to the reference cell. However, the wireless connection resulted in RSRP fluctuations from -88 dBm to -103 dBm even though the mobile phone was placed at the same position without moving around, and the attenuation level stayed constant.

	1st LTE-CA	2nd LTE-CA	3rd LTE-CA	4th LTE-CA	5th LTE CA
Volume	930 kB	1860 kB	1860	2000	4000
UDP Size	1000 Byte	1000 Byte	4000 Byte	5000 Byte	5000 Byte
Δ_{min}	500 ms	500 ms	500 ms	500 ms	500 ms
Send rate	450 MBit/s	450 MBit/s	450 MBit/s	450 MBit/s	450 MBit/s

Table 5.1: CRUSP configurations

Test Procedure Following steps were taken for this experiment:

1. SET constant attenuation level at the base station
2. PERFORM five iPerf3 measurements for 10 seconds
3. FOR EACH CRUSP configuration
   a) PERFORM ten CRUSP measurements with CRUSP for Mobile

Results As Figure 5.4 shows, the CRUSP measurements have a massive gap to the iPerf3 reference measurements. However, the higher the data volume and the bigger the datagram size, the closer the results get.

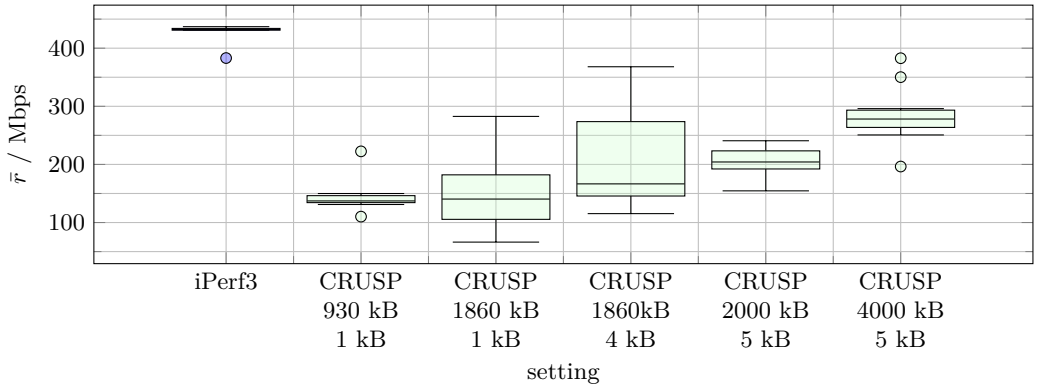

Figure 5.4: CRUSP for Mobile estimations and iPerf3 reference measurements with LTE carrier aggregation

The massive deviation can have multiple causes:

- Slow *measurement-service*: This assumption can be rejected because the service has been tested in the university network for up to 500 Mbit/s.
- Slow connection from *measurement-service* to Internet: This problem is unlikely because the university has a fast connection to the Internet.
- Slow mobile phone: There may arrived too many datagrams at a short time. Consequently, the processing of the datagrams prolongs the overall measurement time. However, a custom CRUSP configuration may alleviate the problem.
- Inappropriate CRUSP configuration: As Figure 5.4 hinted, an increase in total volume or packet size may help to align the CRUSP measurements with the iPerf3 reference measurements.

In summary, CRUSP for Mobile was inaccurate when handling data rates of about 420 MBit/s. These results suggest that CRUSP for Mobile might have difficulties to estimate data rates higher than 150 MBit/s with its standard configuration. Consequently, future work should investigate appropriate CRUSP configurations. Furthermore, a closer look at CRUSP's performance limitations on UEs needs to be taken.

CHAPTER 6

Applicability on 5G

With the emergence of 5G mobile communication networks, a variety of new areas of application areas will open up for CRUSP clients. However, what are the potential use cases for it? Moreover, how does 5G shape the requirements for a CRUSP implementation?

6.1 Applications for 5G

This section will identify potential use cases for a CRUSP deployment. Thereby, the three usage scenarios eMBB, URLLC, and mMTC defined in Chapter 1 serve as a basis. In order to get a feeling for possible CRUSP use cases, Figure 6.1 presents an overview of general use cases in 5G networks.

The following subsections will present five use cases in need of a quick and non-intrusive benchmarking tool.

6.1.1 eMBB: Multimedia Application Quality Detection

Video streaming and multimedia applications are still crucial use cases in 5G. According to Brockmeyer et al. [7, p. 3], the data rate is the most important KPI for multimedia applications. The upcoming 5G standard lowers latency and increases the data rate even further, enabling new kinds of multimedia applications [119]. For example, Sun et al. [120] introduced a 360-degree video streaming solution to support virtual reality (VR) and augmented reality (AR) technologies. Thereby, the increased data rates of the 5G networks enable the streaming of a 360-degree video to the UE. During the streaming process, a streaming client may predict the download bandwidth in order to choose the best quality for the next video chunks to load [120]. Thereby, a CRUSP client offers a way to estimate the available bandwidth. Based on this estimation, the application can choose the best quality for the available network resources.

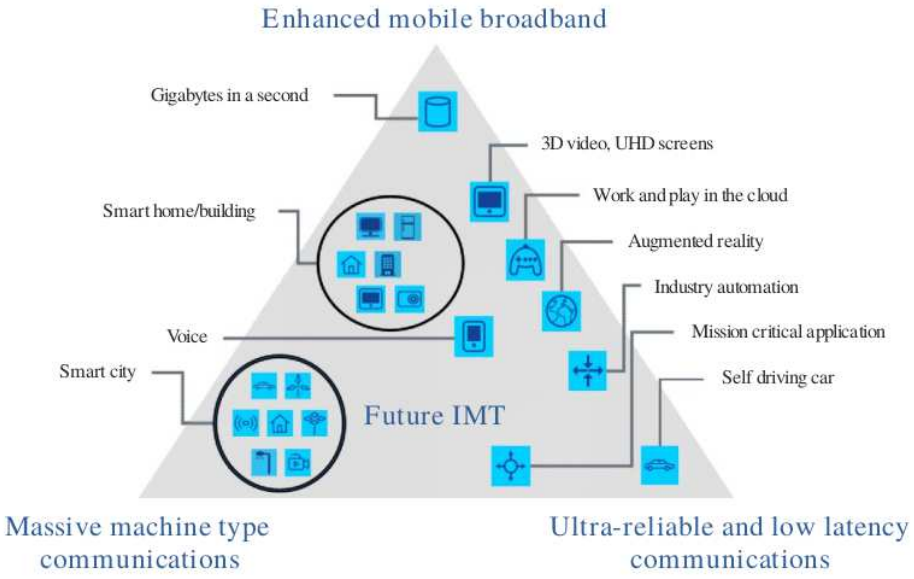

Figure 6.1: 5G use cases, from [1]

6.1.2 mMTC: Crowdsourcing for Coverage Maps

With 5G, millions of new devices like autonomous tractors, drones, and sensors will connect to the Internet. For example, drones can provide high-definition (HD) footage of crops and soil. Additionally, thousands of sensors may measure the humidity and upload it to the cloud for further computation.

