University of Zurich^UZH

MASTER THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Quality-of-Experience (QoE) Estimation for Mobile Data Networks through Mobile Measurements

Tomas Ludrovan
Zurich, Switzerland
Student ID: 13-746-813

Supervisors:
Prof. Dr. Burkhard Stiller, Prof. Dr. Thomas Grechenig,
Christos Tsiaras, Thomas Bocek, Anuj Sehgal, Sebastian Seeber,
Andreas Ehringfeld
Date of Submission: August 29, 2014

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

Mobile networks became popular over last years and technology enabled data delivery to mobile devices. Mobile data is nowadays used for different applications where each of them has different requirements for a good quality. Bandwidth can be measured by available tools, however protocol-specific measurements for testing connection quality under quality of service regulations of network providers are missing. In the scope of this master thesis a new software system called BonaFide+ Provider was developed for performing protocol-specific measurements and detailed visualization of quality on map. In addition, completely new functionality was implemented, which can estimate the experienced quality perceived by users based on type of activity being performed using DQX mathematical model.

Acknowledgments

This master thesis was written in cooperation between the University of Zurich in the scope of Erasmus programme and the Technical University of Vienna. I would like to thank everyone who supported me and the project, especially my coordinators Prof. Dr. Burkhard Stiller at the University of Zurich and Prof. Dr. Thomas Grechenig at the Technical University of Vienna and my supervisors Christos Tsiaras, Andreas Ehringfeld, Anuj Sehgal and Sebastian Seeber. I would also like to thank for support to our project partners and to people who performed measurements, which were evaluated in this master thesis.

This work also contributed to the research of the FLAMINGO[1] project, on which the University of Zurich, Jacobs University of Bremen and the Universität der Bundeswehr München participate.

[1]Flamingo is a European (ICT-FP7 Network of Excellence) project focusing on network and service management. More informations are available on: http://www.fp7-flamingo.eu/

Contents

Chapter 1

Introduction

1.1 Motivation

"The future of internet is mobile" [29]. Mobility is the requirement of modern people [34] and is defined as "Change of property[1] between defined units of a system" [34]. Franz Lehner defines mobility as location-independence of communication partners [29]. Because of such requirements, wireless technology for delivering mobile data was developed [24]. Mobile devices not only support GSM calls and messaging, but got also connected to the internet [24]. By this fact new requirements and demands arised, because using internet completely differs from the previous offline (without internet connectivity) cell phone usage. Using such technology, people and services became location-independent [34] [29] together with the technology (Figure 1.1). Consumers of mobile technologies are mostly confronted by marketing advertisements, which advertise physical layer peak data rate, which define the technology data throughput in lab conditions without considering any data correction techniques, signal quality, interference, scheduling [33], but how the service is offered in real environment is hidden to them, because the technology can't be easily measured and evaluated with respect to different environmental conditions with available tools. Because the *what do I buy* and *what do I get* deviation, consumers of mobile technologies should be aware of possible reasons for it and should have access to measured data.

With internet connectivity on the phone (smartphone) new possibilities were offered to the end user, such as web browsing, e-mail, internet telephony and video streaming. Even if it sounds easy because people are used to consume that services via computers, mobile technologies make this more interesting. In contrast to cable internet, mobile internet has a lower bandwidth and this is in addition shared by the number of simultaneously connected users via the same frequency (overview of common frequencies is listed in Table 1.1). As written in [33] moving from physical layer peak data rate (lab envirnomment) to application level peak data rate (single user located directly under the base station) the available throughput is still not reflecting real world performance [33]. This can be compared to WiFi for better understanding. If only one user is associated with the access point, the

[1]Meaning: persons, informations, material object etc.

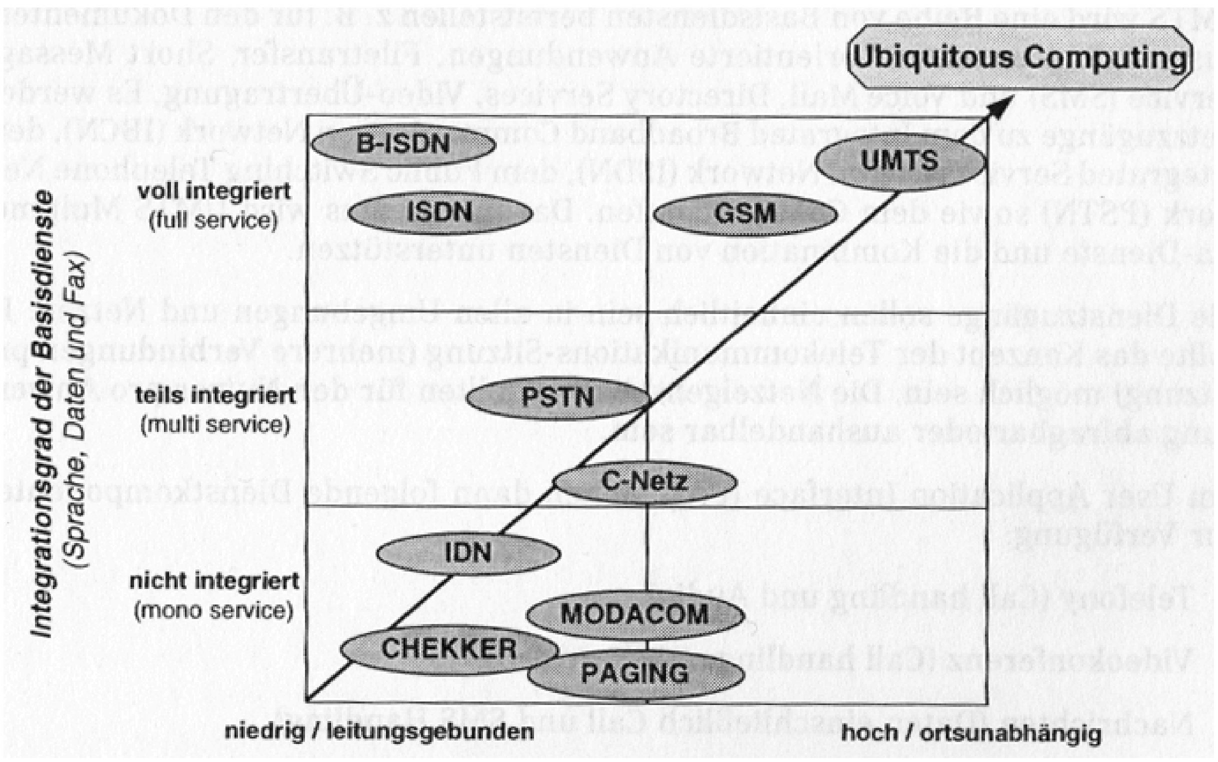


Figure 1.1: Communication technologies and their classification [24]

whole technically available bandwidth belongs to him. If more users are connected, WiFi technology has to coordinate parallel communication and thus schedule the bandwidth to all users. As a result, every user gets lower bandwith as the maximum available. This is also necessary in mobile networks, which also operate on a specific frequency and have a technology- and coverage-specific properties. However, mobile networks are developed with much more effort than WiFi networks because of higher number of consumers and because services provided by mobile network operators (MNOs)[2] are subject of fees and thus should promise a good quality. Specification of the cell system, which is the base of mobile networks, explicitly requires economic utilization of the limited mobile radio frequency [24].

Network performance is also not constant. There are many factors, which give variations to the end-users experience, e.g. network type, number of concurrent users in the same location and at the same time etc. Especially In this master thesis each factor will be described and evaluated. I will call such factors **hard factors**, because their capacity is given by available technology and can't be easily changed. Beside such factors there are also another ones which play an important role in regulating available resources. They can be conFigured and dynamically adapted and I will call them **soft factors**. They are not given by the technology, but by fair sharing of available resources with the goal that every user becomes the feeling that he is not affected by others. I am talking about Quality of Service (QoS), which is used for such regulations by almost every network provider in order to improve overall quality of the users experience when consuming different services in a multi-user network environment.

[2]Franz Lehner define MNOs as companies which deploy and maintain mobile networks [29]

Usage	**Frequency in MHz**
TV	47-68
FM-Radio	87,5-108
Trunked radio system	418,8-430
C-450	451-465
GSM 900	890-960
GSM 1800/ DCS 1800	1710-1880
Radar	960-1215
Satellit	1631,5-1643,5
DCS 1800	1710-1880
DECT	1880-1900, 2110-2120

Table 1.1: Usage of common frequencies [29]

There are many disjoint and correlated factors, which affect the network performance. The expression ”good quality” was mentioned, which is quite vague without proper argumentation. A traditional approach to measure network performance is to perform a bandwidth test. This test provides current network throughput as a number, which can be used to compare performance of different networks and/or network providers. Many such tests (e.g. http://www.speedtest.net/, http://hsi.bluewin.ch/speedtest/, www.dsl.sk) use time-measured data exchange between 2 parties - client and a server - for computing the bandwidth. This can be useful when one would like to compare the network performance in absolute values, however it says nothing about the quality goodness. With result e.g. 1348 kbit/s one can say it is more than 1 Mbit/s and less than 2 Mbit/s, but one can’t say that this is good or bad. This statement is dependent from many conditions. While 1348 kbit/s could be too slow for streaming Full HD videos, it is more than required for Voice over IP (VoIP) [26]. Moving into this direction we will be able to interpret results in a different way - not what is more, what is a big number etc., but what is good for a specific use-case. This interpretation will, in contrast to simple speedtest tools, estimate users experience with the consumed service over the network. This interpretation, called Quality of Experience (QoE), is be able to compute such statements based on measured QoS and parameters, which are developed in lab environments in order to estimate the users experience [37]. Under such interpretation one will be able to measure if the network is good or bad for a particular service and will not confuse by absolute values, which are sometimes much bigger than required. As an example 1 Gbit/s network bandwidth doesn’t make VoIP call better than network with bandwidth of 1 Mbit/s. In both cases the QoE would be excellent (assumed only the bandwidth differs and other parameters such as latency are the same).

There are some tools, which are targeting network quality, however they are very limited and are focusing only bandwidth and not QoE, so they are providing absolute values without their goodness, which is from my point of view more important for end-users than the maximum available bandwidth. For somebody using internet for reading e-mails or VoIP telephony, the bandwidth in ranges greater than e.g. 1 Mbit/s is not required, however the classical end-user understanding says *more is better.* Network providers often try to offer more bandwidth in order to better sell the product, however this is mostly a theoretical value without taking what is needed by the user in real and also provider-specific QoS policies into charge. Hidden QoS regulations can result in different bandwidth for different protocols, such as torrents and video streaming. This gap between offered and provided performance is only partially provided by available tools to consumers. Proper testing of networks targeting specific use-cases (QoE) and for protocol performance - not measured by random data exchange, but by protocol-simulating measurements, which are testing performance under network providers QoS regulations - is missing.

QoE is closely associated with QoS, because experience is connected to a network-specific intent, e.g. making a VoIP call, where QoS has influence on how the experience will be. However pure QoS data are not talking about the experience itself, but about available ressources. While QoS measurement can tell one that there is available bandwidth of 1 Mbit/s for VoIP calls with latency 21ms, QoE will decide if the user will perceive good or bad quality. To make it more clear, one can explain it in a different way. QoS measurement will ask: ”Which bandwidth and latency is currently observed for VoIP protocol?”, but QoE will go beyond it and will ask: ”How good is the quality of VoIP call at the moment?”. In this master thesis the Deterministic Quality-of-Experience model (DQX) [39] is used for the calculation of QoE.

QoS is not constant, but dependent from the underlying network technology and from numerous influence factors, which will be discussed in this master thesis. Because QoE is related to QoS, the QoS fluctuation can result in changes in the QoE, however the QoE will change only if the QoS changes to a degree sensitive to the quality change of the targeted QoE use-case. Every QoE estimation has a set of pre-conFigured values, which decide about the experience for a given service. For example QoE of VoIP will take bi-directional bandwidth into account and QoE of video streaming only bandwidth of one direction. Every use-case is sensitive to different QoS observations and thus fluctuation in the network can have no, slightly or strong influence on the QoE. When a VoIP call requires bandwidth of 8 kbit/s to be considered as good, the network drop-down from 10 Mbit/s to 1 Mbit/s will have no effect in the quality of the call. On the other side, it can result in bad quality while streaming video. All such investigations and decisions will be covered in this master thesis.

There are no tools available, which can tell the user if the network performance results in a good or bad experience with e.g. VoIP call. Because of importance of such informations, this master thesis reflects QoE state-of-art and documents implementation of a distributed system, which is able to measure networks and to provide QoE interpretations. Location, daytime, network technology and other additional factors were considered in order to cover QoE based on different conditions in real world envirnoment as defined by subscriber data rate [33][3]. Measurement clients can be installed on Android mobile devices and are optimized for measurements in mobile networks. WiFi networks are supported as well, but the result is not only dependent from the network quality, but also from the WiFi throughput between the mobile device and the wireless access point. When measuring mobile networks, the maximal throughput can be achieved by the mobile device once it

[3]Subscriber data rate is often expressed as ”up to” a peak, a range of min-max values, and average measurements. This is the view of a single subscriber’s data rate and can vary greatly depending on the conditions and the number of subscribers on a cell using the services, but it gives a realistic expectation of what subscribers are likely to experience on a real life network. [33]

can connect to the given network type in contrast to measuring e.g. fiber internet over WiFi-G access point.

1.1.1 Target group

This work is dedicated for individuals, network providers and research groups in the QoE and QoS field.

Individuals can use the system for testing their internet performance and for viewing networks by measured results of all users, who performed tests before. Users can see real network performance and in addition the estimated experience for different services in particular locations.

Network providers can use this system to map performance of their infrastructure over time and at different locations. Because this system is optimized especially for mobile network operators (MNOs), using system developed in the scope of this master thesis can support them to evaluate their wireless setup and implied signal interference, disturbance and mirroring.

Research groups can use the system for experiments. The system is easily configurable and open-source, so everybody can use it and contribute to its future direction. This thesis can be used as a reference for understanding the system.

1.2 Description of Work

This master thesis consist of a theoretical research and of the system design and implementation.

1.2.1 Theoretical part

Prior to the system design, QoE field was researched in detail to gain in-depth insight into the topic. This was necessary for collecting requirements and preconditions for the system design. In this master thesis related theory will be explained and brought to the future direction. While QoE is still being researched and evaluated, this work will be also contribute to it and will implement current research progress into an open-source software system, which can be used directly by network users and by research groups for experiments and modifications. This master thesis provides documentation and explanation of my decisions for better understanding of the topic and of the system, which can automatically determine users experience.

1.2.2 System design and implementation

As already mentioned, beside the theory research new system was designed and implemented, which is able to perform protocol-specific QoS measurements needed for QoE estimation based on the DQX model and which can provide QoS and also QoE visualizations to end-users. In contrast to existing tools, this system is able to perform protocol-specific measurements, which take QoS regulations such as Application Level Filtering into account. This system not only show detailed measurement results, but also estimate users experience.

The new system extends measurements to protocol specific QoS measurements (targeting protocol performance instead of random data exchange) and adds the ability to estimate by measured data the users experience at specific place, at specific time and for specific use-case.

Chapter 2

Related Work

Currently there are no QoE estimation tools available. This fact initiated the development of a completely new approach for measuring not only QoS, but also for estimation of user experience at particular locations.

Measrdroid (`https://play.google.com/store/apps/details?id=de.tum.in.net.measrdroid.gui.stats`) is an Android application for collecting phone usage statistics, which also include signal quality, traffic usage, WiFi coverage etc. Measurement are passive - only available data is collected and evaluated. No network performance tests are done nor available.

Netradar available at `https://www.netradar.org/` is a software system for mobile internet quality measurements supporting both mobile data and WiFi networks. Results are displayed on a map. This project provides measurement clients for multiple operating systems and measurement results are stored centrally. Measured data are:

- Downstream,
- latency,
- upstream,
- signal strength.

This tool goes into QoS direction, however upstream and downstream measurements are not differentiated for particular protocols and thus they don't reflect application-specific QoS regulations of network providers. QoE estimation is not available and in current implementation also not possible, because QoE is specific for a particular application and without having the specific measurement data (including QoS regulations) the statement about the user experience of the particular application would be not appropriate.

Another Android application for measuring QoS and for finding the connection between QoS and QoE developed by TelecomItalia Laboratories is **TeleAbarth**. This application collect network measurements and end-user quality feedback regarding the use of smartphone applications [40]. However the tool is not able to estimate the user experience based on measured QoS data.

There are many **speedtest tools** such as `http://www.speedtest.net/` or `http://hsi.bluewin.ch/speedtest/`, which are performing download or upload throughput tests and some of them also latency measurements. However no speedtest tool was found which is able to perform protocol-specific measurement, which would incorporate QoS regulations of network providers. QoE aspects are also missing.

Chapter 3

Quality of Service

Internet communication bases on data exchange between peers in the network. Data communication without any regulation can cause bottleneck effect of the whole data flow. In this case, each packet is considered the same, i.e. there is no difference in the network scheduling for any type of packet. This can however lead to problems, because protocols with high-bandwidth usage and with low-priority (e.g. mass downloads, torrents) can consume so much bandwidth that other high-priority services such as VoIP or video streaming become unavailable or limited.

"The notion of quality of service (QoS) has been proposed to capture quantitatively or quantitatively defined performance contract between the service provider and the user applications." [22]. "QoS covers the concepts, parameters and methods needed to manage the interactions between applications, typically running in end-user terminals and in network nodes managed by network operators. QoS parameters include bit rates, delay properties, and packet loss rates." [28]. QoS defines constraints for a specific data communication, e.g. the bandwidth of a HTTP download or latency for an online game. In contrast to raw (unregulated) data flow QoS takes control over the routing scheduling in order to make communication better distributed for different application, each with specific requirement (e.g. low latency, minimum or maximum bandwidth, routing priority). QoS regulations split the available data link in order to manage resources available by the communication channel for different applications.

Figure 3.1 shows an example for a hierarchical QoS. In this scenario QoS regulation divides the *green* communication channel for different customers, where each customer has a guaranteed bandwidth. QoS regulation guarantees in this case that *Customer-1* can't extend the bandwidth of 400 Mbps despite the fact that the physical link can offer up-to 1 Gbps. Given that each customer has a guaranteed maximum speed and each of them can't be affected by the bandwidth usage other customer. QoS is defined in Service Level Agreements of network providers and it guarantees each customer the agreed bandwidth, which is managed by appropriate QoS regulations in the network.

