Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Christian Lewin
Sebastian Fuglesang

A Framework for Benchmarking a Private 5G Network Exemplified with Industry 4.0 Use Cases

Master's thesis in Cyber Security and Data Communication
Supervisor: Thomas Zinner
Co-supervisor: Stanislav Lange, Waqas Ikram
June 2024

NTNU
Norwegian University of Science and Technology

Christian Lewin
Sebastian Fuglesang

A Framework for Benchmarking a Private 5G Network Exemplified with Industry 4.0 Use Cases

Master's thesis in Cyber Security and Data Communication
Supervisor: Thomas Zinner
Co-supervisor: Stanislav Lange, Waqas Ikram
June 2024

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

Title: A Framework for Benchmarking a Private 5G Network Exemplified with Industry 4.0 Use Cases

Student: Fuglesang, Sebastian and Lewin, Christian

Problem description:

A private 5G network is a 5G network deployed to be used primarily by a private organization to provide wireless connection within an area. Private 5G networks can be deployed and configured in a multitude of different ways. For example, it can be built using a mix of LTE and 5G equipment, or only 5G equipment, it can use different classes of frequency bands and have varying configuration setups. Furthermore, 5G is only a specification and the actual equipment is made by companies who often have different proprietary implementations. However, the specifications and many vendors promise high performance in key performance metrics such as delay and throughput, but it is unclear to which extent this holds in practice. As the private 5G networks gain adoption in Industry 4.0 use cases, there is a lack of a common consensus on whether the performance of the network adheres to the use case requirements. Therefore, a methodology to benchmark the performance of a private 5G network would benefit private 5G adopters, when considering what applications can be realistic to run on their network.

Our Master thesis will focus on creating software able to perform reproducible evaluations of a private 5G network, using example use cases relevant to Industry 4.0. For this to be accomplished, a common testbed architecture must be defined. The architecture must be able to capture relevant packet-level metrics, such as throughput, one-way delay and inter-packet delay variation. Because these networks commonly consist of closed vendor equipment, such a methodology must treat the network as a black box. To keep the form factor attractive, the framework should only require minimal modifications of the network.

The framework this Master thesis will produce will be based on an incremental development of the software tool. Firstly, we will design and implement a testbed architecture that is able to measure the metrics of interest. Secondly, we will develop a prototype of the tool that will generate the traffic and perform the measurements we need. Thirdly, we will seek to validate this benchmarking framework. It will be evaluated based on a set of functional and non-functional requirements. Finally, we will attempt to optimize and further generalize our benchmarking framework.

The Master thesis will provide a framework that can be applied to capture the impact of private 5G development on the performance of relevant 5G metrics. This

will help potential adopters to learn about the performance of 5G and its potential in a realistic environment and provide a common benchmark for comparison with other networks.

Approved on:	2024-04-10
Main supervisor:	Professor Zinner, Thomas, NTNU
Co-supervisor:	Lange, Stanislav, NTNU and Ikram, Waqas, ABB

Abstract

5G is envisioned to cater to a plethora of heterogeneous use cases, ranging from delay-sensitive applications relying on Ultra-Reliable Low Latency Communication (URLLC), bandwidth-intensive applications relying on Enhanced Mobile Broadband (eMBB), to energy-conserving applications relying on Massive Machine Type Communication (mMTC). These use cases enable the fulfillment of several of the stringent requirements of Industry 4.0. Therefore, private 5G networks are considered a promising communication technology for Industry 4.0 use cases. However, it is unclear whether the technical realizations of private 5G networks comply with their envisioned requirements. Furthermore, there seems to be no standard, open-source solution to produce measurements for comparing such networks. This thesis explored the design and implementation of a benchmarking framework for private 5G networks. The work followed a feedback-loop cycle, focusing on improving the produced artifacts based on insights from validation.

A benchmarking framework consisting of a software tool and a high-level architecture capable of generating, capturing, and analyzing network traffic has been produced and validated. We have combined existing software utilities with developing custom solutions, leveraging the trade-off between resources spent on the development and validation of the tool and its customizability. The software implementation was validated based on its functional and non-functional requirements. This provided insights enabling enhancements of the benchmarking tool. Furthermore, the efficacy of the benchmarking framework towards a set of common Industry 4.0 use cases has been evaluated on two private 5G networks. The evaluation showed that the benchmarking framework is able to produce benchmarks of networks based on real-world scenarios and that the results are comparable across networks. However, some work remains to enable the benchmarking framework to leverage its testbed architecture maximally, accurately represent Industry 4.0 use cases, and enhance the granularity of comparisons between network implementations.

Sammendrag

5G er tenkt å imøtekomme flere heterogene bruksområder, alt fra forsinkelsessensitive applikasjoner basert på URLLC, båndbreddeintensive applikasjoner basert på eMBB, til energibesparende applikasjoner basert på mMTC. Disse egenskapene gjør det mulig å oppfylle flere av de strenge kravene som Industry 4.0 stiller. Derfor anses private 5G-nettverk som en lovende kommunikasjonsteknologi for bruksområder innen Industry 4.0. Det er imidlertid uklart om de tekniske realiseringene av private 5G-nettverk i virkeligheten oppfyller de forutsatte kravene. I tillegg ser det ikke ut til å finnes noen standardløsning med åpen kildekode for å produsere målinger som gjør det mulig å sammenligne slike nettverk. Denne avhandlingen utforsker derfor utformingen og implementeringen av et rammeverk for benchmarking av private 5G-nettverk. Arbeidet følger en tilbakemeldings-syklus, med fokus på å forbedre programvaren basert på innsikten fra validering.

Vi har utviklet og validert et rammeverk for benchmarking bestående av programvare og en høynivå arkitektur som kan generere, fange opp og analysere nettverkstrafikk. Ved å benytte eksisterende programvare kombinert med egenutviklede løsninger, har vi utnyttet balansen mellom ressursbruk til utvikling og validering versus muligheten til å skreddersy løsninger. Programvaren ble validert på grunnlag av sine funksjonelle og ikke-funksjonelle krav. Valideringen ga innsikt som muliggjorde forbedring av verktøyets funksjonalitet. I tillegg har vi evaluert til hvilken grad rammeverket muliggjør benchmarking av vanlige scenarioer i Industry 4.0 i to private 5G-nettverk. Evalueringen viste at rammeverket er i stand til å benchmarke nettverk basert på virkelige scenarier, og at resultatene er sammenlignbare på tvers av nettverk. Det gjenstår imidlertid en del arbeid for å utnytte høynivå arkitekturen maksimalt, representere brukstilfeller fra Industry 4.0 presist og forbedre detaljnivået i sammenligningene mellom nettverk.

Preface

This thesis was written to conclude the master's degree in Cyber Security and Data Communication class of 2024 at the Norwegian University of Science and Technology.

We want to thank our supervisor, Thomas Zinner, and our co-supervisor, Stanislav Lange, for providing valuable feedback, insight, and support. Furthermore, we want to thank Waqas Ikram from our industry partner, ABB, for giving us access to industry perspectives. We also want to thank Pål Sturla Sæther for valuable support with the technical realization of our testbeds.

I want to thank my parents for their invaluable motivation throughout this degree. Moreover, I want to thank my friends and classmates for stimulating discussions and for making these years exciting and enjoyable.

Christian Lewin

I personally want to thank my parents for their continued encouragement and support through all these years. Finally, I want to thank my partner, Beatrice, and our children for their support through this study program.

Sebastian Fuglesang

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

3GPP Third Generation Partnership Project.

5GC 5G Core.

AMF Access Management Function.

API Application Programming Interface.

BBU Digital Baseband Unit.

CBR Constant Bit Rate.

CDF Cumulative Distribution Function.

DPDK Data Plane Development Kit.

DUT Device Under Test.

ECDF Empirical Cumulative Distribution Function.

eMBB Enhanced Mobile Broadband.

EPC Evolved Packet Core.

gNB Next Generation NodeB.

GTP GPRS Tunneling Protocol.

IAT Inter-Arrival Time.

ICMP Internet Control message Protocol.

IoT Internet of Things.

IP Internet Protocol.

IPDV IP Packet Delay Variation.

ITU-R International Telecommunications Unit Radiocommunications Sector.

KPI Key Performance Indicator.

Linux NAPI Linux New API.

LTE Long-Term Evolution.

MAC Medium Access Control.

MIMO Multiple Input Multiple Output.

mMTC Massive Machine Type Communication.

NIC Network Interface Card.

NR New Radio.

NSA Non-Standalone.

NTNU Norwegian University of Science and Technology.

NUT Network Under Test.

OS Operating System.

OWD One-Way Delay.

PPS Packets Per Second.

RAN Radio Access Network.

RRC Radio Resource Control.

RRH Remote Radio Head.

RTP Real-Time Transport Protocol.

SA Standalone.

SDR Software Defined Radio.

SFP Small Form-factor Pluggable.

SMF Session Management Function.

SSH Secure Shell.

TCP Transport Control Protocol.

UDP User Datagram Protocol.

UE User Equipment.

UPF User Plane Function.

URLLC Ultra-Reliable Low Latency Communication.

Chapter 1
Introduction

This chapter serves as an introduction and overview of the thesis. Section 1.1 describes the motivation for our thesis, and Section 1.2 presents its outcomes. Then, Section 1.3 introduces the research questions of the thesis, before Section 1.4 presents the structure of the thesis. Finally, Section 1.5 discusses how the thesis can contribute to the UN Sustainability Development goals.

1.1 Motivation

The motivation for this thesis is inspired by the work from our pre-project thesis [1]. We have included some of this to provide context below.

The latest mobile technology standard, 5G, is envisioned to support a broad range of usage scenarios, such as eMBB, URLLC, and mMTC [2]. This is a type of flexibility that is new to mobile technologies and enables a vast range of heterogeneous use cases [3].

The forthcoming of Industry 4.0 puts stringent requirements with regard to the Quality of Service (QoS) of communication systems [3]. For example, industrial automation and robotics pose strict requirements for the latency and packet delay variation (PDV) of application traffic. Until now, manufacturing plants have relied on wired communication technologies to achieve the required performance. Wired communication, however, introduces issues related to both scalability and mobility. Requiring a wired connection to sensors, actuators, and robotics necessitates a free wired connection for the unit, regardless of the fraction of the traffic it will use. Moreover, the wire becomes a physical limitation on where the unit can be placed and moved.

A method companies can utilize to solve or mitigate the challenges of using wired connectivity while still meeting the stringent requirements of Industry 4.0 is deploying private 5G networks. The envisioned use cases and requirements for 5G

coincide with those of Industry 4.0 to a great extent [2]. A private 5G network is a local area network for dedicated wireless connectivity within a specified area, built with 5G technology [3]. These networks can provide URLLC, mMTC as well as eMBB while still maintaining security and customized predictable QoS [4]. Moreover, the networks can be tailored to meet the needs of the organization, thus enabling adaption to the various Industry 4.0 use cases.

Private 5G networks include a multitude of deployment options and configuration parameters. The options range from which 5G deployment option to use, i.e., whether to leverage Long-Term Evolution (LTE) infrastructure or to solely deploy 5G infrastructure, to the degree of integration with mobile service providers. Moreover, adopters must choose whether to utilize licensed, unlicensed, or shared spectrum for their Radio Access Network (RAN). This affects the financial aspects of the network, as well as the level of isolation in the radio network. In addition to architectural options, device vendors vary greatly and often offer closed equipment and proprietary interfaces, making interworking between devices from different vendors difficult. 5G has also seen the introduction of several open-source software implementations of the various infrastructure components [3], which may introduce different characteristics. Furthermore, each component has a plethora of configuration options to enable the aforementioned flexibility. The sum of this is that deploying a private 5G network involves a lot of options, and it can be difficult to determine how the interworking of architectural choices, device choices, and configuration choices impact the overall performance of the network.

To our knowledge, no standardized way of testing 5G network performance to compare the different deployment and configuration options exists. This would enable a comparison of the real-world performance of deployments with regard to specific use cases. Such a comparison would aid potential adopters of private 5G networks in designing their network, as they could compare their application requirements to the actual recorded performance.

1.2 Thesis Outcomes

This thesis aims to enable the comparison of the deployments of private 5G networks through the following contributions:

1. A comprehensive design description of the benchmarking framework
2. A technical, open-source realization of the benchmarking framework [1]
3. Validation of the technical realization

[1] https://github.com/Private-5G-benchmarking/benchmarking_tool

4. A case study of the benchmarking framework based on its application on two private 5G networks

To enable comparisons of private 5G networks, we intend to develop a benchmarking framework that can perform reproducible benchmarks. It will consist of a high-level testbed architecture and a software tool that executes on the architecture. We will refer to the combination of the testbed architecture and software tool as the benchmarking framework. The software tool that performs the benchmarking will be referred to as the benchmarking tool. We believe that this framework can enable comparing the possible deployments of private 5G en masse, supporting an acceleration in adopting these networks. To ensure that the technical realization performs according to our design goals we conduct validation on several components and the framework as a whole. Finally, the ability of the framework to compare private 5G networks will be tested by performing a case study, where the framework is used to compare the performance of two real-world networks.

1.3 Research Questions

The decisions and reflections made in this thesis center around a set of four research questions, presented in this section.

RQ1 How can a system capable of performing reproducible benchmarking of private 5G networks be designed?

RQ2 How can the requirements for integration with the NUT and the granularity of information provided by the benchmark be balanced?

RQ3 To what extent can such a system support benchmarking of a defined set of common Industry 4.0 use cases?

RQ4 How can the framework be used to compare 5G implementations?

The first research question provides a basis for discussing the benchmarking of private 5G networks in general. This is the overarching research question of the thesis. Performing such a benchmark involves developing a framework consisting of both a testbed architecture and software tools for producing reproducible results. Without reproducibility, the quality of comparisons between benchmarks is harder to ensure.

The second research question considers the balance between integration into the NUT and the depth of information provided. The NUT refers to the network that is

being benchmarked. It is challenging to provide much detail without measuring at multiple points in the network or having access to any network internals. However, each measurement point added, or network internal that is accessed, increases the requirements for integration with the NUT. If these requirements are high, it takes a longer time to set up on a new network and is more error-prone. Furthermore, depending on the access to the network that can be achieved, the framework might not be possible to set up in certain networks.

The third research question investigates to what extent the framework can support use cases relevant to Industry 4.0. Since one of the main groups of users considered for this framework is industrial adopters, this question is used to explicitly tie the functionality of the framework to Industry 4.0.

The final research question seeks to investigate how the results produced by the framework can enable comparisons between networks. This ties into both the reproducibility aspect of the framework and the depth of information provided by the benchmarks.

1.4 Thesis Structure

The work in this thesis has been performed in accordance with the feedback loop in Figure 1.1. Before we began our iterations of the feedback loop, we investigated the relevant topics in the literature. Our findings are shown in Chapter 2. In the first iteration of the loop, we made the initial design and technical realization of the benchmarking framework. This is presented in Chapters 3 and 4. The outcome of this was an artifact that was validated on a small scale, according to the methodology presented in Chapter 5. This generated results that were used to infer the performance of the artifact and identify areas for improvement. The results from the validation are presented in Chapter 6 and discussed in Chapter 7. This concluded the first feedback loop iteration. After this, we returned to re-designing the artifact to address the areas for improvement, which is described in Chapter 4. Following this, we perform tests on real-world networks, which we refer to as the case study. Chapter 5 describes the methodology of the case studies, and Chapter 6 presents their results. A discussion regarding the networks and the inferences made about the framework is made in Chapter 7. Finally, the insights gained through this iteration are concluded and presented as future work in Chapter 8.

1.5 Supporting the UN Sustainable Development Goals

Our thesis will explore research questions regarding how to perform benchmarking of private 5G networks. As previously discussed, there is some uncertainty about the real-world performance of private 5G networks due to their complexity. Thus,

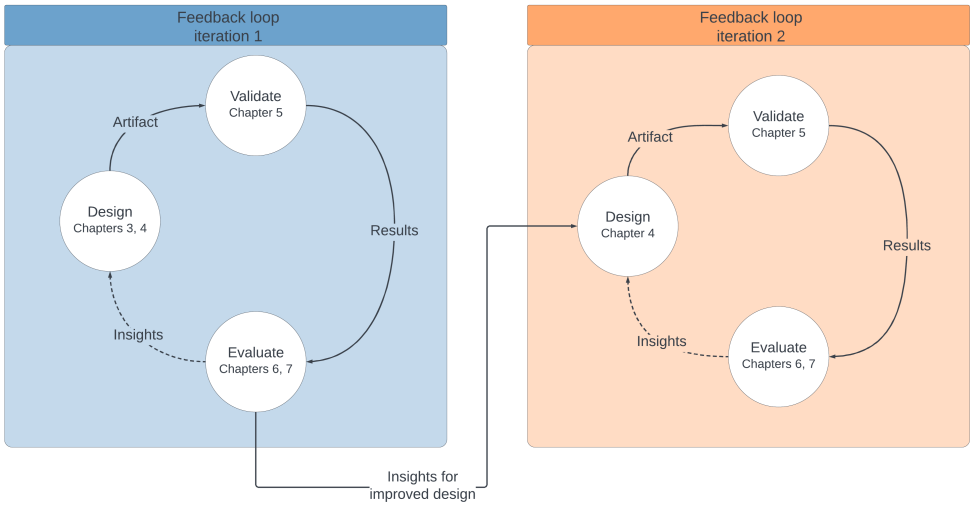


Figure 1.1: Illustration of the structure of the thesis, inspired by a feedback-loop.

benchmarking the performance of private 5G networks can provide Key Performance Indicators (KPIs) about the real-world performance of specific networks. Based on these benchmarks, one can decide if a network is usable for a particular use case. Adopting private 5G can improve performance compared to other communication technologies and can support important use cases such as Industry 4.0 and smart grids. Doing this contributes to several of the UN Sustainable Development Goals. Below, we highlight some of the specific subgoals that could benefit from this technological advancement.

Subgoal 8.2 ("Achieve higher levels of economic productivity through diversification, technological upgrading, and innovation, including through a focus on high-value added and labor-intensive sectors.") is under the eighth UN Sustainable Development Goal [5]. We believe 5G is a technological upgrade that can support innovation and increase productivity in several high-value and labor-intensive sectors. An example of this is how it can be used as a part of the communication infrastructure in Industry 4.0 [6]. Another example is how Private 5G can be used in precision farming to increase the efficiency of resource utilization and automation [6]. Thus, by supporting the adoption of private 5G, our benchmarking framework can help contribute to subgoal 8.2.

Another of the UN Sustainable Development goals our thesis can contribute to is Goal 9, specifically subgoal 9.1. Subgoal 9.1 is "Develop quality, reliable, sustainable and resilient infrastructure, including regional and transborder infrastructure, to support economic development and human well-being, with a focus on affordable and equitable access for all" [7]. 5G promises characteristics such as ultra-low latency,

ultra-high reliability, and high device density while maintaining high throughput [4]. Thus, the adoption of private 5G itself represents an upgrade of our communication infrastructure. Furthermore, we would argue that private 5G networks can support other parts of our modern-day infrastructure. For instance, it can be utilized in smart grids as mentioned in [4]. Another example is how it can improve our industrial infrastructure through Industry 4.0 [6]. Thus, by supporting the adoption of private 5G networks, we can contribute to subgoal 9.1.

In conclusion, by exploring the benchmarking of private 5G networks, our thesis can give a small contribution towards the large goals that are the UN Sustainable Development Goals.

Chapter 2
Background

In this chapter, the theory and literature this thesis is based on is presented. The chapter starts by presenting what benchmarking is and how benchmarking of networks can be performed in Section 2.1, followed by an introduction to measurement of packet-level characteristics in Section 2.2. Section 2.3 presents 5G and private 5G networks. Finally, Section 2.4 wraps up the chapter with an introduction to Industry 4.0.

2.1 Benchmarking

Benchmarking can be referred to as "...an empirical experiment with the aim of comparing learners or algorithms with respect to a certain performance measure" [8]. This could be adapted to define a general benchmarking process as *an empirical experiment that generates data that enables comparing candidates with respect to a certain (set of) performance measure(s)*. We will use this definition of benchmarking for the remainder of the thesis.

Our work in the pre-project thesis included a description of RFC 2544 [9], which presents a benchmarking methodology for networking devices. We have included parts of this work below [1].

A benchmarking methodology consists of reproducible tests that can be performed on a system [9]. The purpose is to derive a set of performance characteristics of the system, which can be used as a reference standard both for the deployment of the technology and for potential replacement technologies. The methodology relies on one or several KPIs to establish a common frame of reference. This resembles the definition of benchmarking presented in the introduction of this section. To perform a benchmark, a testbed architecture must be defined. This may be done in varying degrees of generality. The authors of [9] specify a logical testbed architecture that is intended to work for different types of network devices. The architecture is justified with regard to what they call the Device Under Test (DUT). The DUT

is what the methodology intends to benchmark. The authors also specify in great detail how the benchmarks should be performed. This includes trial duration, traffic pattern, message size and format, and derivation of KPIs. It is interesting to see the level of detail that the different aspects of the benchmarks are described. Thus, to ensure comparability between independently performed benchmarks, it is important to enforce that the benchmark should be reproducible.

A guide to performing reproducible experimental research on networks is described by the authors in [10]. Although the authors clearly state that the article is not peer-reviewed, it provides a reasonable general-purpose guide to ensuring reproducible measurements. The article distinguishes between *repeatability*, *replicability*, and *reproducibility*. Repeatability refers to the ability of the same researcher to perform the same measurement on the same setup. If a measurement is replicable, different researchers must be able to perform the same measurements on the same setup. Reproducibility, conversely, ensures that different researchers can perform the same measurement on different networks. Of these, reproducibility is considered the most difficult to achieve but gives the measurements the most value. The authors present guidelines to ensure reproducible measurements. These include hypothesizing before measuring, always keeping the presentation of the data in mind, iterating consistently by using automation, and accounting for dynamic behavior in the measured systems. Moreover, documenting the setup of the hardware and software, as well as the measurement setup and metadata, contributes to reproducibility. The authors also recommend starting with small measurements to validate your setup before performing the planned measurements. Lastly, they highlight the importance of using existing tools, for instance, software, to perform measurements. The general guidelines presented in the article are also applicable to designing and developing software systems capable of performing reproducible network measurements. For instance, the importance of iteration based on input, which inspired the structure of our thesis, aligns with product development and research frameworks, such as Scrum [11] and Design Science [12]. Moreover, thinking critically about hypotheses and designing measurements before performing the measurements can often reduce the time spent to understand the data. We believe the same can be true when developing software, and we will therefore use this as one of our guidelines in this thesis.

2.2 Measurement of Packet-Level Characteristics

This section starts by presenting methods for calculating different packet-level characteristics, which we refer to as KPIs. Section 2.2.1 presents OWD, Section 2.2.2 presents IPDV, Section 2.2.3 presents availability, and Section 2.2.4 presents throughput. Lastly, we present two examples of performing measurements and calculating KPIs from 5G networks in Section 2.2.5.

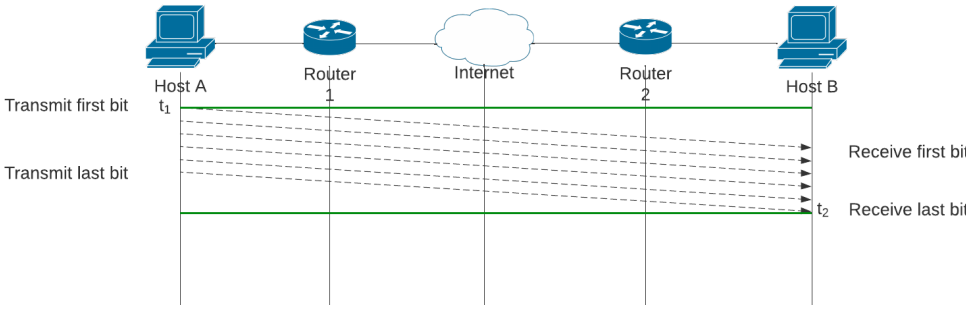


Figure 2.1: Illustration of OWD.

