Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Networked Systems and Services

Performance Monitoring of Radio Access Networks for the Integration of New Network Features and Services

Scientific Students' Association Report

Author:

Balázs Sántha

Advisor:

Dr. Adrián Pekár
Márton Pósfay

2023

Contents

Kivonat

A jelen munka célja egy olyan kifinomult megoldás kialakítása, mely lehetővé teszi internetszolgáltatók számára, hogy objektíven mérjék és értékeljék új funkciók és szolgáltatások bevezetésének hatásait a rádió hozzáférési hálózatokban. Az új funkciók implementálása gyakran kiszámíthatatlan következményekkel járhat a meglévő szolgáltatásokra és a felhasználói élményre, ebből kifolyólag elengedhetetlen, hogy a bevezetésük előtt alaposan felmérjük ezeknek a változásoknak a hálózatra gyakorolt hatásait. Az implementált rendszer virtualizált konténerek használatával képes `iperf3` méréseket végezni, amelyek teljesen függetlenek egymástól. Ezen mérések révén a rendszer egy egységes, valós idejű felügyeleti felületet biztosít, mely segítségével a hálózati adminisztrátorok precíz és megbízható információkhoz juthatnak a hálózat teljesítményéről. Ezzel lehetővé válik számukra a hálózat megfelelő előkészítése, optimalizálása és finomhangolása az új funkciók/szolgáltatások integrálása érdekében, úgy hogy az ne befolyásolja negatívan a már meglévő funkciók működését és a felhasználói élményt. A rendszer használata hozzájárul a megnövekedett stabilitáshoz, megbízhatósághoz és teljesítményhez, ami javítja a felhasználói élményt, növeli a fogyasztói elégedettséget és hűséget. Ezen pozitív változások nem csupán a technológiai innováció szintjét emelik, de közvetetten gazdaságélénkítő hatással is bírnak, mivel ösztönzik a piaci versenyt és a technológiai fejlesztést.

Abstract

The aim of this study is to develop a refined solution that enables internet service providers to objectively measure and assess the effects of introducing new features and services in radio access networks. Implementing new features can often lead to unpredictable consequences on existing services and user experience, making it imperative to thoroughly evaluate these changes' impact on the network before their introduction. The implemented system uses virtualized containers to perform `iperf3` measurements, which are completely independent of each other. Through these measurements, the system provides a unified, real-time monitoring interface, allowing network administrators to access precise and reliable information about the network's performance. This enables them to adequately prepare, optimize, and fine-tune the network for the integration of new features/services so that they do not adversely affect the operation of existing features and the user experience. Utilizing this system contributes to increased stability, reliability and performance, improving user experience, boosting consumer satisfaction and loyalty. These positive changes not only elevate the level of technological innovation but also indirectly have an economic stimulatory effect as they spur market competition and technological advancement.

Chapter 1

Introduction

In a typical mobile telecommunication's system, the network consists of two main parts, the Radio Access Network (RAN) and the Core part. RAN is the main component of a wireless communication system. RAN is implementing a radio access technology (RAT), such as GSM, LTE or 5G NR to handle the radio traffic received from smartphones or any other devices and delivers it to the central core network [41]. Typically, mobile phones and other wireless connected devices are varyingly known as a user equipment (UE), terminal equipment (TE) or mobile station (MS). Figure 1.1 shows the relationship between RAN and Core networks.

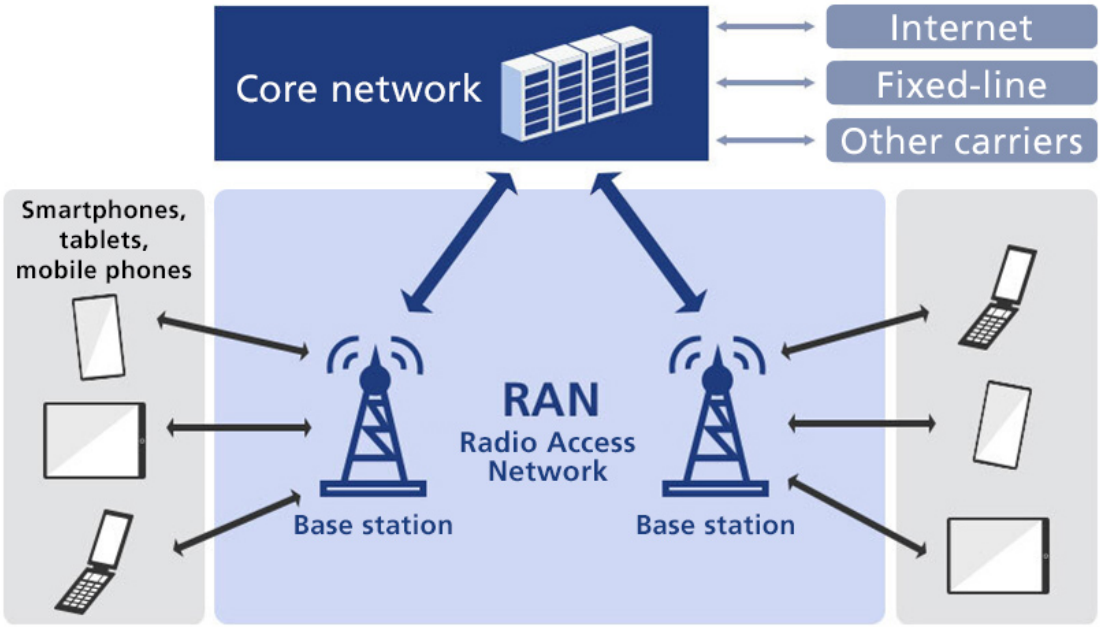

Figure 1.1: Relationship between RAN and Core networks

For most people, when hearing the TCP/IP stack, it refers to a fixed access situation, where a group of nodes in a telecommunication network are interconnected using wired connections such as Ethernet cables or fiber-optic links. In this wired framework, the communication medium is typically stable and less susceptible to external disturbances. Issues such as noise interference, signal attenuation, and communication failures are relatively less frequent and can often be more predictably managed.

This contrasts sharply with wireless communication systems, where the propagation medium is the air, and data is transferred using radio waves. In this setup, many external factors come into play, which can hinder the efficiency of communication. Factors like weather conditions, physical obstacles, electromagnetic interference from other devices, and even the movement of users can have a considerable impact. These challenges introduce variability and can result in fluctuating end-to-end network quality.

Given these dynamics in wireless radio systems, it becomes imperative to consistently monitor and measure network performance. When it comes to introduce and test new RAN functionalities it is essential for the RAN engineers, to have a stable way to reproducibly measure the radio network's Key Performance Indicators (KPI) which enables them to effectively identify the upcoming problems and difficulties.

In an average mobile cell there are sometimes hundreds of active users served at a time, who are constantly generating data up-link (from the UEs in the direction of Core) and down-link (from the Core in the direction of the UEs). Today's traffic runs on top of TCP/IP stack mostly, and the trends are going in the direction of streaming, requesting smaller parts of the data at a certain times. A typical use-case when people are watching online videos on `YouTube`, or `TikTok`. From the end user's point of view, it is expected if the requested content is loaded as soon as possible, e.g.: can start watch the video shortly, without loading the the whole content beforehand. This is also supported by the fact, that in the majority of the cases people tend to watch only the first few seconds of a video, to decide whether they continue watching it or not. On the other hand, from the provider's perspective, such small, but very frequent requests are more demanding when it comes to queuing the requests.

At a mobile telecommunication provider, e.g.: Magyar Telekom Nyrt., new features are tested in a test environment with real-world UEs and radio and core network elements to make a more seamless introduction of newly tested features to the live network. For this purposes mobile telecommunication providers would need to effectively and accurately simulate real-world traffic. As I mentioned earlier, this would include hundreds, even 900 different parties to simultaneously generate traffic (on the radio channel via UEs) with respect to the before-mentioned mobile data traffic trends. However, there are specific hardware and software combinations to generate this kind of distributed traffic, its efficiency in simulating a real-world case is still questionable. Instead of this, engineers needs to have an another way to test the RAN networks capabilities. RAN networks work in a way, that if only one user is in a cell, then the whole spectrum of available bandwidth will be available for that user and allocated to handle it is data transfer if demanded. With this in mind, we can still use the maximum capacity of the radio network and measure the performance of the end-to-end network with the help of network traffic generator tools. This work aims to study the latter approach to measure RAN KPIs and offers an implementation to make this kind of approach a more viable and refined method compared to the former solution used in the company.

Given this backdrop, a proof-of-concept system was implemented utilizing `Vue.js`, `Go`, and `Docker` technologies. Traffic measurement was integrated using `iperf3`, while the parallelization and robustness of the solution were achieved through containerization. In this setup, each measurement can be flexibly run in its own container, effectively isolating and eliminating the cross-impact of multiple, concurrently running measurements and ensuring accurate results while addressing security concerns.

The proof-of-concept validation demonstrated that the proposed solution operates reliably. Measurements conducted in a real-world environment using industry standard devices proved to be valuable and efficient in troubleshooting potential RAN bottlenecks. With

this solution, RAN engineers could optimize RAN networks and facilitate the introduction of new RAN features while ensuring a high-quality user experience. The implications and broader applicability of the solution include its potential to serve as a valuable tool within the company group, extending beyond just Magyar Telekom Nyrt.

The rest of this work is organized as follows: Chapter 2 provides an overview of the theoretical background, the related solutions and the technologies I have used throughout the developmental process. Chapter 3 introduces the developed system, including its main components, and gives an insight into each of them regarding how they operate, what specific technologies they utilize, and what considerations I have made during the development. Chapter 4 overviews the proof-of-concept validation, while Chapter 5 discusses the achieved results and the efficacy of the developed solution. Finally, Chapter 6 concludes this work.

Chapter 2

Background and Related Solutions

In this chapter I will provide a brief overview on network performance measurement and its possible approaches. I will introduce the most important metrics used for the network evaluations. I also took into accounts the possible measurement tools and argued in favor of the chosen instrument that I will use in my proposed implementation. In the last section of the chapter I presented the need for the possibility of parallel measurements and argued for the use of containers, and argued for the chosen container technology.

2.1 Assessing Traffic Characteristics

Measurements to understand network characteristics can be acquired either through internal or external mechanisms. *Internal mechanisms* are often integral parts of a system, such as the NetFlow [26] instrumentation in Cisco IOS software. This provides insights into various aspects of the network's operation, including user statistics, application traffic, routing patterns, and peak activity periods. Typically, these mechanisms are positioned at the interfaces of networking devices. However, as noted in [37], they can also be found at system interfaces, like the `/proc` file system in Linux, offering data on system resource load and performance metrics.

While a significant portion of contemporary networks is equipped with internal mechanisms for traffic analysis, there remain many that lack such built-in capabilities. In these instances, data is obtained through *external mechanisms*. These are not inherently part of the network or its hardware. Commonly, these are sniffers — be they packet, Ethernet, or wireless — that capture and record traffic at the point of observation. These sniffers can be hardware-based solutions like `NetScout` [40], `IPCopper` [32], or `Fluke` [30], or software solutions like `Wireshark` [51] or `NFStream` [23].

External mechanisms employ two primary strategies for network traffic analysis: (i) *active* and (ii) *passive*. While both methods bring distinct results, they are chosen based on the specific requirements of the measurement setup and the intended objectives. *Active* measurements often involve sending test packets to determine network properties, whereas *passive* measurements simply observe existing traffic without introducing any new data.

2.1.1 Active Measurement Approach

Active measurement is rooted in the principle of sending probe packets into a network. The packets are then analyzed to determine the network's characteristics and operational

status. A depiction of this approach can be seen in Figure 2.1. While these probes can sometimes be directed at specific servers or applications to assess the health of network services, the essence of active measurements remains consistent. Two primary features define active measurements: (i) they necessitate the creation of supplemental traffic and (ii) the generated traffic and its properties are synthesized.

Typically, the volume and traits of the artificial traffic can be tailored to the user's needs. For a comprehensive understanding, only a modest amount of this traffic is generally sufficient[1]. This approach offers granular control over various facets like traffic generation, sampling methods, paths, and the dimensions and kinds of packets[2], among others.