Another and for purpose of this master thesis more important QoS regulation is shown in Figure 3.2. In this scenario the traffic classification is specific to different protocols and applications. Thanks to Deep Packet Inspection the QoS regulation is able to classify

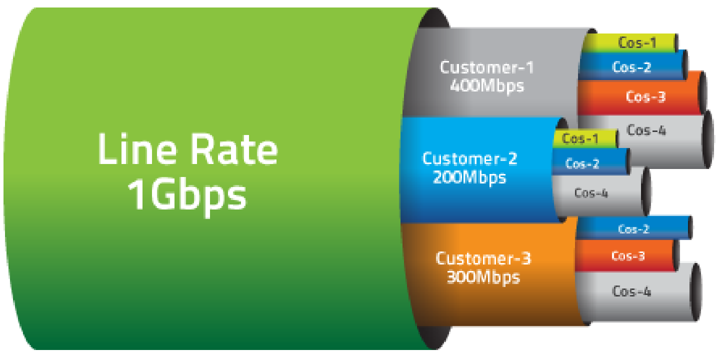


Figure 3.1: Hierarchical QoS [16]

each packet based on its application. This can be e.g. web, video, instant messaging, VoIP. According to this classification the data flow can be managed and prioritized based on the packet type. Classification can be done by the port number or by the deep packet inspection, where packet type is determined by the packet analysis instead of the target or source port, which can vary for each protocol. As a result, the channel is regulated by giving priority to certain applications or by adjusting the minimal and maximal bandwidth dedicated for a concrete application. Similar to the hierarchical QoS shown in Figure 3.1 the available channel bandwidth is divided into an application-specific handling. Given that the data flow can be adjusted in order to give a dedicated bandwidth or latency to specific applications and thus e.g. VoIP can perform well even when the P2P data communication utilizes the whole bandwidth. Such regulations are required in order to make the limited channel bandwidth available for many users and for different applications without harming the required QoS properties of each application.

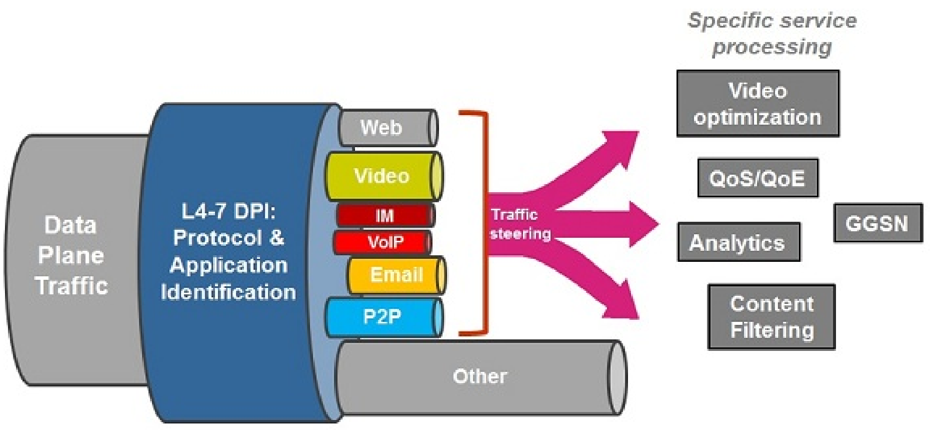


Figure 3.2: Application-specific QoS [18]

QoS can be measured, however to make the measurement application-specific, a desired protocol should be used when performing the test. This can't be reached with random data exchange between two peers in the network, but packets used in the data exchange should carry protocol-specific markers, which can be classified by network providers and handled as a real application-specific communication in the network. Measuring that way the QoS of a particular protocol or application can be measured and QoS specific parameters such as **current bandwidth** and **latency** can be collected. Results can then describe the connection quality for specific applications in numeric (quantitative) values. The quality can be then interpreted based on such values. However the interpretation that bigger bandwidth is better than the slower one is not always correct. More application-specific understanding of this statement will be described in the following chapter.

Chapter 4

Quality of Experience

"Engineers talk about network performance and quality of service, business people talk about average revenue per user and customer churn while behavioral scientists talk about happiness and experiences." [28]. QoE extends the QoS by making statements about the user experience when having a specific bandwidth and latency available on the device. This differs from the numerical model (quantity) and transforms it to qualitative statements such as *application VoIP performs excellent in current network conditions*. Experience is the users opinion and thus such statements rely on the users feedback. [31] describes QoE as "the processes of human perception and experiencing, and of quality formation". In contrast to QoS, QoE doesn't provide numeric values about the bandwidth, but *user experience* statements in given conditions for a particular application considering what the application requires and if the network satisfies its needs [28]. Experience is defined by [28] as direct personal participation or observation. [39] describes QoE as a user-centric concept reflecting the end-user satisfaction of a service while considering various technical variables, such as latency, bandwidth, or jitter.

"The business of network operator is highly dependent on customer satisfaction." [28]. Users are confronted with marketing informations about the offered network performance, QoS can measure if they are satisfied and QoE can say if the user will expect a good or bad quality when using a particular application. Perceived quality is a key criterion for evaluating systems [31]. Figure 4.1 shows the relation between customers, applications, SLA and the network QoS. [31] describes QoE as the degree of delight or annoyance of a person whose experiencing involves an application, service, or system. It results from the person's evaluation of the fulfillment of his or her expectations and needs with respect to the utility and/or enjoyment in the light of the person's context, personality and current state. When talking about wireless communication, QoE is also influenced by radio conditions in the radio access network [23]. An in-detail explanation of the influence is covered in covered in Chapter 5.

"QoE can only be estimated through mathematical models, or it can be measured indirectly through an experimental setup." [37] Because the goal of this master thesis is to provide QoE statements based on QoS measurements, mathematical model had to be used for the QoE estimation. Mean Opinion Score (MOS) reflects the end-user satisfaction at a numerical scale where the higher the score is the higher the end-user's satisfaction is and

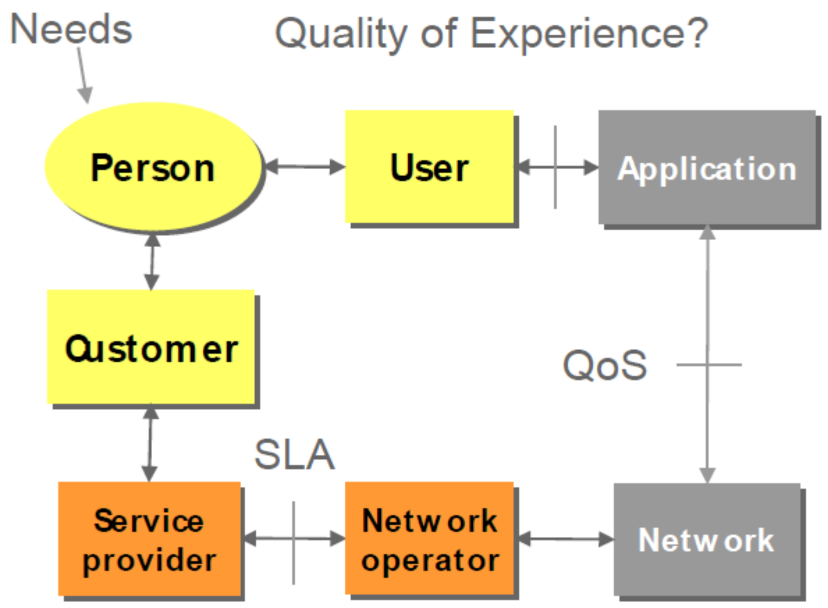


Figure 4.1: Framework for analyzing communications ecosystem [28]

vice versa [39]. The scale is shown in Table 4.1. Because MOS defines subjective opinions, the Deterministic Mathematical Model (DQX) can be used for QoE estimation [39]. This model combines measured QoS results with predefined configuration parameters and computes the resulting estimated QoE. Configuration parameters can be adjusted in order to converge the resulting computed statement to the real experience perceived by users. [37] shows how lab experiments can contribute to parameter adjustments for the DQX model. The mathematical model behind the computation is the same for all applications, however several parameters within the model approximate the perception of quality to the specific application. Because each service has different network requirements, the QoE differs in different conditions for different applications. [38] shows how parameters can be chosen for a particular use-case.

MOS value	**Quality**
5	Excellent
4	Good
3	Fair
2	Poor
1	Bad

Table 4.1: The MOS scheme according to [27]

4.1 Deterministic Mathematical Model (DQX)

DQX is a mathematical approach for QoE estimation. It has a set of predefined parameters listed in Table 6.9, which converge the computation towards the real users experience based on an underlying mathematical formula. The formula can compute values between 1 and 5 as listed in Table 4.1, which are interpreting the measured QoS input for a particular application. The representation in the client application is done by five discrete colors between green (excellent) and red (bad).

MOS can be affected in two way - in a positive and in a negative. Accordingly there are increasing (e_i) and decreasing (e_d) parameters used in the mathematical model. Which parameter is increasing and which is decreasing is defined for each particular application depending on the application requirement. Concrete configuration of the implemented QoE computation in the BonaFide+ provider system is shown in Table 6.9. Equation 4.1 shows the formula for computing.

$$E(X) = 1 + 4 \cdot \prod_{k=1}^{N} \left[\frac{e_{(i \vee d)}(x_k) - 1}{4} \right]^{w_k} \tag{4.1}$$

The formula shown in Equation 4.1 contains $e_{(i \vee d)}$, which means both increasing and decreasing parameters are included in the result, where i stands for increasing and d for decreasing. How the value of e for increasing parameters is computed is shown in Equation 4.2 and how for decreasing parameters in Equation 4.3. The value for m is chosen as shown in Equation 4.4.

$$\begin{gathered} e_i(x) = 4 \cdot \left(1 - e^{-\left(\frac{x}{x_0}\right)^m \cdot \ln 4} \right) + 1 \\ m^- = \frac{\ln\left(\frac{\ln \frac{1+\epsilon}{4}}{-\ln 4}\right)}{\ln \frac{x_0 - \delta}{x_0}} \quad m^+ = \frac{\ln\left(\frac{\ln \frac{1-\epsilon}{4}}{-\ln 4}\right)}{\ln \frac{x_0 + \delta}{x_0}} \end{gathered} \tag{4.2}$$

$$\begin{gathered} e_d(x) = 4 \cdot e^{-\left(\frac{x}{x_0}\right)^m \cdot \ln 4/3} + 1 \\ m^- = \frac{\ln\left(\frac{\ln \frac{3+\epsilon}{4}}{\ln 3/4}\right)}{\ln \frac{x_0 - \delta}{x_0}} \quad m^+ = \frac{\ln\left(\frac{\ln \frac{3-\epsilon}{4}}{\ln 3/4}\right)}{\ln \frac{x_0 + \delta}{x_0}} \end{gathered} \tag{4.3}$$

$$m \coloneqq f(x) = \begin{cases} m^- > 0 & \text{for } x < x_0 \\ 0 & \text{for } x = x_0 \\ m^+ > 0 & \text{for } x > x_0 \end{cases} \tag{4.4}$$

Equation 4.5 shows how parameters are normalized according to the DQX model specification.

$$x \coloneqq \frac{x - x_{min}}{x_{max} - x_{min}} \; \forall \, x \in \mathbb{R} \tag{4.5}$$

How MOS estimation based on DQX model was implemented in the BonaFide+ Provider is described in Subsection 6.1.1.3.5.

Chapter 5

Network Influence Factors

There are factors which have influence on the network quality. This results in QoS variations and in the end also can (but not necessary) affect QoE. In following sections different factors and their drawback will be described.

The following will be also evaluated by the BonaFide+ Provider system for measuring quality of data communication in mobile networks, which is part of this master thesis and which will be described in chapter 6.

5.1 Mobile Network Type

Currently there is not a single network type spread in mobile networks. Because of compatibility between base stations and older mobile devices and because of the reverse compatibility between mobile devices and older base stations, multiple network technologies are used in parallel. Each network type has its characteristics. We will focus on mobile data only. Table 5.1 lists some of them with their maximum available bandwidth. As we can see in the Table, the bandwidth is different for each of them and thus has an influence on the resulting QoS delivered to the client. Figure 5.1

Technology	**Data rate**
GSM	9,6 - 14,4 kbit/s
GPRS	9,05 - 171,24 kbit/s
HSCSD	9,6 - 57,6 kbit/s
EDGE	11,2 - 400 kbit/s
UMTS	64 kbit/s - 2 Mbit/s
LTE MIMO 4x4 20 MHz	50.1 Mbps downlink, 12.7 Mbps uplink

Table 5.1: Mobile network technologies and their data rate [34] [33]

Beside the bandwidth, network technology has also another influence factor on the resulting QoE - the latency. ”With HSDPA networks, a subscriber can expect a two-second or longer delay to set up the first connection, and then between 75 and 150 ms roundtrip

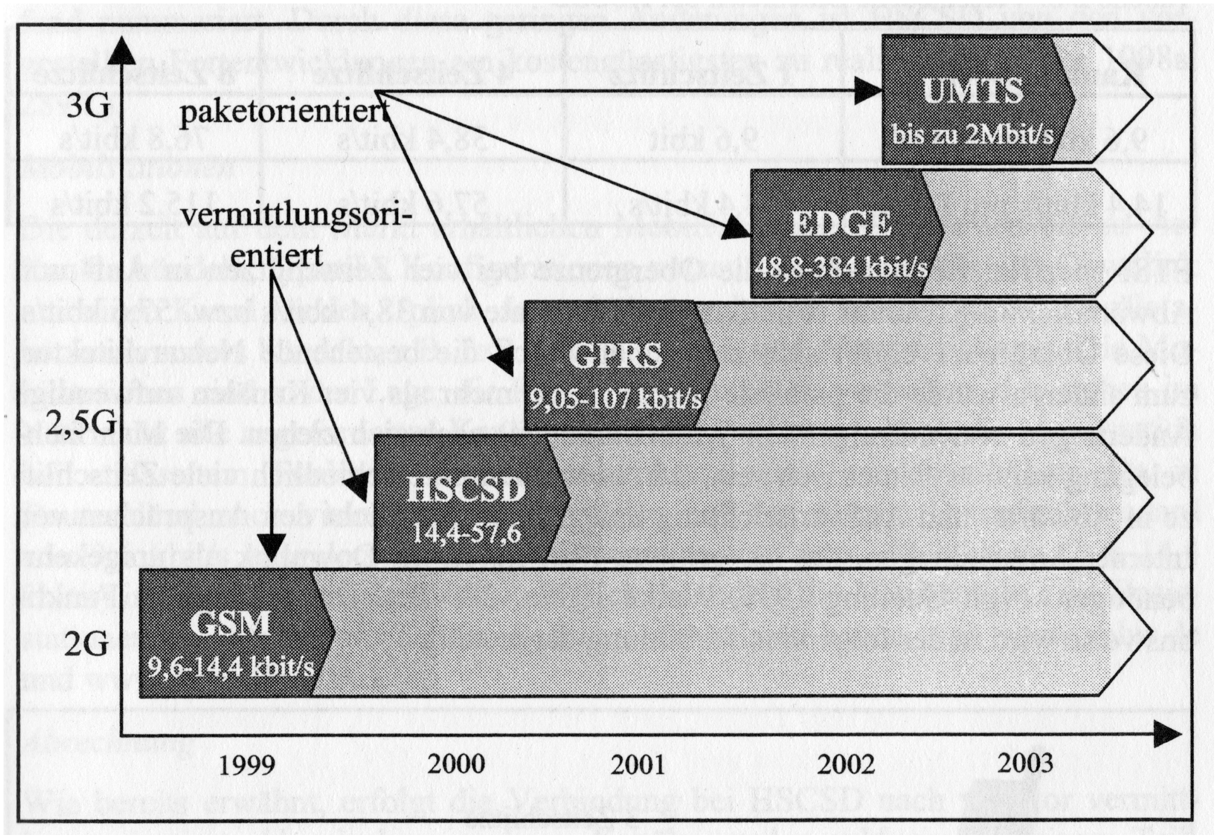


Figure 5.1: Network technologies and their evolution [29] *(paketoriented = packet-oriented, vermittlungsorientiert = exchange-oriented, bis zu = up to)*

latency afterwards. With LTE's all IP, flat architecture, the initial data packet connection is much faster, typically 50 ms, and then between 12-15 ms roundtrip latency afterwards." [33]. Latency will be thus measured by the implemented system.

5.2 Location of the end-user

Quality of mobile networks is also dependent from the location of the end-user. Despite the fact that mobile networks are designed well to offer good quality regardless to the end-user location, there are still location-dependent issues which play role when delivering mobile data to clients:

- Signal interference on same frequencies [33],
- error correction [33],
- number of concurrent users utilizing the network at the same location (i.e. traffic load) [33],
- signal disturbance by buildings, i.e. radio shadows [24],

- Doppler effect [24],
- impulse disturbance [24],
- fading [33],
- attenuation loss [33],
- signal to noise ratio [33].

As shown in Table 5.2 the bandwidth and network coverage are affected by disturbing elements such as written above. Table 5.3 explain other factors, which have influence to the signal quality. Considering all factors which affect the wireless communication the QoS and implied QoE can significantly vary in different locations as shown in Figure 5.2. Location is also an important constraint when making statements about the QoS.

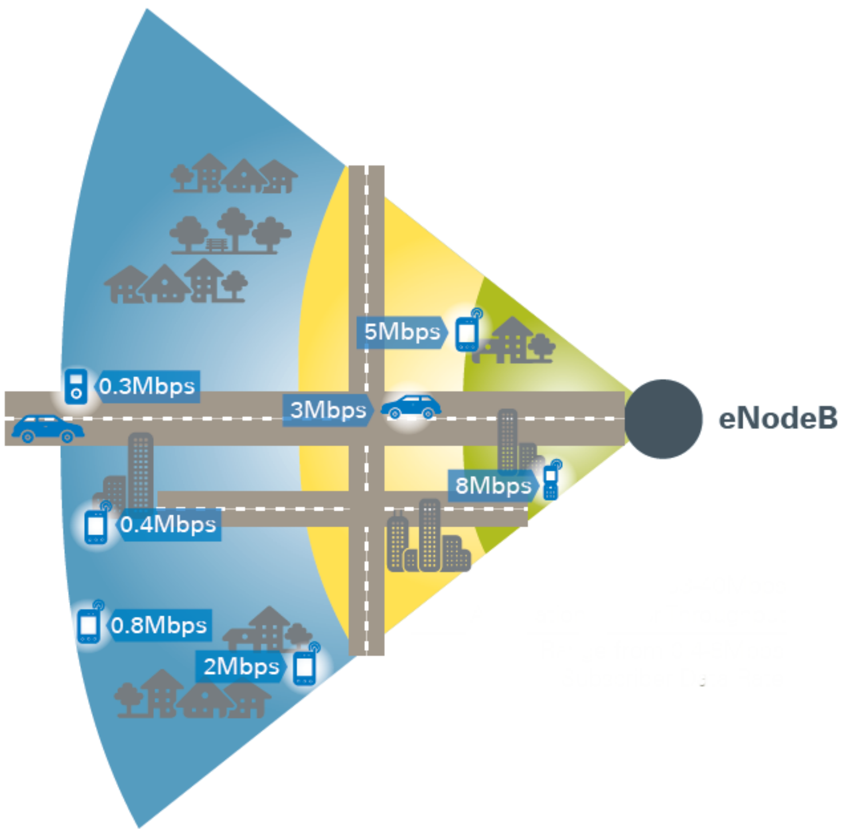


Figure 5.2: Subscriber data rate example [33]

Signal reach	in buildings 500m, outside of buildings up to 10km
Data rate	1200 - 4800 Baud
Limitations	communication not possible from within closed metal rooms

Table 5.2: Mobile coverage variation due to disturbance factors [24]

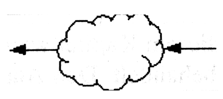
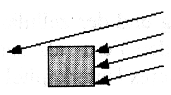
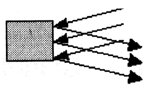
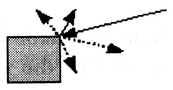
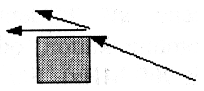
Disturbance reason	**Description**	**Visualization**
Damping	Decreasing signal reach due to atmospheric disturbance or influence of weather	
Shading	Signal can be shaded by barriers such as tunnels or buildings and thus the reception can be limited	
Reflection	Signal is mirrored by flat areas. Such signal is weaker than directly received signal.	
Scattering	Relatively small objects can scatter the signal.	
Diffraction	Cants can bend or distract the signal.	