2.2.1 One-Way Delay

OWD measures how long it takes for a packet to be fully transmitted by a source and fully received by a destination, as illustrated in Figure 2.1. Formally, it can be defined as the difference between the timestamp generated upon having fully received the packet and the timestamp generated upon starting the transmission of the packet [13]. One can assume that the setup in Figure 2.1 generates two timestamps, t_1 and t_2. Using Equation 2.1, the OWD can be calculated by computing their difference.

$$OWD = t_2 - t_1 \tag{2.1}$$

When calculating the OWD, clock synchronization must be considered, because calculating OWD based on timestamps from two clocks offset by each other yields an incorrect OWD. This can be mitigated using the Global Positioning System (GPS), which offers synchronization on a scale of tens of microseconds [13]. If the OWD is required to be precise, one must also consider where the timestamps are generated. Suppose the packets are generated and timestamped in a user program on a Linux computer, i.e., software timestamped. In that case, the actual time of the transmission start is likely after the software timestamp. The same goes for the timestamp at reception. Thus, it is important to consider the various influence factors and the accuracy of timestamps required when performing timestamping.

2.2.2 IP Packet Delay Variation

IPDV describes variations in the OWD between packets. In traditional wired computer networks, IPDV can arise from packets traversing distinct paths from source to destination or non-deterministic queuing delay along the path. In radio networks, the packets must traverse the air interface, where variation can be caused by random access procedures and changing radio conditions. This can cause a

non-zero IPDV. IPDV is often called "jitter", however, we will follow the convention recommended in [14] and refer to it as delay variation, or IPDV, to avoid confusion.

We found no single standardized way of calculating and expressing IPDV. The methods for calculating IPDV can be categorized based on what they calculate - either aggregate IPDV for a stream of packets or singleton IPDV for pairs of packets. Some suggest calculating the standard deviation of the packet OWD, according to Equation 2.2 [15]. This is a simple way of deriving the statistic but is limited in its expressiveness and flexibility. Because it is an aggregate KPI, it only reflects some aggregated variation in OWD, but hides the fluctuations and distributions of the variations. Moreover, it is vulnerable to the sample size being small, especially if there are outliers. Using the standard deviation of the measurements is, therefore, a simple way of calculating IPDV, but it is also limited.

$$J_{std} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(D_i - \bar{D})^2} \tag{2.2}$$

Another approach to calculating IPDV is to use the methodology defined in the Sender Report TCP packet of Real-Time Transport Protocol (RTP) [16]. This approach produces a rolling average, which is useful for computing an IPDV-value for continuous network monitoring. The method uses the successive differences in OWD between two packets to update the IPDV-estimate. Equation 2.3 shows how the relative differences are calculated, where Tx_k and Rx_k are timestamps for transmission and reception of packet k, respectively. Consequently, variables i and j correspond to packet indices. This difference is used to update the IPDV using Equation 2.4, where a is some value used for initializing the measure for $J(0)$. The latter equation updates the IPDV-measure by adding one-sixteenth of the difference between the new OWD-difference and the previous IPDV-value to the previous IPDV-value. By scaling down the update to one-sixteenth, the impact of individual outliers is reduced to some extent. This method is well-suited for producing a IPDV-measure that is continuously updated during monitoring.

$$D(i,j) = (Rx_j - Tx_j) - (Rx_i - Tx_i) \tag{2.3}$$

$$J_{avg\_diff}(i) = \begin{cases} J_{avg\_diff}(i-1) + \frac{|D(i-1,i)| - J_{avg\_diff}(i-1)}{16}, & \text{if } i \geq 1 \\ \text{a}, & \text{otherwise} \end{cases} \tag{2.4}$$

The authors of [14] propose a method for calculating IPDV as singleton statistics. The methodology for calculating the IPDV is to define a measurement interval, measuring the OWD of at least two packets inside the interval and selecting two of the packets using some selection function F, which for example can choose consecutive packets or packets at specified indices. The IPDV statistic can be calculated by calculating the difference in the OWD, using Equation 2.5. This method of calculating a singleton statistic can be extended to calculating the IPDV for a set of packets by dividing the measurements into n intervals and applying the method to each interval. This yields a set of singleton values, forming a distribution from which one can perform inferences. Its expressiveness is thus inherently greater than an aggregate value for IPDV. Additionally, one can study fluctuations in IPDV over time. By choosing the F, one can filter illegitimate outliers. Thus, this method of calculating IPDV solves the shortcomings of the above methods.

$$J_{diff}(i) = \begin{cases} (Rx_i - Tx_i) - (Rx_{i-1} - Tx_{i-1}), & \text{if } i \geq 1 \\ 0, & \text{otherwise} \end{cases} \tag{2.5}$$

2.2.3 Availability

Availability measures the probability of a system or service being operational. Different services require different levels of availability. Its most common definition is that of Equation 2.6, where $MTTF$ is the mean time to (system) failure, and $MTTR$ denotes the mean time to (system) repair [17]. Intuitively, this is equivalent to dividing the total time the system is operational by the total time.

What constitutes a failure depends on the context the system operates in and the scope in which it is analyzed. For a web server, a failure may be anything that prevents it from responding to requests. The inability of a switch to forward traffic can be considered a failure of the switch, but not necessarily the network it operates in. For a latency-sensitive application, an end-to-end latency greater than a given threshold may constitute a failure. The availability of a latency-sensitive application could be described as the total time when packets are delivered, and the OWD is less than the threshold, divided by the total time.

$$A = \frac{MTTF}{MTTF + MTTR} \tag{2.6}$$

The state of a system might not always be precisely monitored. In this case, the exact availability of the system cannot be calculated. However, it can be estimated by sampling the system state at different time intervals. Equation 2.7 outlines a method for estimating the availability. $\tilde{A}$ is the estimated availability, n represents a

sample from the set of all samples, S, and Up is a function that is 1 if the system is in an operational state during sample n. This method uses discrete samples instead of measured time intervals. In Equation 2.7, the numerator represents the total amount of samples where the system is in an operational state. Thus, it maps to $MTTF$ in Equation 2.6. Furthermore, the total amount of samples in the denominator corresponds to the total measured time, i.e., $MTTF + MTTR$, in Equation 2.6. In the case of the latency-sensitive application, Equation 2.7 can be used with Equation 2.8 to obtain $\tilde{A}$. For $\tilde{A}$ to be representative, the sampling should be performed at evenly spaced intervals to mitigate the potential for bias.

$$\tilde{A} = \frac{\Sigma_{n=0}^{n_{max}} Up(n)}{|S|} \tag{2.7}$$

$$Up(x) = \begin{cases} 1, & \text{if } OWD(x) \leq threshold \\ 0, & \text{otherwise} \end{cases} \tag{2.8}$$

2.2.4 Throughput

Knowing how much traffic a network can handle before it starts dropping packets is essential in network characterization. This metric is known as throughput, as defined for network interconnect devices in RFC 1242 [18]. Delay-sensitive applications may not tolerate retransmitting dropped packets. Therefore, the authors argue that the maximum rate at which the network can successfully handle traffic is essential to know. However, calculating the throughput of a network, i.e., a series of devices, based on the aforementioned definition is difficult. For example, the network may be subject to background traffic, which impacts maximum measurable throughput because the amount of background traffic typically is hidden from the measurer.

Throughput can also be defined as the total amount of data that can be transmitted over a network in a given amount of time [19]. Equation 2.9 can be used to calculate throughput according to this definition. $Size(x)$ denotes the size of packet x in bits, $Time(x)$ denotes the timestamp of packet x in seconds, and n_{max} is the last packet in the stream of packets. Consequently, the unit of this measure is bits per second (bps). Care must be exercised when reporting the throughput from this equation. If $Time(n_{max}) - Time(0)$ is small, chances are that the reported value is not representative of the network. Capacity utilization of the network devices and links affects the achievable throughput. This can, for instance, be due to dropped packets or higher queuing delays. Capacity utilization typically varies with time. Therefore, if the period of time is too short, the network conditions may be unrealistically poor or good. Throughput calculated like this may be characterized as either

instantaneous, i.e., over a small period of time, or average, i.e., over a longer period of time.

$$Throughput = \frac{\Sigma_{n=0}^{n_{max}} Size(n)}{Time(n_{max}) - Time(0)} \tag{2.9}$$

2.2.5 Real-World Examples of Calculating Packet-Level KPIs on 5G Networks

Real-world experiments have been carried out to measure packet-level KPIs on a 5G network in [20] and [21]. These articles were presented and discussed in the pre-project of this thesis [1]. The paragraphs below include parts of and draw on these findings.

In [20], the authors developed a testbed architecture designed to measure one-way KPIs for both Non-Standalone (NSA) and SA deployments of 5G. Specifically, they analyzed OWD, packet delay variation, and packet loss, and separated each KPI into uplink and downlink directions. The testbed utilized MoonGen, a high-speed open-source traffic generator that uses Data Plane Development Kit (DPDK) for hardware acceleration [22]. Through the use of port mirroring in the switch connecting the mobile cores and the Digital Baseband Units (BBUs), the authors could measure the individual impacts of the RAN and the mobile cores on the calculated KPIs. Moreover, by capturing all packets on the same physical machine, the need for clock synchronization was eliminated. The authors calculate the OWD based on the same equation presented in Section 2.2.1, and IPDV according to Equation 2.5. Moreover, their definition of downtime resembles how we present the calculation of availability in Equation 2.7. However, their definition of downtime considers *consecutive* packets exceeding the threshold, whereas our definition of availability considers the total number of packets violating the threshold. In summary, the authors presented a testbed architecture that can be used to calculate one-way KPIs, but requires some manual integration in the NUT, i.e., installing a port mirror in a switch.

In [21], the authors perform measurements on a commercial 5G NSA deployment. The testbed includes UEs from different manufacturers, an application server in the cloud, and the mobile network. This particular setup treats the 5G network as a black box, meaning it does not have access to information from the network components. The authors are thus limited to end-to-end characteristics over the network. Through the use of a mobile application, physical layer information related to the air interface could be extracted. Moreover, they utilized `iperf3` and `traceroute` to measure end-to-end throughput and delay consecutively. The article highlights some benefits and limitations of using a black-box approach to testing a 5G network; on the one hand, any arbitrary network the researcher can access can be measured

through existing applications. On the other hand, the level of detail in the reported measurements is limited, and more complex analysis is required to extract insights from the measurements, as highlighted by the transport protocol investigation [21].

By integrating their measurement tool in the NUT, the authors in [20] were able to measure one-way packet-level characteristics of their NUT and separate the uplink and downlink. The authors in [21], on the other hand, were more only able to measure data about the NUT as a whole, due to their black-box approach. This highlights the difference between measurement setups and what results they provide, which is something we consider important to evaluate when designing a measurement setup.

2.3 5G

This section introduces aspects of 5G that are relevant to this thesis. Firstly, the envisioned usage scenarios of 5G are presented in Section 2.3.1. Section 2.3.2 presents the high-level architecture of 5G SA before data bearer establishment is examined in Section 2.3.3. Lastly, Section 2.3.4 presents the notion of private 5G networks.

2.3.1 5G Usage Scenarios

5G is meant to serve a multitude of usage scenarios with heterogeneous requirements [2]. Figure 2.2 shows the envisioned usage scenarios from IMT. The figure relates the usage scenarios to the 5G service categories, namely eMBB, URLLC, and mMTC. eMBB centers around improving the overall bandwidth and mobility of mobile networks. URLLC, on the other hand, poses strict requirements on latency and availability, as it focuses on mission- and safety-critical applications. Lastly, mMTC necessitates the support of a very high density of devices, each transmitting a small volume of non-delay sensitive data while remaining battery efficient.

The multitude of heterogeneous usage scenarios indicates that 5G has to offer a high degree of technical flexibility. This can be observed, e.g., in the multitude of possible deployment architectures and configurations, as well as technically complex enabling mechanisms, such as network slicing. By introducing logically separate networks, network slicing enables various heterogeneous services simultaneously. However, while maintaining virtually separate network slices may theoretically support the various services, all traffic must be mapped to physical networks. The authors of [23] highlighted this as a central challenge of network slicing in 2017 and indicated that supporting multiple virtual RANs to accommodate the flexibility introduces a trade-off between resource utilization and traffic separation.

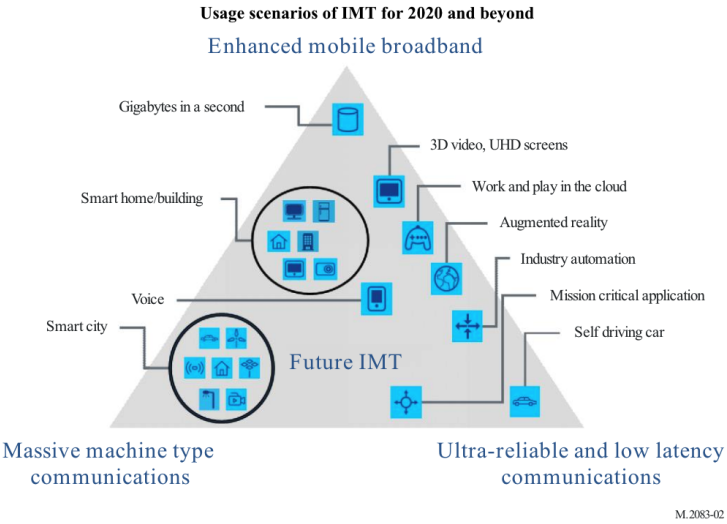


Figure 2.2: Envisioned usage scenarios of IMT for 2020 and beyond from Figure 2 in [2].

The envisioned usage scenarios of 5G indicate that 5G systems should support high bandwidth, high mobility, extremely low latency, high availability, high device density, and high battery efficiency. Some aspects are contradictory, such as high bandwidth and battery efficiency. Moreover, the extremely low latency requirements can impact the performance of high bandwidth usage scenarios, as attempted solved in [24]. This illustrates how the intended flexibility of 5G necessitates care during the design and implementation of the systems.

2.3.2 5G Standalone Architecture

5G introduces significant improvements in both the radio interface (5G New Radio (NR)) and in the mobile core (5G Core (5GC)) [25]. To enable mobile providers to capitalize on this as soon as possible, Third Generation Partnership Project (3GPP) defined several deployment options for 5G networks, which can be broadly categorized in NSA and SA deployments. An NSA deployment typically only rolls out the 5G NR while leveraging existing LTE infrastructure. An SA deployment, on the other hand, deploys both the 5G NR and the 5GC. Its high-level architecture can be seen in Figure 2.3. The network consists of a UE, a Next Generation NodeB (gNB), consisting of an Remote Radio Head (RRH) and a BBU, and a 5GC. The RRH translates digital signals from the BBU into analog signals that can be transmitted over the air interface and vice versa. Signal encoding, decoding, and cell management are among the responsibilities of the BBU. It also connects each 5G cell to the 5GC.

In line with the trend of virtualization and microservices, 3GPP designed a mobile

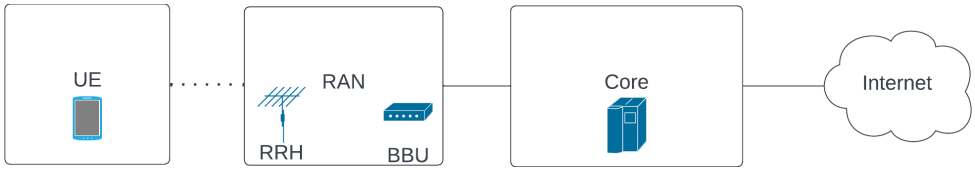


Figure 2.3: High-level architecture of a 5G SA system.

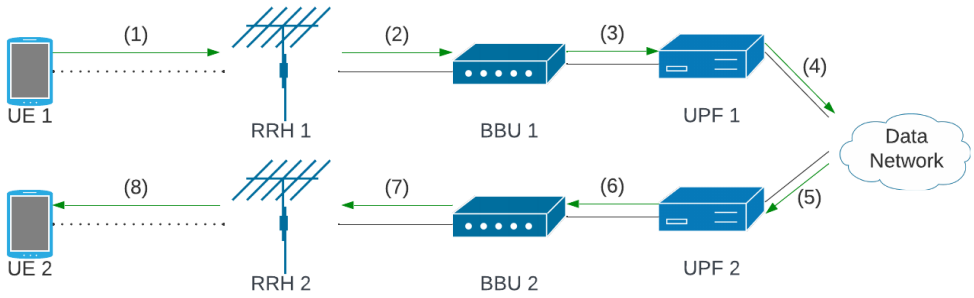


Figure 2.4: Simplified 5G architecture showing the direction of traffic from source to destination UE.

core consisting of individual network functions [25]. Each network function in the 5GC has its own responsibilities and can run on virtualized infrastructure. Some of the functions are the Access Management Function (AMF), Session Management Function (SMF), and User Plane Function (UPF). The AMF handles registration, connection and mobility management, and session management traffic, among other things. Session establishment between a UE and UPF is among the responsibilities of the SMF. The UPF connects the 5G system to external networks and handles Internet Protocol (IP) address allocation and bearer termination.

Figure 2.4 shows a simplified overview of the traffic flow when a UE in a given 5G cell in a SA network wants to transmit data to a UE in another cell, based on information from [25]. Assuming that the bearers for both UEs are set up and the data is ready to be transmitted, the data packets traverse (1) the air interface and RRH before being sent to (2) the BBU. After this, they are forwarded to (3) the UPF the source UE is registered with, which sends the data towards the destination through (4) a Data Network. In this example case, the traffic traverses the Data Network to (5) a second UPF, that the destination UE is registered with. This UPF sends the data to (6) the BBU managing the cell of the destination UE, which sends the encoded data to (7) the RRH, which finally transmits the data over (8) the air interface to the destination UE.

2.3.3 5G Data Bearer Establishment

Enabling the transmission of user data through a 5G system is a process that can be broken down into three steps. The steps are (1) to ensure an Radio Resource Control (RRC) Active state, (2) to register with the 5GC, and (3) to establish a data bearer. In the first step, the UE must become associated with a gNB, which is referred to as an RRC [25]. The UE goes into an RRC Idle state if it has not sent data in a while. In this case, the connection must be refreshed and transitioned into an Active state to transmit data. Secondly, the UE must be registered with the 5GC, which is a responsibility of the AMF. When a UE has an RRC, it registers with a AMF, after which identification and authentication are performed. When this process is complete, the UE has registered with the 5GC and can attempt to establish data bearers. Lastly, the data bearer must be established. To accomplish this, the SMF is responsible for configuring a GPRS Tunneling Protocol (GTP) tunnel between the UE and a UPF. Accordingly, all user plane data is encapsulated in a GTP-U-header between the UE and the UPF [26]. The gNB also configures a radio bearer for the data tunnel. This prevents the UE from having to send user plane data over its signaling bearer. 3GPP has stated that transitioning from an idle state to a state where the UE can transmit continuous data should not take more than 10ms [27].

2.3.4 Private 5G Networks

3GPP has defined a private 5G network, which they refer to as a non-public network, as *"a 5GS deployed for non-public use"* [26]. Furthermore, they state that it can be deployed standalone or through integration with a public network. This section presents how private 5G networks differ from public 5G networks.

Deploying a private 5G network enables an organization to utilize a mobile network with dedicated access and characteristics tailored to their needs [4]. Through configuration, the network can, for instance, adhere to the specific security policies of the organization. This is often seen in contrast with utilizing services from public 5G networks, in which the organization has to adhere to the policies of the mobile service provider. Moreover, because the private network belongs to the organization, they receive dependable communication and service [3].

The infrastructure needs can differ between private and public 5G networks. For instance, private 5G networks can employ a lean core [3]. Moreover, private 5G networks have been envisioned to utilize the progress of open-source software implementations of 5G network components, such as from the OpenAirInterface Software Alliance and the Open5GS Project [28], [29].

2.4 Industry 4.0

One of the possible applications of private 5G is as an enabling technology in Industry 4.0. This concept has been defined as "... the current trend of automation and data exchange in manufacturing technologies" [30]. Other enabling technologies include Internet of Things (IoT), cloud computing, and cyber-physical systems [31]. The integration of these technologies into manufacturing processes allows for the creation of "smart" factories with increased efficiency and data sharing [32].

Industry 4.0 can be utilized to improve a multitude of application areas. Table 2.1 based on [33] shows some of these areas with their corresponding use cases and requirements. The authors of [33] do not provide a definition of reliability. Therefore, when considering these use cases later, we define reliability as the percentage of packets arriving within the required latency. Several of the use cases can be considered safety-critical and, consequently, pose stringent requirements on latency and reliability. Examples include motion control and condition monitoring for safety. On the other hand, use cases such as AR/VR have lower requirements for reliability but pose higher demands on throughput. Finally, some use cases, such as process monitoring, are mainly concerned with supporting high device density. Thus, the networks enabling communication between entities must satisfy a diverse set of requirements to enable multiple use cases in Industry 4.0.

Because of these stringent demands on latency and reliability, wired networks have traditionally been used in Industry 4.0 [3]. However, as the authors discuss, this has downsides, such as increased deployment and reconfiguration costs. These downsides are both related to using wired connections, thus, using wireless communication technologies could mitigate these challenges. Through URLLC, eMBB, and mMTC, 5G is envisioned to satisfy the requirements of the aforementioned use cases [2]. Therefore it is important to investigate if real-world private 5G networks can fulfill these requirements.

Application Area	**Use Case**	**Reliability**	**Latency**	**Data Rate**	**Payload**	**Devices**
Factory automation	Motion control	99.9999%	0.5–2ms	1–5Mbps	20–50B	20–100
	Control to control	—	10–50ms	—	1kB	5–10
	Mobile robotics (co-operative)	—	1–50ms	—	40–250B	100
Process automation	Closed-loop process control	99.9999%	≤10ms	—	20B	—
	Process monitoring	99.99%	50–100ms	0.5–2Mbps	—	100–1,000
	Condition monitoring (safety)	99.9%	5–10ms	0.1–0.5Mbps	—	>1,000
	Condition monitoring (interval/event based)	99.9%	50ms–1s	0.1–0.5Mbps	—	>1,000
HMI and production IT	Mobile control panels with safety control	99.9999%	4–12ms	—	40–250B	2–4
	AR/VR	99.9%	<10ms	5–25Mbps	—	10–20
Logistics and warehousing	Mobile robotics (video operations)	99.9999%	10–100ms	—	15k–250kB	100
	Mobile robotics (standard operations)	—	40–500ms	—	40–250B	100
Monitoring and maintenance	Massive wireless sensor networks	Noncritical, massive devices, and energy aware				

Table 2.1: Industrial use cases and Key Performance Indicators (KPIs) from [33].

Chapter 3
Design of the Benchmarking Tool

This chapter presents the design process of the benchmarking tool. This is the first step in creating the artifact that will be iterated over during this thesis, as illustrated in Figure 1.1. During this process, several aspects of RQ1 (design of system capable of reproducible benchmarking of a private 5G network) and RQ2 (balancing interaction with NUT with the level of detail provided) will be considered. The outcome of this chapter is a list of functional and non-functional requirements for the tool in Section 3.1, and the overall software architecture of the artifact and its sub-requirements in Section 3.2. Additionally, the high-level physical architecture of the artifact is presented in Section 3.3.

3.1 Requirements of the Benchmarking Tool

This section presents the functional and non-functional requirements of the benchmarking tool in Section 3.1.1, and a reflection over these in Section 3.1.2.