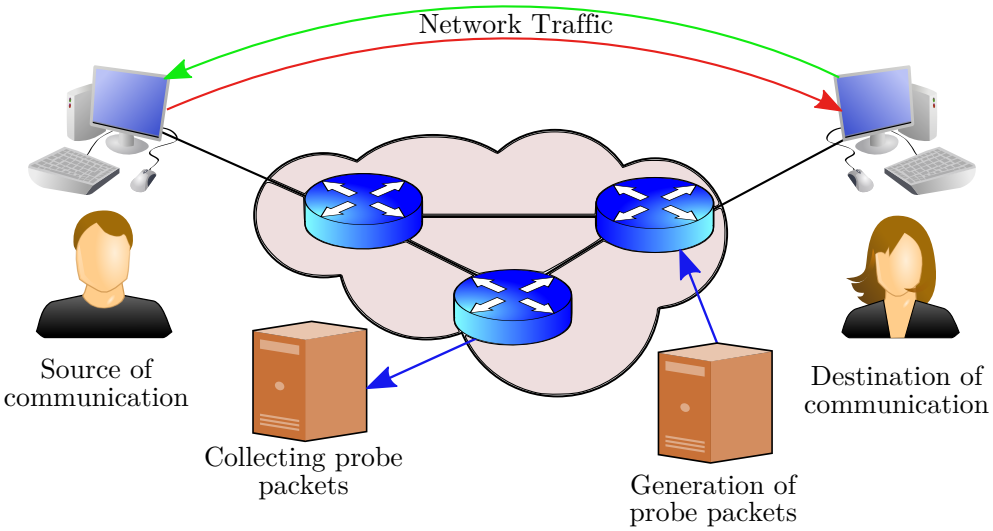

Figure 2.1: Illustration of the active measurement approach

However, active measurements are not without drawbacks. For one, they can amplify network loads, potentially skewing or entirely distorting results. Thus, network managers should be discerning about the relative cost of these measurements compared to standard operations. The primary goal is to ensure monitoring exercises a negligible impact on the network's performance. Another limitation is that active measurements offer a somewhat narrow view at the observation point, concentrating more on the relationship between two nodes. Common metrics derived include performance indicators such as RTT, average packet loss, connection bandwidth, and packet throughput.

There is potential in active measurements to deduce information about asymmetric delays or routing modifications between nodes. Nonetheless, achieving this often necessitates expanding the measurement tools or integrating complementary mechanisms.

2.1.2 Passive Measurement Approach

Passive measurement, as the name suggests, refrains from introducing or altering existing network traffic. Instead, it solely relies on traffic naturally produced by network users and applications. An illustrative approach to passive measurement is presented in Figure 2.2. Passive measurement predominantly employs two strategies. Firstly, it utilizes *inter-networking devices*—like routers, switches, and end devices—that come equipped with mechanisms to gather data about network health and traffic, such as the IPFIX [27],

[1] It is crucial to note that the requirement varies based on the specific objective of the measurement.
[2] Crafting packets of diverse sizes or types is essential for simulating different applications.

NetFlow [26], SNMP [25], and RMON [50] protocols. Secondly, it harnesses *software tools*, including `NFStream` [23] and `Wireshark` [51], tailored to capture and analyze network traffic.

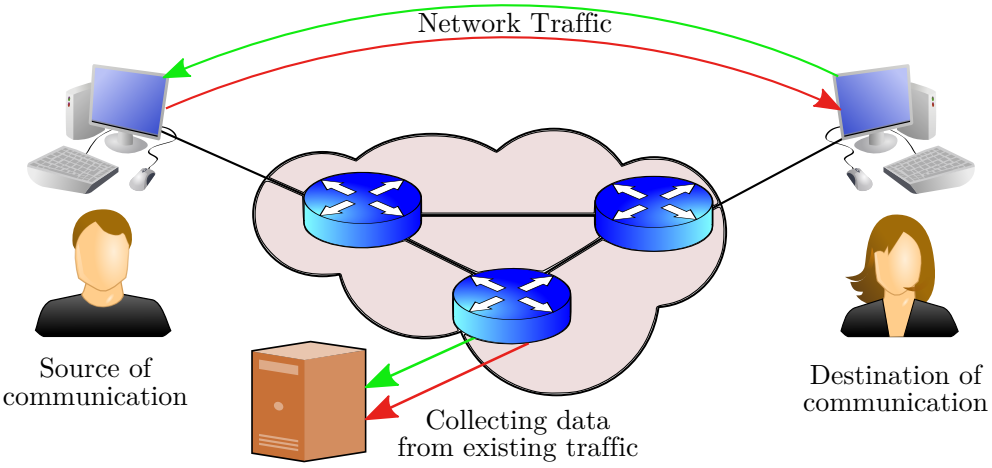

Figure 2.2: Illustration of the passive measurement approach

Interpreting these strategies can be complex, particularly when integrating them into an overarching network monitoring framework. A fundamental challenge lies in the nuanced processes of monitoring and their potential subdivisions. For instance, the IPFIX [27] protocol splits the familiar data collection task into: (i) the measurement and export process, and (ii) the collection process, as outlined in the IPFIX architecture [44].

Regular intervals punctuate data collection, either via inter-networking hardware or software solutions. Subsequent analysis of this data reveals insights into network performance and status.

Passive measurements' intrinsic strength is their reliance on genuine traffic, ensuring no artificial load gets imposed on the network. However, the act of gathering this data can marginally boost network traffic, particularly when every packet gets captured. A solution entails designating a dedicated route for this measurement-related traffic, ensuring its non-interference with the primary flow and preserving result accuracy. Notably, when packet-by-packet capture ensues, privacy and data protection concerns can arise [49, 48].

In contrast to active measures, passive ones offer a granular dataset about the observation point, elucidating traffic composition (data types, services, protocols, multimedia, etc.), packet frequency, timings, and delays.

Beyond real traffic properties, passive measures can also shed light on network infrastructure. This involves scrutinizing the NIB [35] of devices. Protocols exemplifying passive measurement include BGP [43] and OSPF [39].

Packet Capture: An Overview

Packet capture, a cornerstone of network traffic monitoring, employs a passive measurement technique that does not generate supplemental traffic. The primary components of a typical packet capture tool include (i) a mechanism for collecting data traversing a given interface or physical link, and (ii) a storage system designed to house the captured data without negative repercussions, like packet loss.

Considering the stringent performance criteria—encompassing computing power, memory speed, storage velocity, and clock precision—packet capture is predominantly executed on high-end computing systems. While specific monitoring tools like DAG cards [29] can serve the purpose, their expense and integration challenges make standard computer systems a preferred choice. Traffic is generally channeled to the packet capture system's interface via port mirroring, with captured data stored in a suitable format.

The ubiquity of the `PCAP` API across numerous operating systems positions it as the dominant capture file format. On UNIX-like platforms, `PCAP` is realized through the `libpcap` library, whereas on Windows, the `WinPcap` port is adopted. This library allows users to specify interfaces, packet types, and various filtering rules for capture.

However, `libpcap` primarily offers raw data (see Figure 2.3). To decipher captured packets, advanced software like TcpDump [46] or Wireshark [51] is requisite. One notable downside of `libpcap` is its root access requirement, which could introduce security vulnerabilities.

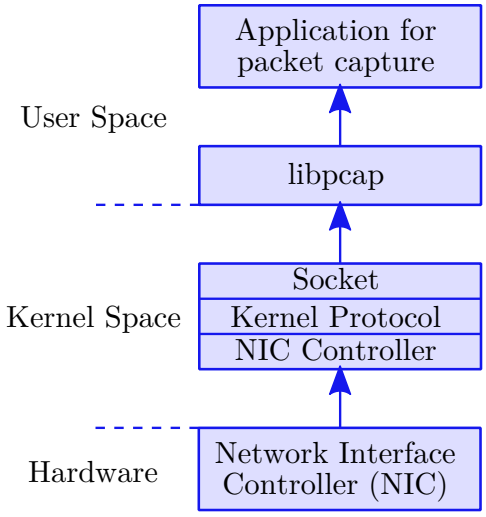

Figure 2.3: Illustration of packet capture via the `libpcap` library

Packet capture, when executed at the granularity of individual packets, offers unparalleled insights into network traffic. Yet, this granularity is not devoid of challenges, especially concerning lossless capture at elevated speeds, and the subsequent processing and storage of vast data volumes.

To manage these challenges, capturing full packet data is not always pursued. Often, only packet headers—along with metadata like data size and timestamps—are retained, while the data payload is omitted. Despite this reduction, the sheer volume of per-packet capture remains substantial, necessitating in-depth discussions on ensuing network monitoring complexities.

2.1.3 Combined Measurement

Efficiency in assessing infrastructure or topology attributes can be significantly enhanced through a combined measurement methodologies. Consider, for example, the challenges inherent in active measurement: in specific scenarios, such as mapping an autonomous system, it demands a substantial number of test packets. Incorporating passive measure-

ment can drastically reduce this requirement, thus counteracting the limitations of each individual method.

An intriguing aspect of combined approaches is the concept of *semi-active* measurements. These augment standard traffic with additional data—like timestamps or unique packet identifiers—to collect enhanced insights into network characteristics. While promising, semi-active measurements come with their own challenges, notably demanding high performance from the measurement points.

2.1.4 Summary of the Approaches

In summary, combining measurement techniques leverages the strengths of individual methodologies while mitigating their drawbacks. This synergistic approach optimizes accuracy and streamlines the measurement process, ensuring a focus mostly on the positive attributes of the used techniques.

Due to the fact, that in my case the main originator of the measurements are to help the introduction of new RAN features and not to assess some already set networks, the proper approach was to follow the active measurement paradigm. Especially to use a network traffic generator program, such as `iperf3` or `bwping` to achieve the maximum possible bandwidth on the IP network.

2.1.5 Determinants of Network Performance

When we discuss network performance, it is a confluence of various integral metrics and influential factors. The interplay between physical components and data traffic determines the actual performance of a network. Understanding this complicated dance is crucial for both network design and its subsequent operations, like testing new features, especially in the case of the radio communication domain.

Three key drivers primarily influence the performance of any network: Quality of Service (QoS) [42, p. 206], Network Congestion [24], and Network Resilience [45]. These can be viewed through different lenses, such as QoS parameters [36] or more generically as network performance attributes. The key indicators of such attributes are called Key Performance Indicators, KPI for short. In this subsection I will list the most important metrics used for the evaluations.

Latency: Latency, commonly referred to as "ping time", encapsulates the time taken for a data fragment, like a packet, to traverse from its origin to its intended destination. Within this broad definition lie specific types of delays: *routing delay*, which refers to the time taken for routers to process and forward a packet; *transmission delay*, the time it takes to push the packet's bits onto the link; and *propagation delay*, the time required for a bit to travel across the link to the router on the other end. Various tools, like ping and traceroute, are employed to evaluate this metric, providing essential insights into network responsiveness.

Jitter: Beyond just measuring the consistency of packet arrivals, jitter highlights the irregularities and variations in latency. Such inconsistencies, especially pronounced in voice and video communications, can cause detrimental effects on call and video quality. It primarily results from network elements, such as congestion, varying route paths, and discrepancies in timing, which lead to uneven packet delivery patterns.

Packet Loss: Not all packets dispatched from a source find their way to the designated destination. Some packets are lost due to an array of reasons, ranging from deliberate drops by routers in congested conditions to corruption en route. Visualizing this metric over time can be revealing, shedding light on network health, congestion levels, and potential areas of concern.

Throughput: Think of throughput as the practical manifestation of a network's capacity. It describes the actual volume of data packets that make their journey successfully, without disruptions, through the myriad of network components. Tools like `iperf3` and `speedtest-cli` prove invaluable in gauging this metric, revealing the efficacy of data transfer and potential bottlenecks.

Bandwidth: A concept often merged with throughput, bandwidth is the maximal rate at which data can potentially be transferred across a network. It is the theoretical upper limit, often marketed by telecommunication service providers as a selling point. However, real-world data transfer rarely hits this ceiling due to various limiting factors, such as throughput restrictions and network barriers.

Error Rate: This metric dives into the integrity of the data being transmitted, representing the fraction of bits that get corrupted during transmission compared to the total bits sent. A high error rate can signal physical issues, such as hardware malfunctions or signal interference.

Round Trip Time (RTT): Expanding on latency, RTT measures the total time taken for a packet to commence its journey from a source, reach its intended destination, and then return. It is an all-encompassing metric, incorporating all forms of delays encountered during this round trip.