However, mobile reception may be weak or not available in secluded areas. The poor signal can lead to an inability to upload or download data. Therefore, the autonomous industry may be interested in coverage maps of the mobile networks. Figure 6.2 shows such a coverage heatmap created by *RTR NetTest* measurements. The dark green areas represent the median downlink bandwidth in that area.

Coverage maps can be also useful for mobile network operators. With a detailed map, they can offer guarantees about the QoS levels in specific areas to a specific time.

In order to create a coverage map, mobile network operators need a sufficient amount of measurement points over space and time. Crowdsourcing can acquire such a massive amount of data by handing over the measurement task to a large number of users or machines [122]. Unfortunately, coverage maps created by traditional bandwidth measurement tools often lack enough measurement points in rural areas. The reason for the lack of data could be the intrusive and lengthy measurement method (e.g., RTR NetTest takes 7 seconds) as well as the unavailability of UEs in secluded areas.

Indeed, the quick and lightweight CRUSP mechanism can be a solution to this problem. It provides UEs and IoT devices with a non-intrusive measurement tool suitable for

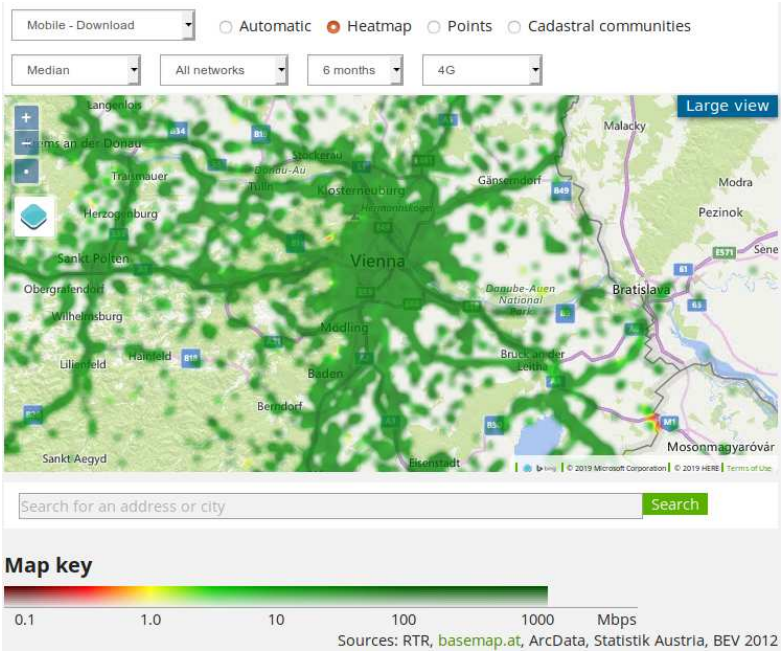

Figure 6.2: RTR heatmap from [121]

crowdsourcing.

6.1.3 URLLC: Vehicle-to-Vehicle Passing Assitant

Car drivers often struggle to overtake large vehicles like trucks because they block their field of vision. In order to solve this problem, Gomes et al. [123] want to leverage 5G's Vehicle-to-Vehicle (V2V) communication for a passing assistant.

As Figure 6.3 (a) shows, a car can get stuck behind a truck, being unable to see in front of the truck. Consequently, passing the truck becomes highly risky. The solution introduced by Gomes et al. [123] provides a see-through-system. Thereby, the truck operates a front camera and streams its footage to the car behind it. The see-through-system then projects a 3D model of the road and landscape extracted by the footage onto the windshield of the car as if the truck is transparent. In order to match projection and reality, the latency must be kept very low. Figure 6.3 (b) contains a visual representation of the transparent truck.

Crucial factors for this passing assistant are low latency and enough available bandwidth to transmit the video. CRUSP can provide the passing assistant with available bandwidth measurements to determine the video quality.

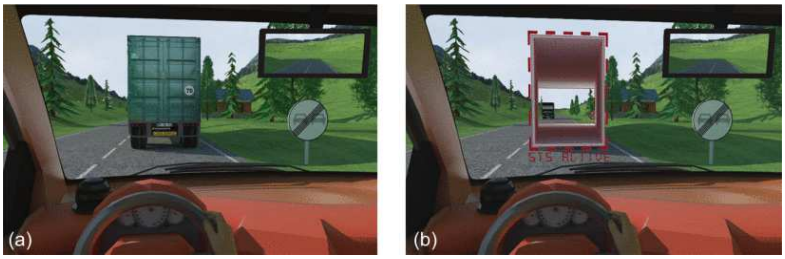

Figure 6.3: Use case *V2V communication* from [123]

6.1.4 Further use cases

5G brings not only new usage scenarios but enables new paradigms in addition to cloud computing. This paradigm shift creates excellent areas of application for CRUSP deployments.

Application/Computation Offloading in Cloud/Fog Computing Cloud computing has shaped the Internet in the last years by making data-heavy and complex computation available for everyone. However, cloud computing also introduced unwanted latency and severe limitations with location-unaware applications [124]. For example, data exchange often happens locally; migrating the data leads to additional traffic to the cloud. To solve these shortcomings, a new paradigm called *Fog Computing* emerged.

Bonomi et al. [124] described fog computing as an extension of cloud computing to the edge of the network. Thereby, the fog computing paradigm uses so-called *fog nodes*. These nodes are small data centers with highly virtualized storage, computing, and networking capabilities, deployed near the edge of the network. In short, fog computing moves the services from the cloud to the fog. Near the mobile user, the services achieve shorter latencies, better location awareness, and more efficient energy use. Additionally, the cloud is relieved from lightweight computation and location-specific content sharing. Consequently, fog computing supports 5G's vision of a sub-milliseconds latency [125]. For example, Yang et al. [126] proposed to deploy those fogs on top of 5G infrastructure.

Computation-intensive applications consume a significant amount of energy and time. In order to save energy and speed up processing time, applications can offload their tasks to the cloud or the fog. Thereby, the UE sends the application state to the cloud or fog. Afterwards, the fog nodes perform the calculation with their more powerful hardware resources. At the end, they send the result back to the UE.

Kumar et al. [127] identified three critical factors deciding about offloading: *available bandwidth*, *computational power* in the cloud (or fog), and the *amount of data to transfer*. Similarly, Ketyko et al. [128] named two essential components for offloading: *transfer time* and *execution time*, whereby the available bandwidth plays a significant role in the transfer time. A CRUSP client embodies an excellent tool to perform a quick and non-intrusive available bandwidth estimation to support the decision about offloading.