3.1.1 Functional and Non-Functional Requirements

To ensure that the benchmarking tool could benchmark private 5G networks, we defined functional and non-functional requirements. These requirements were used when designing the architecture of the tool and the individual modules. The requirements are based on general features required to perform experiments on networks, requests from ABB, and features supporting the reproducibility of experiments.

The functional requirements of the benchmarking tool describe the set of features it offers. These requirements must be satisfied for the tool to enable answering the defined research questions. All of the defined functional requirements are presented below. The benchmarking tool must be able to:

- **TFR1**: Generate traffic that emulates real-world scenarios
- **TFR2**: Capture network traffic
- **TFR3**: Calculate a pre-defined set of KPIs based on network traffic

- **TFR4**: Visualize the result of a trial
- **TFR5**: Provide possibilities for custom analyses of the trials
- **TFR6**: Execute trials automatically
- **TFR7**: Display the status of the executing trial

TFR1 ensures that the traffic used to measure the performance of the NUT resembles the traffic used in the desired scenarios. The performance of a network depends on the traffic it is subjected to. This requirement enables answering RQ3 (supporting Industry 4.0 use cases), which provides value to ABB. **TFR2** is a general feature required to perform network measurements. Capturing network traffic is necessary to calculate the pre-defined KPIs, which is specified in **TFR3**. The calculation of KPIs enables analyses of how the NUT performs. This information can be used to answer RQ4 (compare 5G implementations). **TFR4** focuses on providing an overview of the benchmark results through visualizations. This makes it easier to compare different networks and supports answering RQ4. While **TFR4** provides an overview of the network performance, **TFR5** enables further analyses of the trial results. **TFR6** supports the reproducibility of the benchmarking tool by ensuring that actions are executed in the same order and environment. Finally, **TFR7** is added to provide a method to verify that a trial is running as expected. This provides value as some trials last for longer durations.

Non-functional requirements are concerned with how a system behaves rather than its specific behavior. The benchmarking tool has a single non-functional requirement, **TNFR1**: "The benchmarking tool should be portable to new NUTs". This simplifies benchmarking different NUTs, which enables answering RQ4.

3.1.2 Reflections on Requirements

Based on the requirements presented above, there are some considerations we need to account for in the designs of the modules. This subsection presents a brief explanation of these considerations.

TFR3 and **TFR4** are concerned with the data gathered by the tool and its representation. Some of the interesting KPIs, such as packet loss, require processing to make the information easily understandable. For instance, simply labeling packet loss as either `true` or `false` for each packet can be difficult to interpret. To address this, the data can be aggregated by calculating the rate of lost packets in the entire measurement period. While providing an easily digestible number, this approach simultaneously hides the temporal fluctuations of the packet loss. Care must be taken to balance aggregation and level of detail to provide meaningful information to the user.

To fulfill **TFR5**, files and calculations must be transparent and accessible to the

users. We need to know which files will be relevant for such investigations, such as the captured traffic and potentially intermediate files between capture and final calculation. Ideally, all files of previous trial executions should be made available. However, the resulting file sizes can be in the order of gigabytes, which makes this impractical. Therefore, balancing accessibility and storage limitations must be considered while complying with **TFR5**.

5G can potentially support very high throughput, and thus, the ability to handle a large number of packets is an important consideration for the benchmarking tool. Therefore, employing techniques such as filtering out irrelevant traffic, sampling, and parallelization should be considered. If the processing time of the tool is too high, it will be limited to benchmarking scenarios with low amounts of packets. This limits the extent to which RQ3 can be answered.

During the discussions with ABB, the extensibility of the benchmarking tool was emphasized. However, because neither of the defined research questions considers extensibility, we have not included it as its own requirement. Rather, it is a consideration that is taken for the development of all the modules of the benchmarking tool.

We must consider which functionality we want to build and where we want to utilize existing tools, as discussed in [10]. Using existing tools can save substantial amounts of development time as well as increase reliability through maintenance and documentation. However, it might limit the available functionality and extensibility. Implementing functionality ourselves offers a great deal of flexibility but comes at the cost of increased development time and the need for validation. Therefore, we must consider the trade-off between custom implementation and utilizing existing tools. Avoiding re-implementing existing functionality yields more time to ensure a rich and robust feature set in the tool.

The considerations presented will be taken into account when designing and developing the modules of the benchmarking tool. Awareness of the need to consider how information should be presented, made accessible, and manipulated supports the development of the tool with regards to **TFR3**, **TFR4**, and **TFR5**. Furthermore, being aware of the trade-offs between custom functionality and the usage of existing functionality allows for efficient prioritization of development resources. This will help align the technical realization of the tool with its functional and non-functional requirements.

3.2 Benchmark Tool Software Architecture

This section presents the architecture of the benchmarking tool. It presents the Orchestrator, Traffic Generator, Packet Matcher, Packet Analyzer, and Visualization modules in this order. The requirements for the individual modules are also presented.

3.2.1 Orchestrator

The Orchestrator is responsible for automating trial execution. This involves the coordination of all the other modules. To perform its role, the Orchestrator must fulfill the following functional requirements. The module must be able to:

- **OFR1**: Start, stop, and pass parameters to the other modules
- **OFR2**: Run commands on multiple hosts

OFR1 and **OFR2** encompass the functionality necessary to perform orchestration. This enables **TFR6** (executing trials automatically). **OFR1** emphasizes the ability to coordinate tasks, while **OFR2** focuses on doing so in a distributed system.

3.2.2 Traffic Generator

The Traffic Generator generates traffic based on the trial parameters. This traffic is injected into the NUT and provides the foundation for the benchmark. To fulfill its role, we devised the following requirements. The module must be able to:

- **TGFR1**: Generate traffic patterns adhering to real-world scenarios
- **TGFR2**: Generate traffic based on parameters for packet rate, duration, hosts and packet size
- **TGFR3**: Customize packet fields

TGFR1 and **TGFR2** both enable satisfaction of **TFR1** (generating traffic that emulates real-world scenarios). **TGFR1** emphasizes the traffic patterns of real-world scenarios, such as CBR or the pattern used in a specific Industry 4.0 use case. Furthermore, **TGFR2** focuses on being able to parameterize the pattern. This enables changing pattern characteristics such as the duration or the packet size to accommodate different trials.

TGFR3 focuses on being able to customize the generated packets. Customizing packets enables, for instance, adding identifiers that can be used for packet matching, which is necessary for calculating the KPIs.

It must also meet the following nonfunctional requirement.

- **TGNFR1**: Sustain stable packet transmission over a prolonged duration

The benchmarking tool should be able to execute trials with a long duration. If the traffic generator does not generate stable traffic, it would be difficult to differentiate this from the performance of the NUT. **TGNFR1** addresses this concern.

3.2.3 Packet Matcher

The Packet Matcher is responsible for capturing and matching packets and storing the result. To accomplish this, the module must fulfill the following functional requirements. The Packet Matcher must be able to:

- **PMFR1**: Perform packet capturing on at least two interfaces
- **PMFR2**: Filter out irrelevant traffic
- **PMFR3**: Match packets
- **PMFR4**: Write the result of matched packets to persistent storage
- **PMFR5**: Provide simple updates at a fixed interval during a trial

To capture the packets at both the ingress and the egress of the NUT, the Packet Matcher must be able to capture on at least two interfaces, which is covered by **PMFR1**. By filtering out irrelevant traffic we reduce the demands for storage and processing. This is addressed by **PMFR2**. **PMFR3** considers the need to match the different observations of the same packet from the ingress and egress of the NUT. Comparing the matched packets and their meta information provides the foundation for calculating the pre-defined KPIs. **PMFR4** supports the handover between the Packet Matcher and the Packet Analyzer. For this, the results of the matched packets must be written to storage. Furthermore, it also makes the output of the Packet Matcher accessible, which supports **TFR5** (Provide possibilities for custom analyses of the trials). Finally, **PMFR5** provides the data necessary to comply with **TFR7** (display the status of the executing trial).

It must also meet the following nonfunctional requirement.

- **PMNFR1**: The module should be able to process 1 million packets in at most 60 seconds

PMNFR1 focuses on the need for scaling to trials with a large number of packets, elaborated in the previous section. If the processing time is too high, the usability of the tool is impacted. These values above are chosen to ensure that for our purposes and experiments, the processing time of the Packet Matcher will not be prohibitive. However, depending on the trials to be executed, this requirement can either be too stringent or lenient. Therefore, further analysis and adjustments of these values is required as the tool matures.

3.2.4 Packet Analyzer

The Packet Analyzer is responsible for calculating the pre-defined KPIs. To do this, it must satisfy the following functional requirements. The Packet Analyzer must be able to:

- **PAFR1**: Calculate the set of pre-defined per-packet KPIs
- **PAFR2**: Calculate the set of pre-defined aggregated KPIs
- **PAFR3**: Write the results to persistent storage

PAFR1 and **PAFR2** are both concerned with calculating the pre-defined KPIs, which satisfies **TFR3** (calculating the pre-defined KPIs). The requirement is split into two to emphasize the difference between per-packet and aggregate KPIs. **PAFR3** allows for the handover between the Packet Analyzer and the Visualization module by writing the resulting KPIs to storage. Furthermore, making the results of the Packet Analyzer accessible for further analysis supports **TFR5**.

It must also meet the following nonfunctional requirement.

- **PANFR1**: The module should be able to process 1 million packets in at most 60 seconds

PANFR1 is grounded in the same arguments as **PMFR1**, which is discussed above.

3.2.5 Visualization

The Visualization module is responsible for displaying the results of a trial. To accomplish this, it must satisfy the following functional requirements. The Visualization module must be able to:

- **VFR1**: Give an overview the calculated KPIs
- **VFR2**: Respond to changes in the data used

VFR1 supports **TFR4** (visualizing the results of trials). Emphasis is put on providing an overview that is easily digestible, and provides an identification of areas for further analysis. **VFR2** enables **TFR7** (displaying the status of the executing trial) by being able to show continuously updated data.

3.3 High-Level Testbed Architecture

After having presented the requirements of the benchmarking tool, we now turn our attention to the physical testbed architecture upon which the benchmarking tool runs. This section presents this architecture and discusses its most important design choices.

Figure 3.1 illustrates the main components of the high-level testbed architecture. The architecture can be split into two main components - the benchmarking tool and the NUT. The tasks of the benchmarking tool could be accomplished by a single host, but it has been split up into a traffic generator, a measurement machine, and a 5G gateway. Separating traffic generation onto its own machine is intended to offload the traffic generation from the orchestrating machine since traffic generation could be a CPU-intensive process. Additionally, it emphasizes the separation of concerns. Separating the measurement machine and the 5G gateway provides increased flexibility. The NUT, on the other hand, contains the components of the network that the benchmarking tool tests.

When performing tests on a system, the tester must choose whether to perform black-box, grey-box, or white-box testing. Each alternative differs in the amount of customization necessary and the granularity of the information provided. Black-box testing requires the least configuration and interaction but is only able to provide insights into the performance of the system as a unit. White-box testing, on the other hand, provides detailed information about the system under test and its internals. However, more prior knowledge must be attained, and tests must be configured to perform this type of testing. Finally, grey-box testing enables balancing the complexity of testing with the level of detail in the information it generates [1].

For this project, RQ2 (balancing the requirements for integration with the NUT and the granularity of information provided by the benchmark) focuses on the trade-off between these approaches. Black-box testing of a 5G network has been performed in [21] and yielded interesting findings. However, the results were only insightful after being contextualized with auxiliary information. It lacks the information to enable reasoning about the performance of individual network components and network configurations. White-box testing may provide detailed information about the performance of individual network components and network functions. However, it requires complex customization and a significant amount of hardware and may not be possible in many cases due to closed proprietary interfaces between certain components. Grey-box testing of a private 5G network was performed in [20], and were able to provide what we considered sufficient granularity with only a single port mirror. Inspired by this, we consider a grey-box approach to be suitable. The only integration we perform with the NUT is a wiretap installed between the BBU and the core server (link number 3 and/or 6 in Figure 2.4). This way, we can gain insight into one-way KPIs of the network without configuring clock synchronization across the sender and receiver of the traffic.

The benchmarking framework contains the components that make up our tester, as illustrated in Figure 1 in RFC 2544 [9]. To gauge the network, a method of recording when a packet is transmitted and received is necessary. In our high-level testbed architecture, this is accomplished by measuring when a packet is transmitted (forwarded) by the measurement machine and when it has passed through the network. This way, both the transmit timestamp and the receive timestamp are recorded, and one-way KPIs can be calculated as described in Section 2.2. This is illustrated in Figure 3.1 through the line from the measurement machine to the 5G gateway and between the NUT and the measurement machine. Moreover, by recording both the transmit and receive timestamps at the same host, clock synchronization need not necessarily be configured.

[1] This description is inspired by general knowledge of this type of testing and supplements from GeeksForGeeks (accessed: June 5^{th} 2024).

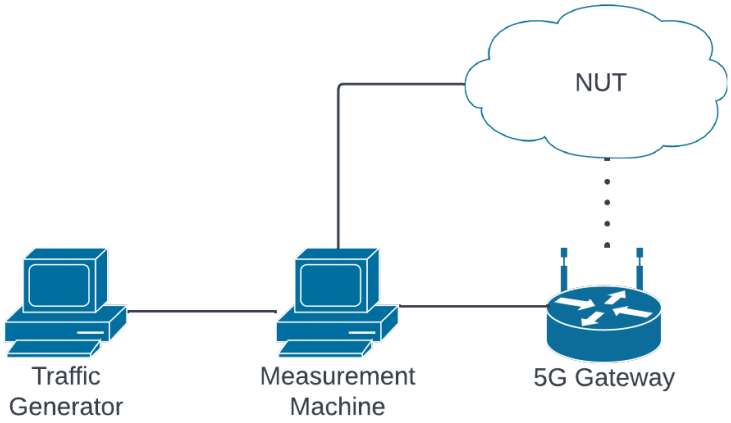


Figure 3.1: Illustration of separation of traffic generation, traffic measurement, transmission, and NUT.

The NUT of the testbed architecture carries the traffic of the benchmarking tool. By keeping this part logically separated from the tester, it is simpler to fulfill **TNFR1** (portability to new NUTs) by developing the tester independently of the NUT, such that it is interchangeable. In theory, the traffic generation and measurement part should be agnostic of the type of network it is testing.

Chapter 4
Implementation of the Benchmarking Tool

This chapter describes the implementation of the different modules of the benchmarking tool. Its outcome is a description of the artifact used for validation, as seen in Figure 1.1. The intention of the chapter is to map the design choices from Chapter 3 to the actual implementation of the benchmarking tool. First, the Traffic Generator module is described in Section 4.1, followed by the Packet Matcher in Section 4.2. After this, the Packet Analyzer module is presented in Section 4.3. The Visualization module is presented in Section 4.4, before the Orchestrator is described in Section 4.5. Lastly, Section 4.6 presents how data is represented and stored between modules.

4.1 Traffic Generator

The Traffic Generator is responsible for producing traffic that will be applied to input to the NUT. This section describes the three alternatives we provide for traffic generation. Each offers a different level of flexibility and ease of use, suiting different use cases of the benchmarking tool. We balanced the development of new tools and utilities with the usage of existing tools to tailor the offered traffic generators to our requirements.

Table 4.1 lists the submodules of the Traffic Generator and maps them to the functional requirements they contribute to. First and foremost, it must be able to generate traffic adhering to specific patterns, contributing to **TGFR1** (realistic traffic patterns). Through **TGFR2** (parameterizable traffic), the traffic patterns it produces must also be parameterizable, increasing the extensibility of the module. **TGFR3** (customize packet fields) necessitates that it needs to be able to craft custom packets that contain identifiers, which we refer to as packet crafting. This enables the system to be able to calculate KPIs such as one-way delay and inter-packet delay variation. The non-functional requirement specified for the Traffic Generator must be considered when developing the module. For instance, **TGNFR1** (stable packet transmission over a long time) entails that the traffic generator cannot impose too

much load on the benchmarking tool, such that the behavior changes significantly over time.

Submodule	TGFR1: Realistic traffic patterns	TGFR2: Parameterizable traffic	TGFR3: Customizable packet fields
Traffic pattern generation	X	X	-
Packet crafting	-	-	X

Table 4.1: Required submodules of the traffic generator module with the requirements they satisfy.

In order to cater to the need for flexibility and customization, we developed three alternatives for traffic generation. Alternative 1 offers a traffic generator able to produce parameterizable CBR traffic, which uses Cisco TRex for transmission, described in further detail in Subsection 4.1.1. Alternative 2 offers a script that parses a capture file provided by the user and can replay it with configurable source and destination addresses. The benchmarking tool can thus generate industry-specific traffic patterns. This is described in Subsection 4.1.2. Alternative 3 offers a specification that enables using a different custom traffic generator while utilizing the remainder of the benchmarking tool. It is described in Subsection 4.1.3. Table 4.2 compares the alternatives based on traffic realism and ease of use.

Traffic generation alternative	Traffic pattern realism	Ease of use
Alternative 1	CBR - low	High
Alternative 2	Any[1]	High
Alternative 3	Any[2]	Moderate/Low

[1] Limited by the realism of the traffic pattern in the provided pcap.
[2] Limited by the realism of the custom traffic generator used.

Table 4.2: Comparison of the offered traffic generation alternatives based on traffic pattern realism and ease of use.

4.1.1 Alternative 1: Constant Bitrate with Cisco TRex

The first traffic generation alternative enables simple traffic generation through the use of a parameterized script able to produce CBR traffic. Its traffic pattern realism is limited to CBR, and its purpose is to provide a simple means for generating traffic. It uses the stateless module of Cisco TRex to produce the specified traffic. Firstly,

a survey presenting various software traffic generators is described, followed by a description of how Cisco TRex is used.

Survey of Software Traffic Generators

Before implementing the traffic generator module, a small survey of software-based traffic generators was conducted. There exist several traffic generator alternatives, each of them suitable for different scenarios. The survey was intended to find a suitable software traffic generator that could be used to implement the submodules in Table 4.1. The traffic generator must be programmable to send CBR traffic. However, to be extensible, the traffic generator should also enable different traffic patterns. To generate identifiable packets, it must be able to customize individual packets. A simple example is the *ping* application, which uses Internet Control message Protocol (ICMP) messages. ICMP requests and replies place identifiers and sequence numbers in the header. This way, correlated packets can be identified. Moreover, the traffic generator must be able to construct packets with an arbitrary protocol structure and payload to ensure that the traffic generator module is extensible.

We considered MoonGen [22], Cisco TRex, Ostinato, Scapy, `iperf3`, and D-ITG. `iperf3` was discarded because of its evident lack of flexibility in traffic pattern customization [34]. Moreover, D-ITG uses the traditional Linux New API (Linux NAPI), meaning that it cannot guarantee sufficiently accurate transmission of packets [35], potentially compromising **TGNFR1**. Therefore, it was excluded from further evaluation.

Table 4.3 summarizes the remaining traffic generator options. It categorizes them based on the capability for hardware acceleration, whether they support stateful or stateless traffic generation, and packet crafting capabilities.

Tool	Hardware Acceleration	Stateful/ Stateless	Packet Crafting
MoonGen	DPDK	Stateless	Per-packet modifications
Cisco TRex	DPDK	Stateful and stateless	Per-packet modifications
Scapy	No (uses Python sockets)	Stateful and stateless	Per-packet modifications
Ostinato	eXpress Data Path	Stateless	Per-packet modifications

Table 4.3: Survey findings of popular traffic generator options.

Ostinato is a traffic generator with a proprietary license. It enables extensive packet crafting and supports a wide range of network protocols [36]. It uses the

eXpress Data Path. Because of the proprietary license, Ostinato was regarded as an unsuitable option and was not followed up further. [1]

Scapy is a library for Python that allows for simple traffic generation and flexible packet crafting [37]. By scripting packet generation and response parsing, Scapy enables operating in both a stateful and stateless manner. However, because it uses Python for processing and the traffic generator machine uses the Linux NAPI and not DPDK for Python sockets, it is limited in terms of the achievable standard deviation of inter-transmission times [38]. Scapy offers sufficient flexibility in packet crafting and traffic pattern modeling. However, the aforementioned limitations make it unsuitable in use cases where throughput and performance are important considerations.

MoonGen is an open-source software-based traffic generator distributed under the MIT license [22]. It uses the DPDK-framework to achieve high throughput. Moreover, by enabling scripting with Lua, programmers can craft packets and send CBR-traffic as well as, for example, Poisson-distributed traffic. MoonGen is scalable in terms of the number of cores used by the traffic generator. It can transmit more than 14 million PPS per core, which has been demonstrated to scale up to 12 cores [22]. However, it only supports stateless traffic, making emulation of client-server communication difficult.

Cisco TRex, subsequently referred to as TRex, is another open-source software-based traffic generator using the DPDK-framework [39]. It is distributed under the Apache license. TRex can operate in both stateful mode and stateless mode and offers great flexibility in both modes of operation. Stateless operation enables packet crafting and programmatic definitions of traffic patterns using its own Python-based software development kit and Scapy. It can generate 10-22 million PPS in stateless mode [39].

Out of these alternatives, both TRex and MoonGen are viable options. However, offering alternatives for both stateful and stateless traffic differentiates TRex from MoonGen. Therefore, we decided that TRex provided the greatest opportunities in terms of flexibility and extensibility and we chose to use this software traffic generator for this traffic generation alternative.

Script for Generating Traffic With Cisco TRex

To generate CBR-like traffic, we wrote a Python script using the stateless API of Cisco TRex. It defines a class that is responsible for creating a traffic stream. A traffic stream is defined by a packet template, the transmit mode, a TRex Field Engine, and the transmit rate, among other things. The packet template we provided

[1] As of April 2024, Ostinato is no longer open-source. https://ostinato.org/open-source

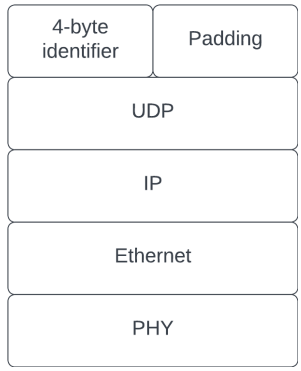


Figure 4.1: Format of packet constructed for transmission with Cisco TRex.

is a simple packet consisting of an Ethernet header, IP header, UDP header, and a fixed payload. This template is illustrated in Figure 4.1. The payload contains a sequence number and a configurable number of bytes of padding. We used the Field Engine functionality of TRex to implement an incrementing 4-byte sequence number. This was done because TRex generates identical copies of the same packet by default and requires the use of a Field Engine to update packets during traffic generation. UDP was chosen as the transport layer protocol because of its stateless properties. To generate CBR traffic, the created stream transmits in continuous mode, which sends packets at the same rate per second for the entirety of the defined trial duration.

The traffic generation script has been made parameterizable by connecting command-line arguments to the script. Source- and destination IP addresses and UDP ports, packet rate, duration, and payload size can be configured through the arguments. This makes it easy to reuse the script for different trials and in different networks.

This traffic generation alternative requires manual configuration of a server to respond to the traffic and make the traffic bidirectional.