In a more refined context, these performance metrics collectively provide a comprehensive analysis of a network's efficacy and efficiency. Each measurement, whether it is throughput, jitter, delay, or latency, acts as an individual piece in the overarching puzzle of mobile telecommunication network evaluation, especially within the RAN. By meticulously examining each metric—like the rate at which data is transmitted or any discrepancies in data relay—we obtain a holistic view of the network's performance. Among these, bandwidth, jitter, and packet loss emerge as predominant indicators. A thorough understanding of these key metrics is pivotal for optimizing and enhancing our network's functionality.

2.2 Existing Tools for Performance Measurement

In this section I will examine possible network testing tools like `iperf3`, `netperf`, and their counterparts, that are following the active based approach to measure a network. These tools are valuable for a variety of reasons in the realm of network administration, feature testing, performance tuning, and troubleshooting. In this section I will give a brief introduction to the several possible solutions to generate network traffic.

2.2.1 Iperf3

`Iperf3` is a widely used open-source tool designed to measure the bandwidth and the quality of a network link. At its core, it measures the maximum TCP and UDP bandwidth performance by creating data streams and assess the throughput of these streams between

a client-server setup. The tool supports both TCP and UDP testing. While TCP is connection-oriented and provides a reliable stream, making it useful for measuring the maximum possible bandwidth, UDP, being connectionless, is employed to test parameters like packet loss, jitter, and datagram delay.

The tool functions on a client-server model. One instance of `iperf3` runs as the server, and another connects to it as the client to perform the test. Notably, there are two main versions of iperf that have been actively used: `iperf2` and `iperf3`. These are distinct tools, with each having its own set of features and command-line options. For instance, `iperf3` is a complete rewrite of the original `iperf2`.

A significant feature of `iperf3` is its ability to handle multiple parallel streams. This becomes invaluable in scenarios where a single TCP or UDP stream does not saturate the available bandwidth, especially in high-speed networks. Moreover, it is capable of testing bidirectional bandwidth simultaneously, shedding light on how a link performs under simultaneous uploads and downloads.

During its operation, `iperf3` can present bandwidth statistics at regular intervals, offering users a detailed view of bandwidth variation over time. Another advantage of `iperf3` is its cross-platform availability, catering to Linux, Windows, macOS, and most importantly some mobile platforms.

Beyond bandwidth testing, `iperf3` has found utility in tuning the performance of specific network paths or equipment, validating QoS (Quality of Service) configurations, and verifying SLAs (Service Level Agreements) with ISPs or cloud providers. Over time, various extended versions and variations of `iperf3` have emerged, like `jperf`, which furnishes a graphical interface for `iperf3`.

In essence, `iperf3` is a versatile tool that provides an uncomplicated avenue for network administrators, engineers, and researchers to evaluate and diagnose network performance, making it an essential asset in many network professionals' arsenals.

2.2.2 Netperf

A networking performance benchmark that tests a variety of different types of networking operations across different protocols (TCP, UDP, etc.). Originating from Hewlett Packard labs, it provides a versatile means of determining raw socket and transport layer throughput.

2.2.3 BWPing

As the name suggests, `BWPing` is a tool to measure bandwidth and the quality of the data channels by sending ICMP echo requests. It allows for real-time data channel monitoring to detect link faults promptly.

2.2.4 MTR (My TraceRoute)

`MTR` integrates the functionality of the 'traceroute' and 'ping' programs in a single network diagnostic tool. As it sends packets, it provides real-time insights into latency and packet loss for each hop along the route, making network troubleshooting more precise.

2.2.5 BWCTL (Bandwidth Test Controller)

A command-line client-server tool used to coordinate performance tests between two hosts, which can use several testing tools like `iperf3`, `ping`, and `traceroute` under its umbrella. It is especially valuable for its scheduling capabilities, allowing for bandwidth tests to run without human intervention.

2.2.6 TTCP (Test TCP)

An older utility for measuring TCP and UDP performance between two systems. It allows both throughput testing and transaction testing.

2.2.7 Synthesis

`iperf3` is an actively maintained and updated tool tailored to modern network environments, offering features that are in step with contemporary performance challenges. The very fact that it is a successor to `iperf2` means it benefits from lessons learned from its predecessor. The cleaner and more efficient code base of `iperf3` results in fewer bugs, which can make it a more reliable choice than some older tools like TTCP.

In terms of reporting precision, while tools like `Netperf` provide a variety of tests across different protocols, `iperf3` has the edge with its interval-based reports, giving users a granular understanding of performance over time. This is a stark contrast to tools that only offer an end-of-test summary.

Performance-wise, `iperf3` has features like support for zero-copy TCP sends, which can significantly improve performance metrics. For instance, when trying to measure the potential of high-speed networks where CPU overhead might be a concern, `iperf3` can outshine tools that lack this feature.

In scenarios requiring the testing of high-speed or multi-gigabit networks, the ability of `iperf3` to handle parallel streams gives it an advantage. This provides a more comprehensive view of network throughput and potential bottlenecks compared to simpler tools that only handle single-stream tests.

When versatility is a concern, `iperf3` shines with its support for both TCP and UDP tests. While tools like `BWPing` are focused on specific protocols, `iperf3` offers a broader range of testing scenarios. Moreover, its capability to specify bandwidth during UDP tests means it offers greater flexibility than many counterparts.

One distinct feature of `iperf3` is its bidirectional testing capability. While many tools require separate runs for upload and download tests, `iperf3` can conduct them simultaneously, providing a clearer picture of full-duplex network conditions. This stands in contrast to tools like MTR, which, while powerful in route diagnostics, do not offer the same level of duplex testing.

Lastly, the ease of use and widespread adoption in the telecommunication industry of `iperf3` cannot be understated. Its straightforward command-line interface is both intuitive for new users and powerful for experienced ones. And since many network professionals are familiar with `iperf3`, there is a rich community ready to offer advice, scripts, and troubleshooting tips, something that might not be as prevalent for lesser-known tools.

2.3 Computing Parallelization and Its Role in Performance Measurement

In the realm of network performance measurement, the significance of parallelization is underscored by two primary considerations in your use case. Firstly, parallelization is indispensable when multiple teams seek to concurrently utilize the application to assess end-to-end network performance for distinct purposes. This concurrent usage ensures that varied network analyses can be conducted simultaneously, enhancing efficiency. Secondly, there are specific scenarios, such as within a given Dynamic Spectrum Sharing (DSS) cell, where both 4G and 5G signals are being broadcasted concurrently. In such instances, parallelization facilitates compatibility tests where data is transmitted over both technologies simultaneously to evaluate potential degradation and mutual interference. By enabling parallelization, your application can comprehensively cater to diverse testing requirements and scenarios, thereby ensuring a thorough and efficient assessment of network performance. In this section I will provide a brief overview what kind of parallelizations are there and argue for the virtualization as a choice of mine for this application.

For comprehensive network performance measurement, it is often imperative to carry out multiple tests concurrently to simulate real-world loads and scenarios. Parallelization plays a crucial role here:

- **Parallel Testing:** By employing software parallelization techniques, multiple instances of a test can be run concurrently, providing a broader overview of network performance under various conditions.
- **Use of Containers:** With tools like `Docker`, multiple containers (lightweight virtualized environments) can be spun up rapidly to run parallel tests. For instance, `iperf3` servers inside `Docker` containers can be utilized to parallelize network bandwidth measurements.

2.3.1 Hardware Parallelization

This refers to the intrinsic parallel architectures of modern computing hardware:

- **Multi-core Processors:** Modern CPUs have multiple cores, each capable of executing its own thread.
- **Graphics Processing Units (GPUs):** Originally designed for rendering graphics, GPUs are now utilized for general-purpose parallel computations due to their many concurrent processing units.
- **Field-Programmable Gate Arrays (FPGAs):** These are integrated circuits designed to be configured by the user for parallel operations.

2.3.2 Software Parallelization

Software parallelization involves designing algorithms and software to execute operations concurrently. This includes:

- **Thread-level Parallelism:** The application is divided into multiple threads that can be executed concurrently.

- **Data-level Parallelism:** Data is divided into smaller chunks, and the same operation is applied to each chunk concurrently.
- **Task-level Parallelism:** Different tasks or operations within an application are run in parallel.

2.3.3 Virtualization as a Form of Parallelization

Virtualization involves creating virtual instances (like virtual machines or containers) of computing resources. While not parallelization in the traditional sense, it allows for multiple virtualized environments to run concurrently on a single physical machine (see Figure 2.4). This can be particularly useful for tasks such as testing different software versions or running isolated applications. Virtualization, at its core, is the act of creating virtual (rather than actual) versions of computing resources, be it storage devices, network resources, or computing environments. This technology has revolutionized the way IT infrastructures are designed and managed.

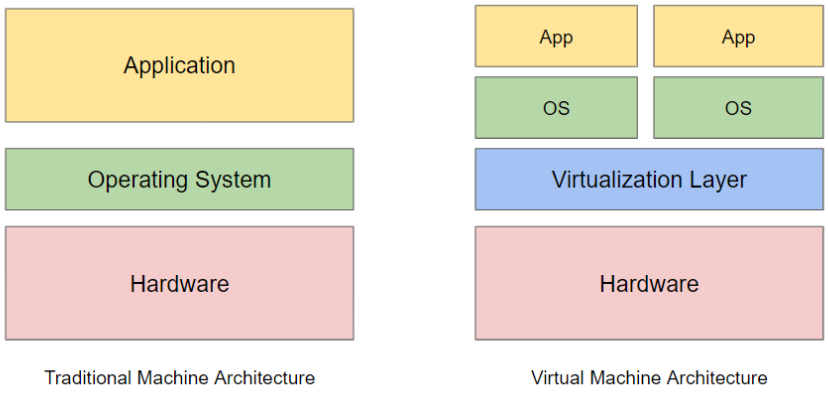

Figure 2.4: Virtualization architecture

Pros:

- **Resource Optimization:** Virtualization enables better hardware utilization by allowing multiple virtual environments on a single physical entity.
- **Flexibility:** Rapid deployment of new virtual environments and easy adaptation to changing workloads.
- **Isolation:** Virtual environments are isolated, ensuring that failures or security breaches in one do not affect others.

Cons:

- **Overhead:** Introducing a virtual layer can sometimes introduce performance overhead.
- **Complexity:** Managing virtualized environments can be complex, requiring specialized tools and expertise.

Containers: The Lightweight Virtualization Solution

Containers are a form of lightweight virtualization (see Figure 2.5) that package an application and its dependencies together. This ensures that the application runs uniformly across different computing environments.

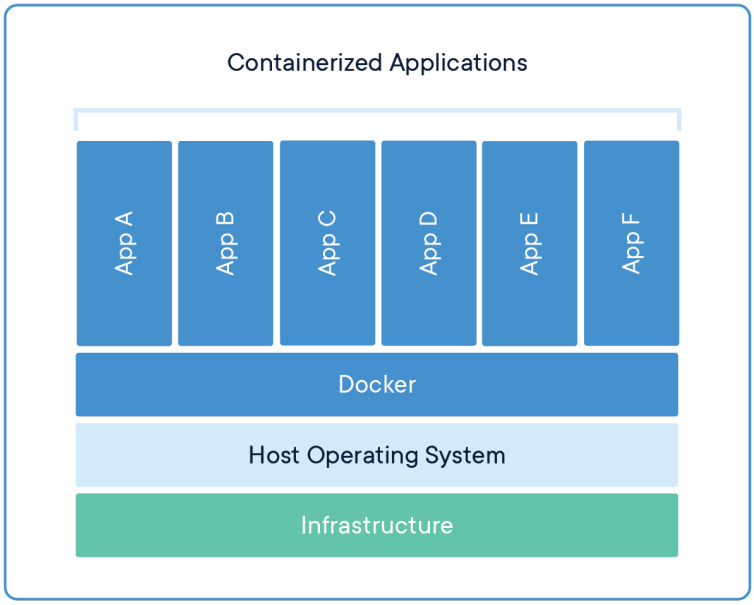

Figure 2.5: Virtualization with containers

Pros:

- **Portability:** Containers ensure applications behave consistently across various environments.
- **Resource-Efficiency:** Unlike VMs, containers do not need a full OS stack, making them lightweight.
- **Rapid Deployment:** Containers can be quickly instantiated, scaled, or decommissioned.

Cons:

- **Isolation:** While containers are isolated, they are not as strictly separated as traditional VMs.
- **Dependency on Host OS:** All containers on a host share the same OS kernel, potentially leading to compatibility issues.

Traditional Virtual Machines vs. Containers

Both VMs and containers are pivotal virtualization techniques, but their operational modalities differ. The conceptual difference can be seen in Figure 2.6.