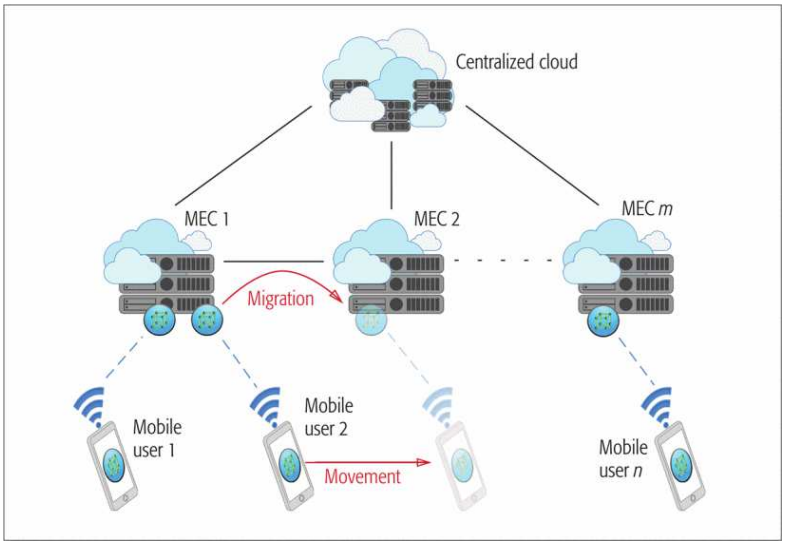

Figure 6.4: *Live Migration* use case on a typical MEC architecture with the *centralized cloud* at the top, multiple MEC servers (MEC1 - MEC m) in the middle, and multiple mobile users at the bottom, from [129]

Live Migration As fog computing, the paradigm *Mobile Edge Computing* (MEC) brings the cloud close to the user in order to achieve an ultra-low latency and location awareness. Interestingly, the terms fog computing and *MEC* are often used interchangeably. Nevertheless, both paradigms are ideally suited for deployment on 5G infrastructure.

In the MEC paradigm, mobile users are associated with an MEC server, which is running services and applications for them. However, the mobile users move around, and therefore the connection to the associated MEC server deteriorates. Consequently, the paradigm suggests to move the applications to a more suitable MEC server. Figure 6.4 depicts a typical MEC architecture. Imagine, *mobile user 2* is moving around, and after some time, is closer to *MEC2* than to *MEC1*. In order to provide *mobile user 2* with an optimal service quality, the application needs to migrate from *MEC1* to *MEC2*.

For that, Machen et al. [129] propose a framework where the application and its state migrate between MEC servers when the predicted migration benefit is higher than the migration costs. An integral part of these migration costs is the available bandwidth. Similarly, Manzalini et al. [130] want to migrate virtual machines across multiple distributed servers in a WAN. Again, the available bandwidth plays a crucial role in determining the migration costs. Luckily, a CRUSP client can provide an excellent way to estimate the available bandwidth.

6.2 Requirements for CRUSP Implementation in 5G

5G undergoes substantial changes to increase bandwidth, reduce latency, and support a massive amount of connected devices. Furthermore, technologies like MEC/Fog Comput-

ing, SDN, and NFV will shape 5G's architecture. All of those influences affect potential CRUSP deployments. However, what requirements must a CRUSP implementation meet to provide useful measurements? Additionally, how do the different usage scenarios eMBB, URLLC, and mMTC affect the requirements?

In order to get an overview, Figure 6.5 shows approximately the current capabilities of *CRUSP for Mobile* and the three usage scenarios. It becomes immediately visible that *CRUSP for Mobile* does not cover all usage scenarios. As we will see in the next paragraphs, a CRUSP implementation needs to make tradeoffs in order to cover specific corner stones.

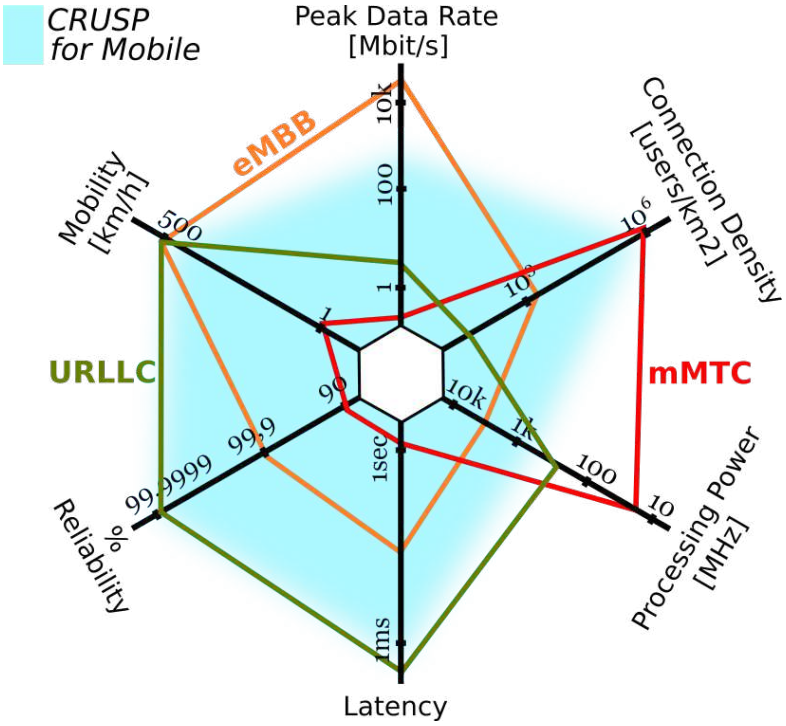

Figure 6.5: Service profiles for the three big 5G usage scenarios. The turquoise area marks the current capabilities of *CRUSP for Mobile*

The following paragraphs use *CRUSP for Mobile* as a basis. They informally describe additional requirements a potential CRUSP client may have in a 5G scenario.

Peak Data Rate *CRUSP for Mobile* does not yet support data data rates up to 20 Gbit/s. In order to improve the peak data rate, the CRUSP services need an ultra-fast connection to the Internet. Unfortunately, the university backbone is limited to 1 Gbit/s. Furthermore, the closer the services are to the Internet backbone, the less the data rate is distorted by long travels through the network. Therefore, a deployment of the CRUSP services in the cloud could solve these challenges.

Connection Density *CRUSP for Mobile* is designed to scale. However, it is running at a university server with limited resources at the moment. A deployment in the cloud or fog makes better use of the scalable architecture.

Mobility *CRUSP for Mobile* uses the GPS sensor of the devices to determine its geographical location. As long as the device is moving slow, the accuracy is high. However, if an eMBB-device is traveling with 200 km/h, it moves around 11 meters in a 0.2 seconds measurement. Consequently, location-sensitive applications need to find the best tradeoff between data rate accuracy and geographical accuracy.