Table 5.3: Signal disturbance factors [29]

5.3 Daytime

Network loads can vary during the daytime. That's why it is important to bind a particular result to the current time, so that locations can be evaluated with respect to the time.

5.4 QoS Regulations in the Network

Specification of the cell system, which is the base of mobile networks, explicitly requires economic utilization of the limited mobile radio frequency [24]. To achieve a fair bandwidth distribution within limited resources, MNOs apply artificial QoS rules, which try to limit bandwidth to the level that it doesn't affect QoE, e.g. by limiting the speed of torrent downloads where the bandwidth doesn't play a significant role and leaving the bandwidth available for other protocols such as video streaming. Applying such regulations the network can serve different services over a limited but utilized channel. However this is possible only to a certain extent. Peak times and traffic loads [33], when many users are connected and using internet services simultaneously, can lead to worse QoE even when the communication channel is regulated by QoS restrictions.

There can be also a business reason for a specific QoS regulation, e.g. limiting VoIP protocol in order to make internet telephony artificially unusable in order to force users to use paid telephony services. According to the research of Vitali Bashko there was found an evidence that such scenarios are implemented by some MNOs [20].

Chapter 6

Towards real-world use-cases of QoE

In previous sections QoS and QoE were described from the technical perspective. In this chapter the theory will be turned into the real-world environment. Internet content is delivered over best effort networks and is consumed on various types of devices such as smartphones [23], but what does the best effort look like, how can it be measured and evaluated - this will be part of expertise in following sections. The focus will be on mobile networks and mobile devices (smartphones or Tablets). This direction is interesting because of it's location, wireless bandwidth and current aggregation dependency. Also the data traffic generated by smartphones increased dramatically over last years [30]. Mobile devices also support the full bandwidth throughput over the current mobile technology over which the device is connected in contrast to e.g. throughput of fiber internet over WiFi-G. Mobile devices are directly connected to the technology they support and thus make the measurement more accurate.

Because the wireless bandwidth is shared by numerous users and the number users has an increasing tendency (Figure 6.1), the bandwidth observed on the users device can vary. There are different sources of generating traffic on mobile devices. [30] states increasing grow of video-on-demand applications, which generate noticeable traffic because of its nature. Music streaming counts also to bandwidth-intensive applications [30]. As we can see, there is growing need for data throughput, but the technology can only provide fixed maximum rate. How networks perform in peek times, or in more general how they perform given different conditions such as daytime, protocol, location etc. will be covered by the BonaFide+ Provider.

6.1 Tool for QoE Estimation: BonaFide+ Provider

The implemented system was not set up in a lab environment, but in real-world. This invited many people to use it for performing real network measurements and QoE estimations as it will be discussed later. Thanks to collaborating universities the systems infrastructure is maintained and provided to everybody. In addition, the system is Open Source and enables everybody to deploy it on own infrastructure, including all kinds of servers and the client application.

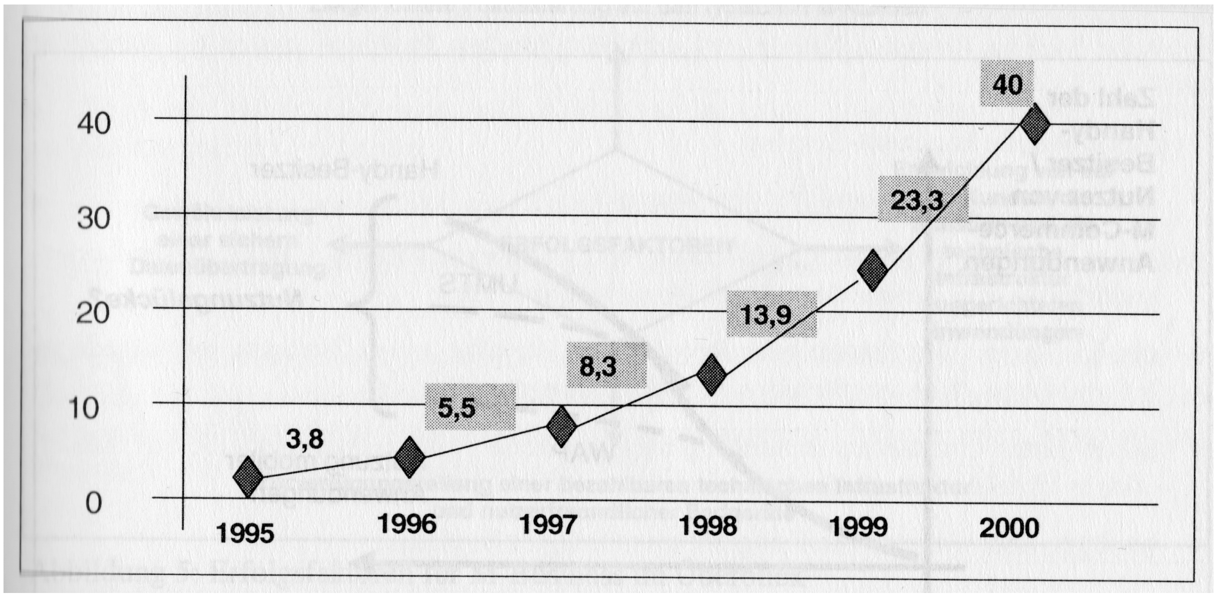


Figure 6.1: Number of mobile network users between 1995 and 2000 [34]

The system, which was implemented in the scope of this master thesis, is called **BonaFide+ Provider**. It is a distributed system containing 2 types of servers and a client application, which is available for Android operating system. The name is derived from the BonaFide Provider system initially developed by Vitali Bashko at the Jacobs University of Bremen [21]. The new one is using measurement engine of the original one and the rest was completely redesigned and turned into managed and distributed system. Details will be described in following sections.

6.1.1 Software Architecture

BonaFide+ Provider is a distributed system, which contains three communication parties:

- central server,
- measurement server,
- measurement and visualization client.

The original BonaFide Provider was able to perform protocol specific QoS measurements, however only single measurement server was supported and the configuration (e.g. IP address and port) had to be set manually in the client application. Protocol performance was already mentioned in previous sections - the new intention was to estimate QoE of specific services. For that, measuring only the random traffic would exclude the influence of QoS restrictions by providers and thus the measurement wouldn't be accurate for the given purpose. That's why I decided to simulate protocol on traffic used for measurements. This extended the *random data flow* to *protocol data flow* and made the measurement

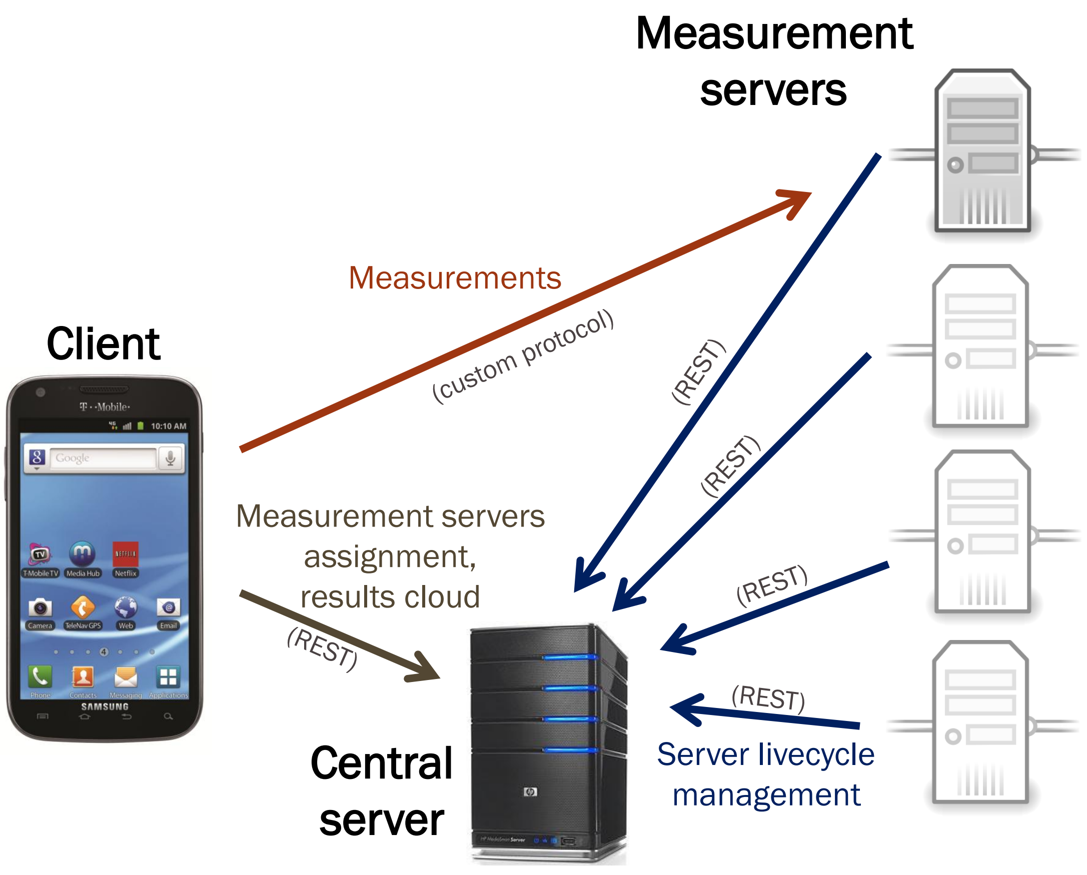


Figure 6.2: BonaFide+ Provider - infrastructure diagram

protocol-specific. By using such data, one can talk about bandwidth available for a specific protocol at the given moment. Measurement engine with the ability to simulate protocols while testing the network throughput was already implemented in the BonaFide Provider and because the code was open, it was used and extended it in the new system. The BonaFide Provider only contained a measurement server and a client application and the measurement engine was implemented between them. Results were only available locally, i.e. in the application on which measurements were performed. This is useful when somebody would like to test the actual network performance - without respect to location, MNOs and other parameters, which could improve later analysis. Measurement results were stored locally, but they provided detailed performance results only and without other parameters such as the BonaFide+ Provider does. The new system stores results in own cloud - i.e. centrally. This adds more flexibility to the data analysis, because not only own results can be analyzed and shown, but also results aggregated from others. Added the location awareness, one can see geographically distributed results and performance of networks in a global space.

Figure 6.2 shows how system parties of the new BonaFide+ Provider are deployed and how they communicate. The meaning of arrows is presence of communication between parties. Basically every party has implemented communication channel with each other party, but how we can see on the Figure, arrows are unidirectional. The direction shows originator and target of communication. For example an arrow originating at the client and pointing to the measurement server implies that the client attempts to establish connection to the listening (waiting for communication) measurement server. The communication is established only by directions shown in the Figure, however the communication itself is bidirectional, i.e. on top of the TCP/IP protocol. The Figure also explains the subject of each communication with labels colored by the same color as the corresponding arrow. The information in braces stands for the communication protocol used on the given path. Basically the whole system uses **REST** (Representation State Transfer [36]) services over HTTPs for communication except for the communication between client and the measurement server, where custom protocol was necessary for measuring the protocol-sensitive data throughput. The custom protocol was initially implemented and described by Vitali Bashko in his master thesis [20] and was reused and extended for performing measurements in the scope of this master thesis.

6.1.1.1 Central Server

Central server is the central unit of the whole system. There is only one central server in the system given by the design. However, to support higher loads and availability, the central server can be clustered into a load-balancing and/or fail-over cluster. Because all the communication with the central server uses REST services, which are data-oriented [36] and provided over a HTTP server running on it, adding such feature is not subject of new research, but can be easily implemented by already existing extensions of available HTTP servers, e.g. extension to the popular Apache web server: `http://httpd.apache.org/docs/2.2/mod/mod\_proxy\_balancer.html`. Also the Secure Sockets Layer (SSL) extension on top of HTTP was used and thus the communication interface is provided via HTTPs only to ensure integrity, authenticity and encryption between the central server and other parties of the system. The decision to implement the communication with REST services was based on its platform- and programming language independence. REST services are universal interface and can be consumed and called by every system. Claudio Riva and Markku Laitkorpi define REST services as "an architectural style derived from the Web, and its architectural elements and constraints aim at collecting the fundamental design principles that enable the great scalability, growth and success of the Web." [36]. The decision to implement REST interface was result of a long discussion between participating universities about the technology, which should be used to implement communication between the central server and other parties of the system. Another approach was to use Enterprise Java Beans (EJB), but after performing experiments with the server technology a big memory overhead was observed. Another disadvantage is the update process of the central server, e.g. when updating a single line in the central server, the whole application needs to be recompiled and deployed to the server. This process took a long time compared to another technology such as PHP. PHP was also evaluated and brought better performance (lower memory usage), portability (every PHP-enabled server can host the central server and every client - not only Java-based - can consume the service - in contrast to EJB) and maintenance (application can be edited on-the-fly by simply editing the source scripts - e.g. when changing parameters such as p, which will be discussed later). Because in the system PHP runs on Apache web-server, it is natively reachable via HTTP requests, which include GET, POST, PUT and DELETE methods. The decision to use this technology was seen as the best one available. In addition MySQL server [17] is used for storing and manipulating data.

Figure 6.2 shows many arrows pointing at the central server and we can notice the zero outdegree. It means that the central server act as a passive unit in the system. It provides services for other parties and manages the system without initiating requests to them. Each other party requires it in order to make the whole system available. In the following its necessity within the system will be described.

Starting with focus on the right side of Figure 6.2, i.e. the scope of communication between measurement servers and the central server, the system is able to contain an arbitrary number of measurement servers. However, at least one is essential for the system to work. Why there is need for many measurement servers will be discussed in the following 6.1.1.2 section. The main role in the scope of measurement servers is to manage their life-cycle, availability for clients and distribution.

6.1.1.1.1 Life-cycle Management

Life-cycle management takes care of managing the state of measurement servers. State can be either available or not-available. The central server has to be aware of all measurement servers, which are currently online. This is necessary to provide client applications with the information, with which server they should perform measurements. The most important perquisite is that maintained measurement servers are currently online and central server should take care of providing only such servers to the client applications.

One way how to achieve this is to open and maintain a persistent TCP/IP connection between each measurement server and the central server. In this case the central server would have to keep opened sockets with every measurement server and with every socket also to run separated thread. When the connection would be interrupted (central server encounters exception from the socket connection), the measurement server would be deleted from the measurement servers list. The delay would be dependent from the keep-alive timeout of the TCP/IP connection. This would be a working solution, however costs for maintaining threads and sockets on the central server were seen as high and a better solution was searched.

Another approach to maintain statuses of measurement servers should work in a similar way, but with significant improvement of reserved resources maintained by the central server. Instead of using TCP/IP sockets another non-persistent connections for keep-alive messages were considered. In this model the central server doesn't have to manage a persistent connection to every measurement server, but measurement servers will send a short keep-alive message (we can describe them also as *heart-beat messages*) to the central server at fixed intervals to update the central server about their availability. Every connection will be closed immediately after the information is delivered to the server and thus there is no need for any socket management. In order to implement this, an available approach was used and the central server is available for such communication via REST services.

The state is determined by the central server and because the communication is unidirectional, notification service in the measurement server was implemented, which sends periodical notification messages to the central server. That messages carry additional informations about the measurement server and the meta-information that the server is available and running. By collecting such messages, the central server is able to evaluate the state of each measurement server on-the-fly, i.e. to manage its life-cycle. There are several challenges for the life-cycle management:

- Measurement server can be started and integrated into the system easilly.
- Measurement server can go offline due to any reason such as server shutdown, network interruption etc.
- Measurement server must be accessible over the internet.

The life-cycle management is based on keep-alive messages received from measurement servers. If p is period of keep-alive messages, c is current time, e is the time when last keep-alive message was received (time of an *event*) and l is network latency, the measurement server availability a is shown in 6.3 and will be considered as follows:

$$\text{a is true if: } e \geq c - (p + l), \text{ false otherwise}$$

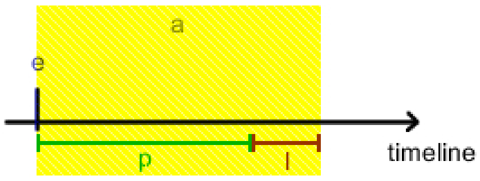


Figure 6.3: Measurement server availability timeframe

This approach dynamically manages online servers over time and satisfies all challenges written in 6.1.1.1.1. The central server only delegates a measurement server to clients when it gets keep-alive message from it within a. When no keep-alive message arrives within the availability threshold a, central server considers the server as offline. By setting p to an appropriate value, measurement servers can leave without notifying the central server and without affecting the overall functionality. Because clients are resistant against connecting to offline measurement servers, they can pick another server without returning with an error, so delegating an offline server for a short time-frame ($\leq a$) will have no effect on the functionality of the whole system.

This implementation features another advantage as well - measurement servers don't require to have a static IP address. When the IP address changes over time, the central server will be notified from the new one, adds a new server with this IP address to the measurement server list and removes the server with the old one after the availability threshold a. Again, by setting p to an appropriate value makes this transition invisible.

An *appropriate value of* p was mentioned several times. Because p is the main variable deciding about the measurement server status, it has to be chosen with respect to the following challenges:

- Measurement servers state should be always up-to-date.
- Updating the state of the measurement server shouldn't affect the performance of the central server.

To combine this two challenges, p should not be too big (in conflict with the first challenge) and also not too small (in conflict with the second challenge). Setting p should be adjusTable over time and should satisfy both of them. Initially the system was be set up with the value of $p = 2$ and its implementation makes it easy to change its value later.

6.1.1.1.2 Availability to Clients

Now when the list of online measurement servers is available, the design can move to the second aspect - to the **availability to clients**. Even with the information about online measurement servers, we are not that far to initiate the connection to them. At the moment we have their IP addresses and the information if each of them are online or not. The IP address is determined automatically by the central server from the recieved connection request and thus can't be faked by incorrect setting. But for a successful connection we need more than the IP address:

- name, which is human-readable and which can identify the server for humans,
- port, to which the measurement server is bound to,
- location of the measurement server, which will be used for performance issues discussed later.

The keep-alive message was extended to contain also these parameters. The notification service, which updated the central server about its state was extended to advertise the measurement server completely to the central server, including parameters mentioned in the Listing 6.1.1.1.2. Implemented like this, the central server can provide complete technical information for the connection (state, IP address and port) and identification for end-users (name, location). Location is used also for performance optimalization and will be described later.