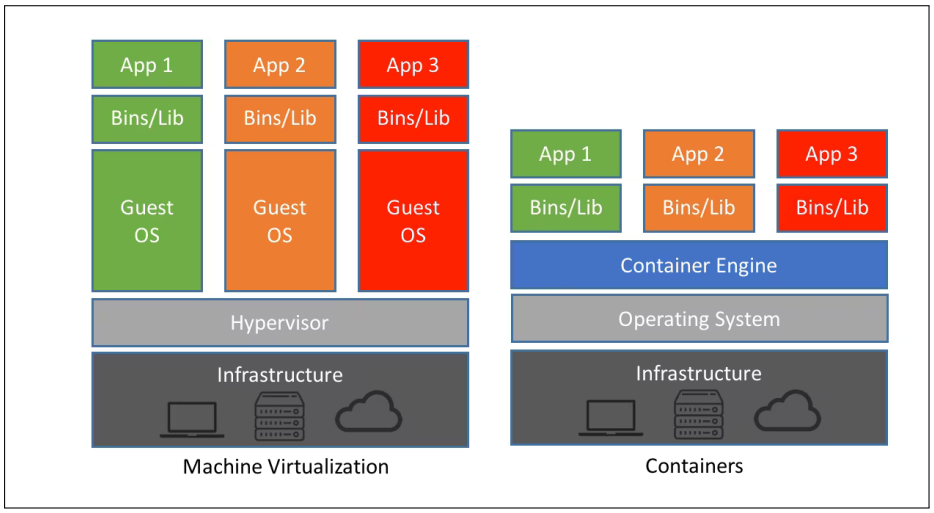

Figure 2.6: Containers vs. Virtual Machines [4]

- **Virtual Machines:** VMs mimic physical hardware, running comprehensive operating systems. This results in bulkier, resource-intensive entities.
- **Containers:** Containers run on the same OS kernel but isolate the application processes. This architecture ensures that containers are lightweight, efficient, and quick to instantiate.

For network performance measurement endeavors, especially when deploying tools like `iperf3`, containers offer clear advantages. Their lightweight nature, coupled with the speed of deployment, makes them apt for quickly spawning multiple `iperf3` server instances, thus endorsing parallelized network bandwidth assessments.

2.3.4 Synthesis

Given the extensive examination of virtualization techniques, containers emerge as the most fitting solution for our network performance measurement tasks. Their inherent advantages, notably over traditional VMs, in terms of resource consumption, speed, and scalability, make them the optimal choice for hosting `iperf3` servers. Thus, our design gravitates towards the strategic adoption of containerization.

Chapter 3

Design and Implementation

In this chapter, I give an overview over my proposed architecture to the application (see Figure 3.1). In the preceding sections I have made an conclusion to choose `iperf3` as the main measurement tool. This chapter seeks to introduce the fundamental building blocks of the system.

At its core, the architecture encapsulates a web-based user interface, a single-page frontend application that communicates with the backend services via a RESTful API. Through the API endpoints users are able to instantiate new, or handle (start or stop) existing measurement servers, in our case, `iperf3` servers. To let users make parallel measurements on the network, I have chosen a form of virtualization, the container-technology as a tool. Each `iperf3` server will be launched in its own container. In the following subsections I will provide an overview over the three main components of the application, and reason behind the chosen technologies I have used to make them.

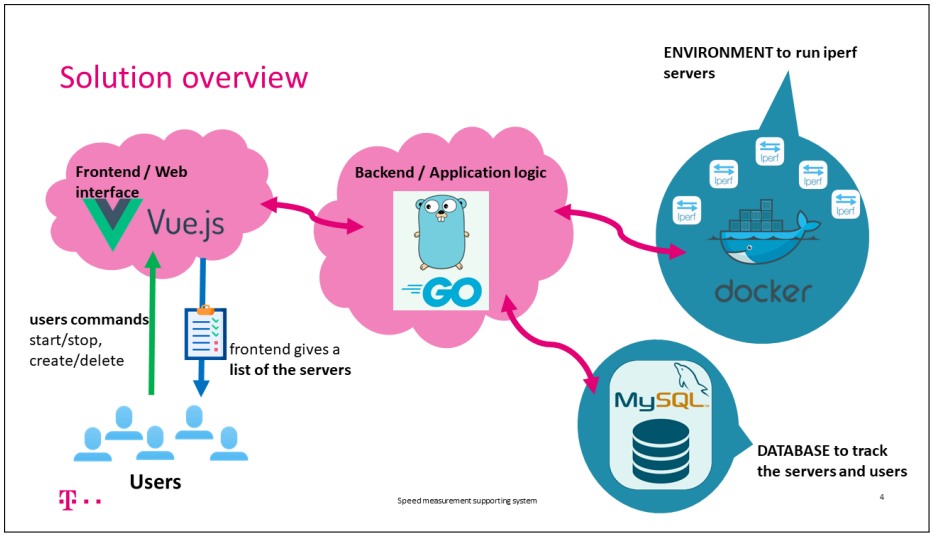

Figure 3.1: Architecture overview

3.1 Framework Selection

In the realm of modern web development, frontend frameworks such as `Vue.js` [22], `Angular` [2], and `React` [17] are pivotal. Rather than relying on vanilla `JavaScript`,

these frameworks offer developers streamlined tools and structures for crafting dynamic single-page applications. In my use-case it was imperative to use such a framework, as end-users do not want to deal with SSH and terminal to launch a measurement server. In the following sections I will briefly introduce the frontend frameworks I considered using when I was planning the architecture of the application.

React

`React`, a product of Facebook's innovation, has carved a niche with its component-based architecture. It introduces a virtual DOM, uses JSX for templating, and complements itself with tools like Redux for managing application state. Owing to its robust capabilities and backing, `React` has seen immense adoption and boasts a vast community that contributes resources, plugins, and libraries.

Angular

`Angular`, another popular choice in this domain, founded by Google. Unlike `React`, which is more library-like, `Angular` is comprehensive. Its features, including two-way data binding, dependency injection, and a modular structure, are its main features. It uses `TypeScript` as its primary language. `Angular`'s long history, especially with its earlier version `AngularJS`, makes it a popular choice for developers building big applications.

Vue.js

`Vue.js`, despite its relatively recent introduction, competes effectively with established frameworks such as `React` and `Angular` backed by multi-million dollar companies like Facebook and Google. Developed by Evan You, `Vue.js` incorporates a component-based architecture similar to `React`. It adeptly positions itself between `React`'s library-focused model and `Angular`'s comprehensive framework methodology. Features like the virtual DOM, bidirectional data binding, and the distinct single-file components contribute to its robustness. The swift increase in its adoption for large-scale applications attests to its efficacy and value in the industry.

Synthesis

Frontend frameworks offer several advantages. They speed up development with reusable components. Their reactive design ensures that the user interface updates immediately when data changes. These frameworks have large, active communities, providing developers with many resources and solutions. Performance is a top priority, with features like the Virtual DOM in `React` and `Vue.js` improving the rendering. They support single-page applications, which improves user experience by reducing full page reloads. Moreover, they encourage a modular approach, crucial for effectively managing large applications.

To wrap up, `React`, `Angular`, and `Vue.js` each bring unique strengths to the table. `Vue.js`, however, shines brightly with its user-friendliness, adaptability, and seamless integration capabilities. It effectively blends `Angular`'s systematic approach with `React`'s versatility. A focus on performance in `Vue.js` ensures efficient application execution, supported by its streamlined build. The framework's design and API are coherent and straightforward, streamlining development efforts. Additionally, its well-organized documentation eases

the entry for new adopters. Because of the aformentioned reasons I have chosen `Vue.js` as my applications frontend framework.

3.2 Frontend Component

To ensure a polished and professional appearance for the frontend, I utilized Vuetify [1], an open-source UI library constructed atop Google's Material Design guidelines. This library empowers novice frontend developers to create pages that adhere to industry-standard aesthetics. The landing page seamlessly redirects users to the login section of the application, as illustrated in Figure 3.2. Unregistered users can navigate to the `SignUp`

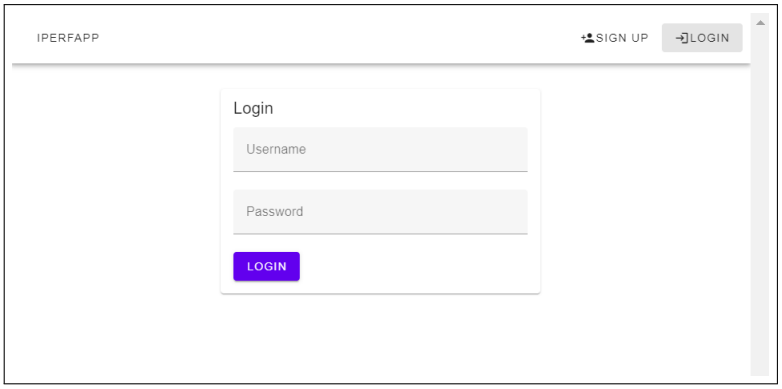

Figure 3.2: Login page

page (see Figure 3.3) using the navigation bar situated at the top of the page. Upon

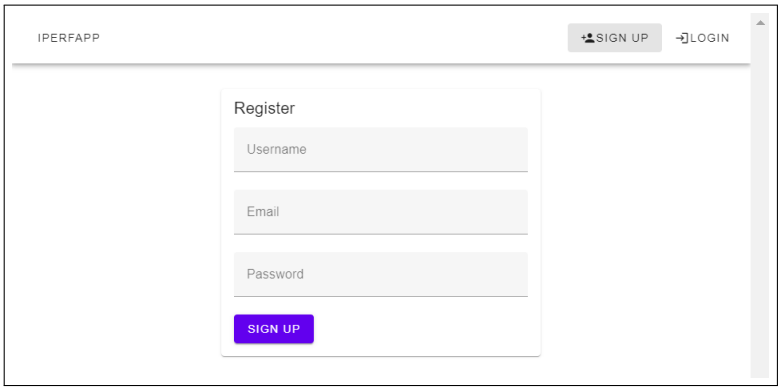

Figure 3.3: `SignUp` page

successful login, users are redirected to the `Profile` page (see Figure 3.4). The `Profile` page displays fundamental user information such as username and email, a segment of the authentication token used for backend communication, the assigned port range manageable by the user and the current authorities and roles assigned to the logged-in user.

Utilizing the navigation bar, logged-in users can access the `Home` page (see Figure 3.5), which serves as the gateway to the application's main functionalities. For newly created users, this view is initially empty as no servers are associated with the user. To create new servers, users can click on the *"Add Server"* toggle button, revealing the corresponding form shown in Figure 3.6. This form enables users to define attributes for the new server, such as its name, the port for exposure, the server type (only `iperf3` is supported by default),

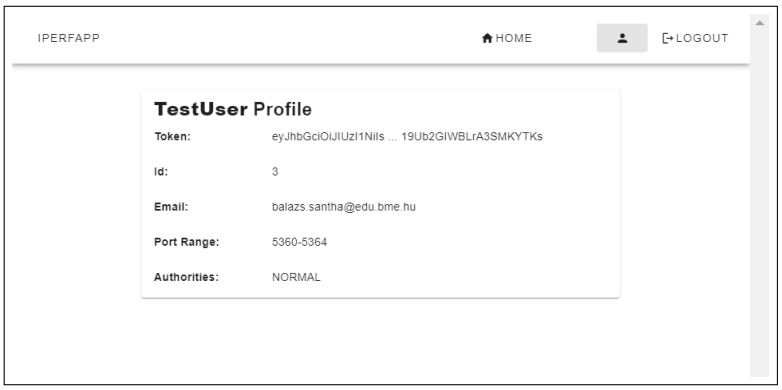

Figure 3.4: `Profile` page

Figure 3.5: Home page

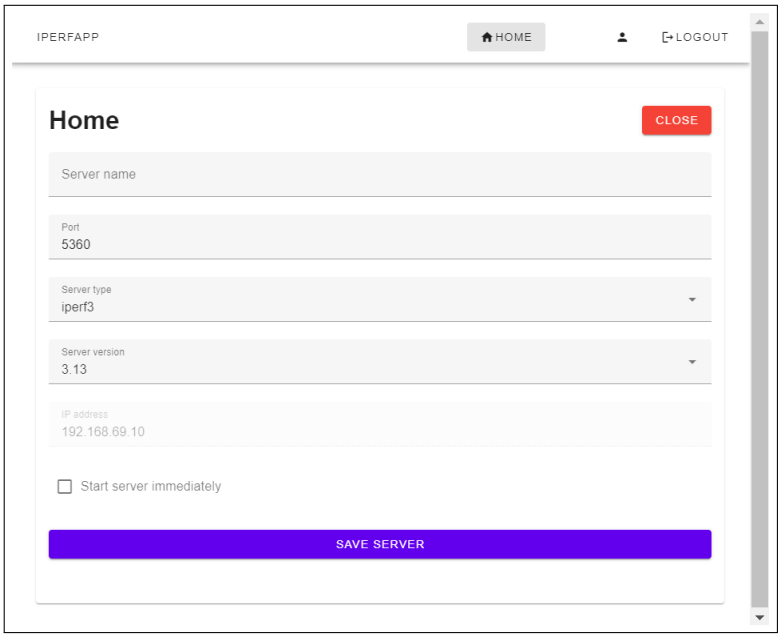

Figure 3.6: `AddServer` view

the version (`iperf3` 3.13 and 3.0.11 versions are supported by default) and whether to initiate the server immediately. The form incorporates numerous validation rules for each field, demonstrated in Figure 3.7. Once a server is successfully created, it appears in the list and the form recalculates itself to suggest the next available port for use, as shown in