4.1.2 Alternative 2: Packet Capture Replay

To enable traffic patterns emulating real-world protocols, the benchmarking tool enables replaying capture files. In this context, replaying means transmitting the packets from a capture file in the same format, in the same order, and with the same inter-packet transmission time. `tcpreplay` is used to replay traffic. Cisco TRex also offers functionality for this but was deprioritized due to the inconvenience of copying

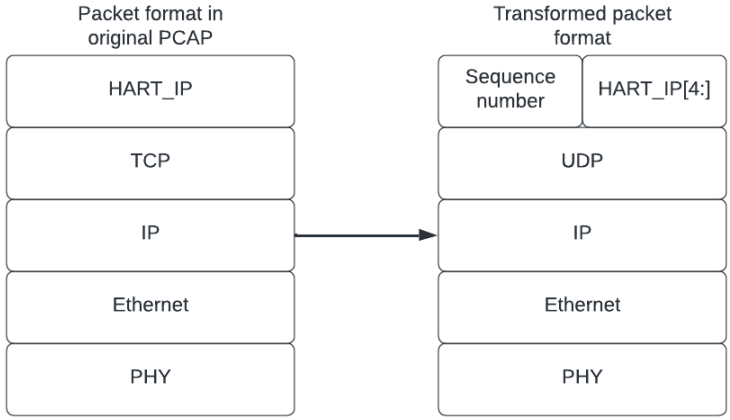


Figure 4.2: Illustration of synthesizing of an example packet format to replay format.

files between hosts [2]. The tool expects a capture file that contains the application traffic. The file can also contain background traffic, which will be filtered out of the file. The packet format of the application data is updated according to Figure 4.2. This adds an identifier to the packets. When the capture file has been parsed, it can be transmitted using `tcpreplay`.

This traffic generation alternative also does not provide bidirectional traffic without the manual configuration of a responding server. To emulate bidirectional traffic, the destination of the traffic needs to actively respond to received packets.

4.1.3 Alternative 3: Custom Traffic Generator

As a third alternative, a custom traffic generator can be integrated with the benchmarking framework, enabling full control over the traffic. This approach requires more setup but makes it possible to use stateful protocols and applications, such as TCP, for trials. To properly integrate the traffic generator, two requirements must be fulfilled. Firstly, the traffic generator must be connected to the measurement machine such that the traffic can be captured before being injected into the NUT. This can be done either by sending the traffic through it using the L2 bridge as done in Figure 3.1 or by port mirroring to the measurement machine as in Figure 4.3. Secondly, the generated traffic must comply with **TGFR3** to enable identification. An example of this would be a machine running `iperf3` client connected to the measurement machine, sending traffic to an `iperf3` server.

[2]The maintainers of TRex have mentioned functionality for extracting traffic patterns from a capture file into a Python script in the forthcoming features [40]. This functionality would be interesting for this traffic generation alternative.

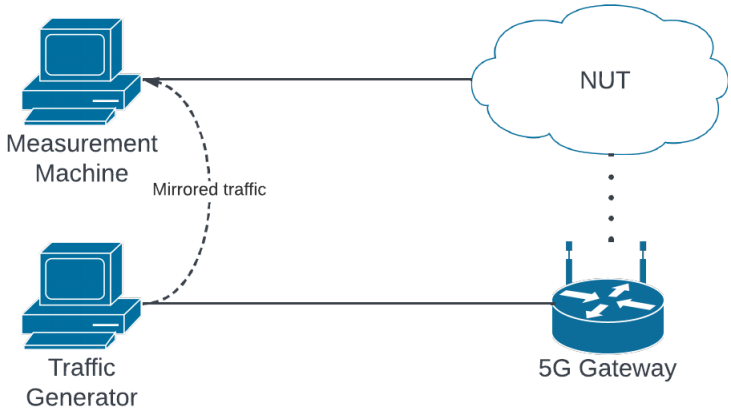


Figure 4.3: High-level testbed architecture with port mirror instead of using L2 bridge on measurement machine.

4.2 Packet Matcher

The Packet Matcher captures the traffic, processes it, and forwards it to the Packet Analyzer where the defined KPIs are calculated. Table 4.4 shows the submodules of the Packet Matcher. The Packet Capture submodule is responsible for capturing network traffic. Within the Packet Matching submodule, the timestamps at the different measurement points of a specific packet are extracted and then stored in a CSV file, in the Data Extraction submodule. Finally, the Live Update submodule writes the number of captured packets to InfluxDB at a pre-defined interval. This section describes each of the submodules in the order they were presented.

Submodule	PMFR1: capture packets	PMFR2: filter during capture	PMFR3: match packets	PMFR4: write to storage	PMFR5: semi-live simple updates
Packet Capturer	X	X	-	-	-
Packet Matching	-	-	X	-	-
Data extraction	-	-	-	X	-
Live update	-	-	-	-	X

Table 4.4: Required submodules of the Packet Matcher with the requirements they satisfy.

4.2.1 Packet Capturer

The Packet Capturer submodule captures traffic on a defined set of interfaces. This is a use case already implemented in several tools, and therefore we wished to utilize an existing solution.

Table 4.5 summarizes some characteristics of `tcpdump`, `tshark`, and `dumpcap`. We chose `tshark` as it can perform capturing on multiple specific interfaces, and we had experience with the tool.

Feature	`tshark`	`dumpcap`	`tcpdump`
Capture on multiple interfaces	Yes	Yes	Either one or all
Timestamp granularity	Nanosecond[1]	Nanosecond[1]	Nanosecond[1]
Filtering abilities	libpcap	libpcap	libpcap
Command line tool	Yes	Yes	Yes
Capturing framework	libpcap	libpcap	libpcap

[1] If hardware timestamping with support for nanosecond precision is used.

Table 4.5: Comparison of features offered by `tshark`, `dumpcap` and `tcpdump` based on [41], [42], and [43].

The Packet Capturer applies a filter when capturing and writes the results to a file and to `stdout`. The filter reduces the amount and impact of background traffic when this is present. Furthermore, the packets written to the file are used as the input for the Packet Matching, while the `stdout` is utilized for the semi-live updates.

The Packet Capturer also performs timestamping on the captured packets. `tshark` enables timestamping both based on global system time and based on the network adapter time if the Network Interface Card (NIC) supports this. We opted for the global system time to prevent issues with clock synchronization because we capture on multiple interfaces.

4.2.2 Packet Matching

As the packets traverse the network, they will be captured once at the measurement machine before it is injected into the NUT and once again at the interface connected to the wiretap. By matching the two observations of the same packet, we can extract one-way information. We employ a sliding window to perform packet matching without searching through every packet in the capture file.

The Packet Matching submodule has had two designs. Firstly, we utilized a simple First In First Out queue as our sliding window, which is described in Algorithm 4.1. The algorithm reads the packets from the capture file sequentially, and for each new packet, it searches the sliding window for a match. If no match is found, the packet is added to the sliding window. If a match is found, the match is removed from the sliding window, and information from both packets is stored.

After performing our validation, we re-designed this submodule to be less vulnerable to the sliding window filling and consequently introducing artificial packet loss during trials with either high OWD or high packet loss. With the initial design, all the captured packets were written to a single file, which was iterated through. Thus, if the PPS were high, especially when combined with high OWD or packet loss, the sliding window would get filled with packets from the first interface. Depending on the size of the sliding window, it was then possible for the window to fill up before matching packets arrived. If this happened, the corresponding packet already in the sliding window would be ejected without a match. Consequently, both packets would be stored as packets without a match. Furthermore, the match of a packet observed at a given interface can only be found at the other interface. Therefore, searching through multiple packets from the same interface is wasteful.

The new implementation of the Packet Matching mitigates these issues by using a separate capture file and sliding window for each capture interface. Alternating the capture files should make the sliding windows less full and reduce the time used to search through the sliding windows. The new design is described in Algorithm 4.2. This version is used for the case study.

Algorithm 4.1 Packet Matching with the naive implementation of the sliding window.

```
PacketSource ← Pcap
SlidingWindow ← Empty FIFO queue
File ← CSV file
for each packet p in PacketSource do
    matched ← false
    for each packet s in SlidingWindow do
        if match(s, p) then
            Remove s from SlidingWindow
            Write s and p to File
            matched ← true
            break
        end if
    end for
    if not matched then Add p to SlidingWindow
    if isFull(SlidingWindow) then
        Remove the first element from SlidingWindow
        Write the removed element to File
    end if
end for
for each packet p in SlidingWindow do Write p to File
end for
```

The matching function used in Algorithm 4.1 and 4.2 can be changed depending

Algorithm 4.2 Packet Matching with the improved implementation of the sliding window.

```
PacketSourceTx, PacketSourceRx ← Pcap1, Pcap2
SlidingWindowTx, SlidingWindowRx ← Empty FIFO queue
source1Exhausted, source2Exhausted ← false
File ← CSV file
while not source1Exhausted and not source2Exhausted do
    if source1Exhausted then continue
    p1 ← PacketSourceTx.Next()
    if p1 is null then source1Exhausted ← true
    matched ← false
    for each packet prx in SlidingWindowRx do
        if match(p1, prx) then
            Remove prx from SlidingWindowRx
            Write p1 and prx to File
            matched ← true
            break
        end if
    end for
    if not matched then Add p1 to SlidingWindowTx
    if isFull(SlidingWindowTx) then
        Remove the first element from SlidingWindowTx
        Write the removed element to File
    end if
    if source2Exhausted then continue
    p2 ← PacketSourceRx.Next()
    if p2 is null then source2Exhausted ← true
    matched ← false
    for each packet ptx in SlidingWindowTx do
        if match(p2, ptx) then
            Remove ptx from SlidingWindowTx
            Write p2 and ptx to File
            matched ← true
            break
        end if
    end for
    if not matched then Add p2 to SlidingWindowRx
    if isFull(SlidingWindowRx) then
        Remove the first element from SlidingWindowRx
        Write the removed element to File
    end if
end while
for each packet ptx in SlidingWindowTx do
    Write ptx to File
end for
for each packet prx in SlidingWindowRx do
    Write prx to File
end for
```

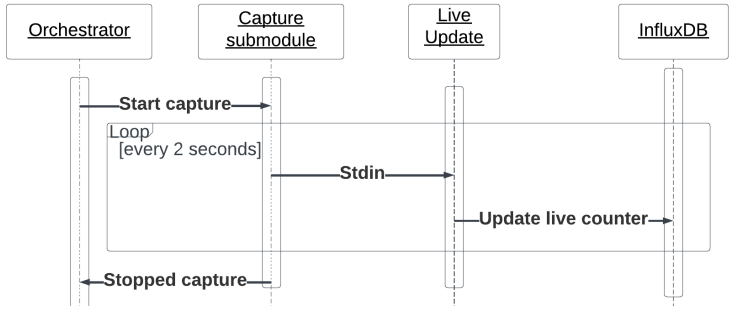


Figure 4.4: A sequence diagram showing how the live update submodule works together with the packet capture submodule.

on the trial. This increases the flexibility by making it simple to change the matching criteria. When the trial uses TCP traffic, it matches based on TCP sequence numbers [3]. If the trial uses UDP traffic, as described in Traffic Generation Alternative 1 and Alternative 2, it matches based on the sequence number within the custom payload.

4.2.3 Data Extraction

The Data Extraction submodule is responsible for parsing the relevant information from the packets and writing it to a CSV file. It stores the source and destination IP addresses, the packet size, timestamps for when the packet was transmitted and received, and if the match was found. The CSV contains all the information the Packet Analyzer requires and enables the handover between the Packet Matcher and Packet Analyzer.

4.2.4 Live updates

The Packet Matcher module also supports providing limited updates on a given interval. It works by having the `tshark`-command use the flags -w and -P. This allows the `tshark` output to be both written to a file and sent to so the captured packets are both written to the file and can be piped into this script. At the given interval, the script reads from `stdin` and extracts the packet count. For every 10,000 packets, the new packet count is written to an InfluxDB measurement that the Visualization module can later utilize. An overview of how the Live Update submodule works with the Packet Capture submodule can be observed in Figure 4.4.

[3]We are aware that because of the matching on TCP sequence numbers, it is vulnerable to retransmissions because they reuse the sequence numbers.

4.3 Packet Analyzer

This section describes the Packet Analyzer module, which calculates KPIs based on the packet data generated by the Packet Matcher module. It distinguishes between per-packet KPIs, such as OWD, and aggregate KPIs, such as packet loss. The calculators for aggregate KPIs group packets on a per-second basis and then compute its KPIs for each second. This preserves the major temporal fluctuations and simultaneously provides a meaningful number of packets for calculating aggregate KPIs. All of the calculated KPIs are subsequently written to InfluxDB. Table 4.6 shows the submodules of the Packet Analyzer and the functional requirements they contribute to. The data reader and parser reads the results of the Packet Matcher and forwards it to the KPI calculation submodule. This submodule is responsible for calculating all the configured KPIs and forwards the data to the Database Connection submodule, which writes the results to a database.

Submodule	PAFR1: calculate per-packet KPIs	PAFR2: calculate aggregate KPIs	PAFR3: write KPIs to storage
Data Reader and Parser	-	-	-
KPI Calculation	X	X	-
Database Connection	-	-	X

Table 4.6: Submodules of the Packet Analyzer module with the requirements they contribute to.

4.3.1 Data Reader and Parser

The Data Reader and Parser submodule are responsible for reading a CSV file containing summarized packet information. The records are read sequentially, and each record is parsed into a Go-struct containing source and destination IP addresses, packet size(s), transmit and receive timestamps, and whether a match was found.

4.3.2 KPI Calculation

For each KPI the benchmarking tool supports, a calculator is defined in the KPI Calculation submodule. Each calculator expects a slice of pointers to the aforementioned struct and returns a map with transmit timestamps as keys and the calculated KPI as values. This ensures that the correct order of timestamps and values is preserved when the values are stored. The input slice must be ordered in ascending order based on the transmit timestamps because some of the calculators compare successive packets [4]. All of the per-packet calculators return values whose units

[4]The transmit timestamp is chosen as the default point of reference for each packet. The original order of the packets is more closely preserved at transmission than at reception because of

are seconds. We calculate OWD, IPDV, and IAT as the per-packet KPIs. For the aggregate KPIs, we calculate instantaneous throughput, instantaneous packet loss, and delay threshold satisfaction. The calculators are presented in this order.

One-Way Delay

The calculator for OWD is based on the method described in Section 2.2.1. It subtracts the transmission timestamp from the reception timestamp to find the OWD.

IP Packet Delay Variation

Calculating IPDV is based on [16], as described in Subsection 2.2.2. We chose this equation since it considers the last 16 packets and not just the last two. This reduces the impact of individual outliers. Each IPDV-value is calculated according to Equation 2.4, which is why the ordering of the input slice matters. Otherwise, the evolution of the IPDV along the time axis becomes meaningless.

Inter-Arrival Time

The Packet Analyzer also defines a calculator for IAT for the transmit timestamps. This calculator is intended as a verification for the traffic pattern, e.g., for CBR traffic. The calculator finds the successive differences in transmit timestamps and assumes an ordered input slice.

Throughput

The throughput calculation is based on the definition from Subsection 2.2.4 and Equation 2.9. The calculator computes the sum of the total bits transmitted per second, which yields an instantaneous throughput measure. It uses the non-GTP encapsulated packet sizes to yield a consistent value independent of the NUT. Moreover, the calculator ignores packets without a match to avoid taking potentially lost packets into account.

Packet Loss

The packet loss is simply calculated as the rate of packets that do not have a match in the input slice. The calculator yields the packet loss per second of the trial for the same reason as the throughput calculator.

---

non-deterministic procedures in the network internals. Therefore, using the transmit timestamp yields more stable results.

Delay Threshold Satisfaction

Lastly, the Delay Threshold Satisfaction calculator accepts a map of delay thresholds in addition to the input slice. This calculator is conceptually similar to the packet loss calculator. It treats the packets whose OWD exceeds the threshold as lost. Based on this, it calculates the rate of packets satisfying each delay threshold. It is based on Equation 2.7. Even though it is based on the definition of availability presented in Section 2.2.3, we renamed it to delay threshold satisfaction to remove the connotations to the actual uptime of a system. For each threshold, the delay threshold satisfaction calculator returns a map like the other calculators.

4.3.3 Database Connection

The Database Connection submodule receives credentials to connect to a time series database instance. When it receives KPIs from the calculators, it writes them into the database.

4.4 Visualization

The benchmarking tool uses the Visualization module to show the KPIs calculated in a trial. It uses different Grafana dashboards to show a simple measurement during a trial and detailed measurements after a trial is completed.

4.4.1 Grafana

Grafana is a tool for data visualization. It has built-in support for various data sources for its visualizations [44]. Its core features are the dashboard and panel features, which can host a multitude of visualizations. A dashboard consists of one or more panels and is represented as a JSON object [45]. Dashboards can have variables that get passed to panels, allowing for dynamic queries and visualizations [46]. A dashboard can also be configured to re-run its queries at pre-defined intervals [47] to ensure up-to-date data. Grafana has several built-in panels, such as time series, tables, and metrics [48]. It is available both as a cloud-hosted service and as an open source with self-hosting [49].

We used a self-hosted instance of Grafana to implement our Visualization module. Grafana was chosen because it meets all the module requirements, and one of our supervisors was familiar with it.

4.4.2 Configured Grafana Dashboards

The benchmarking tool provides two dashboards. One shows a count of the captured packets during the run of a trial. This primarily serves as a verification that the

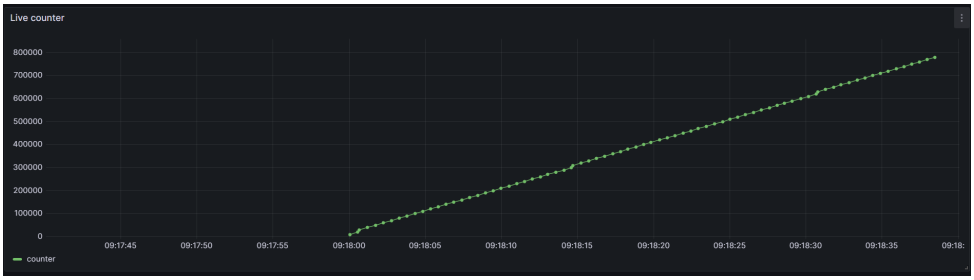


Figure 4.5: A screenshot of the live counter from the semi-live Grafana-dashboard.

Figure 4.6: A screenshot from the detailed dashboard showing the visualizations for the OWD and the IPDV.

trial is running as expected to prevent having to re-do long trials. The data source for this dashboard is described in Section 4.2.4. This dashboard re-runs its query every 2 seconds, resulting in a close-to-live counter. Figure 4.5 shows the live counter from the semi-live dashboard, which displays the number of captured packets at the ingress of the NUT.

The other dashboard displays the KPIs calculated after the trial has finished. It shows both the per-packet KPIs and the aggregated KPIs, some of these can be observed in Figure 4.6 and Figure 4.7. The data source of the dashboard is the data written to InfluxDB by the Packet Analyzer. The user is able to choose which trial to display results from by either choosing from a set of pre-defined trial names or typing in one. This dashboard enables the user to get a broad overview of the performance of the network. The dashboard elements were chosen based on input from our supervisors.

Figure 4.7: A screenshot from the detailed dashboard showing the visualization for the delay threshold satisfaction.

4.5 Orchestrator

The Orchestrator module is responsible for coordinating all the modules aside from Visualization. It needs to be able to satisfy the functional and non-functional requirements specified in Section 3.2. Its most important functionality is being able to run tasks on remote machines both synchronously and asynchronously. This is required to satisfy **OFR1** (coordinating other modules with parameters) and **OFR2** (running commands on multiple hosts). During the development of this module, we considered if any existing solutions could satisfy these requirements to avoid re-building existing functionality.

Ansible is an open-source automation platform [50]. Actions are performed either through ad-hoc commands through the Ansible CLI or through the execution of playbooks [51] [52]. Ansible playbooks are YAML files consisting of a set of plays, which again consist of a set of tasks [52]. Tasks in a play can be run either synchronously or asynchronously at different hosts [52]. Ansible supports defining a set of hosts in inventory configuration files that decides which hosts the plays are run at [53]. Playbooks can also accept variables from files and the command-line [54].

We chose to use Ansible for the Orchestrator because it satisfies all the requirements. The benchmarking tool has three Ansible playbooks for running trials, one for each alternative for generating traffic discussed in section 4.1. All playbooks receive variables from a file, and the variables describe all the individual parameters for a trial. Depending on the playbook, a different set of variables are required.

The only significant difference between the playbooks is how they handle traffic generation with regard to the three alternatives. A sequence diagram showcasing the communication between the different modules can be observed in Figure 4.8.

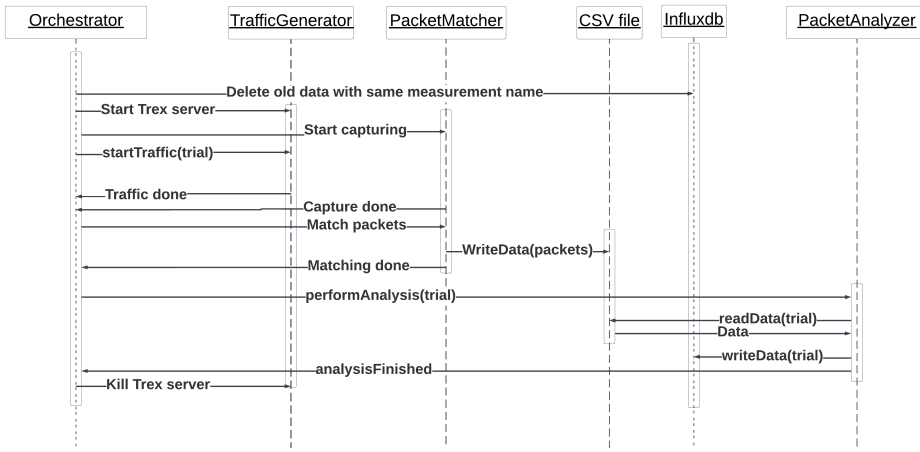


Figure 4.8: A sequence diagram showing an overview of orchestration for a trial using traffic generation alternative 1.

First, a series of tasks are performed to ensure that the current trial is unaffected by any previous trials. One such task is removing previous data from the same trial from InfluxDB. Unless Alternative 3 for traffic generation is chosen, the traffic generation is started. The Packet Capture submodule is run parallel to the traffic generation. Once the traffic generation is done and the traffic capture has been completed, the Ansible playbook resumes executing tasks synchronously. Next, the playbooks run the Packet Matching and Data Extraction submodules of the Packet Matcher. Once completed, the Packet Analyzer module uses the output CSV from the Packet Matcher module to perform the KPI calculations. At this point, the data is available in InfluxDB for the Visualization module.

4.6 Data Storage

The benchmarking tool must store data at various stages, from the initial capture of packets to the final data visualization in Grafana. By storing the data from each step of the pipeline, the benchmarking tool ensures well-defined handovers between modules. Furthermore, storing the data from each step allows for transparency for the user and enables the user to make custom investigations, supporting **TFR5** (providing possibilities for custom analysis). The storage formats and methods used are pcap, CSV, and InfluxDB. Figure 4.9 shows the different files created during a trial.

The benchmarking tool overwrites existing pcaps, CSVs, and InfluxDB measurements when a new trial with the same name is run. This is done to reduce the storage

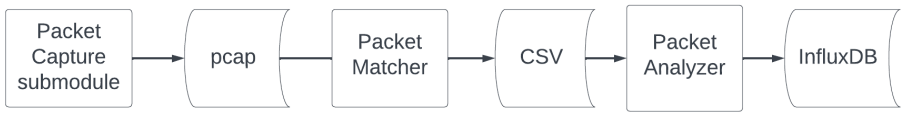


Figure 4.9: A flowchart showcasing the files created at the different steps of a trial run.

demands of the tool. If a user wishes to keep historical data, they must perform these backups themselves.

4.6.1 Packet captures

Once the packet capture submodule has captured a packet, it is added to a pcap. This file contains all the packets that passed the filters in a given trial. The benchmarking tool keeps all the pcaps in a folder reserved for pcaps, where each pcap is named after its trial name. This enables simple access to the capture files, if further investigation is necessary.