The measurement server needs to be configurable due to the need of providing informations about itself. The configuration is implemented directly on the measurement server and can be provided by two ways:

- by configuration file *BonaFide.conf* as shown in Listing 6.1,
- or by providing parameters via command-line arguments.

```
# BONAFIDE CONFIGURATION FILE

# Path to the central server service
centralserverurl=https://bonafide.pw/rest
# Name of this measurement server
name=University of Zurich
# Port Number used to run main socket
port=4000
# Path to file that contains list of protocols to load
list=../list
# Path to a local folder to store submitted measurement
   results
storage=
# Define level of logging. Possible values sorted by priority
   : OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE. INFO is
   default value.
verbose=INFO

# Geograpgic position of the measurement server
latitude=47.414242
longitude=8.549490
```

Listing 6.1: Configuration file *BonaFide.conf* of the measurement server

The first option, i.e. setting parameters in the *BonaFide.conf* configuration file, makes it easier to (re)start the server (no start-up are required when starting the server). If there is a need of setting arguments dynamically on server start-up - for example when the measurement server should be started dynamically by call from another program - there is the second option for setting parameters. The second option has precedence in case of configuration conflicts. By this implementation, particular saved parameters in *BonaFide.conf* (if set) can be easily overwritten by setting them as arguments when starting the measurement server. All available configuration parameters are listed in Listing 6.1 and each of them can be set either via the configuration file or via command-line arguments. When providing them via the command line, they have to be appended to the server start call as follows:

<BonaFide server start call> -argument1 value1 -argument2 value2

6.1.1.1.3 Measurement Server Distribution

Configuration parameters are checked during the measurement server start. If any parameter is missing, the measurement server will not start and inform the user about the problem. The measurement server uses parameters loaded by listed methods for its advertisement to the central server.

BonaFide+ Provider system added support for multiple measurement servers. Each server has a properts *location*, which differs it from others. Figure 6.4 shows current **measurement server distribution**. In order to achieve the most accurate results when measuring connection performance, the routing distance should be taken in charge in order to minimalize routing delays, which affect the measurement result. The measurement should text the throughput of the network connection and thus the optimal model would be to have measurement server directly connected to the base station to which the client is connected to. Because this is not possible, I was considering other strategies to eliminate side-effects of long-distance routing. The goal of a correct measurement is to perform it over the minimal amount of hops between the client and the measurement server. However the decision about which server should be chosen from the list of available measurement servers had to be researched in depth. First and probably most correct decision about which is the nearest server would be to measure the routing distance by itself, however to check the number of immediate hops on the routing path between the client and the measurement server is time-consuming and linearly growing with the amount of available measurement servers. The routing distance can't be cached and reused by other clients in the system, because routing decisions can vary for each node in the network (in mobile networks the routing decision can also be affected by the current base-station association) and thus each client would need to measure routing distance to each available measurement server prior to each single measurement. However, this is very inefficient.

Another approach would be to ping each measurement server from withing the client and decide based on the response time. To have a correct response, factors such as initial establishment of wireless data connection should be excluded from the time measurement. However, this is still time consuming and the time spent during the measurement server picking process is growing with the number of available measurement servers.

A solution was searched, which would make the number of available measurement servers independent from the picking process. The direct approach would not be able to achieve it. The intention was to bridge the goal to a different strategy, which could pick the best-suited measurement server instantly and as correct as possible. University of Maryland showed in their study [19] that there is a connection between geographical distance and round-trip times (RTT) between server and client. According to the study with increasing geographical distance the RTT was increasing as well. This behavior sounded very hopeful, because based on [19] I would be able to dynamically find out the server with the lowest RTT with respect to the current location of the client. Because measurement servers are distributed over multiple locations as shown in Figure 6.4 and their location is known to the central server, the goal is to take advantage of the connection between RTT and geographical distance and compute the nearest measurement server picking strategy. Nearest server now became a double meaning - what is the geographically nearest can be now considered as the one with the lowest RTT on the path. In this thesis a mechanism for minimizing the geographical distance and in conclusion for minimizing the RTT for measurements between client and measurement server will be designed and applied.

Distance between measurement servers and clients can dynamically vary, as we focus measurements of mobile networks. Client can move and change their location and thus the distance to the measurement server. The location of a measurement server is considered to be constant. As I already mentioned, the measurement server advertises its location to

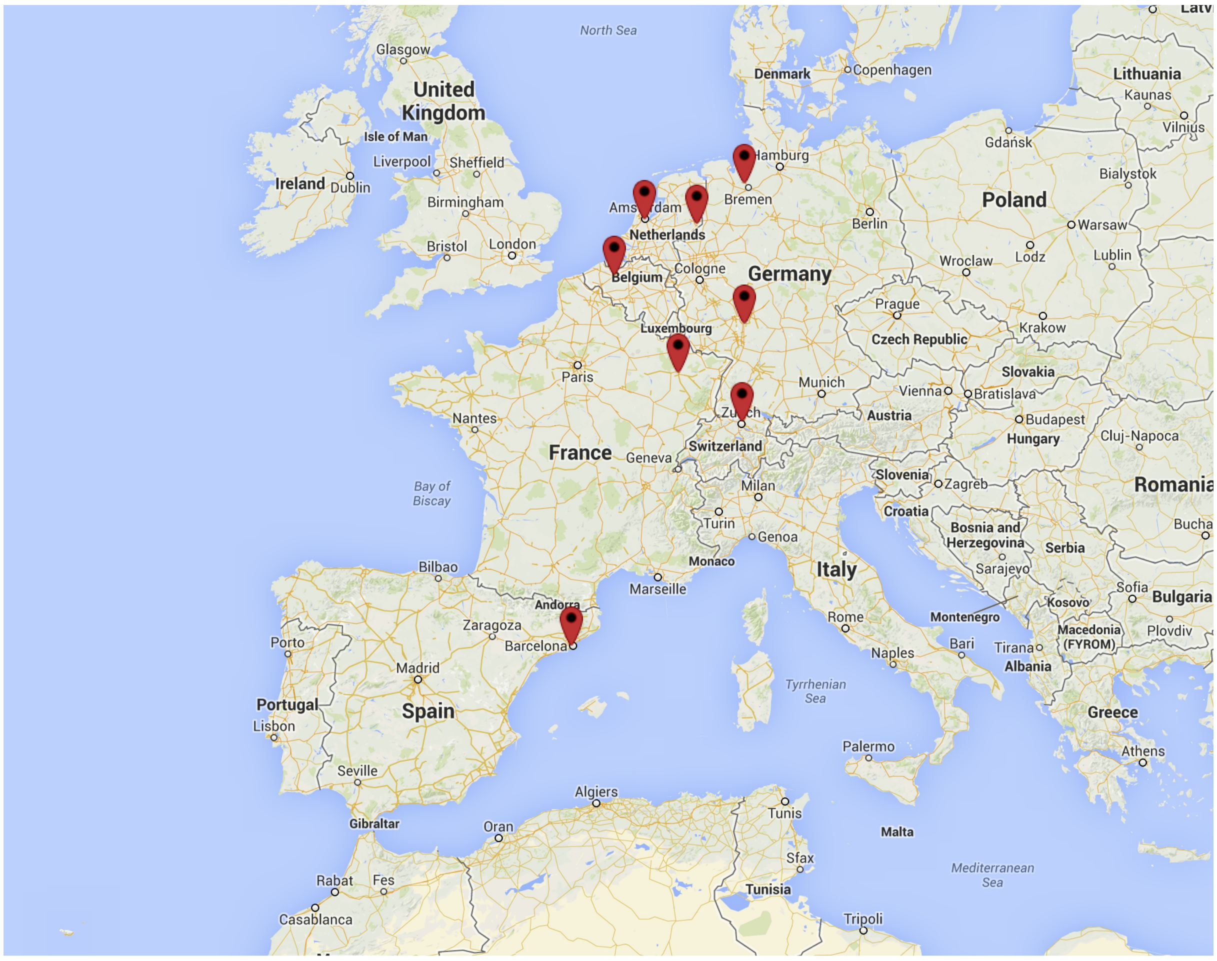


Figure 6.4: Measurement server locations

the central server. At this point, the location became much higher importance than just locating the server on the map for the users better overview. For distance calculation also the second point needs to be known - the location of the client. Because most devices, which will be supported by the Application, are equipped by position sensors [3] and different location strategies [1], the client was implemented in the way that it can request the list of available measurement servers given its current location retrieved by any of the available location strategies on the mobile device. As a result, the central server will know the clients location at the time the list of available measurement servers is requested. Because the client should perform a measurement against the nearest measurement server, the central server has to adapt the list and provide the client with the best decision based on the current context (distance between the client and each measurement server). The implementation involves a list sorted by the geographical distance between client and the measurement server in the ascending order. Given such list, the client can process measurement servers in the order they are distanced from it, e.g. try to measure against the nearest and if it fails, measure against the second nearest server and so on.

The next question was how to compute the distance between two geographic points. Point class of the MySQL database management system [5] is used for storing location data.

MySQL added support for spatial extensions since version 5.0.16[1], however the distance computation between two points was not implemented in versions before 5.6 according to the official MySQL database management system (DBMS) documentation [8]. Because of missing detailed documentation of the function and because of limited support in MySQL DBMS versions I decided to research distance computation independently from the current implementation and to implement a custom variant, which I will be able to reproduce and explain in detail in following paragraphs and which will be supported by all versions of MySQL DBMS with spatial extensions.

Julien Montavont and Thomas Noel describe in their article the way of computing the distance between two points given by their coordinates: "The distance between two points can be calculated using the **Haversine formula**." [32].

Let us denote coordinates of $point1$ and $point2$ as (lat1, long1) and (lat2, long2) respectively, latitude separation with Δlat and the longitude separation with $\Delta long$, where angles are in radians, and R is the Earths radius (R = 6, 371km). The distance d between the two points is calculated by the formula as follows:

$$haversin\left(\frac{d}{R}\right) = haversin(\Delta lat) + cos(lat1) * cos(lat2) * haversin(\Delta long)$$

where the Haversine function is given by

$$haversin(\delta) = sin^2\left(\frac{\delta}{2}\right)$$

After resolution on the right side we get:

$$haversin\left(\frac{d}{R}\right) = sin^2\left(\frac{\Delta lat}{2}\right) + cos(lat1) * cos(lat2) * sin^2\left(\frac{\Delta long}{2}\right)$$

Let h denote the $haversin(d/R)$.

$$h = sin^2\left(\frac{\Delta lat}{2}\right) + cos(lat1) * cos(lat2) * sin^2\left(\frac{\Delta long}{2}\right)$$

One can then solve for d either by simply applying the inverse Haversine or by using the $arcsin$ (inverse of sine) function:

$$d = R * haversin^{-1}(h) = 2R * arcsin(\sqrt{h})$$

The computation can be done in the source code of the central server or directly in the database by a stored function. Implementing it in the code has several disadvantages:

[1]Support was added for InnoDB engine. Source: https://dev.mysql.com/doc/refman/5.0/en/spatial-extensions.html

- Distance can be computed only via code. Custom analytical SQL queries executed during data analysis directly in the DBMS will not be able to take distance in charge. Every distance computing will require an external script.
- Distance-related queries will be processed via the central server. All rows will be fetched from the DBMS and processed by the central server. This can include huge amount of data and thus the processing procedure would be of an additional overhead. This also includes distance-related sorting, which will be part of the nearest measurement server search.

Implementing distance computation directly in the database would lead into better performance and query flexibility. The haversine formula can be implemented as a stored function in MySQL DBMS. Implementation is shown in 6.2. This function will take coordinates (latitude and longitude) of two points and will return the distance between them in kilometers. This function also enables sorting by distance between given point and stored coordinates. In other words, with stored coordinates of measurement servers we can sort their distance to the client by given coordinates in the ascending order on-the-fly and enable the client to perform measurements against the nearest server.

```
DELIMITER //

DROP FUNCTION IF EXISTS DISTANCE; //

CREATE FUNCTION DISTANCE(lat1 FLOAT,lon1 FLOAT,lat2 FLOAT,
   lon2 FLOAT)
    RETURNS FLOAT NO SQL DETERMINISTIC
    COMMENT 'returns␣distance␣(km)␣between␣2␣points␣on␣Earths
       ␣surface'
BEGIN
    SET lat1=radians(lat1);
    SET lat2=radians(lat2);
    SET lon1=radians(lon1);
    SET lon2=radians(lon2);
    RETURN 2*6371*asin(sqrt(pow(sin((lat2-lat1)/2),2)+cos(
       lat1)*cos(lat2)*pow(sin((lon2-lon1)/2),2)));

END; //

DELIMITER ;
```

Listing 6.2: Haversine function in MySQL

With this stored function, MySQL DBMS is not only able to tell one how far the server is, but also perform sorting operations. Listing 6.3 shows an example query for such sorting, which is used by the central server. Variables starting with **$** are input or configuration parameters and are injected into the query by the central server.

```
SELECT id,ip,name,port,latitude,longitude,DISTANCE($latitude,
   $longitude,latitude,longitude) AS distance FROM
   bonafide_measurement_servers WHERE last_seen_timestamp>=
   $minimum_timestamp_of_alive_servers ORDER BY distance ASC
```

Listing 6.3: Distance-based sorting

Variables *$latitude* and *$longitude* are coordinates of the clients current position. They are received from the client on measurement server listing request. As seen in the query in Listing 6.3, the central server can then use the built-in function shown in Listing 6.2 for obtaining the distance between client and measurement servers. When the distance is present, the *ORDER BY* clause can then sort the result set by the distance in ascending order. This ordered result set is list of measurement server in the order they are distanced from the client. The calculation is directly done in the DBMS and thus only the final (ready-to-use) ordered result set is sent back to the central server.

Central server manages besides measurement servers covered by previous subsections also **measurement results**. Following subsections will focus on the left part of Figure 6.2.

6.1.1.1.4 Centralized Storage of Measurement Results

BonaFide+ Provider system stores measurement results in the central server storage instead of using local storage of each client. The central server takes care about their collection, distribution and protection. There were several challenges when implementing this part of the system, because handling of such data is essential when obtaining trust of end-users. While everybody would expect in-depth overview of own results, providing the same to other users could significantly harm the trust to the system. This involves not only other users of the system, but also data interception by third parties such as man-in-the-middle attacks. Each potentially unexpected behavior of the system by end-users should be eliminated and thus following aspects were taken in charge when designing the measurement results centralized storage.

- **Usability, privacy:** each user should have full and detailed access to **only** own measurement results,
- **Accessibility, portability:** each user should have the possibility to access own results from an arbitrary device and should be able to migrate results from device to device,
- **Anonymity, privacy** stored informations should be limited to the extent that nobody can identify the end-user, even not the provider of the central server,
- **Privacy, authenticity, encryption:** data communication between clients and the measurement results cloud should be encrypted and authenticated in order to avoid unauthorized access to data (e.g. by network providers, which can in combination with the communication origin and the transmitted information reveal the users identity),

- **Accessibility, anonymization:** data visible to all users should provide only statistical overview.

While considering listed challenges, the central server, which is provider of the measurement results cloud, had to implement all of them in order to make the whole system safe and robust. Without that the future of the system would be endangered.

In order to implement **usability**, each user has simple (one-click) access to detailed own measurement results stored on the central server. The overview contains location, network provider, country and detailed measurement results such as bandwidth, data usage etc. With such overview the end-user has full control of own measurements and access to its own results, which can be used for network quality interpretations. By adding **privacy** the detailed overview is only accessible by the user of origin. No other user can retrieve detailed results of other users. This is achieved by custom implementation of transparent authentication, which will be described in Subsection 6.1.1.1.5. In order to make this access **accessible and porTable** each user can use an arbitrary device (supported by the client application) and can migrate results from end-device to end-device. The exact mechanism will be described in Subsection 6.1.1.1.5.

Another point of interest is the communication between clients and the central server. Because the communication carries the whole detailed overview when storing and retrieving own measurement results, the system should also make sure that the information is only available to desired parties, i.e. to the central server and to the particular client. This is achieved by forcing the communication between clients and the measurement server as shown on Figure 6.2 to be fully **encrypted and authenticated**. The communication interface is secured by using HTTPS in combination with signed server certificate, which is verified prior to any data transmission between the central server and clients. This makes man-in-the-middle attacks and unauthorized data interception impossible.

By adding next level of security, the system is designed to protect end-users even from identification by central server providers (**anonymity**). Data, which is stored along with measurement results, don't contain any information which can reveal the identity of the end-user. In contrast to services such as WhatsApp, where the user is required to provide its phone number[2], BonaFide+ Provider doesn't require or even collect such data in background. No e-mail address, phone number or other data available on the device, which uniquely identify the end-user, are sent to the central server.

End-users of the system can access statistical overview of all measurement results collected by all users. This overview doesn't contain any detailed informations about other users or about their detailed performance and coordinates. Such results are **anonymized** and **accessible** in the way that no privacy is violated nor the detailed overview owned by each user is provided to everybody. Having access to such statistical overview users can compare performance of networks at specific locations, using specific mobile network technology or by using different protocols (QoS and QoE relevant). How the overview is provided to end-users will be described in Section 6.1.1.3.4.

[2]Citing terms and conditions of WhatsApp available on the official website http://www.whatsapp.com/legal/: " In order to access and use the features of the Service, you acknowledge and agree that you will have to provide WhatsApp with your mobile phone number."

6.1.1.1.5 Identification of Measurement Results

Measurement results are stored and managed by the central server. Each result itself is anonymized, i.e. there is nothing stored about the users identity. Despite this fact, each user should have access to own measurement results, including the whole history. In order to achieve that and to preserve all privacy issues discussed before a custom mechanism was implemented.

Because identification by phone number or by other *personal* information would reveal the users identity and everything about his network, a *pseudonym* identification was considered. By applying such identification, results still can be aggregated and stored under unique identification, but without any connection to the end-user.

Implementing such mechanism involves following procedures:

- transparent automatic generation of the artificial identification *key*,
- maintaining the identification key by both central server and client,
- linking stored measurement results to the identification key,
- providing detailed results if correct identification is supplied by the client.

In the first step the artificial identification key was added to each result and an extra identification table was added to the system in order to link results to the identification key. The relation is shown in Figure 6.5. All keys are stored in this table and results are mapped by a foreign key to them. This implementation enables fast indexed check if a provided identification key is available in the system and joined set of results belonging to the particular identification key.

An identification key will be called **identification token** in the scope of this master thesis. It is an upper-case string of length 15 characters and it is randomly generated as shown in Listing 6.4. Initially the client has no identification token assigned and thus the first request of measurement history and the first submit of measurement result is identified with an empty string. When such request arrived to the central server, two cases are considered:

- Client requests list of previously submitted measurement results with an empty identification token. In this case the central server can't find any connection to submitted results, because the identification token is missing and thus an empty list will be returned to the client.
- Client submits new measurement result with an empty identification token. In this case the measurement result needs to be stored with proper identification, which doesn't exist yet.
  - At this moment - prior to saving the result - the identification token is generated as shown in Listing 6.4 and stored in the identification token Table. The identification token is generated until it is unique across all already persisted identification tokens.