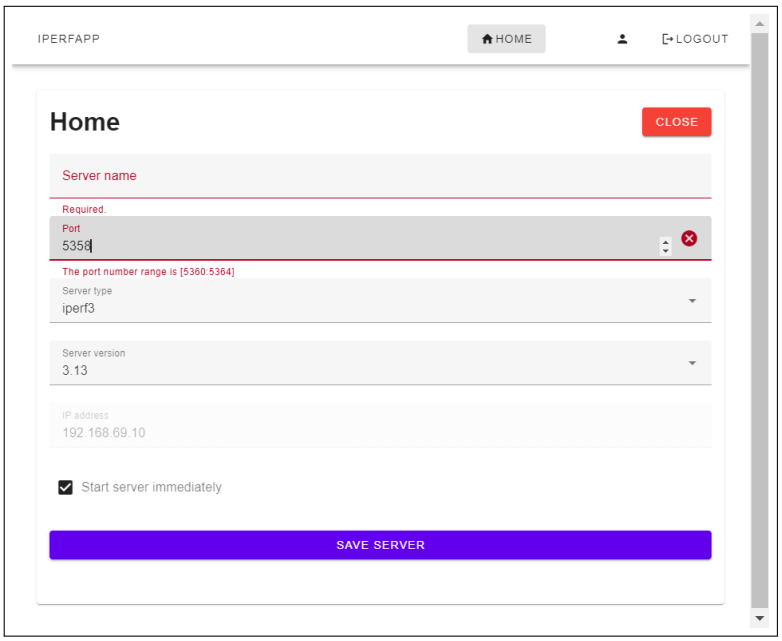

Figure 3.7: Field validations in the `AddServer` form

Figure 3.8. Users can toggle the status of each server between running and not running.

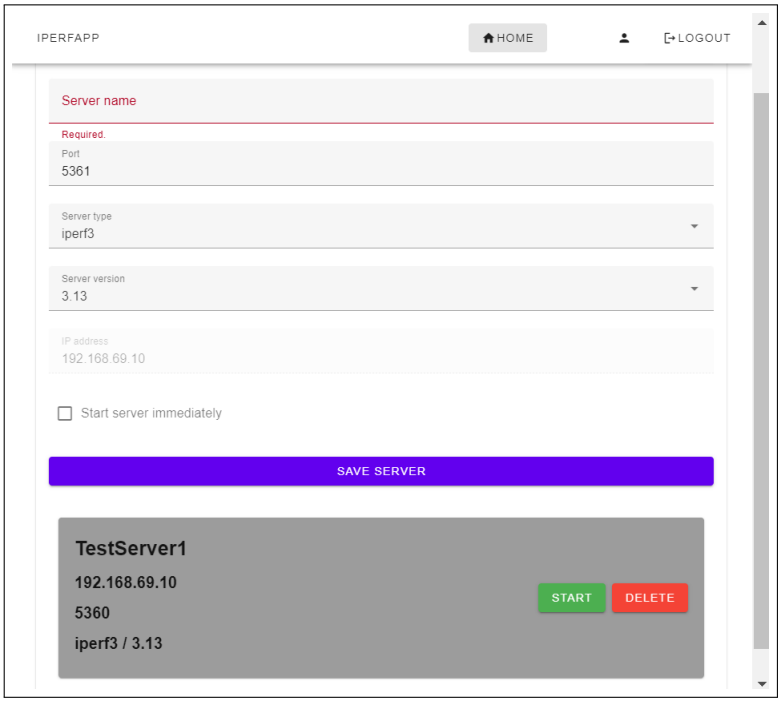

Figure 3.8: The form automatically offers the first available port

Servers that are active are highlighted with a light green background as indicated in Figure 3.9. Users also have the option to delete servers using the Delete button.

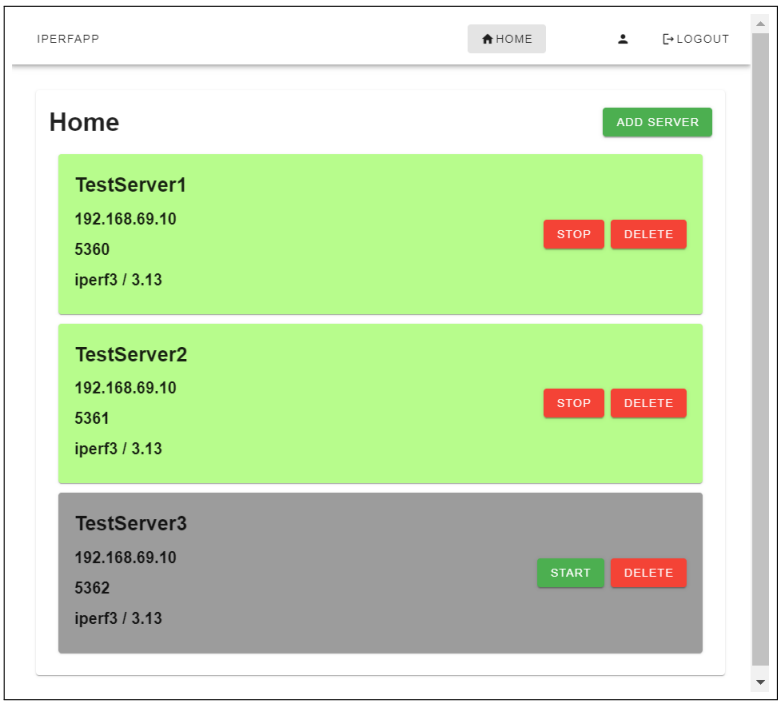

Figure 3.9: Running servers are with green background

3.3 Backend Component

The backend component serves as the central element of the application, orchestrating its various functions. It exposes a variety of API endpoints, processes incoming requests, and interfaces with the `Docker` engine on the host to manage container operations, all while persisting data to a database. This component is also responsible for user authentication, ensuring that each user is presented with the content specifically intended for them. As illustrated in Figure 3.1, every use case interacts with the backend component.

3.3.1 GoLang by Google

When it comes to backend development, there are several programming languages you can choose from, each with its strengths and trade-offs. After considerations of different languages like Ruby [19], Python [16] or PHP [14] etc. the backend architecture springs to life using the `Go` [10] programming language, commonly referred to as `GoLang`. This language, birthed by Google, shines in scenarios that demand superior performance.

`Go`, developed at Google in 2007 by Robert Griesemer, Rob Pike, and Ken Thompson, is a modern programming language that is both statically typed and compiled. What sets `Go` apart is its focus on simplicity and developer productivity. With performance that often rivals languages like `C` or `C++`, it also incorporates built-in support for concurrent programming with `goroutines` and channels. This focus on parallel execution makes it especially powerful for today's multi-core systems. `Go`'s syntax is clean and straightforward, making it easy for developers to read and write code. Additionally, it offers a robust standard library that is particularly adept for tasks like building web servers, handling I/O, and manipulating common data structures. Cross-compilation is another feature where `Go` shines, allowing developers to compile code for different platforms from one machine. On the memory management front, `Go` simplifies things with automatic garbage

collection, reducing the risk of issues like memory leaks. Furthermore, Go's static type system, combined with tools like go vet and gofmt, ensures code quality and helps catch potential errors early in the development process. The introduction of Go modules has also made dependency management and versioning more streamlined and intuitive. With the growing popularity of Go year-by-year I have opted for it as my main programming language for the application's backend component.

3.3.2 Web Frameworks in the Go Ecosystem

For the creation and management of the RESTful endpoints, there are several noteworthy frameworks in Go designed for web development. These choices include Gin [9], Echo [7], Gorilla/Mux [11], Revel [18], Fiber [8], Beego [3], and Buffalo [3].

- Gin: A high-performance web framework that is minimalist and efficient.
- Echo: A fast and unfancy HTTP server framework.
- Gorilla/Mux: A powerful URL router and dispatcher.
- Revel: A high-productivity web framework for Go that comes with a lot of features out-of-the-box.
- Fiber: Inspired by Express (Node.js framework), it's designed to jump-start your web application development.
- Beego: An all-in-one framework with functionality for ORM, caching, and more.
- Buffalo: Aims to be a holistic web development environment, providing tools to address a variety of development needs.

Gin stands out for its outstanding speed, making it one of the quickest web frameworks for Go. The framework supports a vast array of middleware, offering developers flexibility in crafting their applications. Gin does not impose any particular architecture pattern or directory structure, allowing developers the freedom to structure their projects as they see fit. Gin's API is intuitive, establishing a smooth development experience. For this reasons I opted for the Gin web framework out of the other choices I have considered. While each library presents its advantages, Gin's robustness and efficiency suited the project's demands the most.

3.3.3 Operation of the Backend

At the outset of this section, it was mentioned that the application uses a client-server architecture, in which the frontend component interacts with the server through its Application Programming Interface (API). This API uses the Representational State Transfer (REST) paradigm. The backend comprises two primary groups of endpoints.

- **AUTH group: login and register related endpoints**
  - `POST /auth/register` - User registration.
  - `POST /auth/login` - User authentication.
- **PROTECTED group:** (Requires authentication for access)

- `GET /protected/servers` - Retrieve all servers.
- `GET /protected/server?id=` - Fetch specific server details.
- `POST /protected/servers` - Add a new server.
- `DELETE /protected/server?id=` - Delete a specific server.
- `PUT /protected/server?id=` - Toggle a server state between halted and running.

3.3.4 Docker

`Docker` [6] is an open-source set of platform as a service, designed to facilitate the development, deployment, and execution of applications in containers. The platform encompasses several specialized tools for container-related tasks, such as:

- Software to run containers.
- Tools for developers to wrap applications into container images.
- A repository for storing container images.

However `Docker` is one of the most popular containerization tools, it is noteworthy to say that it is not the only one. There are several other tools and technologies that enable containerization like `LXC` [13], `Podman` [15] or `CRI-O` [5].

`Docker`'s immense popularity within the containerization landscape can be attributed to a combination of timing, simplicity, a robust ecosystem, and innovation. `Docker` introduced `Docker Hub`, a cloud-based registry where users could share and access container images. This centralized repository played a massive role in promoting `Docker`'s widespread use. Resulted from the massive popularity, knowledge base and community behind `Docker` is massive. Additionally great documentation and tutorials make `Docker` a great choice for newcomers in the field of containerization. For this reasons I have also opted for `Docker` as my containerization tool.

`Docker` is integral to the architecture of the application, serving as the primary virtualization technology responsible for initiating multiple `iperf3` servers concurrently. When processing specific requests, the `Go` application communicates with the `Docker` engine operating on the physical server hosting the entire application. Utilizing various endpoints, users can indirectly instantiate new containers, as well as halt and remove existing ones.