Processing Power IoT-devices may have slow processors unable to measure the time accurately. For example, low powered CPUs running on 800 MHz can produce latencies up to 150 ns per system call to get the current time [131]. Consequently, a CRUSP client needs to find a way for exact timestamping when the processor of the device becomes the bottleneck.

Latency *CRUSP for Mobile* can handle ultra-low latencies in the sub-millisecond range. However, it also depends on the processor of the UE. Ultra-low latencies require exact timestamping with nanosecond accuracy.

Availability The CRUSP services are currently running on a university server. In order to achieve ultra-high availability, the services need to run in the cloud or fog.

Data Volume *CRUSP for Mobile's* configuration allows high and low data volumes for each measurement. However, IoT devices may require ultra-low data volumes where every byte counts. Therefore, uploading the complete measurement data for persistence may not be feasible anymore.

CRUSP Configuration 5G increases the flexibility of mobile communication networks drastically. For example, an eMBB application can achieve data rates of up to 20 Gbit/s, whereas a URLLC application may achieve 100 Kbit/s. Selecting an appropriate CRUSP configuration plays a crucial role in the accuracy, intrusiveness, and duration of a CRUSP client. For instance, if the CRUSP data volume is too low, it leads to inaccurate results [66]. In contrast, if the data volume is too high, the sent data will congest the network and block other communication for the UE. Consequently, finding appropriate configurations and adapting them to the circumstances is a crucial challenge to guarantee quick, accurate, and non-intrusive measurements.

Mobile Data Offloading A UE can connect to several networks (e.g., LTE and WiFi) at the same time and use them simultaneously. Thereby, it can decide to split the data and send one part via the first network (e.g., LTE) and another part via the second network (e.g., WiFi). Using multiple links at the same time is called *mobile data offloading* [132].

With the advent of 5G and its architectural changes, mobile offloading has the chance to make the breakthrough. For instance, Aujla et al. [133] proposes an intelligent mobile data offloading scheme by utilizing SDN. Consequently, a CRUSP client needs to ensure to use only one network in order to produce meaningful measurements.

Bypassing NAT and Firewalls Bonafiglia et al. [134] propose SDN controlled and virtualized network functions for 5G. Indeed, network functions like *Network Address Translation (NAT)* and *firewalls* are already widespread in mobile communication networks [135].

Unfortunately, NAT and firewalls pose problems for a CRUSP client. NAT shields the inside interface of the router from the outside traffic [136]. If there is no previous connection, routers can not find a corresponding address behind the NAT. For example, when a server wants to send a UDP datagram via NAT-routers to a UE, it needs to know the destination address of the UE behind the NAT. However, if it does not know this address, the UDP datagram can not be delivered.

Moreover, many firewalls block all incoming connections or only allow outgoing connections. Therefore, it is essential to find ways to deal with blocked traffic by NAT or firewalls.

CRUSP for Mobile uses a method called *UDP hole punching* [136] to bypass NATs and firewalls. Thereby, not the sender but the receiver sends the first message to punch a hole in the NAT or firewall. Afterwards, NAT and firewall recognize the established connection. Consequently, they allow the datagrams to pass.

Portability In the *Live Migration* use case, services migrate from one fog to another fog. Consequently, the CRUSP services need to be lightweight and portable in order to perform quick migrations.

Passive Reusability In case of absolute non-intrusiveness, a CRUSP client may use existing traffic to estimate the available bandwidth. The Internet is home to a lot of existing UDP traffic. Consequently, a client could utilize a Youtube video or firmware update to estimate the available bandwidth. A passive CRUSP client would reduce the injected data volume to 0 Byte. However, passive reusability is a topic for future work.

Background Measurements *CRUSP for Mobile* can not trigger measurements in the background of another Android application. Indeed, repeated measurements in the background of a client may be valuable for specific use cases (e.g., Coverage Map). However, this requirement may be in conflict with the *asynchronous access* aspect of mMTC [2].

Frequency of Measurements *CRUSP for Mobile* can perform automated measurements each second. Repeated measurements may be useful for specific use cases (e.g.,

Coverage Map). However, a high frequency of measurements stands in conflict with energy efficiency aspect of some IoT devices. Therefore, a CRUSP client needs to find a tradeoff between sufficient measurement frequency and energy efficiency.

In summary, the architectural changes add further requirements to a CRUSP implementation. Additionally, the three 5G usage profiles eMBB, URLLC, and mMTC, force it to tradeoffs. Subsequently, it may not be possible to serve all edge cases of 5G at once. Developers have to adapt a CRUSP system precisely to the needs of the application in order to achieve quick, non-intrusive, and accurate measurements.

CHAPTER 7

Conclusion

This thesis aimed to find a way for a fair benchmarking of mobile communication networks from the perspective of an end-user. Based on this research question, the author developed a software system able to estimate the currently available downlink and uplink data rate in a quick, data-saving, and accurate way. The user only needs an Android application. After pressing a button, the application displays the data rate in a fraction of a second.

The implementation of this application was the end product of a step-by-step process to answer the research question. The following contributions were made in the course of this work:

- Analysis of characteristics used in research to describe mobile networks: As a result, bandwidth and delay were selected as the most significant characteristics. Indeed, suitable methods for latency already exist, so the focus shifted to *bandwidth*.
- Analysis of metrics to describe the bandwidth: In the end, *available bandwidth* (ABW) was chosen to be the most meaningful metric due to its broad applicability and excellent approximation of throughput.
- Analysis of suitable methods to estimate the ABW in mobile networks: Based on this review, *CRUSP* was picked as the most suitable method for implementation on an agent due to its short duration, high accuracy, and data-saving properties.
- Development of a novel measurement system that determines the ABW at the push of a button in only a fraction of a second: In order to be suitable for crowdsourcing, Android was picked as an operating system. Additionally, the software system persists those measurements and offers a measurement analysis tool. The development process included phases for requirements engineering, project planning, software design, implementation, and testing.

- Evaluation of the created software in terms of accuracy in a configurable and shielded reference cell: Thereby, repeated measurements of the ABW led to an average error of 2.44% to the well-known reference tool iPerf3 in a standard LTE cell. Compared to traditional tools, this is an excellent result considering that only a fraction of the time (~200 ms) and data (930 kB) was used. However, when measuring on a UE with activated carrier aggregation (with data rates of about 420 MBit/s), the *CRUSP for Mobile* results revealed a massive difference to the reference measurements.

- Elaboration of potential use cases for CRUSP and its software implementation with focus on the different usage scenarios eMBB, URLLC, and mMTC in 5G: Thereby, the potential use cases *multimedia application quality detection*, *V2V passing assistant*, *crowdsourcing for coverage maps*, *computation offloading*, and *live migration* were identified.