  – Now when the token is created the result is stored in the database and linked to the newly created entry in the identification token table as shown in Figure 6.5.
  – The new identification token is sent back to the client and the client replaces the identification token in its internal database. At this point, the client is assigned an identification token, which will be used in future communication requests related to the measurement results management with the central server.

```
function generate_random_token() {
return strtoupper(substr(md5(rand(100000000,999999999)),0,15)
   );
}
```

Listing 6.4: Function for generating the identification token

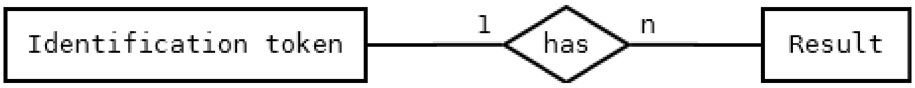


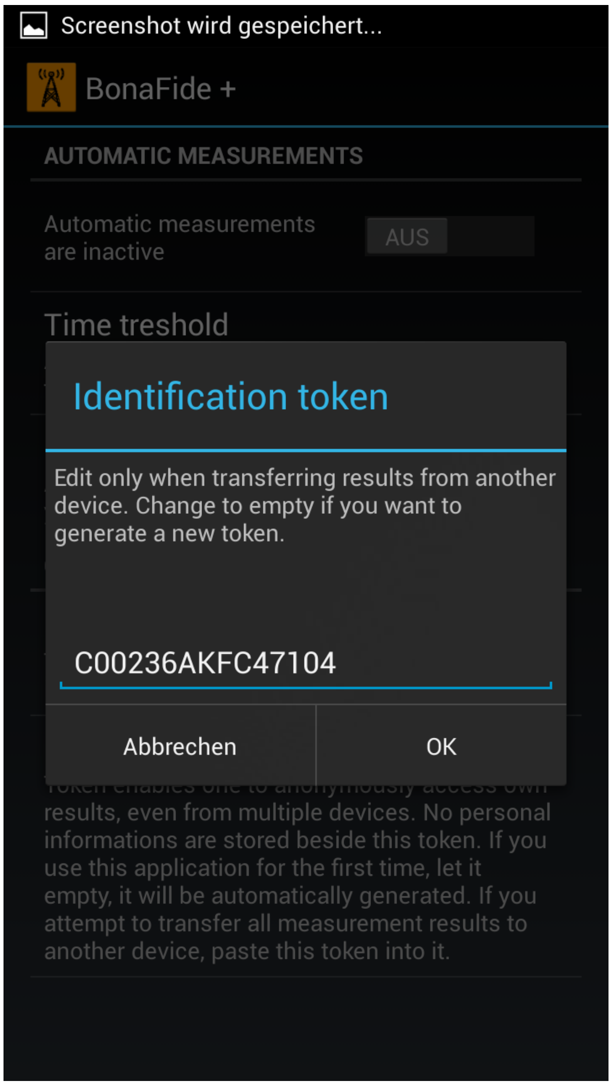
Figure 6.5: Identification token relation

Once this phase is done, submitting new measurement results and requesting list of measurement results for the particular user are bound to the maintained identification token. This token is maintained by the central server and by the client. The identification token is hard to guess because of the number of possible combinations and even if it would be guessed (can happen when the system will manage huge number of identification tokens), there is no connection to the end-user and thus this approach can be considered as safe for this particular purpose. Identification token only identifies detailed measurement results and results history and is sent by the client when performing both operations - result retrieval and submit.

Identification tokens are mainly assigned by the central server when the client submits first measurement result with an empty or invalid token. There is no action required from the user. Client then stores the token locally and uses it for future communication. For portability purposes the system supports multiple clients with the same identification token. Because the token can be changed in the client as shown in Figure 6.6, users can:

- set the same identification in multiple devices and share results across them. Example scenario: Company XY would like to test their mobile data plans and thus each employee is requested to collect measurements. Shared identification token enables centralized analysis of measurement results collected by all employees of the company.
- transfer collected measurement results to a new device.

Given this flexibility, users can find out new scenarios which will fit their needs by just editing the configuration and without the need to set up their own BonaFide+ infrastructure.


Screenshot wird gespeichert...

BonaFide +

AUTOMATIC MEASUREMENTS

Automatic measurements are inactive — AUS

Time treshold

Identification token

Edit only when transferring results from another device. Change to empty if you want to generate a new token.

C00236AKFC47104

Abbrechen | OK

results, even from multiple devices. No personal informations are stored beside this token. If you use this application for the first time, let it empty, it will be automatically generated. If you attempt to transfer all measurement results to another device, paste this token into it.

Figure 6.6: Identification token setting in the client application

Following subsections will handle the internal structure of the data management and the communication interface of the central server.

6.1.1.1.6 Stored Data

This subsection is dedicated to the data model behind the central server. The central server uses relational database for storing and manipulating data. Data are interconnected by foreign keys and thus the InnoDB database engine of MySQL DBMS is used [7]. There are three tables in the database and they are defined in Listings 6.5, 6.6 and 6.7.

```
CREATE TABLE IF NOT EXISTS `bonafide_measurement_servers` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `ip` varchar(15) NOT NULL,
  `hostname` varchar(50) NOT NULL,
  `port` int(11) NOT NULL,
  `name` varchar(50) NOT NULL,
  `last_seen_timestamp` int(11) NOT NULL,
  `latitude` float(10,6) NOT NULL,
  `longitude` float(10,6) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `ip` (`ip`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COMMENT='Collection␣of␣
   measurement␣servers␣with␣additional␣informations'
   AUTO_INCREMENT=1 ;
```

Listing 6.5: DB table for storing data about measurement servers

```
CREATE TABLE IF NOT EXISTS `bonafide_tokens` (
  `user_token` varchar(15) NOT NULL,
  `created_datetime` datetime NOT NULL,
  PRIMARY KEY (`user_token`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='Tokens␣-␣
   anonymous␣identifiers␣of␣users';
```

Listing 6.6: DB table for storing identification tokens

```
CREATE TABLE IF NOT EXISTS `bonafide_measurement_results` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `user_token` varchar(15) DEFAULT NULL,
  `measurement_datetime` datetime NOT NULL,
  `measurement_location` point DEFAULT NULL,
  `measurement_server_id` int(11) DEFAULT NULL,
  `measurement_server_name` varchar(50) NOT NULL,
  `protocol_specification_name` varchar(30) NOT NULL,
  `is_mobile_network` enum('true','false') NOT NULL,
  `network_type` varchar(30) NOT NULL,
  `country` varchar(30) NOT NULL,
  `operator` varchar(30) NOT NULL,
  `operator_name` varchar(30) NOT NULL,
  `signal_strength` varchar(30) NOT NULL,
  `latency` bigint(20) NOT NULL,
  `upload_random_roundtrip_time` bigint(20) NOT NULL,
  `upload_random_bytes_sent` int(11) NOT NULL,
  `upload_random_bandwidth` bigint(20) NOT NULL,
  `upload_random_completness` text NOT NULL,
  `upload_protocol_roundtrip_time` bigint(20) NOT NULL,
  `upload_protocol_bytes_sent` int(11) NOT NULL,
  `upload_protocol_bandwidth` bigint(20) NOT NULL,
  `upload_protocol_completness` text NOT NULL,
  `download_random_roundtrip_time` bigint(20) NOT NULL,
  `download_random_bytes_sent` int(11) NOT NULL,
  `download_random_bandwidth` bigint(20) NOT NULL,
  `download_random_completness` text NOT NULL,
  `download_protocol_roundtrip_time` bigint(20) NOT NULL,
  `download_protocol_bytes_sent` int(11) NOT NULL,
  `download_protocol_bandwidth` bigint(20) NOT NULL,
  `download_protocol_completness` text NOT NULL,
  `upload_total_bytes` int(11) NOT NULL,
  `download_total_bytes` int(11) NOT NULL,
  `error_message` text NOT NULL,
  PRIMARY KEY (`id`),
  KEY `token_id` (`user_token`),
  KEY `measurement_server_id` (`measurement_server_id`),
  KEY `measurement_location` (`measurement_location`(25))
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COMMENT='Storage␣for␣
   measurement␣results' AUTO_INCREMENT=1 ;

ALTER TABLE `bonafide_measurement_results`
  ADD CONSTRAINT `bonafide_measurement_results_ibfk_1`
     FOREIGN KEY (`user_token`) REFERENCES `bonafide_tokens`
     (`user_token`) ON DELETE SET NULL ON UPDATE CASCADE,
  ADD CONSTRAINT `bonafide_measurement_results_ibfk_2`
     FOREIGN KEY (`measurement_server_id`) REFERENCES `
     bonafide_measurement_servers` (`id`) ON DELETE SET NULL
     ON UPDATE CASCADE;
```

Listing 6.7: DB table for storing measurement results

Each table contains self-explanatory variable names for stored data. In the *BonaFide_measurement_re* table definition there are multiple columns for storing upload and download performance. That columns contains *random* and *protocol* keyword. Random measurements are performed by using random data exchange between the measurement server and the client. It can be supposed as simple throughput measurement, i.e. without taking specific QoS regulations in charge. Protocol stands for protocol-specific measurement. When measuring protocol bandwidth, measurement server and client simulate the protocol in order to achieve protocol-related bandwidth. In-depth description follows in the subsection about the measurement server.

Table 6.1 explains in detail why and for what are different upload and download columns used.

Specification of *upload_ and *download_** DB columns**		
[random/protocol]_roundtrip_time	Duration of the data exchange during the test	μs
[random/protocol]_bytes_sent	Amount of data used in the test	bytes
[random/protocol]_bandwidth	Bandwidth	bytes/s
[random/protocol]_completness	Statement about the test completeness	

Table 6.1: REST service - submission of measurement results

6.1.1.1.7 REST Interface Specification

Central server only provides communication with other parts of the system via REST services and it only listens to incoming request. No outgoing connection is initiated by the central server. The communication interface is reachable via the Apache webserver with *mod_ssl*[3] extension, which protects the communication from eavesdropping. This is essential for privacy reasons, because clients are sending e.g. their location over the internet network, where intercepting such information could lead into locating the application end-user. HTTPS, which is currently the only possible way to communicate with the central server, protects the whole system from man-in-the-middle attacks and enables only encrypted and authenticated communication channel for the data exchange.

The REST interface is published on the following URL:

https://bonafide.pw/rest/

This is the **base path** for all services, which are provided by the central server. This URL can vary and thus it is configurable in all parts of the system, i.e. in the measurement server and in the client application. The second reason for making this parameter configurable is the nature of the project - the whole system is open-source, what means that everybody can get the code and deploy the system on his own infrastructure. In this scenario, the URL will be different from the official one and thus it needs to be conFigured in other parties as well. I enabled the configuration via settings (client) and via configuration parameters (measurement servers), so the change doesn't require recompilation of the source code and can be easily changed even by non-technical user.

In addition to the base path of the REST services URL, there are subpaths pointing to particular services. There are different services and different suffixes to the base path. They are described in the following part. Details of the implementation can be seen in the attached source code.

Retrieval of the measurement server list sorted by name	
Listing of measurement servers sorted by name should be used when no location of the client is available.	
URL suffix:	/measurement-servers/list
Method:	GET
Input format	*no input*
Output format:	JSON

Table 6.2: REST service - measurement server list sorted by name

[3]mod_ssl extension provides SSL v2/v3 and TLS v1 support for the Apache HTTP Server. Source: http://httpd.apache.org/docs/2.2/mod/mod_ssl.html

Retrieval of the measurement server list sorted by distance	
Listing of measurement servers sorted by distance should be used when location of the client is available. The distance is computed between the clients location and the location of measurement servers. The first server in the returned list will be the nearest measurement server and the last one the one with the longest distance.	
URL suffix:	/measurement-servers/list/{latitude}/{longitude}
Method:	GET
Input format	Input is provided in form of path parameters
Output format:	JSON

Table 6.3: REST service - measurement server list sorted by distance

Advertising measurement to the central server	
This service should be used by measurement servers for registering them into the central server	
URL suffix:	/measurement-servers/add
Method:	POST
Input format	JSON
Output format:	JSON

Table 6.4: REST service - advertising measurement to the central server

Listing of measurement results	
This service should be used by the client application for reading all own measurement results from the central server	
URL suffix:	/measurement-results/list
Method:	GET
Input format	*no input*
Output format:	JSON

Table 6.5: REST service - listing of measurement results

Listing of measurement results for a viewport	
This service should be used by the client application for reading own measurement results from the central server in the given area (viewport). This is used for rendering results on a map. The output will contain definition of result squares on the map and the view will be adapted according to the requested viewport. The viewport is the map area currently displayed to the end-user by the client application. Results are also filtered based on the provided filtering request.	
URL suffix:	/measurement-results/list-for-viewport/{south-west-latitude}/{south-west-longitude}/{north-east-latitude}/{north-east-longitude}
Method:	POST (POST method is used because of the need to send many data such as the viewport definition and the definition of filter to the central server. Sending all such data would be too complicated over path parameters and thus I decided to make an exception and to use POST instead of GET for implementing this service. GET doesn't support sending JSON-serialized data.)
Input format	path parameters (viewport definition) and JSON (filter definition)
Output format:	JSON

Table 6.6: REST service - listing of measurement results for a viewport

Submitting results to the central server	
This service should be used by client applications to submit measurement results to the central server.	
URL suffix:	/measurement-servers/add
Method:	POST
Input format	JSON
Output format:	JSON

Table 6.7: REST service - submission of measurement results

This subsection covered the importance of the central server in the BonaFide+ Provider system. In the following subsection I will discuss the measurement server into detail.

6.1.1.2 Measurement Server

Measurement servers are the key component for performing network measurement tests. As described in the previous subsection, the BonaFide+ Provider system supports multiple instances of measurement servers, which are distributed over the globe. This section will focus on the server infrastructure and the measurement procedure.

Measurement server is an endpoint for connection quality measurements. It is running on the internet and waiting for measurement requests. When measurement request arrives, the server provides an interface to perform service-level quality measurements. Every server maintains a list of available protocols, which can be tested on it. This list is provided to the client prior to the measurement. In the scope of this masters thesis, the current single-server implementation was extended to multi-server because of better performance described in Section 6.1.1.1.3. There will be also more than one measurement server, which will be deployed in different locations. They don't require to be under the control of the BonaFide project. The project is open-source and everybody can run his own instance of measurement server.

Because measurement servers are a dynamic environment (they can appear and leave), a suiTable mechanism was implemented to use and utilize them in a maximal feasible way. Their role is to deliver bandwidth informations to the client application and thus they should offer the best possible data throughput to the client application to minimize routing delays between the measurement server and the client application. The mechanism for measurement server life-cycle management was described in the previous subsection.

Each measurement server is a standalone Java application. Java programming language was chosen because of its portability to multiple operating systems. Thanks to "Write software on one platform and run it on virtually any other platform" [15], the measurement server is not dependent from the server architecture nor from the operating system running on it. With additional libraries such as described later on the programming language is well suited to implement the measurement server including the custom communication protocol. Client application is implemented in Java as well because of the Java-limited support on Android devices, which leads to improved code-reusability across the client application and the measurement server, where especially the measurement protocol implementation is shared by common libraries between them.

According to the infrastructure diagram on Figure 6.2 the measurement server communicates with the central server and accepts connections from clients. REST interface is used for communication with the central server and custom protocol for communication with clients. The custom protocol features the measurement interface and thus REST was not applicable. The custom protocol was initially implemented by Vitali Bashko at Jacobs University of Bremen [20] and was adapted and used in the new BonaFide+ Provider system.

6.1.1.2.1 REST Implementation

Because RAW socket data exchange between the measurement server and the central server would be too much overhead and because the central server already provides REST communication interface, which is suiTable for the required data exchange between that two parties, the measurement server communicates with the central server via REST services. Measurement servers consume service specified in Table 6.4 for their advertisement. Measurement servers contain notification service, which takes care of telling the central server about their presence on internet and about their availability to measurement clients, including the initial IP address, port, name and location. The service sends periodic updates to the central server in order to refresh its availability as described in Subsection 6.1.1.1.1 about the measurement server life-cycle management in the central server. Updates are sent at:

- the measurement server start,
- at the time p according to Figure 6.3.

The period of keep-alive messages p is received from the central server at each REST service call. Thanks to this implementation the central server dynamically manages the delay between updates and the client is supposed to follow it.

While researching the technology EJB, which was supposed as candidate for the central server implementation, the framework Jersey [13] was part of the research, which features *RESTful Web Services in Java* and which is used according to the documentation also in the GlassFish Application Server [12] for REST services implementation. Jersey can be used for both REST server and REST client implementation. Because measurement servers only consume such services as described in Figure 6.2, Jersey API was chosen for the client-side REST implementation side [14]. The notification service is based on Jersey in addition with Jackson [9] extension for JSON object serialization and deserialization. This technology allowed me to work in high-level abstraction environment and without the need of RAW socket management, generating JSON strings etc. This is also a good example for the REST independence and portability. Despite the fact that the REST provider is implemented with PHP, there is no limitation for other programming languages, i.e. I avoided the single-technology-lock-in. The measurement server is implemented with Java and can consume the standardized and documented REST interface of the central server. This will be essential for future system extensions as well, e.g. by implementing an iPhone application, which require implementation in different programming language than PHP or Java.

6.1.1.2.2 Measurement Protocol

Measurement servers provide custom communication interface for performing measurements. This protocol was developed by Vitali Bashko and it is documented in his master thesis [20]. The BonaFide+ Provider system uses the same protocol implementation with a new extension for **measuring latency**.

Latency is measured after each measurement when the wireless connection is already established (connection initialization as described in [33] doesn't negatively affect the measured latency). The implementation on the client measures time of the following procedure:

1. Establish a TCP socket connection to the measurement server.
2. Send *"PING"* message as a string to the measurement server.
3. Receive *"PONG"* message as a string from the measurement server.
4. Close the TCP connection.

6.1.1.3 Client Application

The client application is the part of the system which will be in the hand of the end-user. Since the data collection will rely on the application use, one of the biggest challenges was to make the application user-friendly. Central server and measurement servers are used by the client application as shown in Figure 6.2. The client application was implemented for Android operating system version >= 4.0 and it can be installed directly from the Google Play store: `https://play.google.com/store/apps/details?id=de.jacobs.university.cnds.BonaFide.plus`. This makes distribution and update process easy.

6.1.1.3.1 Infrastructure Integration

The application uses the central server as a *bootstrap server*, i.e. it is the first place which is contacted when the client application is started. Because the BonaFide+ provider system is an open-source software project, everybody can set up own central server and thus the URL pointing to the central server REST interface is configurable directly in application settings (Figure 6.8), i.e. without the need of recompiling the application. The communication uses the HTTPS secured REST interface of the central server for communication. The implementation on the client side uses Jersey and Gson library. Jersey is also used by measurement servers for communication with the central server, however running it on the Android platform caused an application crash by NullPointerException[4] and thus a workaround had to be implemented. To get Jersey working on Android the ServiceIteratorProvider had to be extended as shown in Listing 6.8 and used instead of the default Jersey implementation by calling *ServiceFinder.setIteratorProvider(new AndroidServiceIteratorProvider());*. With this workaround Jersey was working as usual.

For **object-JSON (de)serialization** the library Gson was used [10] instead of the library Jaskson, which was used in the measurement server implementation because of missing SAX annotation support on the Android platform. This library is able to easily convert objects to JSON representation and vice-versa and thus it fully replaces the Jackson library. Object data fields are automatically serialized into JSON and the name of the variable is used as a key in the JSON output. This behavior can be changed by annotating the appropriate data field with *@SerializedName("another_name").* In combination with Jersey everything was ready to implement the REST service client for communication with the central server.

Client application initially is not aware of any measurement servers available in the system. They are managed by the central server and provided to the client application based on the particular intention. As described in Subsection 6.1.1.1.3 the central server is able to sort list of available measurement servers by their distance from the client in order to minimize routing delays when performing measurements. Measurement servers are sorted by the distance when the client application provides its own location to the central server. If the location is not available, random measurement server is chosen.