3.3.5 Role of Security

Today's worlds trends are indicating and force developers to take into account the security aspects of newly developed products more than ever. Various strategies and layers exist to protect systems from malicious activities. In terms of user authentication, there are two predominant approaches to consider. The first approach involves session-based authentication, as illustrated in Figure 3.10. Alternatively, token-based authentication can be employed, as depicted in Figure 3.11. A traditional approach employed on the web involves utilizing cookie-based, server-side sessions. In this method, a user sends their username and password to a server via an HTTP request. Upon validation, the server initiates a session in the database and returns a session ID to the client, which is then stored in the browser's cookie jar. Each subsequent request to the server includes this session ID, allowing the server to validate the session and respond with content tailored

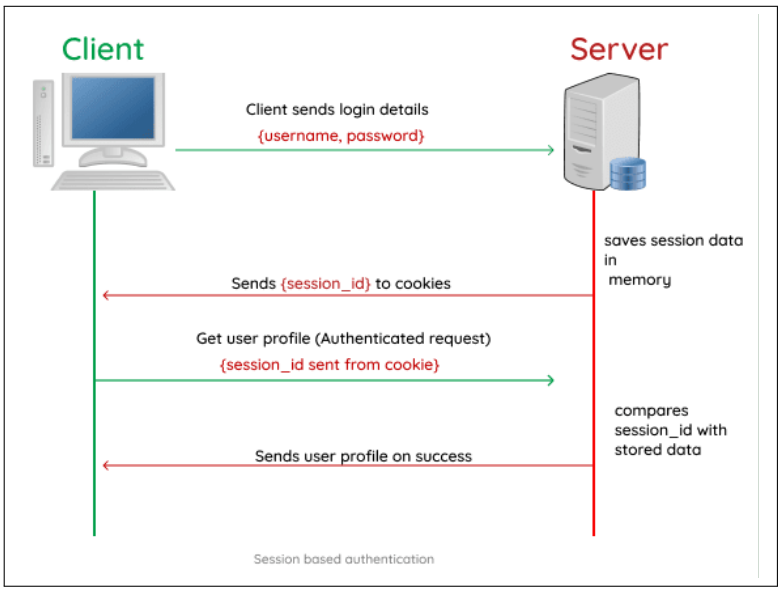

Figure 3.10: Session based authentication [20]

to the authenticated user, thereby establishing a stateful session between the client and server. This method, however, presents certain shortcomings. For instance, it is vulnerable to Cross-Site Request Forgery, although this risk is mitigated when using contemporary frameworks. A more significant issue arises from the need for the server to retain the session ID either in the database or in memory. Modern applications often employ horizontal scaling (as is the case in this solution), makes worse this challenge. Token-based authentication can be implemented as a solution (see Figure 3.11), though it comes with its own set of limitations. The initial phase of the process remains consistent with the

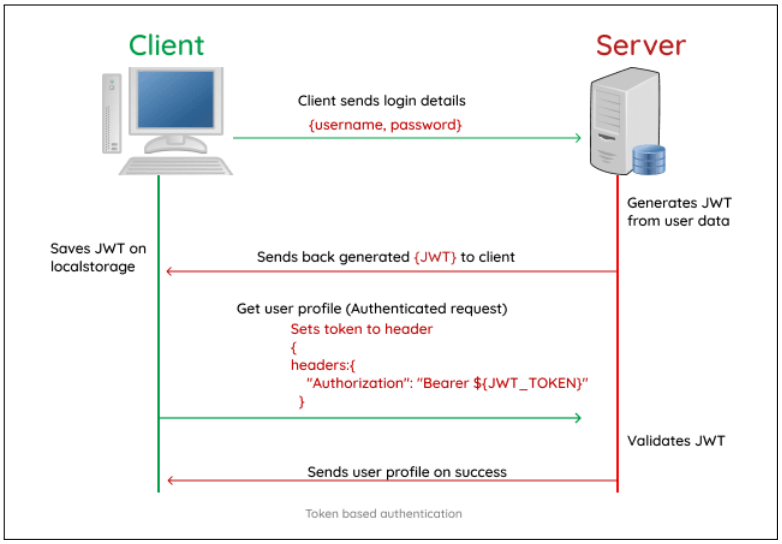

Figure 3.11: Token based authentication [20]

session based methods: upon validation of the username and password, the server generates a JSON Web Token (JWT). This token is signed using a secret securely stored on the server and subsequently transmitted to the client, where it is retained in the browser's local storage. Thereafter, for every subsequent request, the client appends this token to the Authorization Header of the HTTP request, specifically in the Bearer field. Consequently,

the server's only task is to validate the token's signature, eliminating the necessity to retrieve information from the database or retain information in memory. Nonetheless, there are potential drawbacks, such as the vulnerability of tokens to hijacking and challenges associated with their invalidation. After the consideration of the two dominant approaches to user authentication, I have chosen the token-based one due to its scalability, statelessness, ease of implementation, and cross-domain compatibility, which align well with the requirements of the application.

Implementation of Token Based Authentication - JSON Web Tokens

JSON Web Tokens [33] (JWT) form the backbone of the authentication system in our application and follows the token based authentication paradigm. JWTs are compact, URL-safe tokens that can be encoded with specific claims. Once a user logs in, the server generates and returns a JWT containing claims such as the username, user ID, token expiration date, and other pertinent information. This token is subsequently attached to every request from the client, serving as a proof of authenticity. Middleware on the server-side then validates this token, ensuring that only legitimate requests gain access to protected resources.

3.3.6 Database - MySQL

The persistence layer of the system rests on MySQL, a renowned relational database management system. This layer is tasked with storing intricate details about servers and users. Its role is essential to track which server corresponds to which users, also to track the allocated port ranges of each user. Two main structures dominate this realm - the `Server` and `User` as can be seen below.

```
type Server struct {
gorm.Model
UserID      uint   `json:"userID" gorm:"column:user_id"`
Name        string `json:"name" gorm:"column:name"`
Port        int64  `json:"port" gorm:"column:port"`
Type        string `json:"type" gorm:"column:type"`
Version     string `json:"version" gorm:"column:version"`
Running     int64  `json:"running" gorm:"column:running"`
ContainerID string `json:"containerID" gorm:"column:container_id"`
}
```

```
type User struct {
gorm.Model
Username  string `gorm:"size:255;not null;unique" json:"username"`
Email     string `gorm:"size:255;not null;" json:"email"`
Password  string `gorm:"size:255;not null;" json:"password"`
StartPort int    `json:"startPort" gorm:"column:start_port"`
EndPort   int    `json:"endPort" gorm:"column:end_port"`
Role      Role   `gorm:"type:ENUM('ADMIN', 'NORMAL');default:'NORMAL'" json:"role"`
}
```

The Object-Relational Mappings (ORMs) for these structures are facilitated using the `GORM` [12] library. `GORM` aids in abstracting intricate database operations into more intuitive `Go` methods, enhancing code readability and maintainability. Its features like auto migrations, hooks, and associations made it a valuable asset in my development process.

Chapter 4

Proof of Concept Validation

In this section I will present a concrete real-life case, where an UE was examined, due to its weaker speed compared to the expected results. The examination shed light on interesting phenomenon which is caused by the capacity limitation of the specific hardware component. Due to legal restrictions, I will avoid any specific information on the used hardware and software.

4.1 The Test Environment

The UEs, such as mobile phones and tablets, are placed inside a Faraday cage or in a so called RF-shielded box (see Figure 4.1 and Figure 4.2) to ensure a controlled testing environment. In this way, they can simulate real network conditions and perform measurements on the devices' radio performance, data transfer speed, error rate, etc., all without external radio signal interference affecting the results. Within this isolated environment, the UEs run an `iperf3` client to initiate connections for testing network performance. These UEs connect to `iperf3` servers hosted on an application, utilizing the underlying radio technology for the connection. The data then traverses the core network, ultimately reaching the `iperf3` servers situated at a different location. By conducting the tests within this controlled environment, comprehensive end-to-end measurements of the network's performance and capabilities can be achieved.

4.2 Setup for the Examination

In this subsection I will provide a brief introduction to the used radio technologies and to the used nomenclature to properly establish the context of the measurements and to establish the clarity of the future analysis. During our investigations, we tested a specific type of device, due to its weaker speed compared to the expected results. The device was tested in a Carrier Aggregation (*cf.* Section 4.2) (CA) setup on B28, B3, B1 and N1 carriers, in configurations of 10MHz 2x2, 20MHz 4x4, and 20MHz 4x4 MIMO (*cf.* Section 4.2).

Figure 4.1: Shielded box for RF signal testing [21]

Figure 4.2: Shielded box for RF signal testing [21]

Primary Cell, Secondary Cell and Carrier Aggregation in Wireless Communication

Carrier aggregation is a method employed in wireless communication to enhance the per-user data rate by allocating multiple frequency blocks, known as component carriers, to a single user [28]. The per-user data rate potential escalates with the assignment of more frequency blocks to a user. This also results in an elevation of the cell's aggregate data rate due to improved resource utilization. Furthermore, CA enables load balancing [31]. Channel selection strategies for CA systems, which consider optimal values for training length and power, the count of probed sub-channels, and the feedback threshold to maximize the sum rate, are crucial for achieving optimal capacity [47].

In the realm of 4G LTE and 5G networks, the notions of Primary Cell (PCell) and Secondary Cell (SCell) are integral to understanding the operation of Carrier Aggregation.

- PCell: This is the primary cell or sector to which a mobile device connects when it enters a network area. The PCell is responsible for basic communication services, such as initiating and receiving calls, sending and receiving SMS, and data transmission. Most of the communication and device control take place through the PCell.
- SCell: The SCells are additional, secondary cells or sectors to which a mobile device can connect to increase capacity and speed. CA technology allows a device to use multiple cells (PCell + one or more SCell) simultaneously, combining their bandwidth and thereby increasing data transfer speed. Thus, SCells assist the network in using multiple frequency bands and resources at the same time to optimize data transmission.

MIMO in Wireless Communication

Multiple-input and multiple-output (MIMO) is a technique that enhances radio link capacity by utilizing multiple transmit and receive antennas to leverage multipath propagation [38, 34]. MIMO is integral to various wireless communication standards such as IEEE 802.11n/ac (Wi-Fi 4/5), HSPA+ (3G), WiMAX, and LTE. MIMO techniques allow for simultaneous transmission and reception of multiple data signals over the same radio channel, exploiting signal propagation differences between antennas. Multi-user MIMO (MU-MIMO) extends this concept by sending distinct data signals to different receivers.

Frequency Bands, BLER and MCS metrics

In telecommunications, a frequency band is a specific range of frequencies in the radio frequency (RF) spectrum that is allocated for a particular type of communication or wireless service. These bands are designated by regulatory authorities, such as the Federal Communications Commission (FCC) in the United States or the European Telecommunications Standards Institute (ETSI) in Europe, to ensure organized and interference-free communication.

Different frequency bands have varying characteristics: lower frequencies can cover longer distances and penetrate obstacles more effectively, while higher frequencies can offer higher data rates but may have a more limited range.

Used frequency bands in the testing:

- B1 (2100 MHz Band): This is one of the most widely used bands for 3G (UMTS) and 4G (LTE) technologies. It operates in the range of 1920-1980 MHz for uplink and 2110-2170 MHz for downlink.
- B3 (1800 MHz Band): This band is also widely used for 2G (GSM), 3G (DCS), and 4G (LTE) services. It operates in the range of 1710-1785 MHz for uplink and 1805-1880 MHz for downlink.
- B27 (800 MHz Band): This band is used for LTE services and is known for its good coverage and building penetration characteristics. It operates in the range of 807-824 MHz for uplink and 852-869 MHz for downlink.
- N1 (2100 MHz NR Band): Similar to B1 in LTE, the N1 band operates in the frequency range of 1920-1980 MHz for uplink and 2110-2170 MHz for downlink. It is one of the bands identified for the deployment of 5G networks, ensuring a smooth transition and coexistence with existing 4G LTE networks.

Block Error Rate (BLER) is a metric used in wireless communication to quantify the quality of a transmitted data stream. It measures the proportion of incorrectly received data blocks relative to the total number of transmitted blocks within a specific time period. A lower BLER indicates better signal quality and more reliable data transmission.

Modulation and Coding Scheme (MCS) is a parameter used in wireless communication that defines the combination of modulation type and error correction coding rate used for transmitting data. The MCS index typically ranges from 0 to 31, and each index corresponds to a specific combination of modulation (e.g., QPSK, 16QAM, 64QAM) and coding rate (e.g., 1/2, 3/4).

4.3 First Test Case

In the first test case, the device was set up to use three LTE cells with the Carrier Aggregation feature, where PCell was on B3, SCell_1 was on B28, and SCell_2 was on B1 frequency band. Every diagram shows two identical measurements after each other, that is why the line drops to almost zero at half-time. The first diagram (see Figure 4.3) shows the total user throughput, while the other ones Figure 4.4, Figure 4.5 and Figure 4.6 show each cells Physical Data Shared Channel (PDSCH) throughputs. In this case, the device approached the theoretical maximum with a few percent per-cell speed reduction, achieving an MCS27 value with less than 5% BLER per cell.