4.6.2 CSV

After the Packet Matcher module has completed processing a pcap, the result must be stored somewhere the Packet Analyzer module can access it. The benchmarking tool uses a CSV file for this purpose. A CSV file is used because only a simple standard display of information is required. There is no need to query this data or make changes to it once it has been written. Furthermore, several analysis tools support CSV as an input format. At the same time, it is relatively human-readable.

The CSV file is stored in a designated folder for the output of the Packet Matcher module. This makes it accessible and enables further custom investigation.

4.6.3 InfluxDB

Once the Packet Analyzer has calculated the KPIs, the data is stored in InfluxDB. Storing the data in InfluxDB provides several advantages. First, the tool can use the built-in query functionality of InfluxDB. The Visualization module, for instance, uses this to fetch the relevant data for each panel. Second, using a time series database simplifies utilizing the temporal aspect of the data for queries. Finally, storing the data in InfluxDB makes it simpler for different hosts to access the data by utilizing the authentication features provided by InfluxDB.

Chapter 5
Experiments

This chapter presents the various experiments we performed in our thesis. After designing and implementing the benchmarking framework in each iteration in Figure 1.1, we performed some kind of validation. The first validation of the benchmarking framework is further described in Section 5.1. Based on the insights gained during this validation, we re-designed some components of the framework. Finally, we performed a larger-scale validation, referred to as the case study, with the improved framework on two private 5G networks. This case study is described in Section 5.2.

5.1 Validation

This section presents the method for validating the benchmarking framework in the first iteration of the feedback loop in Figure 1.1. The results of this are then used to identify areas for improvement before the tool is used in the case study. Section 5.1.1 describes the testbed architecture used for performing the measurements. The validation questions and the procedures used to answer them are presented in Sections 5.1.2 and 5.1.3. By validating the tool, we ensured it performed as intended and enabled discovery of potential areas for improvement.

The validation is inspired by the design science framework, which separates validation into effect questions, trade-off questions, sensitivity questions, and requirements satisfaction questions [12]. Because of the novelty of the system, trade-off questions are omitted because of the difficulty of finding comparable alternatives. Moreover, we will not present the method for answering requirement satisfaction questions. The satisfaction of requirements will be addressed in the discussion of the validation. Finally, we will not be validating the Orchestrator or Visualization modules. These use solutions that are considered industry standards, and we assume they function as described.

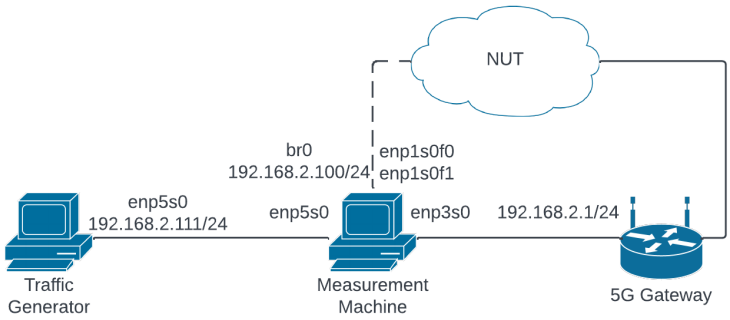


Figure 5.1: Diagram of the physical realization of the benchmarking tool.

5.1.1 Validation Testbed Setup

During the development and validation of the benchmarking tool, we used the Beyond 5G Lab at Norwegian University of Science and Technology (NTNU) as a basis for the testbed. The high-level testbed architecture was described in Section 3.3 and consists of the benchmarking tool and the NUT. This section maps the high-level testbed architecture and the physical architecture used during development and validation. First, the benchmarking tool testbed is presented. After this, the testbed architecture used for some select validation procedures is presented.

Benchmarking Tool Physical Architecture

This section describes the architecture of the benchmarking tool. It consists of a traffic generation machine, subsequently referred to as the traffic generator, and a machine performing everything related to measurement and analysis, subsequently referred to as the measurement machine, as well as a 5G gateway. Figure 5.1 illustrates how this setup was realized during the development of the benchmarking tool. The figure also includes the 5G gateway used in the testbed.

Traffic Generator The traffic generator is an Intel NUC 8i7hvk with the following NICs:

- Intel Corporation I210 Gigabit Network Connection
- Intel Corporation Ethernet Connection (2) I219-LM

The former NIC is DPDK-compatible and is used for outputting generated traffic, and the latter is used as a management interface, primarily for Secure Shell (SSH)-connections. The machine is operated by the Ubuntu 22.04 Operating System (OS). The NIC used for outputting generated traffic is directly connected to an interface of the measurement machine. The measurement machine is transparent to the traffic generator.

Cisco TRex v3.04 is installed on the traffic generator. TRex is set up using

Software	Version
InfluxDB	v2.4.7
Grafana	v10.3.3
Golang	v1.18.1
Python	v3.10.12
Ansible	v2.15.9
Tshark	v3.6.2
Tcpreplay	v4.3.4

Table 5.1: Measurement machine installed software and corresponding versions.

the aforementioned traffic generation NIC and a dummy port. TRex is assigned 192.168.2.1/24 as a default gateway. The default gateway corresponds to the 5G gateway connected to the network infrastructure in Figure 5.1.

Introducing a physically separate traffic generator is seemingly unnecessary and increases the demand for hardware. This design choice was primarily motivated by the lack of traditional network interface control when it is DPDK-bound [55]. When an interface is DPDK-bound, a user space process can directly access the interface, bypassing the Linux NAPI. Due to this, it becomes difficult to police access to the interface, and thus, control usually resides in a single application. Therefore, performing both traffic generation and packet capturing at the same interface is cumbersome, and the traffic generation and traffic capturing were distributed over two physically separate machines. The machine performing packet capturing is described next.

Measurement Machine The measurement machine is a Dell Precision 3630 Tower with the following NICs:

- Intel Corporation Ethernet Connection (7) I219-LM
- 2x Intel Corporation I210 Gigabit Network Connection
- 2x Intel Corporation Ethernet Controller X710 for 10GbE Small Form-factor Pluggable (SFP)+

The first NIC is used as a management interface, and the second type of NIC is used for interconnection between the traffic generator and gateway. The last NIC type is connected to SFPs, which is connected to a wiretap from within the network infrastructure. The measurement machine is operated by the Ubuntu 22.04 OS. Table 5.1 summarizes installed software and their corresponding versions.

The design goal of the measurement machine was that it should operate transparently in the eyes of the traffic generator and gateway. To achieve this, the measurement machine implemented a software bridge `br0` between interfaces `enp3s0`

and `enp5s0` from now on, referred to as the L2 bridge. We configured a Linux bridge, which is a virtual network device [56]. It uses the bridge-functionality of iproutes2. The following commands were used to configure it.

```
$ sudo ip addr flush dev enp5s0
$ sudo ip addr flush dev enp3s0
$ sudo ip link add br0 type bridge
$ sudo ip link set enp3s0 master br0
$ sudo ip link set enp5s0 master br0
$ sudo ip addr add 192.168.2.100/24 dev br0
$ sudo ip link set dev enp3s0 up
$ sudo ip link set dev enp5s0 up
$ sudo ip link set dev br0 up
$ sudo iptables -A FORWARD -i br0 -o br0 -j ACCEPT
$ sudo ip route add 192.168.2.1/32 dev br0
$ sudo ip route add 172.30.0.0/16 via 192.168.2.1
```

These commands enabled bidirectional traffic between the traffic generator and the 5G gateway, unaware of the presence of the bridge. Deploying the L2 bridge between the traffic generator and the gateway introduces a potential performance bottleneck. This could be mitigated by connecting the traffic generator directly to the gateway and attaching a wiretap to the link that connects to the measurement machine. However, this was not feasible due to a lack of project resources.

The 5G gateway this machine is connected to is a Teltonika RUTX50. The testbed setup used during the development of the benchmarking tool included a physical separation of the measurement machine and the gateway. The motivation was to keep the benchmarking tool decoupled from the connection to the NUT. Ensuring that the tool is agnostic of access technology, whether wired or wireless, becomes simpler by doing this.

Base Case Validation Architecture

A simplified model of the intended operating context of the benchmarking tool was used for the validation of the base case. This architecture is illustrated in Figure 5.2. Its purpose is to be able to validate the performance of the benchmarking tool itself, void of stochasticity that arises from the external radio network. All measurements performed on this architecture will show the performance of the benchmarking tool itself, i.e., only measuring OWD between the ports of the measurement machine. Some of the procedures performed during validation simplify the operating context even further. When validating individual modules, we attempt to validate them in

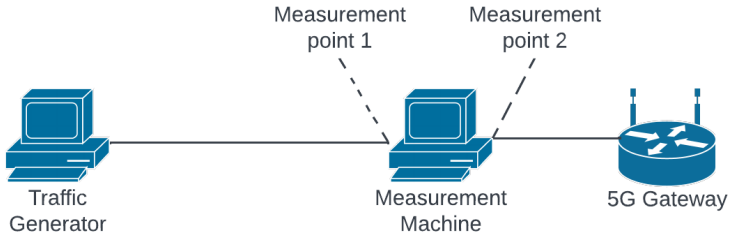


Figure 5.2: Testbed architecture used for some select validation questions for the benchmarking tool.

isolation from the other modules so that errors do not propagate between modules during validation.

5.1.2 Effect Questions

Effect questions are intended to investigate what effects are produced by the tool in its operating context [12]. Firstly, the effect questions for the first two traffic generation alternatives are presented. Thirdly, the method for validating the Packet Capturing submodule is described, followed by the description of the validation of the Packet Matching submodule. Lastly, the effect question for the Packet Analyzer module is described.

Traffic Generation Alternative 1

For traffic generation Alternative 1, we are interested in answering ”Does traffic generation Alternative 1 produce the prompted traffic?”, which seeks to answer whether the traffic pattern that the traffic generator produces adheres to the given parameters. The question tests the predictability of its behavior and is important to ensure both valid results of a single trial and consistent results across trials. This effect question is answered by issuing commands to generate traffic according to the parameters specified in Table 5.2. The traffic is then captured, and analyses are performed to find the distribution of inter-arrival times and recorded durations of the trials. This is intended to measure the accuracy and precision of the traffic generator, both in cases with high and low packet rates and with varying payload sizes.

Traffic Generation Alternative 2

For traffic generation Alternative 2, we are interested in answering ”Is `tcpreplay` able to replay the traffic accurately?”. To answer this question, we created a script that generates a pcap where the packets have an IAT taken from a normal distribution with a configurable mean μ and standard deviation σ. This script generated three capture files, and their respective μ and σ can be observed in Table 5.3. Then, we

	Packet Rate	Duration	Payload Size
1	10pps	100s	16B
2	10pps	100s	426B
3	10pps	100s	1432B
4	5,000pps	50s	16B
5	5,000pps	50s	426B
6	5,000pps	50s	1432B
7	10,000pps	30s	16B
8	10,000pps	30s	426B
9	10,000pps	30s	1432B

Table 5.2: Combinations of packet rates, durations, and payload sizes for validating traffic generator Alternative 1.

used `tcpreplay` to replay the capture file while we performed capturing with tshark on the same interface. Finally, we compared the original capture files with those replayed with `tcpreplay`.

	μ	σ
1	0.01s	0.0015s
2	0.0005s	0.0001s
3	0.0001s	0.00001s

Table 5.3: The μ and σ values used in for generating the files for validation of `tcpreplay`

Packet Capturer Submodule

The effect question to be answered for the Packet Capturer submodule is "Does the Packet Capturer submodule capture as many packets as expected?". This question determines if the submodule captures all the relevant offered traffic. Determining this is important because if it captures less than what is expected, the tool produces artificial packet loss.

To answer this question, we generated a pcap with a known number of packets that we replayed while the Packet Capturer submodule was capturing. The pcap was generated by having our traffic generation module generate 60,000 custom UDP packets over one minute while we used `tshark` to capture this traffic.

We performed three replays and captures to validate the submodule. First, we began by using it to capture on the interface `enp3s0` with a filter that will be used on this interface. This should result in a pcap with 60,000 of our custom packets.

Second, we used it on the interface `enp1s0f1` with a filter that will be used on this interface. In this case, it should result in a pcap with 60,000 of our custom packets encapsulated in GTP. Finally, we performed capturing on both interfaces with the associated filter for each interface. Each of our packets should then be captured once at each interface, resulting in a pcap with 120,000 of our packets.

Packet Matching Submodule

For the Packet Matching submodule, we pose two effect questions. Firstly, we are interested in finding out "Does the Packet Matching submodule match the expected number of packets?". This question seeks to determine if the submodule performs the expected number of matches for a given pcap where the number of matches is known for both UDP and TCP traffic. The second effect question is "Is a sliding window size of 20,000 adequate to not add artificial packet loss for 30,000PPS?". This question seeks to validate that a size of 20,000 is adequate for at least up to 30,000PPS. We have a sliding window size for a given packet rate that we know is adequate.

To answer the first question for UDP traffic, we made two pcaps and used them to test the number of matches. The first pcap is the one created in the subsection for Packet Capturing. For this pcap, the number of matches should be 60,000. The second pcap was used to verify that it identifies packets without a match. This pcap is the same as the first pcap, but we removed ten packets that are not related to each other using `editpcap`.

To answer the first question for TCP traffic, we followed the same methodology as for UDP traffic. We started by creating two pcaps and checking the number of matches that occurred when running them through the Packet Matching submodule. The first pcap was created by capturing the traffic generated from a simple TCP client sending packets to a simple TCP server on our network. The second pcap was created by removing ten unrelated packets from the first pcap.

To answer the second question, we generated a pcap file with a known number of matches. The pcap file was generated using Alternative 1 from the traffic generator with PPS set to 30,000. Then, the Packet Matching submodule was run on the pcap with varying sliding window sizes, going up in increments of 250 from 250 until no artificial packet loss was introduced.

Analysis Module

The effect question for the analysis module is "Does the Packet Analyzer provide the expected analysis of input data with known KPIs?" which seeks to find out whether the values of the KPIs that are calculated correspond to those of the input data.

	Inter-arrival time	$E[OWD]$	$Std(OWD)$	Packet loss probability	Throughput
1	1ms	12ms	5ms	1%	459,360bps
2	1ms	32ms	7.5ms	5%	440,800bps
3	1ms	20ms	10ms	20%	371,200bps
4	1ms	12ms	0ms	0%	464,000bps
5	1ms	32ms	0ms	0%	464,000bps

Table 5.4: Parameters used for generating input to the analysis module during validation of the analysis module.

To validate this in isolation, we manually generate the input data to the analysis module. This way, we can verify the IAT of packets, the distribution of the OWD, the packet loss, and the throughput. We define five combinations of parameters, specified in Table 5.4. The first three include stochasticity in the OWD and packet loss probability. The throughput is not configured explicitly but calculated using Equation 5.1. In total, 1,000,000 rows of input data are generated per definition in the table. Data generation for this validation uses the *random* module of the `numpy` v1.26.3 Python library. The generated data is then provided as input to the analysis module, the output of which is analyzed and compared against the values in Table 5.4.

$$Throughput = Packet\_size \cdot 8 \cdot Packets\_per\_second \cdot (1 - Packet\_loss\_probability) \tag{5.1}$$

5.1.3 Sensitivity Questions

Sensitivity questions validate how the created tool works in varying operating conditions [12]. The Packet Capturer and Packet Analyzer modules are the ones that have been identified to be affected by changing conditions. Therefore, these modules will be given special attention in this section. Firstly, the sensitivity question for packet loss is presented, followed by a similar question for network delay. Both of these questions introduce changing conditions using NetEm [57]. In the following sections, the sensitivity is tested by changing the payload size and the packet rate of the generated traffic. Table 5.5 summarizes the trials used to generate data for answering the sensitivity questions. This section presents sensitivity questions for packet loss, network delay, payload size, and packet rate in this order.

Packet Loss Sensitivity

Packet loss can potentially impact the results and performance of the Packet Matching submodule. Therefore, we are interested in answering "What happens to the performance and results of the Packet Matcher and Packet Analyzer modules when packet loss increases?". We used NetEm to introduce packet loss to the network to answer this question. The three trials specified in Table 5.5 are used for each packet loss value. We tested for 2% and 15% packet loss. In many cases, this high packet loss is unrealistic, but it still indicates how the benchmarking tool performs under these conditions.

We analyzed the results and performance of the Packet Matching submodule and the Packet Analyzer module. We expected the number of packets without a match to exceed the configured packet loss because of the sliding window containing packets that will not be matched.

Even though NetEm performs stateless dropping of packets based on a given probability, which is unrealistic in real scenarios, we argue that the validation results are still relevant. This validation aims to show how the packet matching submodule and packet analyzer behave under varying conditions, which is illustrated despite the conditions being unrealistic.

Network Delay Sensitivity

The delay between measurement points in the benchmarking tool can potentially impact the results and the performance of the Packet Matching submodule. Therefore, we ask "What happens to the results of the Packet Matcher module when the network delay increases?". To answer this question, we use NetEm to introduce delay on interface `enp3s0` on the measurement machine. We subsequently execute trials with the parameters shown in Table 5.5. We tested each trial for three network delays: $5ms$, $50ms$, and $100ms$. We believe these values capture a broad range of delay values that the tool should be able to handle.

After the trials were finished, we analyzed the results and performance of the

Trial	Packets per second	Duration
1	5pps	7,200s
2	1,000pps	600s
3	10,000pps	60s

Table 5.5: Specifications of trials for testing configurations under sensitivity questions.

Packet Matching submodule. The sliding window in the packet matching submodule can be affected the greatest by an increased delay, which can impact the number of packets matched and the processing time of the submodule. Intuitively, the number of matched packets should decrease as the delay increases because the sliding window fills with packets observed at the first measurement point, and the capture file is read sequentially. Moreover, we expect that the sliding window is, on average, fuller, leading to an adverse effect on the processing time. To trigger this happening, we reduced the size of the sliding window such that no packets from the second capture interface are examined before the sliding window is full. The size of the sliding window was 510, just above $10,000PPS \cdot 50ms = 500P$, which makes it likely that some packets will be ejected before the match is read from the pcap.

A study from 2011 found that the accuracy of NetEm declined for configured delays below $50ms$, with errors around $0.5ms$ [58]. Moreover, the author of NetEm states that the accuracy of the kernel timer limits the accuracy [59]. Despite this potential inaccuracy, we believe that using NetEm still provides interesting information when analyzing the performance trends when the delay values are increasing.

Payload Size Sensitivity

The Packet Matching submodule must be able to handle traffic with varying payload sizes. We ask, "How does the Packet Matching submodule perform when the payload size increases?". The procedure for answering this question is the same as for packet loss and network delay sensitivity. The same trials from Table 5.5 are used, with configured payload sizes of $16B$, $426B$, and $1432B$. The latter values were chosen to achieve $512B$ and $1518B$ consecutively when they become GTP-encapsulated. Based on our observations, this is close to the Maximum Transmission Unit.

When the payload size increases, we expect the processing time of the Packet Matching submodule to grow as well. This is because it has to read and parse a larger file.

Packet Rate Sensitivity in L2 Bridge

For the packets to reach the NUT, they must traverse the L2 bridge. Thus, knowing the behavior of the network bridge under high loads provides insights into the scalability of the hardware setup of the benchmarking framework. We pose the question, "How do changes in packet rates affect the L2 bridge?".

To answer this question, we generated traffic with our Traffic Generator using Alternative 1 and performed capturing at both the ingress and egress ports of the bridge. We subsequently compared the number of captured packets at each interface to see whether any packets were dropped. This was done three times for all the values

between 5,000PPS and 30,000PPS in increments of 5,000. We chose this interval because this would validate that the bridge could handle the traffic used for our case study.

Furthermore, to observe if the packet sizes impacted the performance of the L2 bridge we used different payload sizes. First, we used a payload of 16B to represent smaller packets. Then, we used a payload 426B to represent medium-sized packets. Finally, we used a payload of 1432B to represent large packets.

5.2 Case Study

This section describes the case study performed during the second iteration of the feedback-loop in Figure 1.1. The case study consisted of using the benchmarking framework on two different private 5G networks, and comparing their results. Section 5.2.1 describes the methodology used, while Section 5.2.2 describes the architectures of the networks.

The purpose was to provide more data that could be used to answer RQ2, RQ3, and RQ4. Firstly, using the framework on multiple networks would provide more practical insights into the requirements for integration with the NUT. Secondly, choosing trial parameters that were representative of industry 4.0 use cases would allow some verification of the degree to which the framework could support such use cases. Finally, by benchmarking multiple networks and comparing the results we could understand how the framework could be used for comparison of networks.

5.2.1 Method for Performing Case Studies

For each network, we performed trials with varying parameters as shown in Table 5.6. The trial parameters were chosen specifically to mimic certain Industry 4.0 traffic scenarios. Trial 1 resembles a single sensor sending continuous updates over a long period. The results from this trial emphasize how the networks perform under a low load over a longer period. The duration of the trial was originally intended to be significantly longer, but due to connection stability issues in the open-source network, it was shortened. Trial 2 resembles a scenario with higher bandwidth, such as transferring live surveillance camera feeds. The last trial can resemble a scenario where a few devices flood the network with packets. All three trials represent but do not map directly to a use case from [33]. The first trial represents condition monitoring (safety), while the second represents motion control. Finally, the third trial represents AR/VR. The trial parameters do not map directly to the use cases because we wanted to consider other aspects as well, such as very low throughput in trial 1.

Trial	Packets per second	Duration	Payload size
1	5pps	1,800s	16B
2	1,000pps	600s	1200B
3	30,000pps	60s	16B

Table 5.6: Specifications of trials for performing the case study.

To generate the traffic for all three trials we used the Traffic Generation Alternative 1 because of its ease of use as well as a lack of suitable pcaps.

Each trial was repeated five times to reduce the impact of transient performance factors.

It should be noted that the B5G Lab at NTNU is a testbed for several concurrent research projects. Consequently, we could not perform trials in isolation from external traffic. Therefore, the produced results from this network reflect how it behaves when it is in active use by other entities. However, by repeating the trials five times, the effects of this factor are reduced. On the other hand, the Open-source network was configured specifically for our trials, and we could execute the trials without any interfering traffic.

5.2.2 Architectures Used for Case Studies

This section describes the architecture and components of the networks where the case study was performed. First, the architecture for the Open-source network at NTNU is presented. Following this, the B5G Lab network at NTNU is described. Both architectures resemble the testbed architecture in [20].

Network Internals at NTNU 5G Open-Source Network

For the first case study, we executed trials on a 5G network consisting of open-source components. The architecture of this network is depicted in Figure 5.3. The network consists of a RRH, a BBU, and a 5GC. The RRH and BBU are co-located on the same physical machine, which is a Dell OptiPlex 7040 operated by Ubuntu 18.04.6 LTS x86_64 with an Intel i7-6700 CPU. The RRH and BBU are controlled by software from the OpenAirInterface repository. The code was pulled from the repository on commit `56c4a6ec07` [1]. It operates in frequency band `n78`. The 5GC runs on a Dell Precision 3630 Tower operated by Ubuntu 22.04.4 LTS x86_64 with an Intel i7-8700 CPU. It uses open-source core software from Open5GS v2.7.1 [2].