[4]This behavior is known also on the community Q&A website http://stackoverflow.com/questions/9342506/jersey-client-on-android-nullpointerexception

```
/**
 * This class is workaround to make Jersey-client work on
    Android. It extends and fixes the default
    ServiceIteratorProvider
 */

package de.jacobs.university.cnds.bonafide.plus.rest;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;

import android.util.Log;

import com.sun.jersey.spi.service.ServiceFinder.
   ServiceIteratorProvider;

public class AndroidServiceIteratorProvider<T> extends
   ServiceIteratorProvider<T> {

  private static final String TAG =
     AndroidServiceIteratorProvider.class.getSimpleName();
  private static final String MESSAGE = "Unable to load
     provider";

  private static final HashMap<String, String[]> SERVICES =
     new HashMap<String, String[]>();

  private static final String[]
     com_sun_jersey_spi_HeaderDelegateProvider = {
      "com.sun.jersey.core.impl.provider.header.
         MediaTypeProvider",
      "com.sun.jersey.core.impl.provider.header.
         StringProvider" };

  private static final String[]
     com_sun_jersey_spi_inject_InjectableProvider = {};

  private static final String[]
     javax_ws_rs_ext_MessageBodyReader = {
      "com.sun.jersey.core.impl.provider.entity.
         StringProvider",
      "com.sun.jersey.core.impl.provider.entity.
         ReaderProvider" };

  private static final String[]
```

```
    javax_ws_rs_ext_MessageBodyWriter = {
      "com.sun.jersey.core.impl.provider.entity.
         StringProvider",
      "com.sun.jersey.core.impl.provider.entity.
         ReaderProvider" };

static {
  SERVICES.put("com.sun.jersey.spi.HeaderDelegateProvider",
       com_sun_jersey_spi_HeaderDelegateProvider);
  SERVICES.put("com.sun.jersey.spi.inject.
     InjectableProvider",
       com_sun_jersey_spi_inject_InjectableProvider);
  SERVICES.put("javax.ws.rs.ext.MessageBodyReader",
     javax_ws_rs_ext_MessageBodyReader);
  SERVICES.put("javax.ws.rs.ext.MessageBodyWriter",
     javax_ws_rs_ext_MessageBodyWriter);
  SERVICES.put("jersey-client-components", new String[] {})
     ;
  SERVICES.put("com.sun.jersey.client.proxy.
     ViewProxyProvider", new String[] {});
}

@SuppressWarnings("unchecked")
@Override
public Iterator<Class<T>> createClassIterator(Class<T>
   service, String serviceName,
    ClassLoader loader, boolean ignoreOnClassNotFound) {

  String[] classesNames = SERVICES.get(serviceName);
  int length = classesNames.length;
  ArrayList<Class<T>> classes = new ArrayList<Class<T>>(
     length);
  for (int i = 0; i < length; i++) {
    try {
      classes.add((Class<T>) Class.forName(classesNames[i])
         );
    } catch (ClassNotFoundException e) {
      Log.v(TAG, MESSAGE, e);
    }
  }
  return classes.iterator();
}

@Override
public Iterator<T> createIterator(Class<T> service, String
   serviceName, ClassLoader loader,
    boolean ignoreOnClassNotFound) {
```

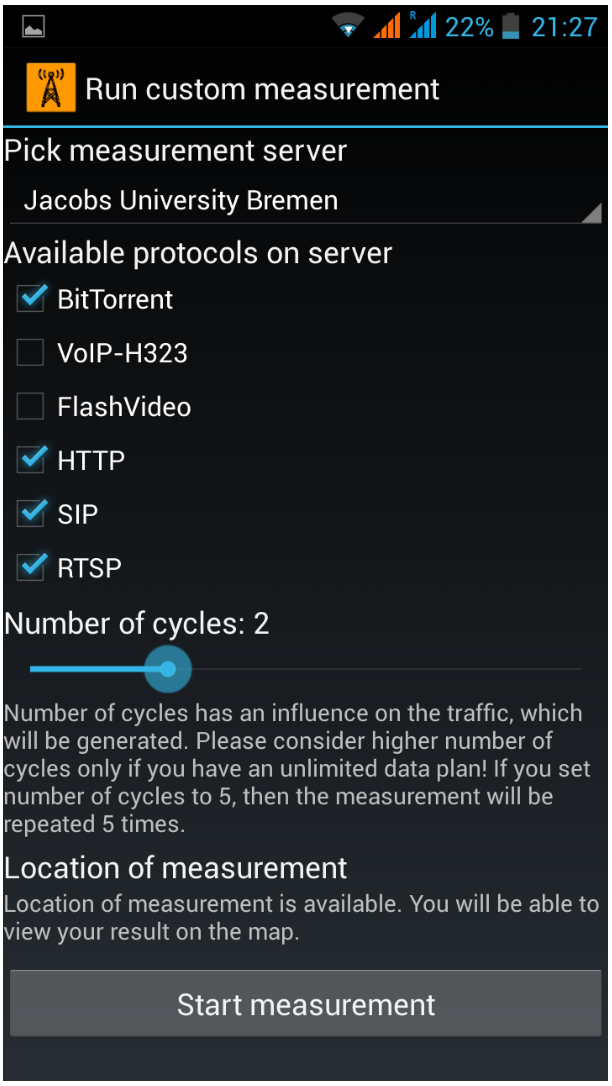
```
    String[] classesNames = SERVICES.get(serviceName);
    int length = classesNames.length;
    ArrayList<T> classes = new ArrayList<T>(length);
    for (int i = 0; i < length; i++) {
      try {
        classes.add(service.cast(Class.forName(classesNames[i
          ]).newInstance()));
      } catch (IllegalAccessException e) {
        Log.v(TAG, MESSAGE, e);
      } catch (InstantiationException e) {
        Log.v(TAG, MESSAGE, e);
      } catch (ClassNotFoundException e) {
        Log.v(TAG, MESSAGE, e);
      }
    }

    return classes.iterator();
  }
}
```

Listing 6.8: Workaround implementation of ServiceIteratorProvider for Jersey on Android

6.1.1.3.2 Ways of Measurements

The client application provides two ways for performing network quality measurements. The most advanced way, where the end-user can set additional configuration, is the **custom measurement**. Custom measurements let the user choose the measurement server from list of available measurement servers, protocol(s) and number of cycles (i.e. how many times the measurement should be repeated). User decides when to start such measurement. User interface for configuration of custom measurement is shown in Figure 6.7.

Second option for performing measurements is an automatic measurement service. This service is running in the background and according to application settings measurements are automatically started without users interaction. This option helps to collect measurement results over longer time and without the need of manually starting measurement results. Choice of measurement server is not controlled by the user and is optimized by the *nearest measurement server* algorithm described in Subsection 6.1.1.1.3. The service tries to pick the nearest measurement server and thus avoids routing delays in order to make the measurement as exact as possible. The service performs one measurement cycle for all available protocols on the chosen measurement server. While the measurement is running, the service prevents the device from going into the sleep state [4], because this would interrupt the measurement process. Once the measurement is done, sleep mode is enabled again.


22% 21:27

Run custom measurement

Pick measurement server

Jacobs University Bremen

Available protocols on server

- [✓] BitTorrent
- [ ] VoIP-H323
- [ ] FlashVideo
- [✓] HTTP
- [✓] SIP
- [✓] RTSP

Number of cycles: 2

Number of cycles has an influence on the traffic, which will be generated. Please consider higher number of cycles only if you have an unlimited data plan! If you set number of cycles to 5, then the measurement will be repeated 5 times.

Location of measurement

Location of measurement is available. You will be able to view your result on the map.

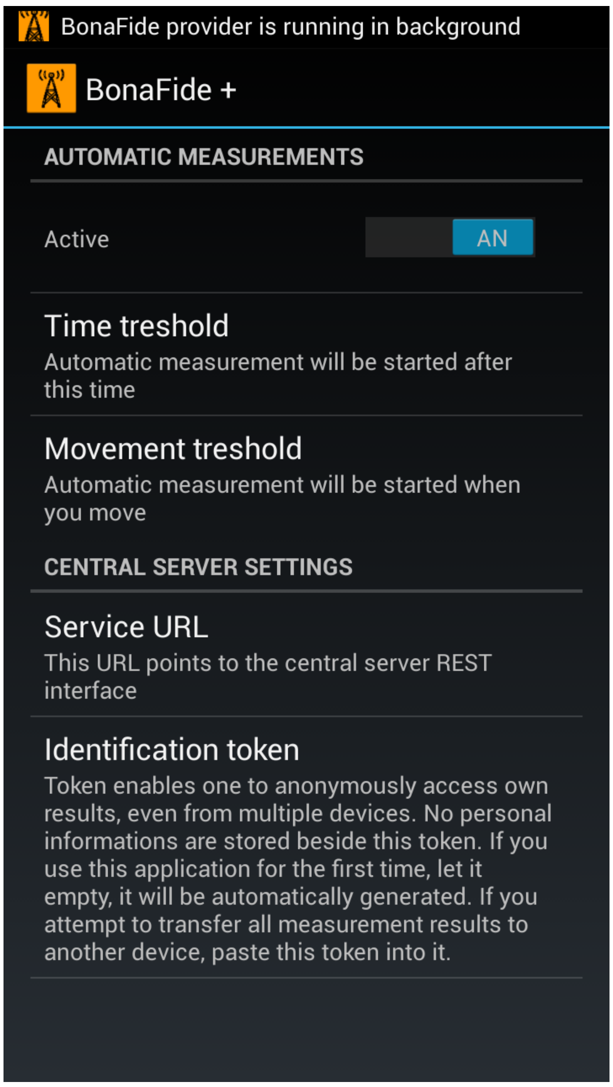
Figure 6.7: Custom measurement

Automatic measurements can be conFigured by application settings as shown in Figure 6.8. There are two parameters which have influence on how automatic measurements will be performed:

- Time threshold,
- Movement threshold.

Time treshold is the delay between measurements in minutes. Setting it to e.g. 10 minutes, the service performs measurement every 10 minutes (even if the device is in sleep state). This option is useful when one would like to collect measurement results in one place over time. If the end-user would like to measure quality of a particular area, the time threshold may be not the best option, because only time counts and not the location change. For this purpose the second threshold would be preferred. **Movement threshold** defines the distance from the last measurement in meters. Setting this threshold to e.g. 300, the service will start measurement every 300 meters of movement. This option is preferred for end-users who are moving and collecting measurement results with respect to locations on the path of movement.

Both thresholds are OR-combined, i.e. measurement is started after the specified time threshold counted from the last measurement or when the location of the device changes and the distance between the current location and the location of the last measurement is longer or equal than the specified movement threshold. Setting both thresholds to appropriate values makes the automatic measurement service well suited for collecting both - static and dynamic measurements.


BonaFide provider is running in background

BonaFide +

AUTOMATIC MEASUREMENTS

Active AN

Time treshold
Automatic measurement will be started after this time

Movement treshold
Automatic measurement will be started when you move

CENTRAL SERVER SETTINGS

Service URL
This URL points to the central server REST interface

Identification token
Token enables one to anonymously access own results, even from multiple devices. No personal informations are stored beside this token. If you use this application for the first time, let it empty, it will be automatically generated. If you attempt to transfer all measurement results to another device, paste this token into it.

Figure 6.8: BonaFide+ application settings

The client application is also able to measure performance of wireless local area networks (WiFi), however results may not be accurate due to throughput limitation of the particular WiFi technology. In case the internet connection is faster than the WiFi connection between the client device and the wireless access point, measuring performance of such network would be misrepresented, e.g. the statement about bandwidth would be affected by the WiFi throughput and the latency would reflect the real internet performance. Because of such limitations, using WiFi networks is not the preferred connection type, however it is supported by the application and results are denoted by the origin. When both connection types - mobile and WiFi - are turned on on the device in parallel, the WiFi connection is used by the device and thus such measurements are classified as measurement of WiFi network.

When using WiFi networks for performing measurements, there was another problem observed while testing the application. Android devices don't work well with WiFi networks in roaming setup [32], i.e. when multiple WiFi access points with the same SSID are overlapping. The device was randomly changing association from one access point to another one followed by interruption of each established connection. This caused interruption of the measurement process and thus measurements were not always finished correctly. An example of such case is shown in Figure 6.9.

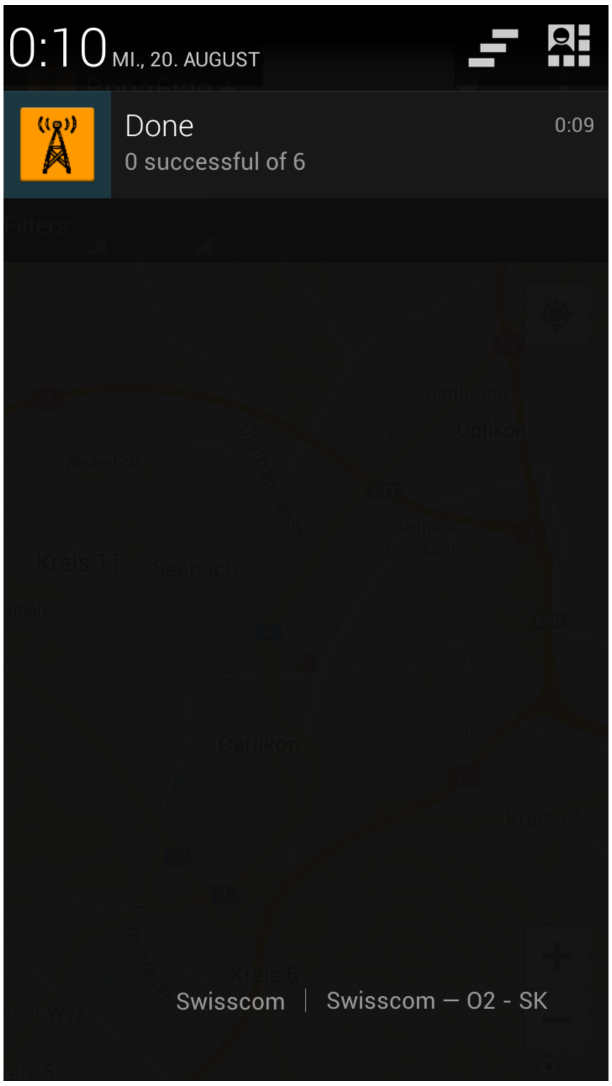
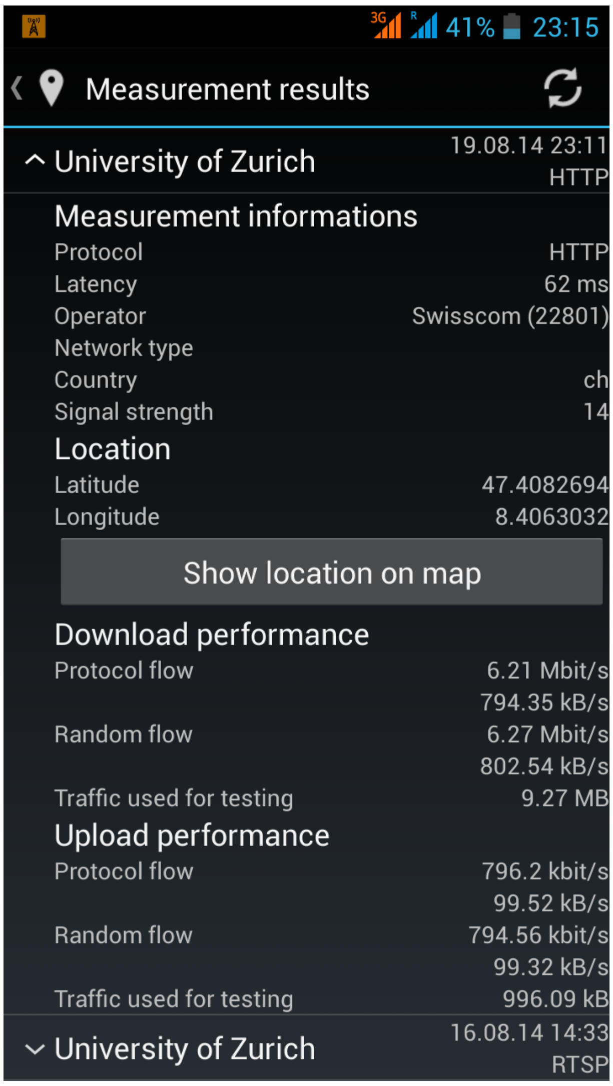


Figure 6.9: Problem with Android and roamed WiFi networks

6.1.1.3.3 Measurement Results

Each user has a detailed overview of own measurement results in the application, including unlimited measurement history. Figure 6.10 shows an example for a detailed measurement result.


41% 23:15

Measurement results

University of Zurich 19.08.14 23:11 HTTP

Measurement informations

Protocol	HTTP
Latency	62 ms
Operator	Swisscom (22801)
Network type	
Country	ch
Signal strength	14

Location

Latitude	47.4082694
Longitude	8.4063032

Show location on map

Download performance

Protocol flow	6.21 Mbit/s 794.35 kB/s
Random flow	6.27 Mbit/s 802.54 kB/s
Traffic used for testing	9.27 MB

Upload performance

Protocol flow	796.2 kbit/s 99.52 kB/s
Random flow	794.56 kbit/s 99.32 kB/s
Traffic used for testing	996.09 kB

University of Zurich 16.08.14 14:33 RTSP

Figure 6.10: Detailed measurement results

6.1.1.3.4 Overall Results Visualization

In addition to the detailed measurement overview, which is only available to the user of origin, there is an overall statistic overview of all measurements performed by the BonaFide+ provider system.

Data visualization is an important approach in the BonaFide+ provider project, because it forms the basis of measurement understanding for the end user. Different types of data listed in Table 6.8 are subject of visualization and every type has different characteristics.

Focus	**Type of data**
Location of measured data	geographic
Operator	enumeration
Mobile technology used	enumeration
Protocol/ Service	enumeration
Call plan type (pre-paid/subscription)	enumeration
Time of data input	temporal

Table 6.8: Data types used in BonaFide+ provider

To be able to visualize such data, an appropriate method had to be chosen for each of them. While designing this part of the client application, following challenges were considered:

- different data types - some of them intersected and some of them disjoint - needs to be visualized clearly and without confusing the user - filtering may help to change focus to e.g. location, provider, technology etc.,
- users need to understand the data - a good visualization method have to be chosen.

Figure 6.10 shows how detailed overview is presented to the user. However this overview is not applicable for visualizing geographical data and/or for showing aggregated results collected by many users. For such purpose another presentation was designed.

Because of combination of different data types and because of anonymization of single results, a combined presentation interface was intended. The design was influenced by the fact that measurements are bound to particular locations, which can be efficiently visualized on a map. Map was taken as the underlaying layer for the visualization. The map area was available for placing another data on top if it while keeping the privacy issue of obfuscating single measurements.

Each position is bound to a single location given by latitude and longitude. On the map this could be visualized by a point marker as shown in Figure 6.4. However keeping limited resources of mobile devices in mind, the amount of shown markers displayed on the map should be minimized. This has also an usability reason, because many overlapping markers would make them unreadable. From the privacy point of view, single locations shouldn't be displayed either, because they can reveal single location such as companies, flats etc. Considering everything written in this paragraph, an **aggregated method of displaying results** was developed.