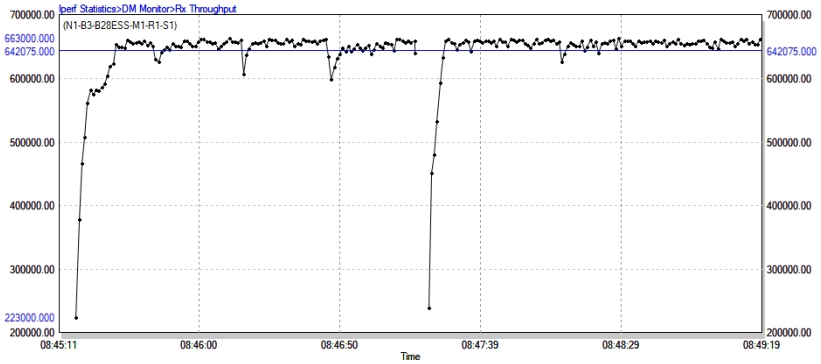

Figure 4.3: Total User Throughput (kbps)

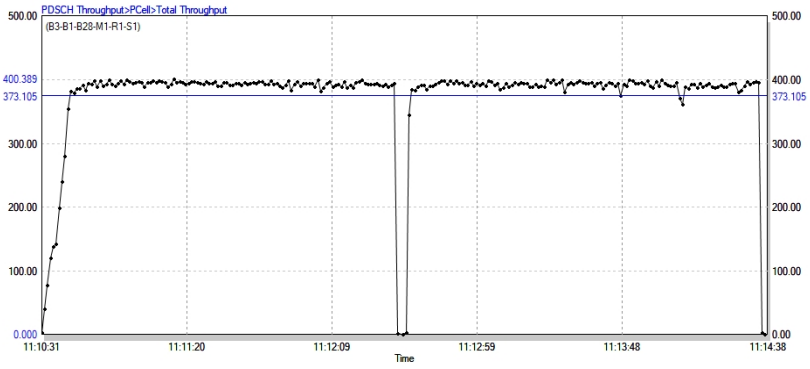

Figure 4.4: PDSCH Throughput - PCell - B3

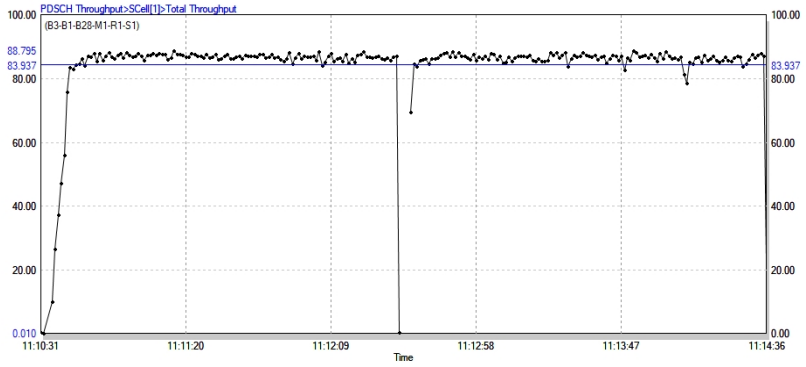

Figure 4.5: PDSCH Throughput - SCell_1 - B28

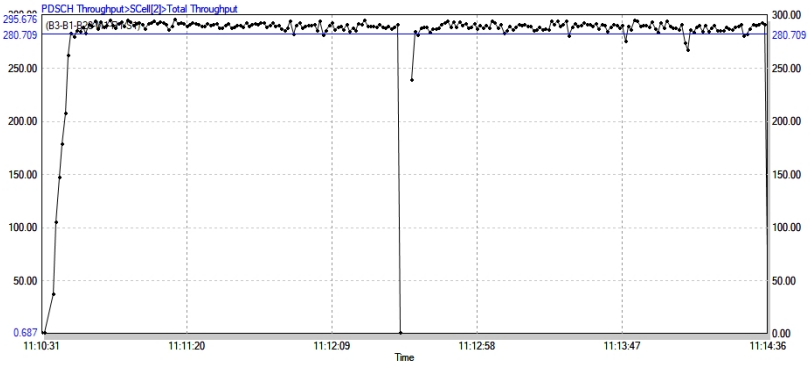

Figure 4.6: PDSCH Throughput - SCell_2 - B1

4.4 Second Test case

In the second test, the device was set up to not have access to the B1 cell. Instead, we used an NR cell with the same bandwidth and configuration, with the LTE PCell being B3 and SCell B28 again. This time, the LTE side continued to weaken negligibly (see Figure 4.7, Figure 4.8, Figure 4.9), but the N1 NR cell has four notable sharp drops (see Figure 4.10) where the throughput goes down drastically. In one case the throughput dropped to almost 50% of its theoretical maximum, as you can see in Figure 4.10.

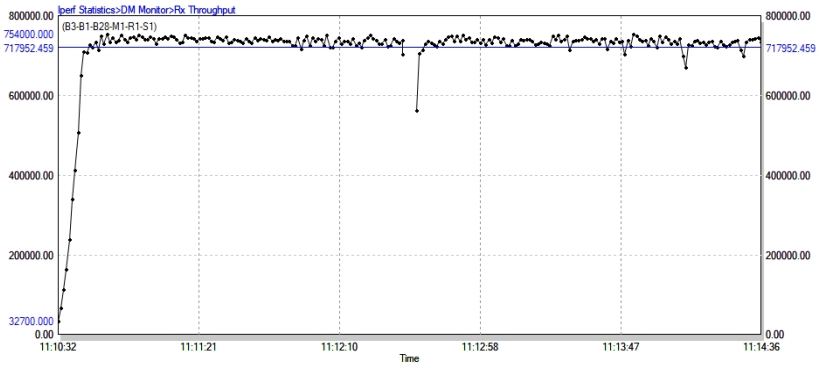

Figure 4.7: Total User Throughput (Mbps)

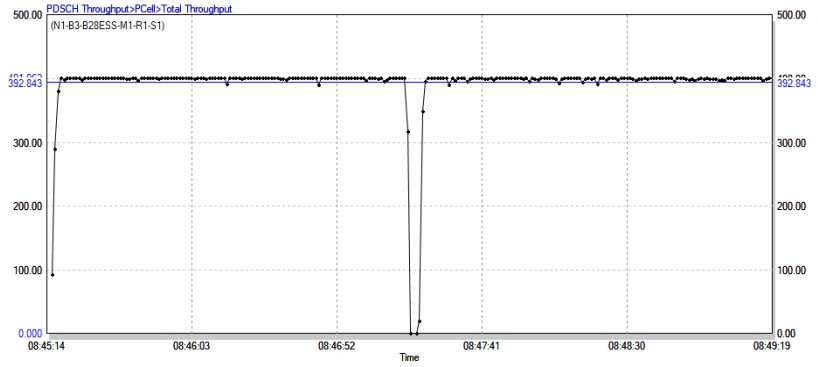

Figure 4.8: PDSCH Throughput - PCell - B3

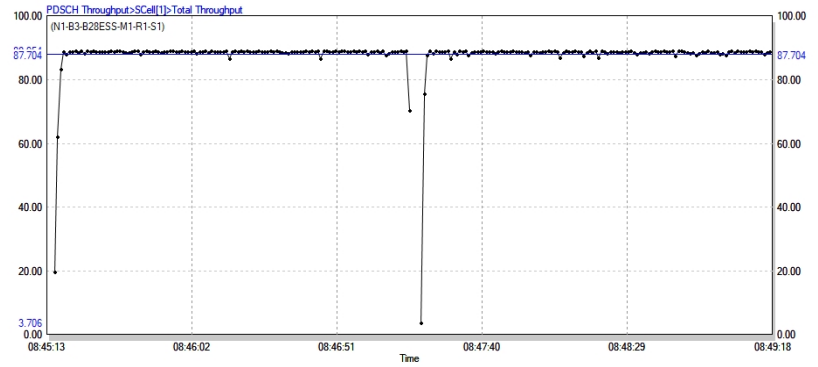

Figure 4.9: PDSCH Throughput - SCell_1 - B28

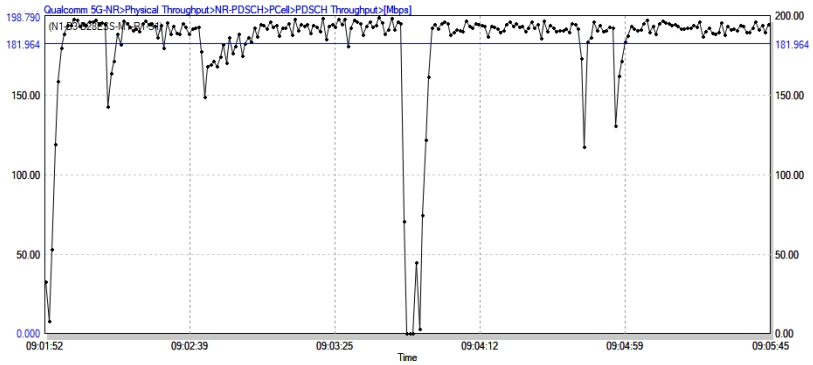

Figure 4.10: PDSCH Throughput - NRCell - NR1

Chapter 5

Discussion

In this chapter, I discuss the results and the presumed explanation of my throughput tests, robustness, scalability and future enhancements of the proposed solution.

5.1 Analysis and Presumed Explanation of the Results

These results may have happened because the device's HW/SW configuration is more sensitive to interference in 5G technology, causing an increase in BLER and SINR. To reduce these, the UE will limit its modulation depth, which we measure with the MCS number. With this, the device will be unable to achieve higher speeds but will not generate as much noise for itself and surrounding devices. Additionally, the device may increase its power up to the value specified in the standard, but this will generate additional noise.

It is clearly visible that on the last graph (see Figure 4.10), the NR on 2100 Mhz is weaker with an offset compared to the LTE on the same frequency (see Figure 4.6), and drops in download speeds are also visible, due to reduced MCS to maintain the target Signal-Noise Ratio (SINR). These observations are valuable, as we believe that NR cells are at least the same or better by default then LTE cells, due to the fact that NR spectrum usage is more efficient in general. These discrepancies can be observed because this phone has a lower noise tolerance and receives more weakly under the same radio environment.

This type of issue can often be resolved with a software update, partly because the antenna system is designed so that transmission and reception occur on multiple antennas and a different antenna pair may be more sensitive per carrier. Also, the device's behavior can be modified at any time by optimizing resource utilization, in compliance with the relevant standards.

In summary, the testing activity resulted that in case of LTE with 3 carriers in a CA setup, more per-cell speed can be achieved on the same bandwidths than with 1 NR + 2 LTE cells. These results were revealed during the `iperf3` measurements - using my proposed solution - and can be attributed to the fact that the phone supports the use of 3 specific carrier frequencies on both NR and LTE, but has different physical capabilities for each technology, thereby resulting in a reduced Overall Throughput and lower N1 layer throughput. These observations indicated the capacity defects caused by the specific hardware components in the UE.

5.2 Robustness

In this section, I will present the results of experiments and evaluations conducted on the proposed solution, focusing on its scalability and robustness.

A critical aspect of robustness involves ensuring that the solution is adaptable to future modifications, thereby accommodating evolving customer requirements within the selected technological framework and stack. A potential consideration may involve extending the application's capabilities beyond initiating `iperf3` servers to include other types of measurement servers. As explored in Section 2.2, there exist several powerful tools besides `iperf3` that could be employed. The potential integration of diverse tools could enhance the range of perspectives supported by the application or respond to shifts in specific requirements. The virtualization strategy implemented using `Docker` containers, along with the dynamic `Docker`-image creation functionality, effectively addresses this need.

An additional dimension of robustness applies to input validation within the frontend interface. As presented in Section 3.2, the frontend incorporates multiple input validations, as illustrated in Figure 3.7. Furthermore the AddServer component automatically re-renders to display the first available port number, as depicted in Figure 3.8.

5.3 Scalability

To evaluate the scalability of the application, I focused on assessing whether the frontend's performance would be impacted by horizontal scaling. Specifically, I examined its ability to manage an increased number of `iperf3` server containers. The initial configuration allowed for 5 servers per user, and I expanded this setup to accommodate 10 servers. The test results demonstrated that the frontend maintained its responsiveness without any slowdowns, indicating that horizontal scaling had no adverse effect on the user experience. The consistency in the frontend component's speed did not degrade at all, aligning with expectations. This can be attributed to the fact that the frontend is not engaged in heavy computation but is merely fetching data from the backend and populating a list to present to the user.