- Requirements a CRUSP implementation must meet in order to work for an application in 5G: In short, it is not possible anymore to develop a one-size-fits-all solution as in LTE. Each implementation must be tailored to the applications needs. Among other requirements, it needs to bypass NAT and firewalls, avoid mobile offloading, and find a suitable CRUSP configuration.

Despite finding a suitable method to determine the ABW, this work has some limitations. The instantaneous measurements bypass traffic shaping due to its short duration. For users who are thinking of replacing their traditional bandwidth estimation application with the one developed during the course of this thesis, it could lead to overestimated data rates.

In order to ensure quick and accurate measurements, CRUSP needs a suitable configuration for the current radio technology and signal strength. However, there are no studies on this topic yet. This system can act as a data-gathering tool for further research on suitable configurations for CRUSP in UMTS, LTE, and 5G.

Indeed, previous work already analyzed the CRUSP method with LTE live data and found it to be accurate. However, this software system lacks an evaluation with real data. Future work could take the developed system and test it for its accuracy with UMTS, LTE, and 5G data.

Another future research topic is *passive measurements* with CRUSP. For example, a client may use existing UDP traffic (e.g., youtube video) to infer the ABW.

List of Figures

List of Tables

Bibliography

[1] Radiocommunication Sector. Framework and overall objectives of the future development of IMT for 2020 and beyond. Technical report, International Telecommunication Union, September 2015.

[2] Javier Campos. Understanding the 5g NR Physical Layer. Technical report, Keysight Technologies, November 2017.

[3] Philipp Svoboda. SEMONE Projektantrag. Technical report, TU Wien.

[4] Federico Boccardi, Robert W. Heath, Angel Lozano, Thomas L. Marzetta, and Petar Popovski. Five disruptive technology directions for 5G. *IEEE Communications Magazine*, pages 74–80, February 2014.

[5] Vern Paxson. End-to-end Internet Packet Dynamics. In *Proceedings of the ACM SIGCOMM '97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 139–152, 1997.

[6] Andrew S. Tanenbaum and David J. Wetherall. *Computer Networks*. Prentice Hall Press, 2010.

[7] Monica Brockmeyer and Jawwad Shamsi. The principles of network monitoring. In *Selected Topics in Communication Networks and Distributed Systems*, pages 433–462. April 2010.

[8] Guy Almes, Matthew Zekauskas, Sunil Kalidindi, and Al Morton. A One-Way Delay Metric for IP Performance Metrics (IPPM). RFC 7679, January 2016. `https://tools.ietf.org/html/rfc7679`, visited on 2020-01-03.

[9] Matthew J. Zekauskas, Sunil Kalidindi, and Guy Almes. A Round-trip Delay Metric for IPPM. RFC 2681, September 1999. `https://tools.ietf.org/html/rfc2681`, visited on 2020-01-03.

[10] Ye Tian, Kai Xu, and N. Ansari. TCP in wireless environments: problems and solutions. *IEEE Communications Magazine*, pages S27–S32, March 2005.

[11] Cisco. Cisco Visual Networking Index: Forecast and Trends, 2017–2022 White Paper. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html, visited on 2020-01-03.

[12] Dixon Salcedo, Cesar Guerrero, and Roberto Martínez. Available Bandwidth Estimation Tools Metrics, Approaches and Performance. *International Journal of Communication Networks and Information Security (IJCNIS)*, December 2018.

[13] Fatih Abut and Martin Leischner. An Experimental Evaluation of Tools for Estimating Bandwidth-Related Metrics. *International Journal of Computer Network and Information Security*, page 1, July 2018.

[14] Ravi Prasad, Constantine Dovrolis, Meg Murray, and Kc Claffy. Bandwidth estimation: metrics, measurement techniques, and tools. *IEEE Network*, pages 27–35, November 2003.

[15] Mukta Airon and Neeraj Gupta. Bandwidth Estimation Tools and Techniques: A Review. Preprints. unpublished, October 2017.

[16] Constantine Dovrolis, Parameswaran Ramanathan, and David Moore. What do packet dispersion techniques measure? In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, April 2001.

[17] *Measurements and Analysis of End-to-End Internet Dynamics*. PhD thesis, University of California, Berkeley.

[18] Jacob Strauss, Dina Katabi, and Frans Kaashoek. A Measurement Study of Available Bandwidth Estimation Tools. pages 39–44, 2003.

[19] Giuseppe Aceto, Fabio Palumbo, Valerio Persico, and Antonio Pescapé. Available Bandwidth vs. Achievable Throughput Measurements in 4G Mobile Networks. In *2018 14th International Conference on Network and Service Management (CNSM)*, pages 125–133, November 2018.

[20] Matthew Mathis and M. Allman. A Framework for Defining Empirical Bulk Transfer Capacity Metrics. RFC 3148, July 2001. https://www.hjp.at/doc/rfc/rfc3148.html, visited on 2019-11-20.

[21] Eneko Atxutegi, Fidel Liberal, Eduardo Saiz, and Eva Ibarrola. Why we still need standardized internet speed measurement mechanisms for end users. In *2015 ITU Kaleidoscope: Trust in the Information Society (K-2015)*, pages 1–8, December 2015.

[22] Vern Paxson. Towards a Framework for Defining Internet Performance Metrics. In *Proc. INET '96*, page 13, June 1996.

[23] Rundfunk und Telekom Regulierungs-GmbH. RTR - NetTest. `https://www.netztest.at/en/Test`, visited on 2018-08-14.

[24] Shilpa Shashikant Chaudhari and Rajashekhar C. Biradar. Survey of Bandwidth Estimation Techniques in Communication Networks. *Wireless Personal Communications*, pages 1425–1476, July 2015.

[25] Hyung Joon Park and Byeong-hee Roh. Accurate Passive Bandwidth Estimation (APBE) in IEEE 802.11 Wireless LANs. In *2010 Proceedings of the 5th International Conference on Ubiquitous Information Technologies and Applications*, pages 1–4, December 2010.

[26] T. Pögel, J. Lübbe, and L. Wolf. Passive client-based bandwidth and latency measurements in cellular networks. In *2012 Proceedings IEEE INFOCOM Workshops*, pages 37–42, March 2012.

[27] Zhenhui Yuan, Hrishikesh Venkataraman, and Gabriel-Miro Muntean. iBE: A Novel Bandwidth Estimation Algorithm for Multimedia Services over IEEE 802.11 Wireless Networks. In *Wired-Wireless Multimedia Networks and Services Management*, pages 69–80, 2009.

[28] Taner Arsan. Review of bandwidth estimation tools and application to bandwidth adaptive video streaming. In *High Capacity Optical Networks and Emerging/Enabling Technologies*, pages 152–156, December 2012.