[1] https://gitlab.eurecom.fr/oai/openairinterface5g/-/commit/56c4a6ec07e2093351112314beeda3228beb66f1
[2] https://github.com/open5gs/open5gs/releases/tag/v2.7.1

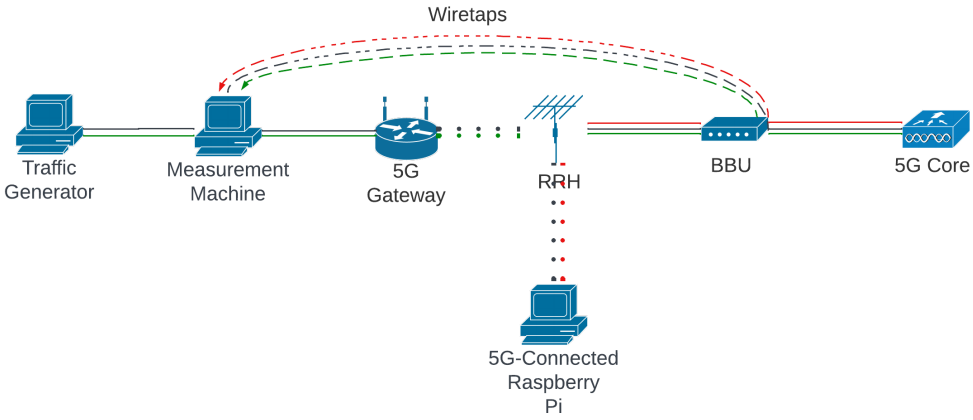


Figure 5.3: Diagram illustrating network internals and direction of traffic from the benchmarking tool to 5G-connected Raspberry Pi in the open-source network.

The direction of the traffic is also illustrated in Figure 5.3. The green lines illustrate uplink traffic before it has been processed by the UPF in the 5GC, and the red lines illustrate uplink traffic after it has been processed by the UPF. It is especially worth noting what traffic passes through the wiretaps. The green line of the wiretap is connected to interface `enp1s0f0` of the measurement machine, while the red line is connected to `enp1s0f1`.

The wiretap is connected to the link between the BBU and the 5GC. This enables capturing traffic after it has been processed by the 5GC. The traffic destination in this setup is a Raspberry Pi connected to the same RRH as the benchmarking framework.

Network Internals at NTNU B5G Lab

The network in the B5G Lab at NTNU is illustrated in Figure 5.4. The network consists of two pico RRHs, a single BBU, a 5GC, as well as a gateway towards the Internet. All of the 5G infrastructure is delivered by Nokia. The RRHs are Nokia 5G AirScale Indoor Radios, the BBU is a Nokia 5G Airscale System Module, the switch is a Nokia 7520 IXR-e Interconnect Router, and the 5GC runs Nokia software on an HPE EL1000.

The figure also illustrates the direction that the traffic traverses in the network. The color scheme is the same as for the figure for the Open-source network.

The endpoint of the traffic in Figure 5.4 is a Raspberry Pi with a 5G-module, which is reachable through a second RRH. The choice of endpoint for the trials was arbitrary. The only criterion was that the traffic traversed the given BBU that the

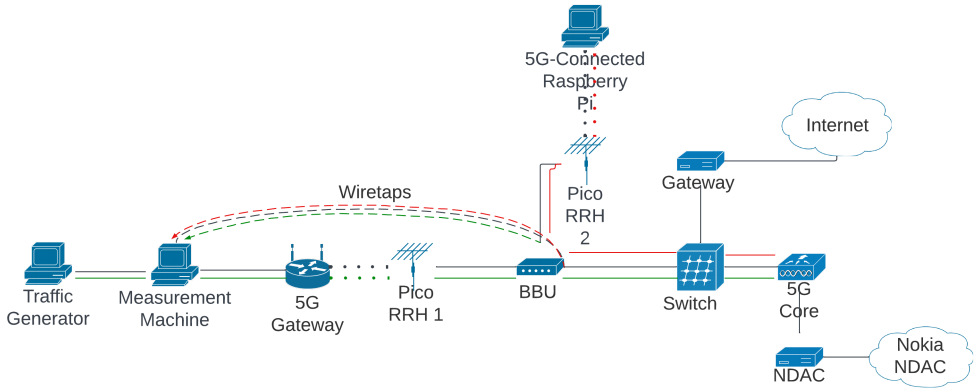


Figure 5.4: Diagram illustrating the network internals and direction of traffic from the benchmarking tool to the 5G-connected Raspberry Pi in the B5G Lab.

wiretaps were connected to in both directions. The placement of the wiretap is the same as for the Open-source network. This helps ensure that the results from the two architectures are as comparable as possible. Since the destination used for our traffic is not the traffic generator, the calculated one-way KPIs is an approximation as in [20].

Chapter 6
Results

This chapter presents the results of the trials outlined in Chapter 5. These results will be used to evaluate the performance of the benchmarking framework as illustrated in Figure 1.1, both from the validation and the case study. Section 6.1 presents the results from the validation of the benchmarking tool. Sections 6.3, and 6.2 present the results from the case studies. Lastly, Section 6.4 presents the comparison of the case studies.

6.1 Validation Results

This section presents the results from performing the validation as described in Section 5.1. First, the results of the effect questions are presented in Section 6.1.1, followed by a presentation of the results of the sensitivity question in 6.1.2.

6.1.1 Results of Effect Questions

The results of the effect questions are presented in the same order they were presented in Chapter 5. The two Traffic Generator alternatives are presented first, followed by the Packet Capturer, the Packet Matching and the Packet Analyzer.

Traffic Generation Alternative 1

The effect question for traffic generation Alternative 1 is "Does traffic generation Alternative 1 produce the prompted traffic?".

The results from running the trials specified in Table 5.2 are displayed in Table 6.1. As the table shows, the standard deviations of the IATs are consistently large for the trials with high packet rates on rows 4 through 9. The IAT oscillates between a low and a high value, as illustrated in Figure 6.1.

Figure 6.2 shows the ECDFs of IAT for varying payload sizes and packet rates. For 10PPS, the ECDF is more or less symmetric around the mean IAT, and there is

	Duration	$E[IAT]$	$Std(IAT)$
1	100.127s	100.127ms	0.033ms
2	100.028s	100.128ms	0.009ms
3	100.028s	100.128ms	0.009ms
4	50.063s	0.200ms	0.399ms
5	50.063s	0.200ms	0.393ms
6	50.063s	0.200ms	0.377ms
7	30.038s	0.100ms	0.298ms
8	30.037s	0.100ms	0.289ms
9	30.037s	0.100ms	0.264ms

Table 6.1: Results of running Traffic Generator Alternative 1 with the trials specified in Table 5.2.

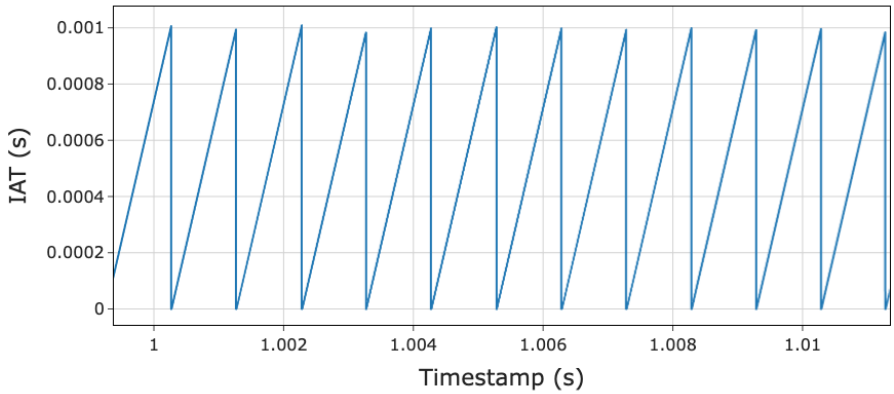


Figure 6.1: Time series plot of IATs for CBR of 10,000 packets per second.

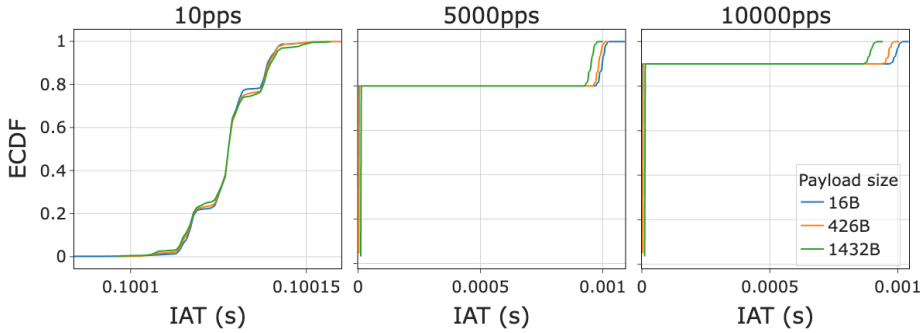


Figure 6.2: ECDFs for all packet rates, for payload sizes 16B, 426B and 1432B.

little to separate the different payload sizes. The ECDFs for 5,000PPS and 10,000PPS in Figure 6.2 shows that around 80% and 90% of the packets, respectively, arrive with an IAT close to $0ms$. Moreover, the graphs display observable differences between the payload sizes.

The oscillation in Figure 6.1 and ECDFs in Figure 6.2 suggest that TRex utilizes batching for packet transmission. Further investigation of packet inter-arrival times from TRex supports this claim. It appears to perform batching when the inter-transmission time is less than $1ms$. The expected inter-transmission time resembles Equation 6.1, with a representing the expected batch size calculated using Equation 6.2, and b is the packet transmission time. The packet transmission time seemingly scales proportionally with the packet size. This is natural with what could be expected from a DPDK-based traffic generation tool [60], [61]. Batching increases the performance but reduces the precision of the generated CBR traffic. An increase in a results in more packets being transmitted in a burst in each batch. We expect this to result in a decrease in the standard deviation of inter-arrival times. The standard deviations of the inter-arrival times in Table 6.1 demonstrate this effect between 5,000PPS and 10,000PPS.

$$E[IAT] = \frac{(a-1)\cdot b + (10^{-3}s - b)}{a} \tag{6.1}$$

$$a = \lfloor \frac{PPS}{10^3} \rfloor + \frac{1}{(PPS \mod 10^3)} \cdot 10 \tag{6.2}$$

The measured duration in Table 6.1 is consistently between $25ms$ and $130ms$ longer than the configured duration, and the average IAT of packets is often greater than expected. This indicates a skew in the IAT of packets. However, because the durations and average inter-arrival times never exceed their configured values by more than 0.2%, this effect is not considered detrimental to the effect of the Traffic Generator.

Based on the results in Table 6.1, while traffic generation Alternative 1 meets the expected IAT on average per millisecond, it potentially compromises **TGNFR1** when the packet rate exceeds 1,000PPS due to packet batching. This makes it unsuitable for scenarios with stringent requirements for stable IATs. TRex evidently does not enable a batch size of 1, which would improve the precision of its CBR traffic. MoonGen, on the other hand, does [35]; hence, using MoonGen could have alleviated this problem. However, if the requirements for inter-transmission times do not necessitate stability on a scale of less than $1ms$, traffic generation Alternative 1 complies with both **TGFR1**, **TGFR2**, **TGFR3**, and **TGNFR1**.

Traffic Generation Alternative 2

The effect question for Traffic Generation Alternative 2 is, "Is `tcpreplay` able to accurately replay the traffic?". Table 6.2 shows the results when comparing the generated capture files with the replayed ones. It shows the difference in the duration of the traffic stream, the difference in the total amount of packets, and the difference in the distributions.

μ	σ	Difference in duration	Difference in packet amount	Difference in μ	Difference in σ
0.01s	0.0015s	47μs	0	4.7ns	8.4ns
0.0005s	0.0001s	30μs	0	2.9ns	8.3ns
0.0001s	0.00001s	17μs	0	1.7ns	7.2ns

Table 6.2: Comparison of the generated pcaps with the replayed pcaps for Traffic Generation Alternative 2.

It can be observed that the difference between the generated pcap and the replayed one is almost indistinguishable with regard to duration and the distribution of the traffic. Based on this, we conclude that `tcpreplay` is adequate for our purposes. Thus, as long the generated or provided capture file is correct, the benchmarking tool is able to fulfill **TGFR1** with Traffic Generation Alternative 2.

Utilizing a negative exponential distribution for the inter-transmission time when generating the capture file could have been more representative of the real world. However, we assume there is little difference regarding `tcpreplay`, whether the inter-transmission time between packets follows a normal or negative exponential distribution. Thus, we would argue that this still provides adequate validation for **TGFR1** with Traffic Generation Alternative 2.

Packet Capturer Results

The effect question for the Packet Capturer submodule is ”Does the Packet Capturer submodule capture as many packets as expected?”. Table 6.3 shows the numbers of captured packets for each replay.

Interface	`enp3s0`	`enp1s0f1`	`enp3s0` and `enp1s0f1`
UDP packets captured	60,000	60,000	120,000

Table 6.3: The number of packets captured by the Packet Capturer submodule when replaying the pcap with 60,000 packets.

As shown in Table 6.3, all the offered traffic has been captured. We did not validate for higher packet rates than 30,000, as this is higher than anything we

intended to use in our case study. Furthermore, the number of captured packets was as expected for each interface. This validates that the filters work as intended and, consequently, do not filter out relevant traffic for the given trials.

Packet Matcher Results

The first effect question for the Packet Matching submodule is "Does the Packet Matching submodule match the expected number of packets?".

The number of matched packets for the Packet Matching submodule can be seen in Table 6.4. The Packet Matching submodule matches the expected number of packets for UDP and TCP traffic.

Transport protocol	All 60,000 packets	Removed 10 packets
UDP	60,000	59,990
TCP	60,000	59,990

Table 6.4: The number of matches made during the validation of the Packet Matching submodule for UDP- and TCP-based traffic.

The second effect question is, "Is a sliding window size of 20,000 adequate to not add artificial packet loss for 30,000PPS?". Table 6.5 shows the percentage of packet loss for the varying sizes of the sliding window. After increasing the sliding window size to 2,750, there was no artificial packet loss introduced, and therefore no higher sizes are shown for the sliding window. To verify that the remaining packet loss was not artificial a sliding window size of 100,000 was used once. This gave the same number of matches as a sliding window of 2750.

This shows that the sliding window has the ability to produce artificial packet loss if it is too small. For the duration of the traffic generation, the mean OWD lay between 11.5ms to 16ms. Based on this, it appears that a sliding window size of 20,000 is adequate with a large margin for a packet rate of 30,000PPS and a OWD in the range of 11.5 to 16ms.

Packet Analyzer Results

The effect question for the Packet Analyzer is "Does the Packet Analyzer provide the expected analysis of input data with known KPIs?".

The results for each of the dataframes are displayed in Table 6.6. All floating point results have been rounded to a precision of 3 decimal digits. The IATs are more or less exactly as configured. The standard deviations of the OWD deviate from the configured standard deviations from the normal distribution they are sampled from.

Sliding Window Size	Packet Loss (%)
250	99.34%
500	93.43%
750	88.09%
1,000	66.46%
1,250	58.08%
1,500	4.31%
1,750	3.94%
2,000	0.39%
2,250	0.39%
2,500	0.39%
2,750	0.13%

Table 6.5: The percentage of packet loss for different sliding window sizes.

	Inter-arrival time	$E[OWD]$	$Std(OWD)$	Packet loss	Throughput
1	1ms	12.118ms	4.858ms	0.982%	459,449bps
2	1ms	31.992ms	7.501ms	4.993%	440,803bps
3	1ms	20.552ms	9.417ms	20.003%	371,187bps
4	1ms	12.0ms	0.0ms	0%	464,000bps
5	1ms	32.0ms	0.0ms	0%	464,000bps

Table 6.6: Results of running the analysis module with the generated data from Table 5.4.

The same goes for the packet loss probabilities. The throughputs are also consistently a little bit greater than expected for the parameters with stochasticity. On the other hand, the parameters without stochasticity exactly match their configured values.

In the first and third rows, the expected OWD is slightly higher than their targets, and the standard deviation is slightly lower than their targets. Upon further inspection, this is due to the standard deviations producing several negative OWDs, as seen in Figure 6.3, which are filtered out when calculating the KPIs. This skews the mean of the resulting distribution, leading to a higher mean and lower standard deviation. We do not see the same effect for the second row because the distribution rarely produces negative values. Moreover, for the rows without stochasticity, the average OWD is the same as its target. Based on this, we conclude that the Packet Analyzer module satisfies **PAFR1** (calculating the pre-defined set of per-packet KPIs).

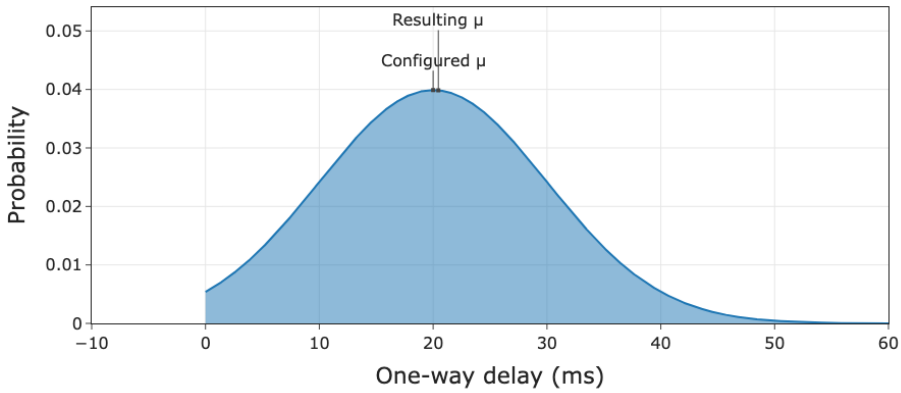


Figure 6.3: Illustration of skewed mean of normal distribution with $\mu = 20ms$ and $\sigma = 10ms$ when negative values are filtered out.

The packet loss in Table 6.6 is more or less exactly as expected. Moreover, because the throughput is a product of packets per second, packet size, and packet loss probability, and the packet loss probability is the only stochastic variable in the product, we should expect that the throughput and packet loss are negatively correlated. The results in Table 6.6 support this, which, in conjunction with the seemingly correct calculation of packet loss, indicates that the Packet Analyzer module fulfills **PAFR2** (calculating aggregate KPIs).

6.1.2 Results of Sensitivity Questions

The results of the sensitivity questions are presented in the same order the questions were presented in Section 5.1.3. We start with presenting the results for the packet loss sensitivity, followed by the network delay sensitivity and payload size sensitivity. Lastly, the results for the packet rate sensitivity are presented.

Packet Loss Sensitivity

For packet loss, the sensitivity question is ”What happens to the performance and results of the Packet Matcher and Packet Analyzer modules when the packet loss increases?”. Our hypotheses for this question are (1) the calculated packet loss will exceed the configured packet loss due to how the sliding window in the Packet Matcher works, and (2) the performance of the Packet Matcher degrades when the packet loss increases.

In Figure 6.4, we observe that the calculated packet loss for all trials with 2% configured packet loss slightly exceeds 2% except for the last trial, but is exactly

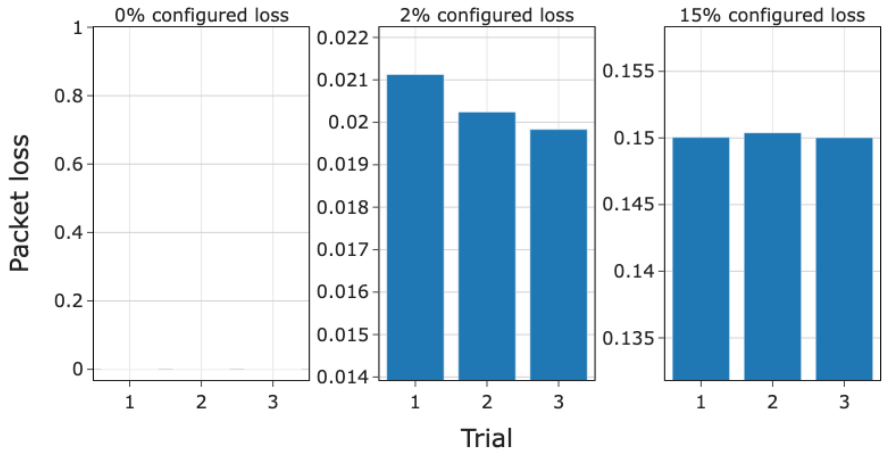


Figure 6.4: Packet loss for all trials in the validation of packet loss sensitivity compared to the base case.

0% when no loss is configured with NetEm. For 15% configured loss, the calculated packet loss lies roughly around 15%. This indicates that the calculated packet loss, in fact, is not consistently higher than the packet loss configured in NetEm for the second and third trials. On the one hand, the deviation between configured and calculated packet loss might be due to how the stateless packet dropping in NetEm works. This is corroborated by the fact that the calculated packet loss for the first trial exceeds its target for 2% packet loss. The first trial only inserts around 18,000 packets into the sliding window, which is less than the window size of 20,000 packets. On the other hand, the deviations in the second trial suggest that the sliding window implementation tends to add artificial packet loss when actual packet loss is high, which supports our first hypothesis. During the second and third trials, a total of 1.2 million packets are expected, corresponding to 24,000 and 180,000 packets without a match for the configured packet losses, consecutively. Because of this the sliding window is likely to be filled at some point, and eject packets that would have been matched later. In this case, the initial ejected packet and the corresponding later packet will be considered packets without a match. Thus, increasing the calculated packet loss.

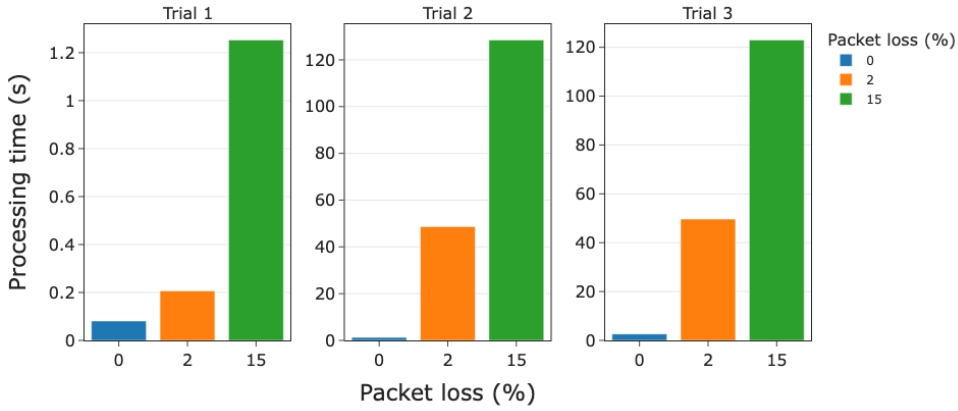
However, according to our first hypothesis, we expect the difference between configured and calculated packet loss to increase as the configured packet loss increases. The results do not indicate this. Moreover, the deviations from the configured packet loss are small, which suggests that the differences in packet loss are due to stochasticity in packet dropping by NetEm.

For the performance, Figure 6.5 displays the relationship between the trial, i.e., the total number of packets and packet rate in the facets, the packet loss percentage on the x-axis, and the processing time in seconds on the y-axis. Subfigure 6.5a shows the processing times for the Packet Matcher module, while Subfigure 6.5b shows the processing times for the Packet Analyzer. The plot for the Packet Matcher for the first trial, i.e., 5PPS, shows that the processing time increases substantially with the packet loss percentage. This trend is not visible in the Packet Analyzer for this packet rate. Moreover, for the second trial, i.e., 1,000PPS for 600 seconds, the difference in processing time for the Packet Matcher is more apparent. For 15% packet loss, the Packet Matcher used close to $120s$ to process the entire file, and roughly $50s$ for 2% packet loss. The Packet Analyzer also took longer to process the packets with 15% packet loss. For the third trial, i.e., 10,000PPS for 10 seconds, the Packet Matcher used roughly the same amount of time as for the second trial. This is as expected because the total number of packets is the same. The Packet Analyzer, on the other hand, used less time than it did for 1,000PPS.