Single results, i.e. points, are aggregated into larger units, which save resources available on the device and also obfuscate single measurement locations. The unit for displaying aggregated results is a **square** - the screen is filled with fixed amount of squares based on its resolution, which divide the screen into a grid. Each square is just a placeholder at the beginning and is not visible to the user. The grid is bound to the screen and not to the map. This is different in similar tools such as Netradar, where squares are bound to the map, i.e. moving the map moves also squares (squares are rendered to the same position on the map). BonaFide+ provider doesn't have any fixation to the map. Once the screen is divided into the grid, the grid doesn't move with the map movement. When the underlaying map is moved, squares stays at the same position on the screen and only the area below them changes.

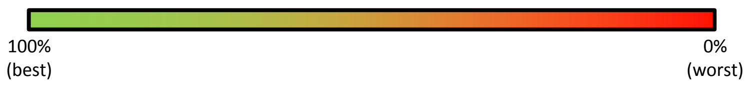


Figure 6.11: Quality gradient for coloring squares

Each square aggregates results below it. If there is no result under the particular square, the square is not displayed on the map to indicate that the area is not covered by any measurements. If there are more that zero measurement results under the area of the square, the square becomes visible. Each square is displayed with 50% alpha value to make also the underlaying area (map) visible. Each displayed square has a color, which indicate the quality of the underlaying area. The quality can be e.g. download protocol bandwidth or upload random bandwidth. The focus can be changed by filters which will be described later. The color is computed based on the average value of all results located below it. The color is red (worst quality), green (best quality) or something between as shown in Figure 6.11. The quality is represented in percentage and the color is picked from the gradient based on the computed quality average. An example for everything described in this paragraph is shown in Figure 6.12.

The average of each square is computed by a special algorithm developed in the scope of this master thesis. Because squares are bound to the display, the user has flexibility to adjust the visualization by moving the map. After the map is moved, color of each square is recalculated based on the new are below it. Until now there was only the expression *average*, but this is not enough to decide which color the square should contain. For that the **minimum** and **maximum** is needed in order to decide about the percentage (percentage is mapped to a particular color between red and green). The minimum value available in the results database will be the value of 0% and the maximum the value of 100%. Then the average for each square will be between 0% and 100% and its color can be determined based on the quality gradient shown in Figure 6.11. In order to add another flexibility to the visualization, the minimum and maximum is always computed for the current map viewport. Map viewport is the area on the map which is currently displayed on the screen. This means that the minimum and maximum is always adapted to the shown map area and to the current zoom level. This approach enables focusing into areas without taking the rest of the world into charge when computing the quality of each square. Regions, which are not displayed in the current viewport, are excluded from the minimum-maximum-computation and thus colors of squares reflect the relative quality based on the minimum and maximum measured in the displayed area.

Figure 6.13 combines view of the viewport, map and of the square orientation.

- green area shows the map viewport,

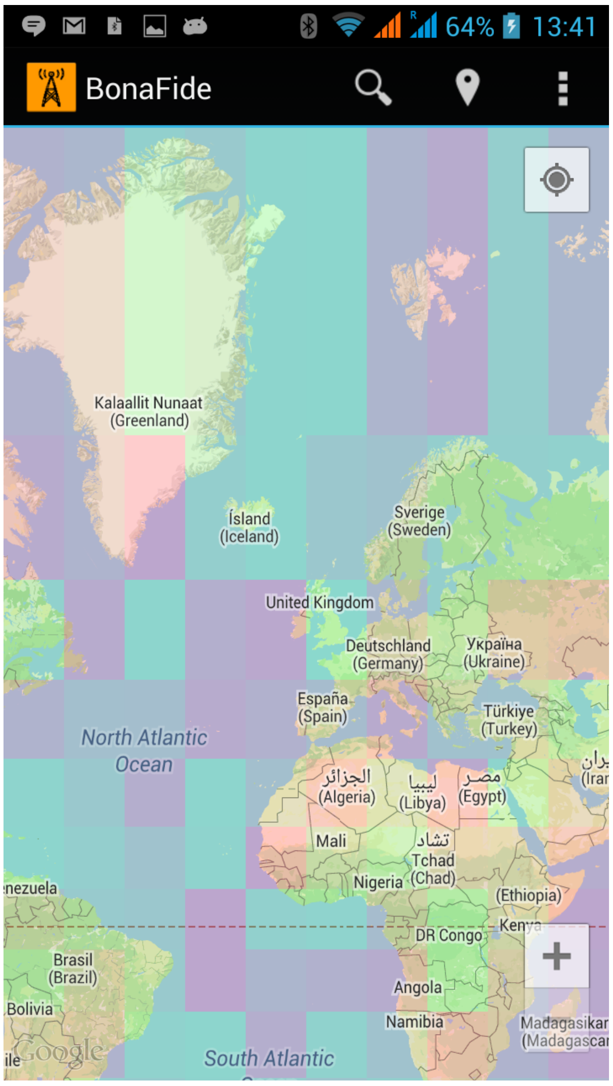


Figure 6.12: Example for rendering of squares

- red area shows the square,
- black arrows and compass show the orientation on the map.

All three views are combined into a single visualization. Map viewport and squares are defined by two points given by coordinates:

- **NE** point stands for north-east,
- **SW** stands for south-west.

Given that two points the rectangle can be reconstructed. NE and SW points of the map viewport are input for the square calculation. Squares are calculated by the central server and rendered by the application given square definitions returned from the central server. The definition contains list of squares and each square is defined by its NE and SW points and by the quality in percents (0-100). The client application then renders all squares

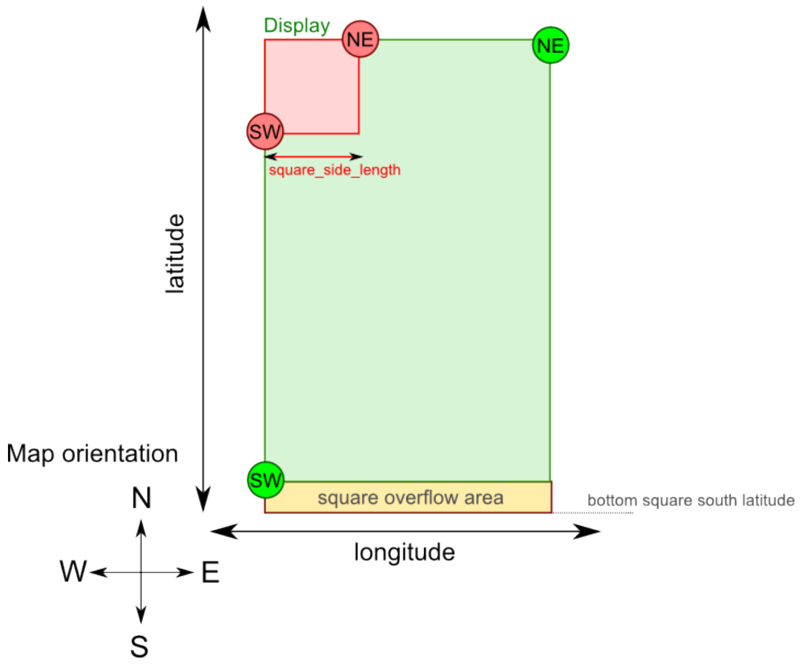


Figure 6.13: Viewport, map and square orientation

defined by the central server for the given viewport. The maximum amount of squares rendered on the screen is defined by the grid. Squares are rendered over the map given coordinates of their NE and SW points. When the underlaying map moves, each square is removed and redrawn according to the definition. The central server knows the width and height of the viewport based on its coordinates. Given the number of squares in the horizontal space by the central server configuration, the central server computes NE and SW points for each square within the viewport. The number of horizontal squares is given by the configuration and can be computed to fill exactly the whole width of the viewport and the number of vertical squares is computed based on the height of the viewport in order to fill the whole screen with squares. Because squares have the same side length (*square_side_length*), it is not always possible to exactly fill the area of the viewport. The yellow *square overflow area* illustrates the area below the viewport, into which the bottom square line may reach. However, the performance is not significantly affected by that and the user has the feeling that the whole display is covered by squares (no empty space is seen at the bottom of the viewport).

The average for each square is queried from the database. The geometry class Polygon [6] is used for selecting results located in the square. An example query is shown in Listing 6.9. This selection is indexed and thus the performance is not significantly affected even when selecting within a huge results space. The DBMS then computes the average value for the requested parameter, e.g. the upload random bandwidth.

```
SELECT count(*) AS aggregated_count, AVG(".$requested_column.
   ") AS ".$requested_column."_avg FROM ".Sql::$db_prefix."
   measurement_results WHERE within(measurement_location,
   GeomFromText('POLYGON((".$square->south_west_latitude."␣".
   $square->south_west_longitude.",".$square->
south_west_latitude."␣".$square->north_east_longitude.",".
$square->north_east_latitude."␣".$square->
north_east_longitude.",".$square->north_east_latitude."␣".
$square->south_west_longitude.",".$square->
south_west_latitude."␣".$square->south_west_longitude."))'
) ) AND ".$requested_column.">0".
$applied_filters_query_injection
```

Listing 6.9: Example query withing the PHP code for average computation for a particular square

6.1.1.3.5 Mean Opinion Score and DQX Model for QoE Estimation

In addition to average computations, the application features also the QoE estimation. Average data are numeric values and they are missing the interpretation. QoE estimation turns them into interpreted expressions such as *at this places the web browsing performs excellent*. For that the same visualization technique is used as for the average visualization, but squares aren't colored by the average anymore, but by the QoE value. Having this, the end-user can interpret colors as the statement about the expected experience for the particular purpose (currently implemented are: webbrowsing, IP telephony and video streaming).

QoE combines different measured values in order to make the statement about the experience as close as applicable. Each application has its specific properties and thus the formula needs to be adjusted for each of them. Table 6.9 shows how the MOS computation is configured in the BonaFide+ Provider system. MOS is computed by the central server and parameters can be adjusted in the configuration file of the central server.

Service	**Protocol(s)**	**Effect**	**Parameter**	**min**	**max**	x_0	$MOS=3$	$MOS=5$	m^-	m^+	w_k
Browsing	HTTP	increasing	downlink throughput	0 $Mbps$	50.1 $Mbps$	1330 $Kbps$	−25%	+100%	2.41	2.58	75%
		decreasing	latency	1 ms	2000 ms	523 ms	+15%	−50%	10.17	6.29	25%
Video	FlashVideo	increasing	downlink throughput	0 $Mbps$	50.1 $Mbps$	1.5 $Mbps$ 480p	−20%	5 $Mbps$ 720p	3.11	1.49	100%
VoIP	VoIP-H323	increasing	uplink throughput	0 $Mbps$	12.7 $Mbps$	8 $Kbps$	5.3 $Kbps$	64 $Kbps$	1.67	0.86	25%
		increasing	downlink throughput	0 $Mbps$	50.1 $Mbps$	8 $Kbps$	5.3 $Kbps$	64 $Kbps$	1.67	0.86	25%
		decreasing	latency	1 ms	2000 ms	120 ms	+50%	−50%	10.17	2.16	50%

Table 6.9: MOS-related Values [38]

6.1.1.3.6 Filtering

In order to view only relevant data on the map, the client application implements two types of filters located at the top of the view:

- **Scope** - which parameter should be averaged and rendered on the map

- **Filter** - which operators, countries, protocols etc. should be contained in the average computation

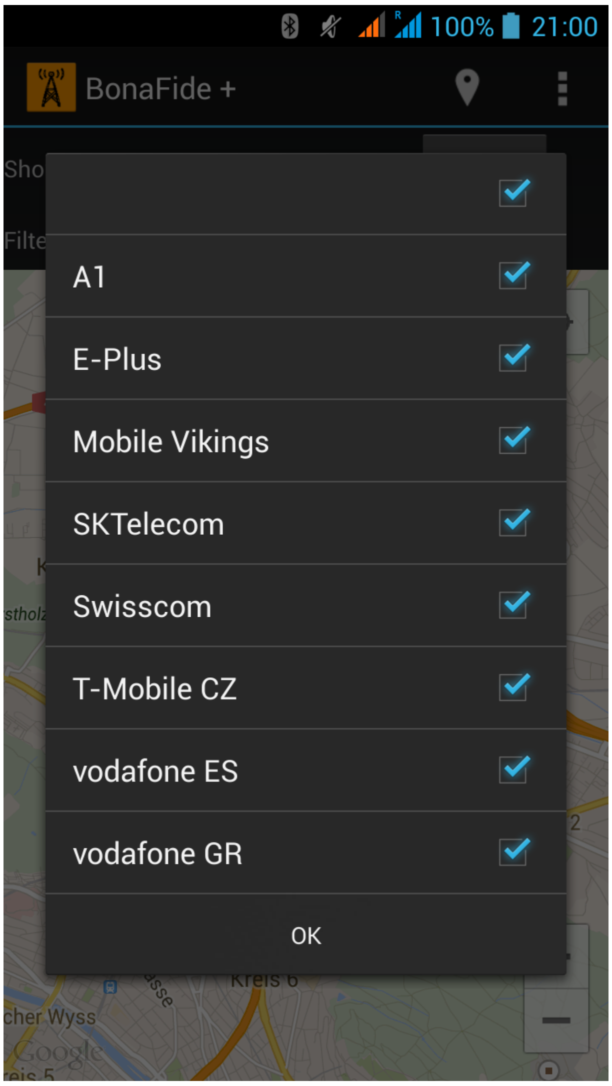
The scope switch decides about the target scope, i.e. what should be visualized in the map. Currently the application supports following target scopes:

- MOS webbrowsing (QoE relevant),
- MOS IP telephony (QoE relevant),
- MOS video streaming (QoE relevant),
- download protocol bandwidth,
- download random bandwidth,
- upload protocol bandwidth,
- upload random bandwidth,
- latency,
- signal strength.

Each target scope can be combined with an appropriate filter. Currently supported filters are:

- protocol,
- network type,
- operator,
- country.

Filters contain additional filtering parameters as shown in Figure 6.14. The list of parameters depends from available results, i.e. unique values from the appropriate database column are selected and displayed. Then only checked (in this case) operators will be applied in the average or MOS computation. Filter can be turned on and off via the switch located at the top right side.



✓
A1 ✓
E-Plus ✓
Mobile Vikings ✓
SKTelecom ✓
Swisscom ✓
T-Mobile CZ ✓
vodafone ES ✓
vodafone GR ✓

Figure 6.14: Filtering operators

6.1.1.3.7 Squares and Map Projection

For the map presentation Google Maps API v2 is used. It is offered as a free service, it can be directly implemented in Android applications and supports shape rendering on top of the map layer [11] as required by the project. Polygon shapes are used for rendering quality squares. Color of each polygon is defined in ARGB format to support the alpha channel, which is required for rendering the square with semi-transparency.

Redrawing of results is initiated by the movement of the map. Google Maps API v2 however don't implement an appropriate listener for *on map idle* event but required by the project, so custom implementation was done in form of a custom listener. This custom listener extends the basic *onMapChange* listener by the *onMapIdle* listener, which is not available by default in the current Google Maps version. This listener ignores frequent immediate position updates caused by scrolling the map and fires onMapIdle event after the position is fixed and not changed for a specified piece of time, which is currently set to 500ms.

Figure 6.12 shows rendered squares, however they appear not to be squares. Squares in the context of the BonaFide+ provider are rendered by coordinates and not by pixels on the screen. The *square_side_length* is always the same for all squares rendered on the map and it is represented by Δ latitude and Δ longitude (Δ latitude=Δ longitude). But as shown in the Figure, they are rendered as rectangles. This is not bug in the application, but squares are distorted because of the map projection. Despite the fact that they appear to be rectangles, they are squares on the real earth surface.

6.1.1.3.8 Location Awareness

The client application is location aware, i.e. it is aware of its current location. This is essential when making measurement results location-specific. Access to the location of the device is dependent from access restrictions defined on the device, but the application requests user to turn on location providers when the access is blocked. There are multiple methods for obtaining the location of the device according to [2]:

- GPS,
- cell towers,
- WiFi signals.

The application supports all available location strategies and uses a fused location provider, which combines input of different location providers available and turned-on in the device and supplies the application with the most accurate location. The fused provider is implemented by Google and its documentation can be found at the following link: `http://developer.android.com/google/play-services/location.html`. Using the fused location provider enables fast access to the location in the indoor space and also the most exact (GPS based) location in the outdoor space (i.e. where GPS is applicable). The application doesn't rely on a single location provider and can get access to the location even if only one provider is enabled.

Chapter 7

Evaluation

In the scope of this master thesis the whole infrastructure was set up and maintained. 29 users were participating on mobile measurements with the client application downloaded from Google Play and running on Android devices connected to internet via mobile data.

There were done 1657 measurements via mobile networks in total in following countries (countries were recognized based on the country of the connected MNO):

- Switzerland (719),
- Germany (410),
- Greece (8),
- Belgium (83),
- Korea (12),
- Czech Republic (357),
- Spain (12),
- *not available (56)*

Controlled measurements in Zurich were done in the city center as shown in figures. This area was crowded and different shapes were placed on the street which have an influence on the signal quality (Figure 7.1). The expectation was that the internet quality will be negatively influenced. During evaluation QoE was focused instead of QoS.

Measurement results for different QoE use-cases are shown in Figure 7.2. The expectation that the quality will be bad was wrong. Major areas performs excellent for all three use-cases. There are some variations caused by application-specific parametrization of QoE computation.

Measurement focusing disturbing elements was performed in area shown in Figure 7.3. When focusing QoE of video streaming, where high bandwidth is required, the QoE sinked

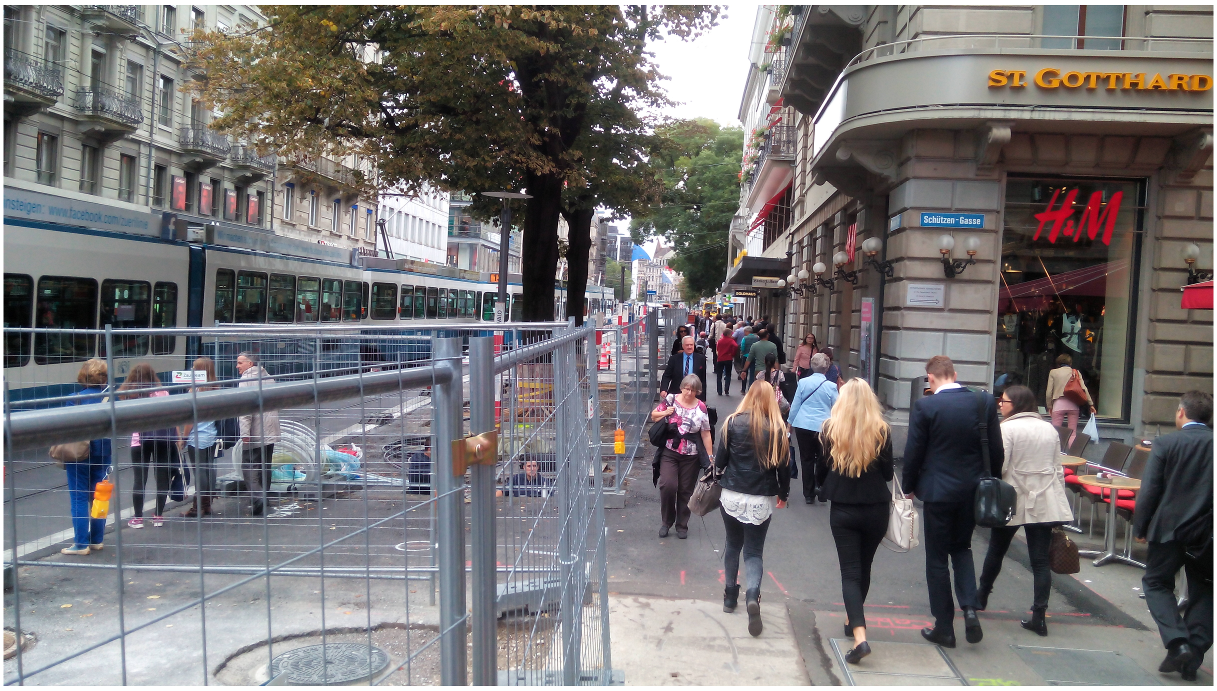


Figure 7.1: Zurich city center - Bahnhofstrasse

because the bandwidth was limited due to signal disturbance caused by the scaffold. However QoE of VoIP and web browsing was excellent in this area - i.e. QoS was apparently influenced by disturbance, but it was still excellent for making an internet call or for web browsing. This disturbance would be noticeable only when one would stream videos in this area.