[29] Ningning Hu and P. Steenkiste. Evaluation and characterization of available bandwidth probing techniques. *IEEE Journal on Selected Areas in Communications*, pages 879–894, August 2003.

[30] Zhenhui Yuan, Hrishikesh Venkataraman, and Gabriel-Miro Muntean. MBE: Model-Based Available Bandwidth Estimation for IEEE 802.11 Data Communications. *IEEE Transactions on Vehicular Technology*, June 2012.

[31] Mingzhe Li, Mark Claypool, and Robert Kinicki. WBest: A bandwidth estimation tool for IEEE 802.11 wireless networks. In *2008 33rd IEEE Conference on Local Computer Networks (LCN)*, pages 374–381, October 2008.

[32] Andreas Johnsson, Bob Mel, and Mats Björkman. Diettopp: A first implementation and evaluation of a simplified bandwidth measurement method. In *in Proc. Swedish National Computer Networking Workshop (SNCNW 2004*, 2004.

[33] Tony Sun, Guang Yang, Ling-Jyh Chen, M. Y. Sanadidi, and Mario Gerla. A Measurement Study of Path Capacity in 802.11b Based Wireless Networks. In *Papers Presented at the 2005 Workshop on Wireless Traffic Measurements and Modeling*, pages 31–37, 2005.

[34] Karthik Lakshminarayanan, Venkata N. Padmanabhan, and Jitendra Padhye. Bandwidth Estimation in Broadband Access Networks. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, pages 314–321, 2004.

[35] Heung Ki Lee, Varrian Hall, Ki Hwan Yum, Kyoung Ill Kim, and Eun Jung Kim. Bandwidth Estimation in Wireless Lans for Multimedia Streaming Services. *Advances in Multimedia*, pages 1–7, 2007.

[36] LLC. Ookla. Speedtest by Ookla - The Global Broadband Speed Test. `http://www.speedtest.net/`, visited on 2018-08-14.

[37] Robert L. Carter and Mark E. Crovella. Measuring bottleneck link speed in packet-switched networks. *Performance Evaluation*, pages 297–318, October 1996.

[38] Vinay Ribeiro, Mark Coates, Rudolf Riedi, Shriram Sarvotham, Brent Hendricks, and Richard Baraniuk. Multifractal cross-traffic estimation. In *Proc. ITC Specialist Seminar on IP Trac Measurement, Modeling, and Management*, pages 15–1, 2000.

[39] Cesar D. Guerrero and Miguel A. Labrador. Traceband: A fast, low overhead and accurate tool for available bandwidth estimation and monitoring. *Computer Networks*, pages 977–990, April 2010.

[40] Jiri Navratil and R Les Cottrell. ABwE :A Practical Approach to Available Bandwidth Estimation. In *in Passive and Active Measurement (PAM) Workshop 2003 Proceedings, La Jolla*, page 11, 2003.

[41] Manish Jain and Constantine Dovrolis. Pathload: A Measurement Tool for End-to-End Available Bandwidth. *Proceedings of Passive and Active Measurement Workshop*, March 2002.

[42] Bob Melander, Mats Bjorkman, and Per Gunningberg. A new end-to-end probing and analysis method for estimating bandwidth bottlenecks. In *Globecom '00 - IEEE. Global Telecommunications Conference. Conference Record (Cat. No.00CH37137)*, pages 415–420 vol.1, November 2000.

[43] Mahboobeh Sedighizad, Babak Seyfe, and Keivan Navaie. MR-BART: Multi-Rate Available Bandwidth Estimation in Real-Time. *Journal of Network and Computer Applications*, pages 731–742, March 2012.

[44] Joel Sommers, Paul Barford, and Walter Willinger. Laboratory-based calibration of available bandwidth estimation tools. *Microprocessors and Microsystems*, pages 222–235, June 2007.

[45] Allen B. Downey. Clink: a Tool for Estimating Internet Link Characteristics. In *SIGCOMM '99*, 1999.

[46] Van Jacobson. Pathchar: A tool to infer characteristics of Internet paths, April 1997. ftp://ftp.kfki.hu/pub/packages/security/COAST/netutils/pathchar/msri-talk.pdf, visited on 2020-01-04.

[47] Stefan Saroiu, Krishna P. Gummadi, and Steven Gribble. Sprobe: A fast technique for measuring bottleneck bandwidth in uncooperative environments. *INFOCOM 2002*, January 2002.

[48] ESnet. iPerf3: A TCP, UDP, and SCTP network bandwidth measurement tool. https://github.com/esnet/iperf, visited on 2019-11-12.

[49] Emanuele Goldoni and Marco Schivi. End-to-End Available Bandwidth Estimation Tools, An Experimental Comparison. In *Traffic Monitoring and Analysis*, pages 171–182. Springer Berlin Heidelberg, 2010.

[50] Alessio Botta, Alan Davy, Brian Meskill, and Giuseppe Aceto. Active Techniques for Available Bandwidth Estimation: Comparison and Application. In *Data Traffic Monitoring and Analysis*, pages 28–43. 2013.

[51] Yin Xu, Zixiao Wang, Wai Kay Leong, and Ben Leong. An End-to-End Measurement Study of Modern Cellular Data Networks. In *Proceedings of the 15th International Conference on Passive and Active Measurement - Volume 8362*, pages 34–45, 2014.

[52] Guowang Miao, Jens Zander, Ki Won Sung, and Slimane Ben Slimane. *Fundamentals of Mobile Data Networks*. Cambridge University Press, March 2016.

[53] Markus Laner, Joachim Fabini, Philipp Svoboda, and Markus Rupp. End-to-end Delay in Mobile Networks: Does the Traffic Pattern Matter? In *ISWCS 2013; The Tenth International Symposium on Wireless Communication Systems*, pages 1–5, August 2013.

[54] Xin Liu, Ashwin Sridharan, Sridhar Machiraju, Mukund Seshadri, and Hui Zang. Experiences in a 3g Network: Interplay Between the Wireless Channel and Applications. In *Proceedings of the 14th ACM International Conference on Mobile Computing and Networking*, pages 211–222, 2008.

[55] Vaclav Raida, Philipp Svoboda, and Markus Rupp. Constant Rate Ultra Short Probing (CRUSP) Measurements in LTE Networks. In *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, pages 1–5, August 2018.

[56] Haiqing Jiang, Yaogong Wang, Kyunghan Lee, and Injong Rhee. Tackling Bufferbloat in 3G/4G Networks. In *Proceedings of the 2012 Internet Measurement Conference*, pages 329–342, 2012.

[57] Weichao Li, Ricky K. P. Mok, Daoyuan Wu, and Rocky K. C. Chang. On the accuracy of smartphone-based mobile network measurement. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 370–378, April 2015.