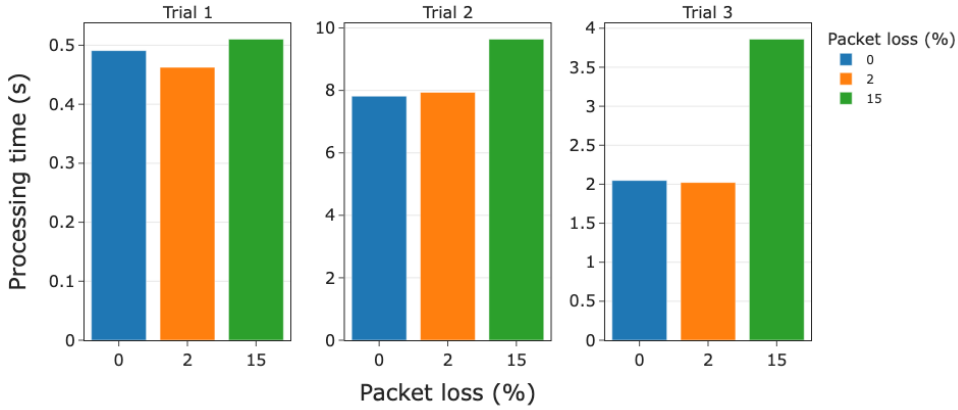
The performance data presented in Figure 6.5 supports our expectations from the second hypothesis about the Packet Matcher. There is an obvious increase in processing time for the Packet Matcher when the packet loss increases. Based on the similarity between the Packet Matcher processing times for 1,000 and 10,000 packets per second, it seems like the packet rate does not impact the processing times as much as the packet loss and total number of packets does. This increase is likely due to the limitations of the sliding window. Packet loss leads to a fuller sliding window, increasing the number of packets searched when matching. We believe this leads to increased processing time in the Packet Matcher.

The processing time of the Packet Analyzer also increases with the number of packets. However, the processing time between trials 2 and 3 is not the same as with the Packet Matcher. The duration of trial 2 is 600 seconds, while the duration of trial 3 is 60 seconds. Some of the KPIs calculated by the Packet Analyzer aggregates the packets into 1-second buckets. Therefore, for the second trial, this results in 540 more buckets, which we believe could cause an increased processing time, even though the total number of packets is the same.

The results are inconclusive regarding the first hypothesis. The results from the first trial suggest that the deviation is caused by stateless packet dropping in NetEm. However, the deviations for the second and third trials support the hypothesis. Finally, we would have expected artificial packet loss to be noticeably higher for 15% packet loss than for 2% packet loss. Therefore, for this combination of sliding window size, OWD, and packet loss, it is hard to either prove or disprove the hypothesis with these results. The second hypothesis, on the other hand, seems to be correct for the Packet Matcher. The trial processing times display a correlation with the packet loss



(a) Processing time of the Packet Matcher module for each configured trial with configured packet loss between 0 and 15%.



(b) Processing time of the Packet Analyzer module for each configured trial with configured packet loss between 0 and 15%.

Figure 6.5: Processing time of the Packet Matcher and Packet Analyzer modules for varying levels of packet loss, for each trial.

and number of packets. Moreover, this also seems to hold for the processing time of the Packet Analyzer. However, it appears also to be impacted by the duration of the trial.

Network Delay Sensitivity

Regarding network delay sensitivity for the benchmarking tool, we were interested in answering "What happens to the results of the Packet Matcher module when the network delay increases?". We hypothesize that the calculated packet loss will, at some point, increase beyond the actual packet loss due to how the sliding window in the Packet Matcher works.

Figure 6.6 shows the calculated packet loss for the third trial and the network delay configured with NetEm. The first and second trials have been omitted from the figure due to not having any artificial packet loss. Before performing an analysis, we verified that the data contained no non-artificial packet loss. For the third trial, which transmits 10,000PPS, it is evident that the sliding window causes packet loss. In the case of $50ms$ delay, $10,000PPS \cdot 0.05s = 500P$ arrived at the first capture interface before any packet arrived at the second capture interface. Because the sliding window could fit 520 packets, a significant amount of packets were likely ejected because the sliding window was filled with packets from the first interface. The same argument holds for the third trial with $100ms$ configured network delay. For $5ms$ configured delay, on the other hand, $10,000PPS \cdot 0.005s = 50P$ arrived at the first interface before any packet arrived at the second capture interface. Therefore, the sliding window was far from full when matching packets were examined.

These results support our hypothesis. We expected packet loss to occur when the product of the packet rate and the configured delay approached the sliding window size, which was exactly what we saw. Thus, we can conclude that increased network delay may cause the Packet Matcher module to introduce artificial packet loss. Moreover, because an increased packet loss rate leads to an increased processing time for the Packet Matcher module, an increased network delay might also increase the processing time.

Payload Size Sensitivity

For the payload size sensitivity, we asked, "How does the Packet Matching submodule behave when the payload size increases?". The implementation of this submodule indicates that only the time to read the capture files into memory should be affected by the increased payload size. After this, the packets are parsed into fixed-size structs.

As can be seen in Figure 6.7, the processing time of the Packet Matcher module appears to be impacted by the payload size for 5 and 1,000 packets per second. However, for 10,000 packets per second, this is not so apparent. Because the total number of packets read in the trials with 1,000 and 10,000 packets per second roughly correspond to each other, we would expect the processing times to be equal. This

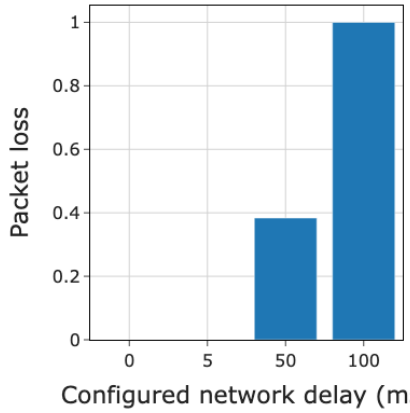


Figure 6.6: Packet loss for the third trial validating network delay sensitivity.

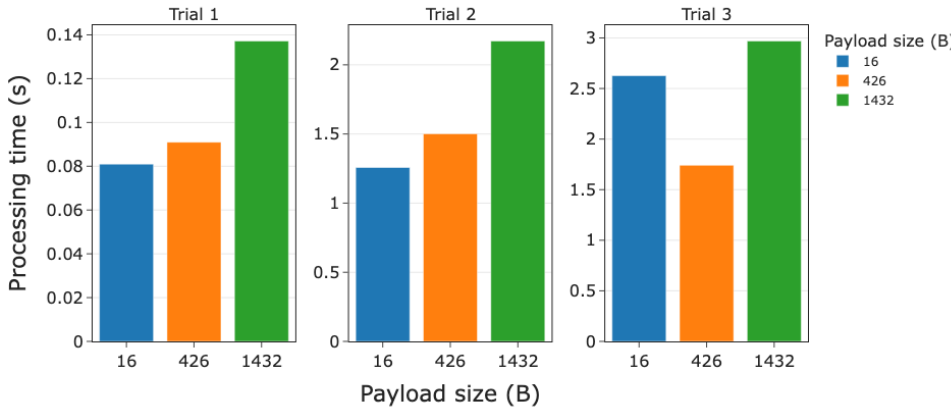


Figure 6.7: Performance of the Packet Matcher module for varying payload sizes varying between 16B and 1432B.

is the case for 426B payload size but not for the other two, indicating that the performance difference comes from differences in the captured pcaps. Due to time limitations, we did not perform any further investigation into the cause of this discrepancy. However, the data does not refute our claim that the payload size negatively impacts the processing time, but the results for the third trial make it more inconclusive. Nevertheless, the noticed impact of the payload size in processing times indicates that the benchmarking tool can handle varying payload sizes without suffering a major performance penalty.

Packet Rate Sensitivity in L2 Bridge

We asked the following sensitivity question for the L2 bridge, "How do changes in packet rates affect the L2 bridge?".

There was no difference between the number of packets captured at the ingress and the egress of the L2 bridge for all three repetitions for payload sizes 16B and 426B. However, with a Payload size of 1432B, there were some instances when the PPS exceeded 10,000 where there was a discrepancy between the number of captured packets. There were cases of more and fewer packets being captured at the egress than at the ingress port. The deviations were in the range -0.131% and 0.072%. For packet rates higher than 10,000PPS, there was no apparent relationship between the packet rate and deviations.

These deviations are either caused by the L2 bridge or `tshark` not being able to handle these higher throughputs. The instances where the egress has fewer captured packets can be explained by packet loss in the L2 bridge. However, the instances where the egress captures more packets than the ingress are more challenging to explain. It could be caused by the bridge sending duplicates or fragmenting packets to fit the MTU, but upon inspection, there are no duplicates or fragmented packets in the pcaps. Furthermore, it could be caused by the high load the machine is under while simultaneously running two instances of `tshark`, as well as the L2 bridge for throughputs above 117,76 $Mbps$. The current setup makes it difficult to discern the cause behind these deviations.

However, we would argue that these results indicate that the L2 bridge can support our use cases. It is stable up to throughputs of approximately 117,76$Mbps$, which exceeds all requirements for data rates of common Industry 4.0 use cases presented in Table 2.1. The sensitivity analysis shows that the L2 bridge may introduce inaccuracies in the traffic offered to the NUT. However, for our purposes this only happens for very high data rates, and the differences are very small, thus unlikely to impact our trials.

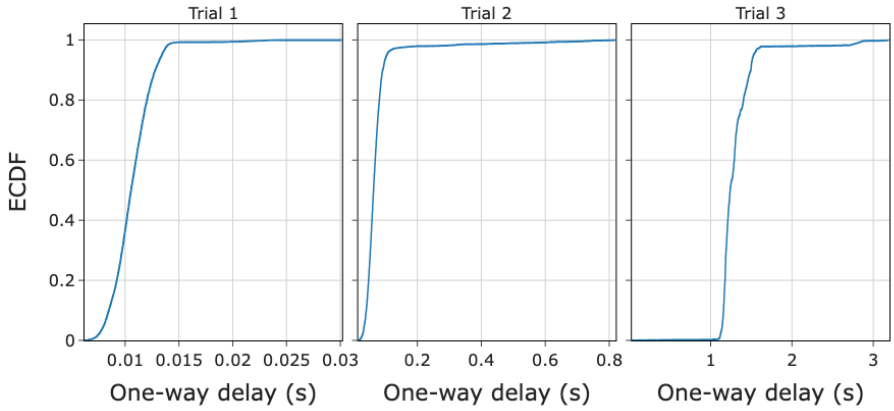


Figure 6.8: OWD for all three trials in the Open-source network.

6.2 Case study NTNU Open-Source Lab Results

This section presents the results produced from the case study trials on the Open-source network. First, the results for the OWD are presented, followed by the IPDV, packet loss, and throughput. Finally, the performance in general is discussed.

6.2.1 OWD

Figure 6.8 shows an ECDF of the OWD for all three trials. The ECDF for each trial is based on the average of five repetitions. In the first trial, nearly all packets have a OWD between 5 and 15ms. For the second trial, the majority of the packets have a OWD in the range 25 to 100ms. Finally, in the third trial, most packets have a OWD in the range 1.1 to 1.5s. All the ECDFs increases close to linearly, indicating an even distribution of OWDs in their respective ranges.

Based on this, we observe that the OWD appears to increase with the packet rate used. Furthermore, as the packet rate reaches 30,000PPS in the third trial, the network performance drastically decreases. This indicates that it cannot smoothly handle a traffic load of this magnitude.

Figure 6.9 shows how the OWD initially evolves for the second trial. As the graph shows, the OWD spikes at nearly 1s for several of the repetitions. The first trial with only 5 PPS did have this spike. This suggests that the initial resource allocation for the 5G gateway is too low, resulting in increased initial OWD.

Moreover, the OWD for all repetitions in the third trial is shown in Figure 6.10. This graph displays signs of queues building up somewhere in the network, eventually

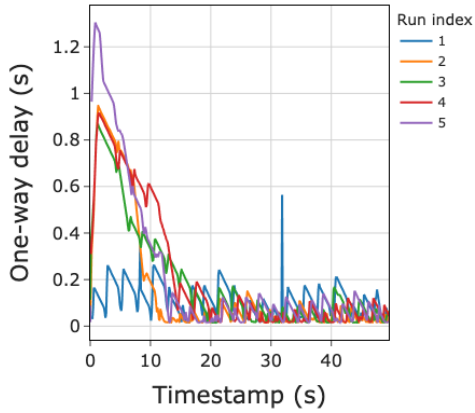


Figure 6.9: OWD for all repetitions of the second trial in the Open-source network with 1,000PPS, zoomed in on the first 50 seconds.

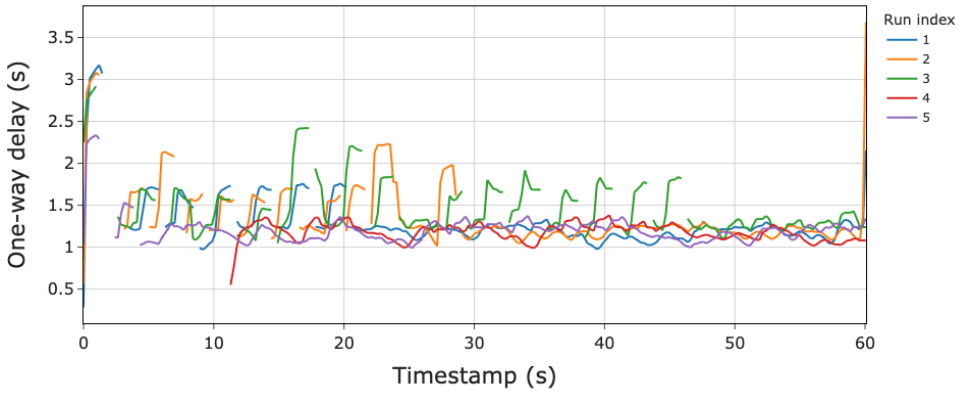


Figure 6.10: OWD for all repetitions of the third trial in the Open-source network with 30,000PPS.

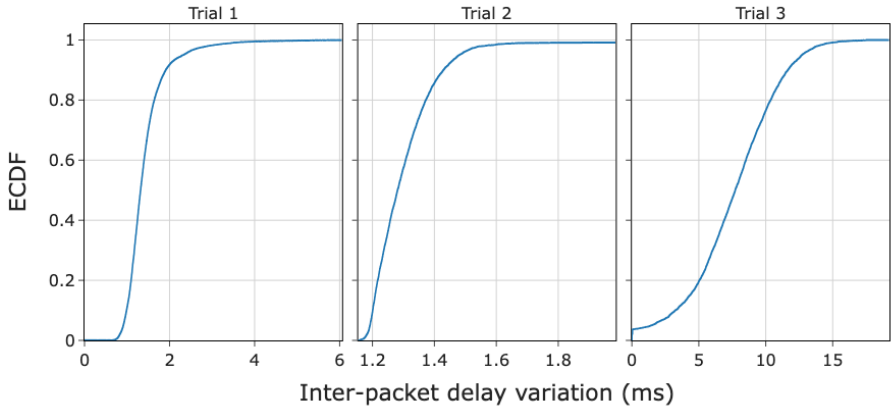


Figure 6.11: IPDV for all trials in the Open-source netowrk.

leading to packet loss, as indicated by the *s*-shapes. This is more prevalent for some of the repetitions and seems to reduce in frequency and magnitude over time. The pattern supports our claim that the network performance degrades significantly when subjected to 30,000PPS.

6.2.2 IPDV

Figure 6.11 shows an ECDF plot of the IPDV for all three trials, each based on the average of five repetitions. For the first trial, most of the packets have an IPDV between 1 and 2.5ms. In the second trial, the majority of packets have an IPDV in the range 1.1 and 1.5ms. Finally, in the third trial, most packets have an IPDV range of 1 to 13ms. When comparing the trials, we observe that the first and the second trials have roughly the same IPDV for their lowest 90%. However, the IPDV of the first trial is notably higher in the final 10%. Furthermore, both the mean and the variance of the third trial are significantly higher than the other trials. This indicates an inability of the network to provide stable performance.

6.2.3 Packet Loss

Figure 6.12 shows a barplot of the packet loss for all trials based on the average of all five repetitions. In the first trial, the average packet loss was just above 0. For the second trial the average packet loss increased to 0.3%. In the third trial, the packet loss increased all the way to 23.6%.

To ensure that the high packet loss is not due to an issue with the sliding window, we increased the size of the sliding windows to 100,000 packets each. We still saw

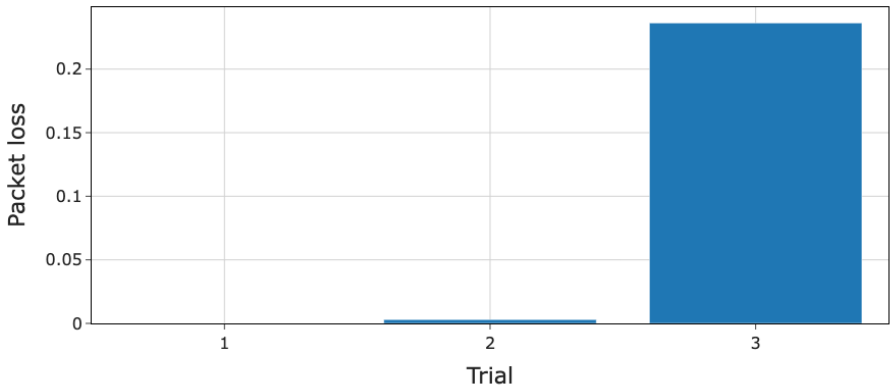


Figure 6.12: Packet loss rate for all trials for the Open-source network.

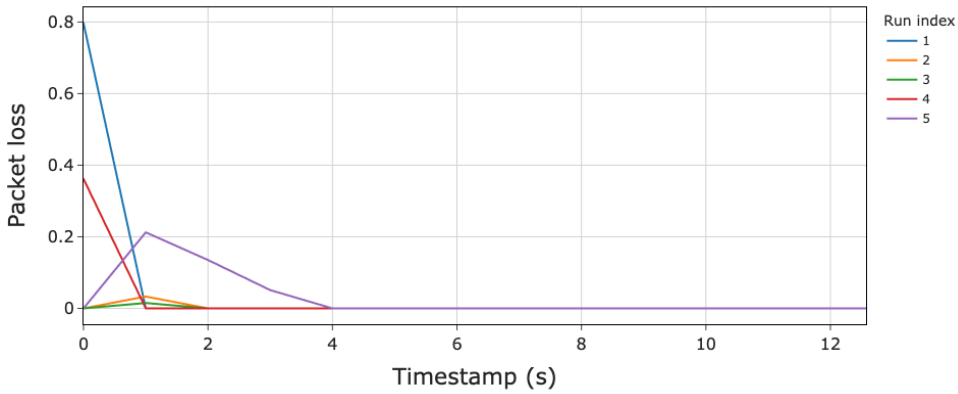


Figure 6.13: Packet loss for all repetitions of the second trial in the Open-source network with 1,000PPS.

the same level of packet loss even with a sliding window of this size, which makes it likely that the packet loss, in fact, was due to the network performance.

Based on this, it appears that the network is not able to handle packet rates of 1,000PPS and above without introducing packet loss. However, as shown in Figure 6.13 showcasing the packet loss over time for all repetitions of the second trial, this packet loss occurs only at the start. This can be seen in accordance with the initial spike in OWD presented above. Thus, it appears the network can handle these packet rates without introducing packet loss.

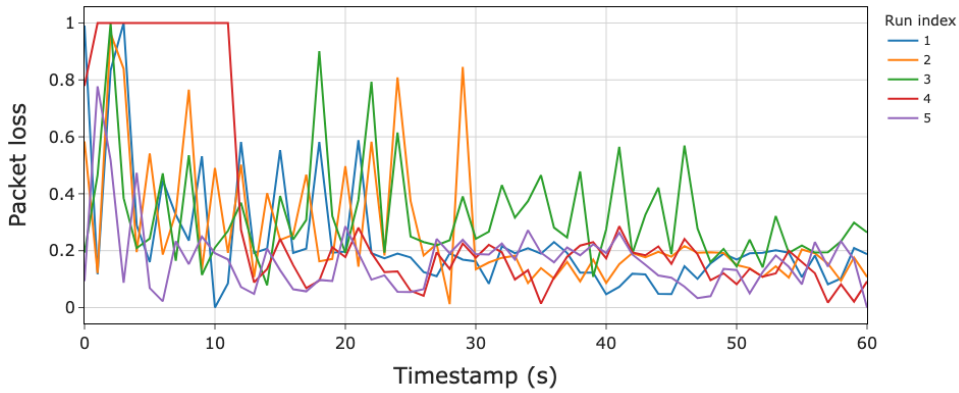


Figure 6.14: Packet loss for all repetitions of the third trial in the Open-source network.

Finally, the high packet loss in the third trial shows how the network cannot sustain a packet rate of 30,000PPS. Figure 6.14 shows how this evolves over time. The spikes in the loss coincide with the *s*-shapes in the OWD in Figure 6.10. This supports our earlier claim regarding the queues filling causing increased OWD and packet loss.

6.2.4 Average Instantaneous Throughout

In Figure 6.15, we can observe the average instantaneous throughput for all the trials based on the average of their respective repetitions. For the first trial, we expected it to be stable at $2.32Kbps$, which it was. In the second trial, the throughput based on the offered traffic was expected to be $9.93Mbps$ but was a bit lower, approximately $9.9Mbps$. The deviation from the expected value is likely caused by the packet loss of 0.3%. In the third trial, the throughput based on the offered traffic was expected to be $13.92Mbps$, but was approximately $10.7Mbps$. This value is roughly as expected based on the packet loss observed for this trial.

6.2.5 Performance

All the trials showed an initial packet loss in the first second. An analysis of the capture files revealed that the packets were never captured at the second measurement interface. If the effect had been produced by the benchmarking L2 bridge, the traffic would not have been captured at the first interface. Therefore, the problem is likely either from the network or gateway. Moreover, all trials have a similar total duration for which the packet loss happens, close to $400ms$. An analysis of the initial

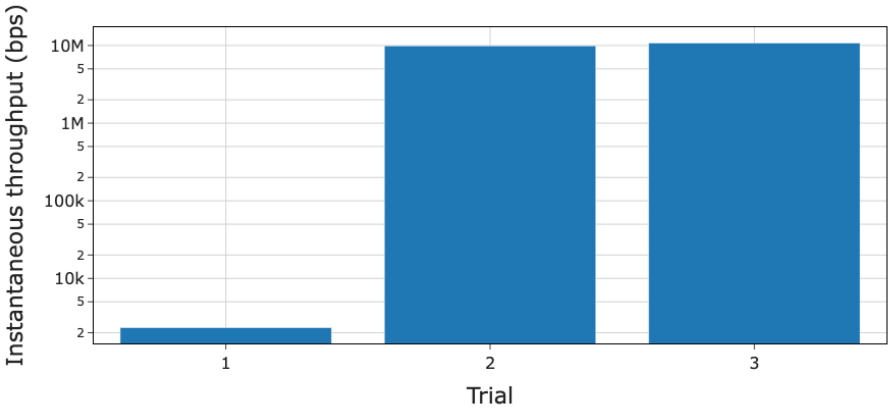


Figure 6.15: Average instantaneous throughput for all trials for the Open-source network.

round-trip time using `ping` when the gateway had been idle for a while revealed that the first round-trip time was close to $400ms$. Therefore, we assume that what we see is the effect of transitioning from an RRC Idle state to the Teltonika gateway receiving a data bearer. Consequently, we believe the data bearer establishment time roughly equals $400ms$ in the network. This process was described in Section 2.3 and should not take more than $10ms$ [27]. If our assumption is correct, this packet loss can be attributed to how the Teltonika gateway handles the queuing of packets while waiting for a bearer, for instance. On the other hand, the evidence may also suggest that the NUT has not allocated sufficient resources, and responds by dropping packets initially. However, because this also happens for the first trial with 5PPS, this is not likely.