In the initial phase of assessing the system's performance, I conducted load testing to understand how the application behaves under stress. For this purpose, I initiated 10 containers and, from another server, concurrently initiated `iperf3` measurements on each server. During this process, I monitored various metrics such as CPU and memory utilization, as well as packet drop rates on the host server, using a combination of a Python script and a Bash script.

The results from these tests were insightful. At maximum load, the CPU utilization peaked at around 8%, while memory usage hovered between 4-5%. Importantly, there were no packet drops recorded. Comparing these results with tests run on native `iperf3` servers, which operated directly on the physical servers and not in a container, I observed similar utilization metrics. This leads me to conclude that there is no significant overhead introduced by the virtualization layer.

In order to optimally run the application, it is essential to host it on a server equipped with a sufficiently powerful CPU and sufficient amount of memory. Additionally, it is crucial to ensure that the network infrastructure used by clients to connect to the servers is robust and capable, preventing any bottlenecks during measurements.

5.4 Future Work

The existing solution serves as a functional application on its own, but there are plans for further enhancements to optimize the entire measurement process by introducing several novel elements. One such enhancement entails the integration of a Grafana dashboard, enabling real-time monitoring of various metrics, such as server utilization during measurements, to prevent possible conflicts with concurrent measurements. Implementing this would necessitate the deployment of a Telegraf server and a Grafana dashboard.

Furthermore, in the realm of authentication and authorization, there is a proposal to incorporate two-factor authentication and synchronize with the company's LDAP, thereby streamlining the authentication process for internal users. External users, who may not have access to this specific LDAP, would continue to utilize the local authentication mechanism.

Another significant aspiration is to enhance the portability of the entire application. Leveraging `Docker` technology is under consideration for this purpose, with the aim of facilitating quick deployment and setup on any server. The plan involves containerizing the entire application, encompassing both frontend and backend components, and bundling them into a `docker-compose` solution. This approach ensures that the benefits of the application extend beyond Magyar Telekom Nyrt., potentially reaching engineers at other telecommunications providers within the Deutsche Telekom group.

In terms of future directions, it would be valuable to explore opportunities for automating the scaling of server resources based on real-time demand and implementing advanced data analytics capabilities to derive deeper insights from the measurement data. Additionally, assessing the feasibility of integrating with other cloud platforms and extending support for more network performance measurement tools could further elevate the application's utility and versatility.

Chapter 6

Conclusion

In conclusion, the development and implementation of the application have demonstrated its significant value and relevance in the field of network performance testing and analysis. The application, built with a `Vue.js` frontend for managing `Dockerized` `iperf3` servers, showcases a well-rounded architecture designed to address the specific needs of testing in networked environments, particularly in RAN scenarios.

The identification of specific hardware and software issues through the presented real-world test case underscored the relevance and usability of the application, affirming it as a robust tool for testing and validating new RAN features. This test case not only highlighted the application's practical significance but also positioned it as a crucial asset for ensuring the quality and reliability of future RAN tests.

By initiating multiple containers and conducting parallel `iperf3` measurements, the tests convincingly demonstrated that the application could handle increased loads efficiently. Metrics such as CPU utilization, memory usage, and packet drop rates were monitored, revealing that the virtualization layer introduced no significant overhead.

Attention to security was also a noteworthy aspect of the project. By incorporating JWT for user authentication, the application ensured secure and authenticated access, thereby addressing potential vulnerabilities. Various common security pitfalls were carefully considered and preemptively mitigated.

Moreover, in the project, careful consideration was given to ensuring both robustness and scalability. With a focus on user experience, the frontend was designed to maintain its resilience and responsiveness even when the number of managed `iperf3` server containers increased.

In summary, this application has the potential to be a major tool when it comes to development, testing, and optimization of new RAN features and other network enhancements.

Acknowledgements

I would like to extend my sincere gratitude to my advisors, Dr. Adrián Pekár and Márton Pósfay for their invaluable guidance and feedback throughout this project. Additionally, I owe a great deal of thanks to my family for their unwavering support and encouragement during this endeavor.

Bibliography

[1] Vuetify. https://vuetifyjs.com/en/. Accessed: 2023-10-31.

[2] Angular. URL https://angular.io/. Accessed: 2023-10-31.

[3] Beego. URL https://pkg.go.dev/github.com/beego/beego. Accessed: 2023-10-31.

[4] Vm vs. container. URL https://www.netapp.com/blog/containers-vs-vms/. Accessed: 2023-10-31.

[5] Cri-o. URL https://cri-o.io/. Accessed: 2023-10-31.

[6] Docker. URL https://www.docker.com/. Accessed: 2023-10-31.

[7] Echo. URL https://echo.labstack.com/. Accessed: 2023-10-31.

[8] Fiber - an express-inspired web framework written in go. URL https://gofiber.io/. Accessed: 2023-10-31.

[9] Gin. URL https://gin-gonic.com/. Accessed: 2023-10-31.

[10] Go: The go programming language. URL https://go.dev/. Accessed: 2023-10-31.

[11] Gorilla/mux, . URL https://pkg.go.dev/github.com/gorilla/mux. Accessed: 2023-10-31.

[12] Gorm - the fantastic orm library for golang, . URL https://gorm.io/index.html. Accessed: 2023-10-31.

[13] Lxc - a userspace interface for the linux kernel containment features. URL https://linuxcontainers.org/lxc/. Accessed: 2023-10-31.

[14] Php. URL https://www.php.net/. Accessed: 2023-10-31.

[15] Podman. URL https://podman.io/. Accessed: 2023-10-31.

[16] Python. URL https://www.python.org/. Accessed: 2023-10-31.

[17] React. URL https://react.dev/. Accessed: 2023-10-31.

[18] Revel - a flexible web framework for the go language. URL https://revel.github.io/. Accessed: 2023-10-31.

[19] Ruby: An open source programming language. URL https://www.ruby-lang.org/en/. Accessed: 2023-10-31.

[20] Session vs token based authentication. URL https://dev.to/thecodearcher/what-really-is-the-difference-between-session-and-token-based-authentication-2o39. Accessed: 2023-10-31.

[21] Tojoin. URL https://www.chbutc.com/. Accessed: 2023-11-01.

[22] Vuejs. URL https://vuejs.org/. Accessed: 2023-10-31.

[23] Zied Aouini and Adrian Pekar. Nfstream: A flexible network data analysis framework. *Computer Networks*, 204:108719, 2022. ISSN 1389-1286. DOI: 10.1016/j.comnet.2021.108719.

[24] Ethan Blanton, Dr. Vern Paxson, and Mark Allman. TCP Congestion Control. RFC 5681, September 2009. URL https://www.rfc-editor.org/info/rfc5681.

[25] J. Case, M. Fedor, M. Schoffstall, and J. Davinn. Simple network management protocol (snmp). RFC 1157, RFC Editor, May 1990. URL http://www.rfc-editor.org/rfc/rfc1157.txt.

[26] B. Claise. Cisco systems netflow services export version 9. RFC 3954, RFC Editor, October 2004. URL http://www.rfc-editor.org/rfc/rfc3954.txt.

[27] B. Claise, B. Trammell, and P. Aitken. Specification of the ip flow information export (ipfix) protocol for the exchange of flow information. RFC 7011, RFC Editor, September 2013. URL http://www.rfc-editor.org/rfc/rfc7011.txt.

[28] Erik Dahlman, Stefan Parkvall, and Johann Sköld. *4G LTE / LTE-Advanced for Mobile Broadband*. Elsevier, 2014. ISBN 9780124199859.

[29] Emulex. Endacedag packet capture cards, 2014. URL http://www.emulex.com/products/network-visibility-products-and-services/endacedag-data-capture-cards/features/.

[30] Fluke. Fluke networks, 2014. URL http://www.flukenetworks.com/content/Centralized-Performance-Monitoring-and-Trou\protect\discretionary{\char\hyphenchar\font}{}{}bleshooting.

[31] Harri Holma, Antti Toskala, and Pablo Tapia. *LTE-Advanced: 3GPP Solution for IMT-Advanced*. Wiley, 2012. ISBN 9781119974055.

[32] IPCopper. Ipcopper, inc., 2014. URL https://www.ipcopper.com/product_itr1000.htm.

[33] M. Jones, J. Bradley, and N. Sakimura. Json web token (jwt). RFC 7519, RFC Editor, May 2015. URL https://www.rfc-editor.org/rfc/rfc7519.txt. URL: https://www.rfc-editor.org/rfc/rfc7519.txt.

[34] Keivan Kaboutari and Vahid Hosseini. A compact 4-element printed planar mimo antenna system with isolation enhancement for ism band operation. *AEU - International Journal of Electronics and Communications*, 134:153687, 2021. DOI: 10.1016/j.aeue.2021.153687.

[35] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, and Scott Shenker. Onix: A distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX Conference on Operating Systems Design*

and Implementation, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.

[36] Hyo-Jin Lee, Myung-Sup Kim, James W Hong, and Gil-Haeng Lee. Qos parameters to network performance metrics mapping for sla monitoring. *KNOM Review*, 5(2): 42–53, 2002.

[37] S. Ligus. *Effective Monitoring and Alerting: For Web Operations*. O'Reilly Media, 2012. ISBN 9781449333485.

[38] Hermann Lipfert. Mimo ofdm space time coding – spatial multiplexing, increasing performance and spectral efficiency in wireless systems, part i technical basis. Technical report, Institut für Rundfunktechnik, Aug 2007.

[39] J Moy. Ospf version 2. RFC 2328, RFC Editor, April 1998. URL http://www.rfc-editor.org/rfc/rfc2328.txt.

[40] NetScout. Netscout systems, inc., 2014. URL http://www.netscout.com/solutions/enterprise/SDMC/Pages/Network_Monitoring_Fabric.aspx.

[41] Thomas O. Olwal, Karim Djouani, and Anish M. Kurien. A survey of resource management toward 5g radio access networks. *IEEE Communications Surveys & Tutorials*, 18(3):1656–1686, 2016. DOI: 10.1109/COMST.2016.2550765.

[42] L.L. Peterson and B.S. Davie. *Computer Networks: A Systems Approach*. Elsevier Science, 2011. ISBN 9780123850607.

[43] Y. Rekhter, T. Li, and S. Hares. A border gateway protocol 4 (bgp-4). Internet-Draft 4271, IETF Secretariat, January 2006. URL http://www.rfc-editor.org/rfc/rfc4271.txt.

[44] G. Sadasivan, N. Brownlee, B. Claise, and J. Quittek. Architecture for ip flow information export. RFC 5470, RFC Editor, March 2009. URL http://www.rfc-editor.org/rfc/rfc5470.txt.

[45] P. Smith, David Hutchison, J.P.G. Sterbenz, M. Schöller, A. Fessi, M. Karaliopoulos, C. Lac, and B. Plattner. Network resilience: a systematic approach. *Communications Magazine, IEEE*, 49(7):88–97, July 2011. ISSN 0163-6804. DOI: 10.1109/MCOM.2011.5936160.

[46] TcpDump. A powerful command-line packet analyzer, 2013. URL http://www.tcpdump.org/.

[47] S. C. G. Tsinos, F. Foukalas, T. Khattab, and L. Lai. On channel selection for carrier aggregation systems. *IEEE Transactions on Communications*, 66(2):808–818, Sep 2017.

[48] L. Vokorokos, A. Kleinová, and O. Látka. Network security on the intrusion detection system level. In *Proceedings of the 10th IEEE International Conference on Intelligent Engineering Systems*, INES '06, pages 270–275. IEEE, 2006. ISBN 1-4244-9708-8.

[49] L. Vokorokos, N. Ádám, and A. Baláž. Application of intrusion detection systems in distributed computer systems and dynamic networks. In *Proceedings of Computer Science and Technology Research Survey*, CST '08, pages 19–24, 2008. ISBN 978-80-8086-100-1.

[50] S. Waldbusser, R. Cole, C. Kalbfleisch, and D. Romascanu. Introduction to the remote monitoring (rmon) family of mib modules. RFC 3577, RFC Editor, August 2003. URL http://www.rfc-editor.org/rfc/rfc3577.txt.

[51] Wireshark. Network protocol analyzer, 2013. URL http://www.wireshark.org/.