Figure 7.4 shows QoE of video streaming in Switzerland. When zooming out, diffused measurement results are aggregated into bigger squares and thus small local deviations will become invisible when the average MOS for the particular square is computed.

Beside QoE interpretations also other parameters such as signal strength can be visualized as shown in Figure 7.5.

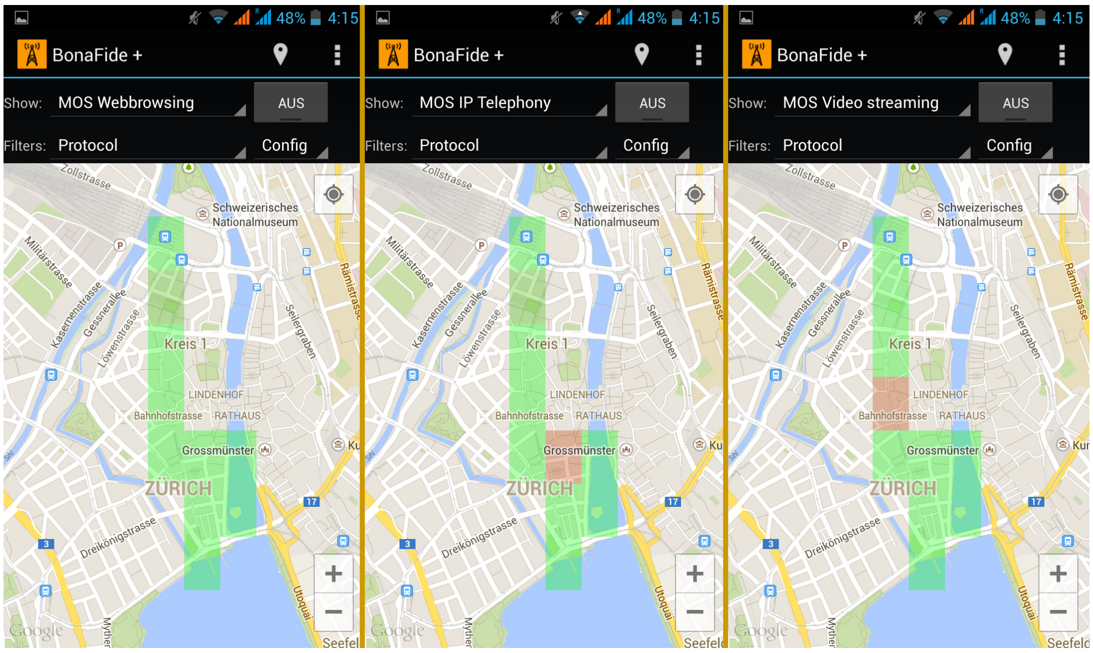


Figure 7.2: Zurich city center - QoS of different use-cases

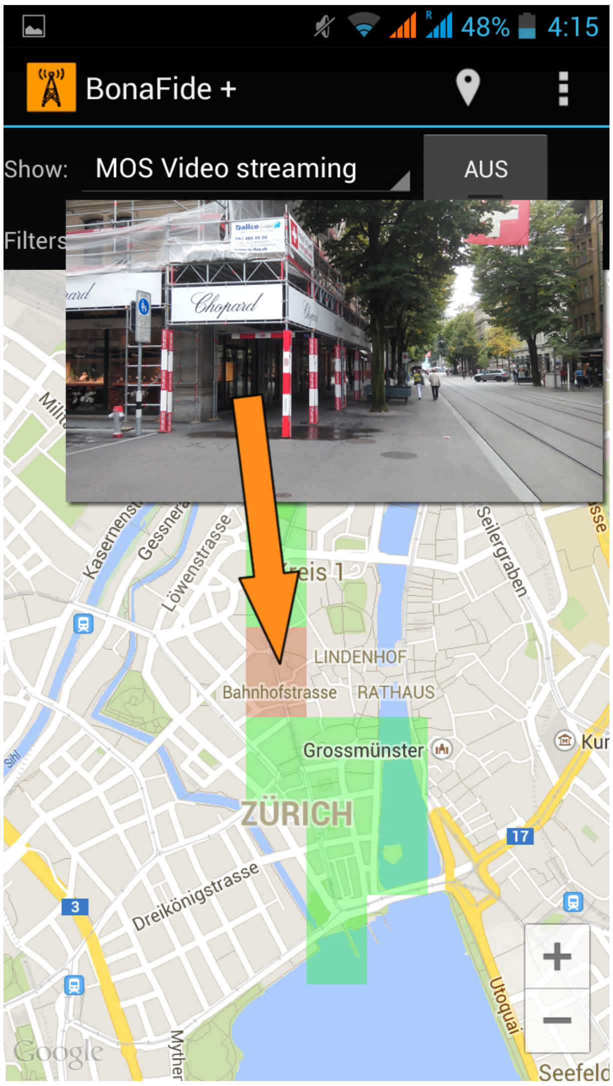


Figure 7.3: QoE for video streaming affected by disturbance element

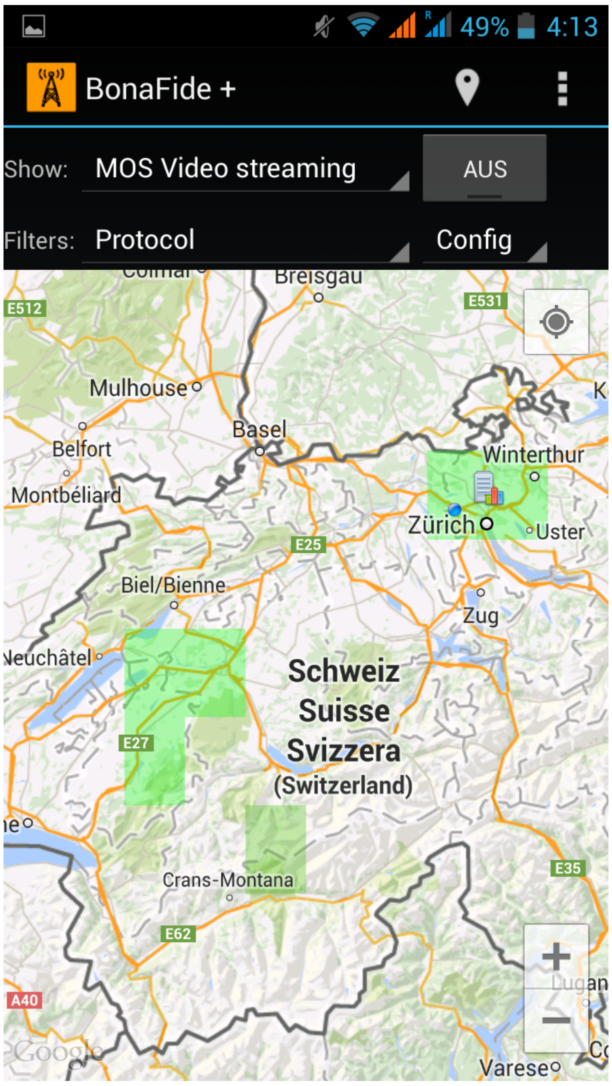


Figure 7.4: QoE of video streaming in Switzerland

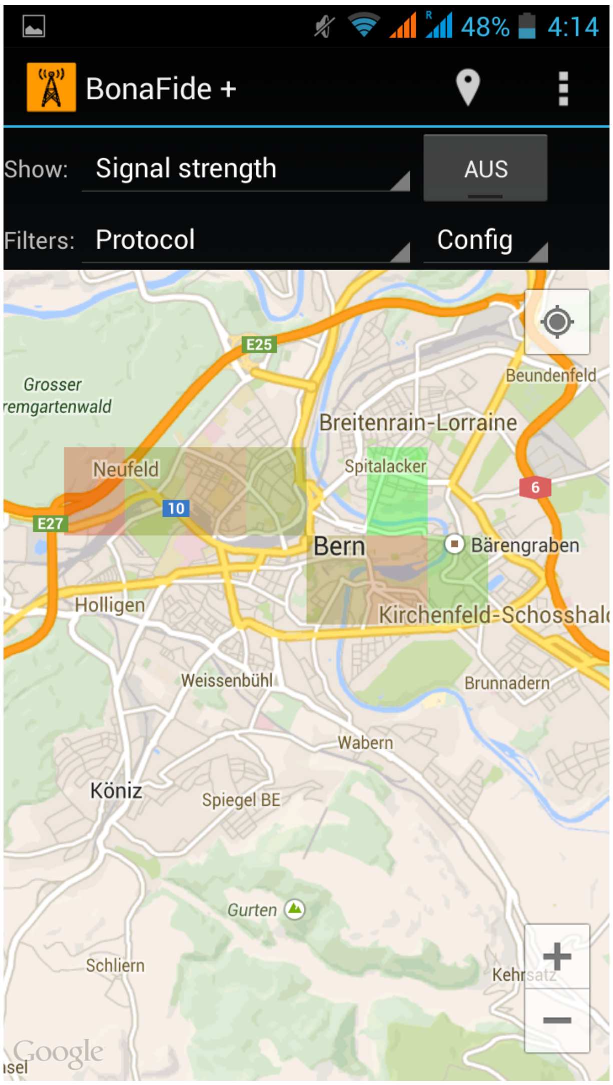


Figure 7.5: Signal strength in Bern

Chapter 8

Future Work

The BonaFide+ Provider system is open for everyone and can be freely extended. This Chapter opens discussion of the future direction of the system.

Integration of measurement servers into the infrastructure is currently not authenticated. Measurement servers can be easily configured and inserted into the system by just specifying the REST URL of the central server. Measurement servers are then automatically managed by the life-cycle management. The origin of measurement servers is not verified and thus also malicious measurement servers would be accepted, which could e.g. provide shaped bandwidth for measurements and thus measurements would be artificially incorrect. Verification is not easy in open infrastructures, because everybody is allowed to deploy a measurement server, however central server is not yet able to filter which measurement server is allowed to join and which not. This can be done similar to communication between WiFi Access Points (AP) and RADIUS servers, where a shared secret is used for AP verification. Measurement servers and central server can share a secret verification string, which can be compared when the measurement server tries to join the infrastructure. The communication between measurement servers and the central server is already encrypted and secured by HTTPS, which would make exchange of shared secret secure.

The connection to measurement servers is not checked when measurement servers join the infrastructure. They are automatically accepted without the guarantee that all necessary ports (i.e. the whole communication interface) are accessible on them. Without checking the reverse connection measurement servers can contain blocked ports and thus measurements could be limited. This reverse-connection check can be implemented on the central server at the point when measurement server tries to join the infrastructure. The IP address and service port is known to the central server and reverse connection can check if the measurement server is configured correctly. If not, measurement server shouldn't be accepted.

Measurement servers are sending periodic keep-alive messages to the central server in order to advertise themselves to the system. Central server responds with the delay value until the next keep-alive message. Measurement servers then follow this received delay instruction. This enables future extensions such as longer update periods for *known measurement servers* or *load balancing of keep-alive updates*. In the current implementation only constant delay is returned by the central server and thus each measurement server keeps sending keep-alive updates in fixed intervals.

Client application are getting assigned an identification token, however there is no security mechanism protecting the central server from issuing huge amount of identification tokens to malicious clients. Implementation of identification token counters and limitations for particular IP addresses would clear this risk.

Evaluation of the system showed a significant data usage with automatic measurements. This is one of major possibilities for improvements, since client applications are measuring mobile networks and thus are connected via mobile data, which may be expensive for end-users. During the evaluation 1GB was consumed within one hour. Automatic measurements perform measurements of all available protocols at the nearest measurement server. Each measurement includes redundant tasks such as random flow measurement used for comparing random and protocol flow, where random flow doesn't need to be measured with each protocol again and again. This would save a lot of data traffic without losing any measurement output. Currently each protocol measurement is handled as a separate task and automatic measurements are performing them one after each other. Adding the possibility to control their subtasks could enable automatic measurements to disable subtasks which are already done and not needed to be done for a second time.

Another important improvement is addition of statistical overview to each square. In the current implementation squares are only drawn and are not labeled nor equipped with pop-up informations. The reason is that Google Maps API v2 doesn't provide on-click listeners for polygons and thus implementing such functionality manually over the map onClick listener was out of scope of this master thesis. This onClick listener can provide screen coordinates of the click and an additional algorithm can be implemented for resolution of the square located in the area where the click was perceived. In addition an overlay information can be shown which would provide more details about the clicked area. Such data are already provided to client applications via the REST data model, but they need to be shown to the user.

Chapter 9

Summary and Conclusions

This master thesis contributed to the QoE research with the open-source distributed system BonaFide+ Provider for protocol-specific QoS measurements and QoE estimation. Estimating the experience of users in different use-cases and environmental conditions is completely new functionality, which was not implemented in any software system before.

BonaFide+ Provider can be used and extended in future works in the QoE direction. Given the ability to deploy the whole system on own infrastructure research groups can adapt QoE estimation parameters and freely experiment with the whole system.

List of Figures

List of Tables

Appendix A

Contents of the DVD

Attached DVD contains source code of the whole BonaFide+ Provider system, including the central server, measurement server and the client application.

Bibliography

[1] http://developer.android.com/guide/topics/location/strategies.html, last access: 29.8.2014.

[2] http://developer.android.com/guide/topics/location/strategies.html, last access: 29.8.2014.

[3] http://developer.android.com/guide/topics/sensors/sensors_overview.html, last access: 29.8.2014.

[4] http://developer.android.com/reference/android/os/PowerManager.html, last access: 29.8.2014.

[5] http://dev.mysql.com/doc/refman/5.0/en/gis-class-point.html, last access: 29.8.2014.

[6] http://dev.mysql.com/doc/refman/5.0/en/gis-class-polygon.html, last access: 29.8.2014.

[7] http://dev.mysql.com/doc/refman/5.6/en/innodb-foreign-key-constraints.html, last access: 29.8.2014.

[8] http://dev.mysql.com/doc/refman/5.6/en/spatial-relation-functions-object-shapes.html\#function_st-distance, last access: 29.8.2014.

[9] http://jackson.codehaus.org/, last access: 29.8.2014.

[10] https://code.google.com/p/google-gson/, last access: 29.8.2014.

[11] https://developers.google.com/maps/documentation/android/shapes, last access: 29.8.2014.

[12] https://glassfish.java.net/, last access: 29.8.2014.

[13] https://jersey.java.net/, last access: 29.8.2014.

[14] https://jersey.java.net/documentation/latest/client.html, last access: 29.8.2014.

[15] https://www.java.com/en/about/, last access: 29.8.2014.

[16] http://www.accedian.com/en/solutions/business-services.html, last access: 29.8.2014.

[17] `http://www.mysql.com/`, last access: 29.8.2014.

[18] `http://www.sdncentral.com/technology/role-of-l4-7-network-intelligence-sdn/2012/10/`, last access: 29.8.2014.

[19] Anurag Acharya and Joel Saltz. A study of internet round-trip delay. Technical report, University of Maryland, Computer Science Department, 1998.

[20] Vitali Bashko. Traffic differentiation detection in mobile networks using android phones. Master's thesis, Jacobs University, 2012.

[21] Vitali Bashko, Nikolay Melnikov, Anuj Sehgal, and Jürgen Schönwälder. Bonafide: A traffic shaping detection tool for mobile networks. In *IFIP/IEEE International Symposium on Integrated Network Management (IM-2013)*, 2013.

[22] Shigang Chen and Klara Nahrstedt. An overview of quality of service routing for next-generation high-speed networks: Problems and solutions, 1998.

[23] Johan De Vriendt, Danny De Vleeschauwer, and David Robinson. Model for estimating qoe of video delivered using http adaptive streaming. IFIP/IEEE IM2013.

[24] Bernardin Denzel, Heidi Heilman, Rolf M. Katzsch, Andreas Meier, Michael Moerike, and Heinz Sauerburger. *Mobilkommunikation.* Hüthig GmbH, Heidelberg, 1995.

[25] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns.* Addison-Wesley, Upper Saddle River, NJ, 2000.

[26] Bur Goode. Voice over internet protocol (voip). *Proceedings of the IEEE, vol. 90, no. 9*, 2002.

[27] TELECOMMUNICATION STANDARDIZATION SECTOR OF ITU. Methods for subjective determination of transmission quality, 08 1996.

[28] Kalevi Kilkki. Quality of experience in communications ecosystem. *Journal of Universal Computer Science*, 2008.

[29] Franz Lehner. *Mobile und drahtlose Informationssysteme.* Springer-Verlag, Berlin, 2003.

[30] Analysys Mason. Global mobile network traffic - a summary of recent trends. report, 2011.

[31] Sebastian Möller and Alexander Raake. *Quality of Experience.* Springer, Berlin, 2013.

[32] Julien Montavont and Thomas Noel. Ieee 802.11 handovers assisted by gps information. *Wireless and Mobile Computing, Networking and Communications*, pages 166–172, 2006.

[33] Inc. Motorola. Realistic lte performance - from peak rate to subscriber experience, 2009.

[34] Ralf Reichwald. *Mobile Kommunikation - Wertschöpfung, Technologien, neue Dienste*. Gabler Verlag, Wiesbaden, 2002.

[35] Flávio Ribeiro, Dinei Florencio, Cha Zhang, and Michael Seltzer. Crowdmos: An approach for crowdsourcing mean opinion score studies. *ICASSP*, 2011.

[36] Claudio Riva and Markku Laitkorpi. Designing web-based mobile services with rest. *Service-Oriented Computing - ICSOC 2007 Workshops, Springer Berlin Heidelberg*, pages 439–450, 2009.

[37] Manuel Rösch. Quality-of-experience measurement setup. Technical report, University of Zurich, Department of Informatics (IFI), 2014.

[38] Christos Tsiaras, Anuj Sehgal, Sebastian Seeber, Daniel Doenni, Burkhard Stiller, Jürgen Schönwälder, and Gabi Dreo Rodosek. Towards evaluating type of service related quality-of-experience on mobile networks. 7th IFIP Wireless and Mobile Networking Conference, 2014.

[39] Christos Tsiaras and Burkhard Stiller. A deterministic qoe formalization of user satisfaction demands (dqx).

[40] Alessandro Verdolini and Stefano Petrangeli. A smartphone agent for qoe evaluation and user classification over mobile networks. Fifth International Workshop on Quality of Multimedia Experience, 2013.