[58] Markus Laner, Philipp Svoboda, Eduard Hasenleithner, and Markus Rupp. Dissecting 3g Uplink Delay by Measuring in an Operational HSPA Network. In *Passive and Active Measurement*, pages 52–61, 2011.

[59] Michael Rindler, Philipp Svoboda, and Markus Rupp. FLARP, fast lightweight available rate probing: Benchmarking mobile broadband networks. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–7, May 2017.

[60] Anup Kumar Paul, Atsuo Tachibana, and Teruyuki Hasegawa. NEXT-FIT: Available Bandwidth Measurement over 4G/LTE Networks – A Curve-Fitting Approach. In *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, pages 25–32, March 2016.

[61] Takashi Oshiba, Kousuke Nogami, Koichi Nihei, and Kozo Satoda. Robust available bandwidth estimation against dynamic behavior of packet scheduler in operational LTE networks. In *2016 IEEE Symposium on Computers and Communication (ISCC)*, pages 1276–1283, June 2016.

[62] Takashi Oshiba and Kazuaki Nakajima. Quick end-to-end available bandwidth estimation for QoS of real-time multimedia communication. In *The IEEE symposium on Computers and Communications*, pages 162–167, June 2010.

[63] Inc. IID. RBB SPEED TEST. `https://play.google.com/store/apps/details?id=com.rbbtoday.speedtest&hl=de`, visited on 2019-12-11.

[64] Shota Shiobara and Takao Okamawari. A Novel Available Bandwidth Estimation Method for Mobile Networks Using a Train of Packet Groups. In *Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication*, IMCOM '17, pages 59:1–59:7, New York, NY, USA, 2017. ACM. event-place: Beppu, Japan.

[65] CRUSP - Internal Report. Technical report, TU Wien.

[66] Vaclav Raida, Philipp Svoboda, Martin Kruschke, and Markus Rupp. Constant Rate Ultra Short Probing (CRUSP): Measurements in Live LTE Networks. In *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, May 2019.

[67] Hyacinth S. Nwana. Software agents: an overview. *The Knowledge Engineering Review*, pages 205–244, September 1996.

[68] MobiPerf. Mobiperf/MobiPerf. `https://github.com/Mobiperf/MobiPerf`, visited on 2019-12-16.

[69] Alexander Schatten, Stefan Biffl, Markus Demolsky, Erik Gostischa-Franta, Thomas Östreicher, and Dietmar Winkler. *Best Practice Software-Engineering: Eine praxiserprobte Zusammenstellung von komponentenorientierten Konzepten, Methoden und Werkzeugen*. Springer Spektrum, 2010.

[70] Martin Fowler. UserStory, April 2013. https://martinfowler.com/bliki/UserStory.html, visited on 2019-10-01.

[71] Andrew Stellman and Jennifer Greene. Software Requirements Specification. In *Applied Software Project Management*. November 2005.

[72] André B. Bondi. Characteristics of Scalability and Their Impact on Performance. In *Proceedings of the 2Nd International Workshop on Software and Performance*, pages 195–203, 2000.

[73] Roger S. Pressman. *Software Engineering: A Practitioner's Approach*. Palgrave Macmillan, 2005.

[74] Martin Fowler. StoryPoint, July 2013. https://martinfowler.com/bliki/StoryPoint.html, visited on 2019-10-23.

[75] Grady Booch, James Rumbaugh, and Ivar Jacobson. *Unified Modeling Language User Guide, The, 2nd Edition*. Addison-Wesley Professional. Part of the Addison-Wesley Object Technology Series series., May 2005.

[76] International Organization for Standardization, Geneva. ISO/IEC 19501:2005 Information technology — Open Distributed Processing — Unified Modeling Language (UML) Version 1.4.2, April 2005. http://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/03/26/32620.html, visited on 2019-10-23.

[77] Martin Fowler. Difference Between Use Cases and User Stories, August 2003. https://martinfowler.com/bliki/UseCasesAndStories.html, visited on 2019-10-23.

[78] James Lewis and Martin Fowler. Microservices, April 2014. https://martinfowler.com/articles/microservices.html, visited on 2019-09-16.

[79] David Messina. 5 years later, Docker has come a long way, March 2018. https://www.docker.com/blog/5-years-later-docker-come-long-way/, visited on 2019-10-24.

[80] Docker Inc. Docker. https://github.com/docker, visited on 2019-10-24.

[81] kubernetes.io. kubernetes/kubernetes. https://github.com/kubernetes/kubernetes, visited on 2019-10-24.

[82] Docker Inc. docker/swarm. https://github.com/docker/swarm, visited on 2019-10-24.

[83] Docker Inc. docker/compose. https://github.com/docker/compose, visited on 2019-10-24.

[84] Platform9. Compare Kubernetes vs Docker Swarm, July 2017. https://platform9.com/blog/kubernetes-docker-swarm-compared/, visited on 2019-10-24.

[85] James Gosling, Bill Joy, Guy Steele, Gilad Bracha, and Alex Buckley. The Java® Language Specification, February 2015.

[86] Stackoverflow. Stack Overflow Developer Survey 2019, 2019. https://insights.stackoverflow.com/survey/2019/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2019, visited on 2019-10-02.

[87] spring.io. spring-projects/spring-boot. https://github.com/spring-projects/spring-boot, visited on 2019-10-24.

[88] spring.io. Spring Cloud. https://github.com/spring-cloud, visited on 2019-10-24.

[89] Netflix Inc. Netflix/Hystrix. https://github.com/Netflix/Hystrix, visited on 2019-10-24.

[90] Netflix Inc. Netflix/ribbon. https://github.com/Netflix/ribbon, visited on 2019-10-24.

[91] Netflix Inc. Netflix/eureka. https://github.com/Netflix/eureka, visited on 2019-10-24.

[92] ISO/IEC JTC1 (Joint Technical Committee 1) / SC22. C++. https://isocpp.org/, visited on 2019-10-24.

[93] The Rust Project. rust-lang/rust. https://github.com/rust-lang/rust, visited on 2019-10-24.

[94] The Go Authors. golang/go. https://github.com/golang/go, visited on 2019-10-24.

[95] Mohit Agrawal. Why we chose Rust as our programming language, June 2019. https://bitbucket.org/blog/why-rust, visited on 2019-10-24.

[96] The Rust Core Team. Announcing Rust 1.0 | Rust Blog, May 2015. https://blog.rust-lang.org/2015/05/15/Rust-1.0.html, visited on 2019-09-19.

[97] The Rust Project. Production users. https://www.rust-lang.org/production/users, visited on 2019-10-24.

[98] The Computer Language Benchmarks Game. Rust vs Go - Which programs are fastest? https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/rust-go.html, visited on 2019-09-19.

[99] Sergio Benitez. Rocket Web Framework. https://github.com/SergioBenitez/Rocket, visited on 2019-10-24.

[100] Sean McArthur. reqwest. https://github.com/seanmonstar/reqwest, visited on 2019-10-24.

[101] Netflix Inc. Netflix/zuul. https://github.com/Netflix/zuul, visited on 2019-10-24.

[102] OpenFeign. OpenFeign/feign. https://github.com/OpenFeign/feign, visited on 2019-10-24.

[103] JUnit Team. junit-team/junit4. https://github.com/junit-team/junit4, visited on 2019-10-24.

[104] Facebook. facebook/react. https://github.com/facebook/react, visited on 2019-10-24.

[105] angular.io. angular/angular.js. https://github.com/angular/angular.js, visited on 2019-10-24.

[106] vuejs.org. vuejs/vue. https://github.com/vuejs/vue, visited on 2019-10-24.

[107] Sophia Martin. Angular vs React vs Vue: Which is the Best Choice for 2019?, May 2019. https://hackernoon.com/angular-vs-react-vs-vue-which-is-the-best-choice-for-2019-16ce0deb3847, visited on 2019-10-24.

[108] axios Team. axios/axios. https://github.com/axios/axios, visited on 2019-10-24.

[109] recharts Team. recharts/recharts. https://github.com/recharts/recharts, visited on 2019-10-24.

[110] postgresql.org. postgres/postgres. https://github.com/postgres/postgres, visited on 2019-10-24.

[111] Timescale Inc. timescale/timescaledb. https://github.com/timescale/timescaledb, visited on 2019-10-24.

[112] InfluxData Inc. influxdata/influxdb. https://github.com/influxdata/influxdb, visited on 2019-10-24.

[113] The Apache Software Foundation. apache/couchdb. https://github.com/apache/couchdb, visited on 2019-10-24.

[114] H2 Database. h2database/h2database. https://github.com/h2database/h2database, visited on 2019-10-24.

[115] Boxfuse GmbH. flyway/flyway. https://github.com/flyway/flyway, visited on 2019-10-24.

[116] StatCounter. Mobile Operating System Market Share Worldwide, August 2019. https://gs.statcounter.com/os-market-share/mobile/worldwide, visited on 2019-09-26.

[117] Distribution dashboard, May 2019. https://developer.android.com/about/dashboards/, visited on 2020-01-24.

[118] 3GPP. Carrier Aggregation explained, June 2013. https://www.3gpp.org/technologies/keywords-acronyms/101-carrier-aggregation-explained, visited on 2020-01-22.

[119] Mansoor Shafi, Andreas F. Molisch, Peter J. Smith, Thomas Haustein, Peiying Zhu, Prasan De Silva, Fredrik Tufvesson, Anass Benjebbour, and Gerhard Wunder. 5G: A Tutorial Overview of Standards, Trials, Challenges, Deployment, and Practice. *IEEE Journal on Selected Areas in Communications*, pages 1201–1221, June 2017.

[120] Liyang Sun, Fanyi Duanmu, Yong Liu, Yao Wang, Yinghua Ye, Hang Shi, and David Dai. Multi-path Multi-tier 360-degree Video Streaming in 5g Networks. In *Proceedings of the 9th ACM Multimedia Systems Conference*, pages 162–173, 2018.

[121] RTR - NetTest. https://www.netztest.at/en/Karte, visited on 2019-10-30, author = Rundfunk und Telekom Regulierungs-GmbH.

[122] Christian Heipke. Crowdsourcing geospatial data. *ISPRS Journal of Photogrammetry and Remote Sensing*, pages 550–557, November 2010.

[123] Pedro Gomes, Cristina Olaverri-Monreal, and Michel Ferreira. Making Vehicles Transparent Through V2v Video Streaming. *IEEE Transactions on Intelligent Transportation Systems*, pages 930–938, June 2012.

[124] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog Computing and Its Role in the Internet of Things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, pages 13–16, 2012.

[125] Tarik Taleb, Sunny Dutta, Adlen Ksentini, Muddesar Iqbal, and Hannu Flinck. Mobile Edge Computing Potential in Making Cities Smarter. *IEEE Communications Magazine*, March 2017.

[126] Peng Yang, Ning Zhang, Yuanguo Bi, Li Yu, and Xuemin Sherman Shen. Catalyzing Cloud-Fog Interoperation in 5g Wireless Networks: An SDN Approach. *IEEE Network*, pages 14–20, 2017.

[127] Karthik Kumar and Yung-Hsiang Lu. Cloud Computing for Mobile Users: Can Offloading Computation Save Energy? *Computer*, pages 51–56, April 2010.

[128] István Ketykó, László Kecskés, Csaba Nemes, and Lóránt Farkas. Multi-user computation offloading as Multiple Knapsack Problem for 5g Mobile Edge Computing. In *2016 European Conference on Networks and Communications (EuCNC)*, pages 225–229, June 2016.

[129] Andrew Machen, Shiqiang Wang, Kin K. Leung, Bong Jun Ko, and Theodoros Salonidis. Live Service Migration in Mobile Edge Clouds. *IEEE Wireless Communications*, pages 140–147, February 2018.

[130] Antonio Manzalini, Roberto Minerva, Franco Callegati, Walter Cerroni, and Aldo Campi. Clouds of virtual machines in edge networks. *IEEE Communications Magazine*, pages 63–70, July 2013.

[131] Aleksey Shipilev. Nanotrusting the Nanotime, September 2014. `https://shipilev.net/blog/2014/nanotrusting-nanotime/`, visited on 2020-01-20.

[132] Adnan Aijaz, Hamid Aghvami, and Mojdeh Amani. A survey on mobile data offloading: technical and business perspectives. *IEEE Wireless Communications*, pages 104–112, April 2013.

[133] Gagangeet Singh Aujla, Rajat Chaudhary, Neeraj Kumar, Joel J. P. C. Rodrigues, and Alexey Vinel. Data Offloading in 5g-Enabled Software-Defined Vehicular Networks: A Stackelberg-Game-Based Approach. *IEEE Communications Magazine*, pages 100–108, August 2017.

[134] Roberto Bonafiglia, Gabriele Castellano, Ivano Cerrato, and Fulvio Risso. End-to-end service orchestration across SDN and cloud computing domains. In *2017 IEEE Conference on Network Softwarization (NetSoft)*, pages 1–6, July 2017.

[135] Zhaoguang Wang, Zhiyun Qian, Qiang Xu, Zhuoqing Mao, and Ming Zhang. An Untold Story of Middleboxes in Cellular Networks. In *Proceedings of the ACM SIGCOMM 2011 Conference*, pages 374–385, 2011.

[136] Gertjan Halkes and Johan Pouwelse. UDP NAT and Firewall Puncturing in the Wild. In *NETWORKING 2011*, pages 1–12. Springer, 2011.