In both the second and the third trials the performance of the NUT is initially unstable. This can be seen from the initial spike in OWD and packet loss in the second trial, and the repetitive symptoms of congestion and packet loss in the third trial. However, these symptoms seem to reduce in magnitude with time. For the second trial, this happens rather quickly. However, for the third trial, this does not seem to happen until after roughly $45s$. Based on the data, this might be due to inadequate initial resource allocation to the bearer. This indicates that the NUT is in a transitional phase when starting the trials, before converging on a more stable performance. The throughput for the second and third trial are roughly in the same range. However, the packet rate is substantially higher for the third trial, which seems to affect the performance the most.

The findings presented in this section suggest that the performance of the Open-

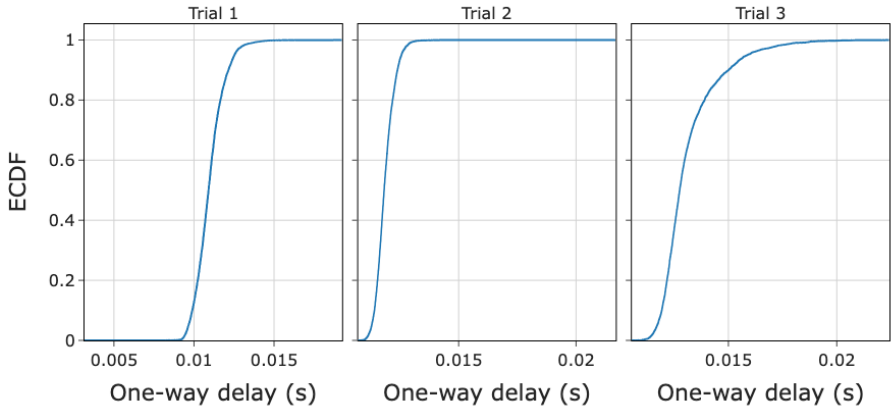


Figure 6.16: OWD for all three trials in the B5G Lab.

Source network degrades when the packet rate increases beyond 1,000PPS. This can be seen in all the presented KPIs.

6.3 Case study NTNU B5G Lab Results

This section presents the results produced from the case study trials on the network in the B5G Lab. First, the results for the OWD are presented, followed by the IPDV, packet loss, and throughput. Finally, the performance of the network in the B5G Lab in general is discussed.

6.3.1 OWD

Figure 6.16 shows an ECDF of the OWD for all three trials based on the average of five repetitions. In the first trial, most of the packets have a OWD in the range of 9.5 to 12.5ms. For the second trial, the majority have a OWD between 11 and 12.5ms. Finally, for the third trial, most are in the 11.5 to 16ms-range.

Figure 6.17 shows the ECDFs of the individual repetitions of the first trial. The first repetition has a OWD that is consistently 2ms higher than the other. This network was also subjected to experiments performed by others when we performed the case study. Because of this, the total traffic load on the network might have been higher than expected by our single trial, which could have impacted the performance. However, we do not have the necessary data to conclude this.

Based on the results in Figure 6.16, we can see that the OWD of the B5G network is not heavily impacted by increasing the packet rate or the throughput to the values

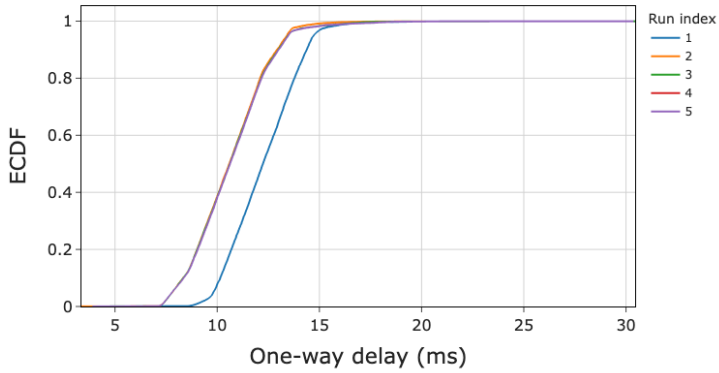


Figure 6.17: OWD for each repetition of the first trial in the B5G Lab network with 5PPS.

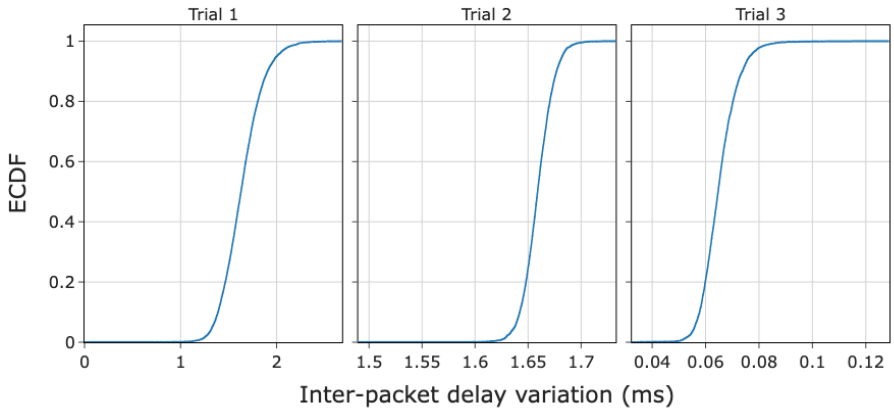


Figure 6.18: IPDV for all three trials in the B5G Lab.

tested here.

6.3.2 IPDV

Figure 6.18 shows an ECDF plot of the IPDV for all three trials, each based on the average of five repetitions. The majority of the packets in trial one have an IPDV in the range 1.2 to 2.1ms. For the second trial, most packets have an IPDV between 1.62 and 1.7ms. Finally, in the third trial, most packets have an IPDV in the range 0.05 and 0.08ms.

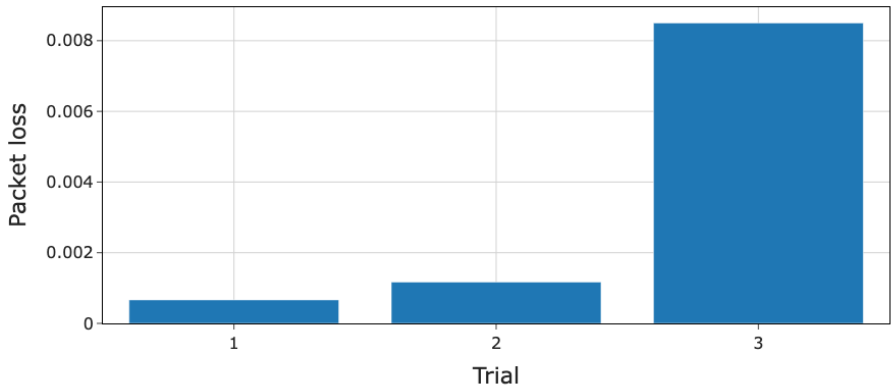


Figure 6.19: Packet loss rate for all trials in the B5G Lab.

The ECDFs of the first and second trials are relatively similar. However, the variance seems to decrease slightly in the second trial.

6.3.3 Packet Loss

Figure 6.19 shows a barplot of the packet loss for all trials based on the average of all five repetitions. In the first trial, the average packet loss was approximately zero. For the second trial, the average packet loss increased to 0.1%. While in the third trial, it increased to 0.85%. The packet loss occurs exclusively within the first seconds for the first and second trials. The same initial packet loss can be observed in the third trial. However, it also contains some minor spikes in packet loss with no apparent pattern, which can be seen in Figure 6.20.

We ensured that the observed packet loss was not due the sliding window by using the same method described for the Open-source network. With sliding windows with a size of 100,000 packets, we observed the same results. Therefore, it is likely that the observed packet loss was introduced by the network, not by the sliding window.

Based on this, it appears that the B5G network experiences some degradation in performance in terms of increased packet loss as the packet rate increases. However, aside from the first second, there was practically no packet loss in the first and second trials and very little in the third trial. Therefore, it appears that the network can generally support these high packet rates without inducing substantial packet loss.

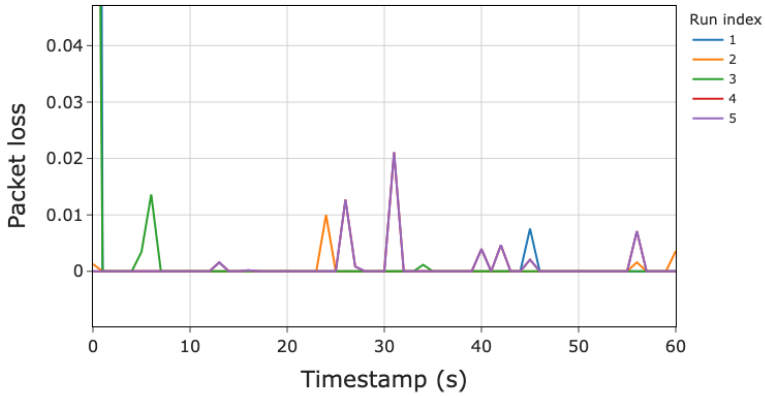


Figure 6.20: Packet loss for all repetitions of the third trial in the B5G Lab.

6.3.4 Average Instantaneous Throughput

Figure 6.21 shows the average instantaneous throughput for all the trials based on the average of their respective repetitions. The throughput of the first trial was expected to be stable at $2.32Kbps$, which it was. For the second trial, the throughput was stably a bit lower than expected based on the offered traffic. It was expected to be $9.93Mbps$ but was a bit lower, approximately $9.9Mbps$. We were not able to find the source of this deviation. In the third trial, the throughput based on the offered traffic was expected to be $13.92Mbps$, but was approximately $13.57Mbps$. Further inspection of the capture files revealed that the L2 bridge dropped, on average, approximately 1,500 packets for each repetition of the third trial. Thus resulting in the offered load being a tiny bit lower than the expected load and impacting the throughput. This is surprising as it was not the case during the validation with $16B$ payloads for 30,000PPS, or in the third trial at the Open-source network. However, the impact on the trials was relatively small, so we do not consider this further. Furthermore, the packet loss also impacts the throughput with some minor valleys for the third trial.

6.3.5 Performance

Almost all repetitions for all trials exhibit packet loss during the first $400ms$ across all repetitions. This resembles what we saw for the Open-source network, further supporting the hypothesis that this is caused by the setup of a data bearer for the 5G gateway.

Figure 6.16 shows that the third trial in the B5G Lab has a noticeably higher

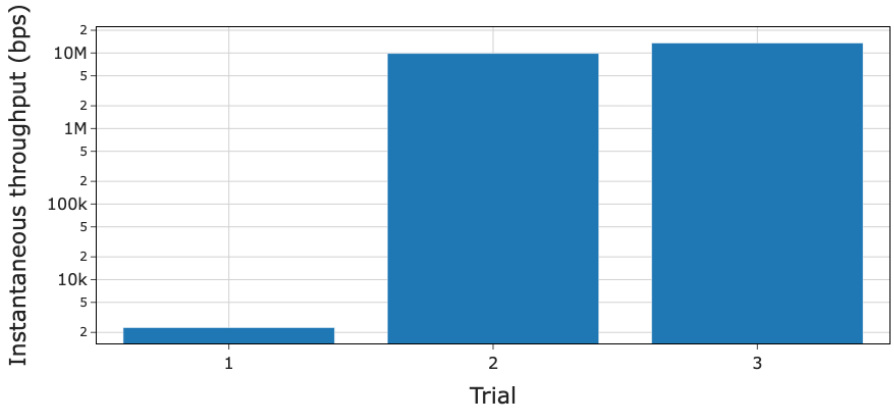


Figure 6.21: Average instantaneous throughput for all trials in the B5G Lab.

variance in the OWD. However, the IPDV in Figure 6.18 does not reflect this. Evidently, because of the high packet rate, the IPDV remains low even when the variance in the OWD is high because the successive differences in OWD are low. However, the differences in OWD on a larger scale are significant. The way we calculate IPDV does not consider the statistical mean; rather, it shows the differences based on a rolling average of the last 16 OWDs and thus indicates the instantaneous variability of the OWD. Therefore, the ECDFs of the OWD and the IPDV are not contradictory.

The findings presented in this section suggest that the performance of the B5G network can handle all the trials without experiencing substantial performance degradation.

6.4 Comparison of Case Study Results

This section compares the results for the Open-source network and B5G Lab discussed in the previous sections. Its structure is the same as the aforementioned sections.

6.4.1 OWD

Figure 6.22 shows the OWD for the average of all trial repetitions for both networks. Due to the large differences in the OWD, we use a log scale on the x-axis. For the first trial, the differences between the networks in the OWD are not that large. The B5G network has a lower variance, while the Open-source network has a slightly lower OWD for 60% of the packets. However, for the second and third trials, the Open-

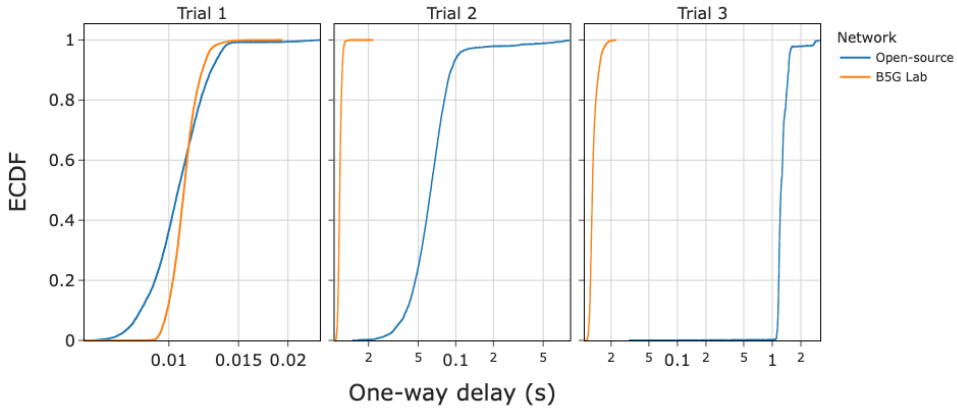


Figure 6.22: Comparison of OWD in both networks for each trial.

source network had substantially higher OWD for almost all packets. Furthermore, the variation in the OWD is also higher in the Open-source network.

Based on this, it appears that the difference in performance is not that large for lower packet rates. In fact, for the first trial, the Open-source network provided a lower OWD for the majority of its packets than the B5G network did. However, the B5G network can provide a far lower OWD than the Open-source network for packet rates exceeding 1,000PPS. Furthermore, the OWD has a significantly lower variance in the B5G network.

6.4.2 IPDV

Figure 6.23 shows the IPDV for the average of all the trial repetitions for both networks. It shows that the IPDV for the first and second trials is relatively close for both networks, with the Open-source generally having a lower IPDV. However, for the third trial, the Open-source network has far greater IPDV than the B5G network. This is shown clearly in 6.24, which shows the ECDFs of the IPDV of both networks using a log-scale on the x-axis.

These figures showcase how the Open-source network generally provides a lower IPDV for low data rates. However, for higher packet rates, as can be observed in the third trial, the Open-source network has a significantly higher IPDV than the B5G network. This indicates that the Open-source network does not provide a stable performance, while the B5G network does for 30,000PPS.

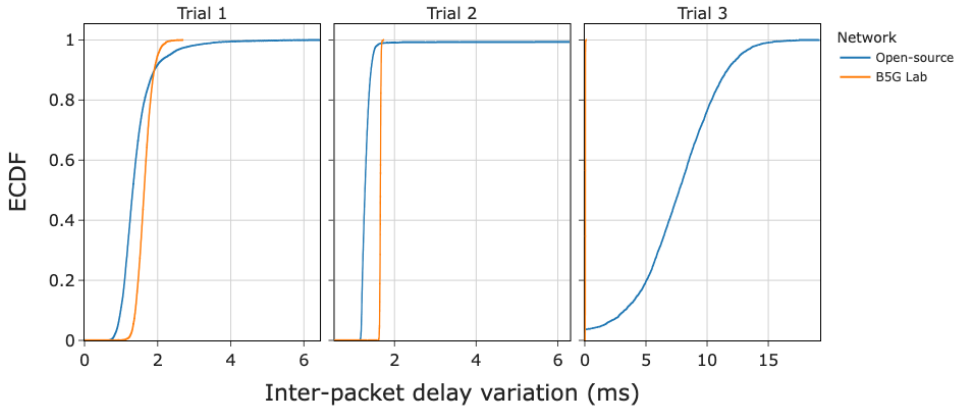


Figure 6.23: Comparison of IPDV in both networks for each trial.

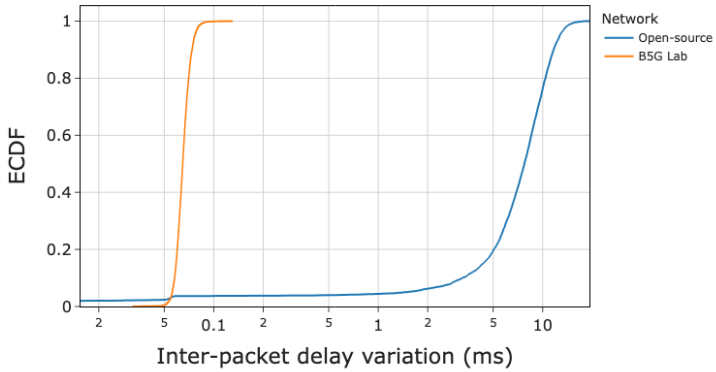


Figure 6.24: IPDV of both networks for the third trial.

6.4.3 Packet Loss

Figure 6.25 displays the average packet loss from all repetitions for each trial in both networks. It shows that the differences between the networks in terms of packet loss are not great for packet rates lower than 1,000PPS. There is a minor difference in the packet loss for the second trial, with the B5G network having a slightly lower packet loss. However, it is clear that for higher packet rates, as shown in the third trial, there is a substantial difference between the networks. The B5G is barely affected by the packet rate, while the Open-source network has large amounts of packet loss each second. Observing Figures 6.20 and 6.14 also show the differences in the profile of the packet loss. In the B5G Lab network, the packet loss is consistently 0% with

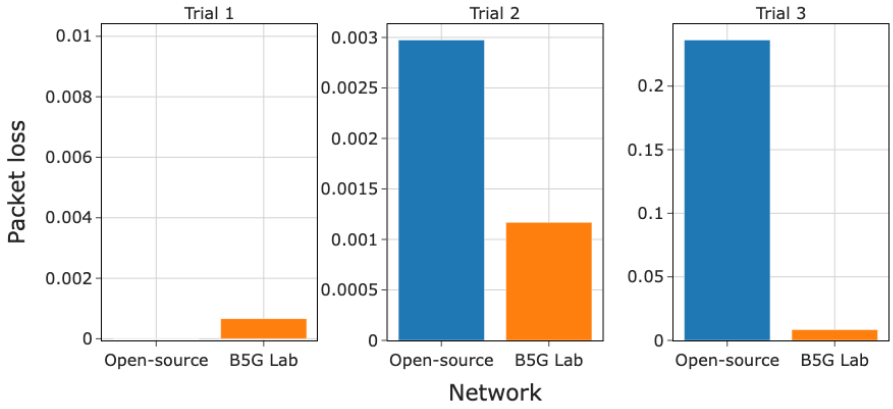


Figure 6.25: Comparison of packet loss rate in both networks for each trial.

minor local peaks, whereas the packet loss in the Open-source network is consistently high and varies greatly. Moreover, the average packet loss across the repetitions decreases after roughly 30 seconds as it stabilizes.

6.4.4 Average Instantaneous Throughout

Figure 6.26 shows that both networks have the expected average instantaneous throughput for the first trial. In the second trial both networks hovered around the same throughput as well. However, in the third trial, the B5G network was stable at around 2.87$Mbps$ higher than the Open-source network. This is most likely due to the difference in packet loss between the networks for this trial.

6.4.5 Performance

By comparing the KPIs between the networks, it is clear that the difference in performance between them is not that large when the traffic load is low. In fact, for the very low packet rate used in the first trial, the Open-source network has a slightly OWD and IPDV for most of the packets. However, for higher loads, the B5G has a significantly better and more stable performance across all KPIs.

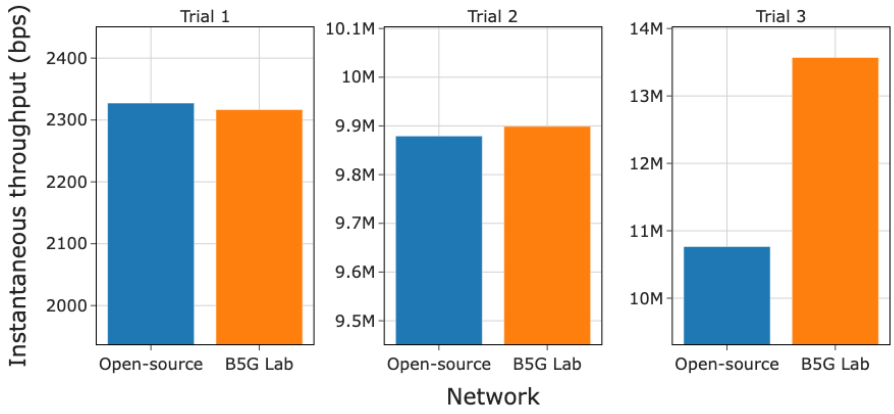


Figure 6.26: Comparison of average instantaneous throughput in both networks for each trial.

Chapter 7
Discussion

This chapter discusses the results from Chapter 6 and relates this to the requirements of the tool and the background from Chapter 2. It produces insights from the results of the validation and case study, which is used in the next iterations of the feedback loop in Figure 1.1. The results of the validation are discussed in Section 7.1. After this, the results from the case study are discussed in Section 7.2. Lastly, Section 7.3 wraps up the chapter with a discussion relating to the research questions.

7.1 Discussion on the Validation

From the validation, we gained insights into weaknesses with the technical realization of the benchmarking framework, such as the implementation of the sliding window and the physical architecture of the benchmarking framework. Section 7.1.1 highlights these and justifies whether they were handled before the next iteration of the feedback loop. Section 7.1.2 discusses whether the functional and non-functional requirements of the modules are satisfied.

7.1.1 Areas for Improvement

As highlighted in the validation of the Packet Matching submodule in Section 6.1.1, and the packet loss and network delay sensitivity questions in Section 6.1.2, the sliding window has several weaknesses. Because the sliding window inserts packets from both the ingress and egress of the NUT into the same queue, many unnecessary comparisons are performed. Moreover, the queue can become congested with packets from a single interface if the delay through the NUT is high. As highlighted, this can lead to a fuller queue and, in the worst case, ejection of packets that, in reality, have a match, which leads to artificial packet loss. Increasing the size of the sliding window would eliminate the issue of introducing artificial packet loss but would penalize the performance. Therefore, we decided to re-implement the sliding window before performing the case study. The improved implementation is described in Section 4.2.2.

Another weakness discovered during validation is within the L2 bridge. As highlighted in Section 6.1.2, it may be unstable for higher throughputs. This does not appear to be an issue for the trial parameters we have chosen for the case study, as previously mentioned. Connecting the Traffic Generation machine directly to the gateway would alleviate this problem by bypassing the measurement machine. A port mirror connected to the Measurement machine could then have been installed on the Traffic Generation machine to be able to capture this traffic as illustrated in Figure 4.3. However, because this was not considered utterly important and because we did not have the resources necessary to accomplish this, we did not implement it.

Based on the insight gathered during the validation, we identified potential for improvements in both the sliding window and the physical architecture of the benchmarking framework. Because the L2 bridge appeared stable for our use cases, we decided to only improve the Packet Matcher module.

7.1.2 Fulfillment of Requirements

After the validation was performed, we gained a foundation to evaluate if the functional and non-functional requirements of the modules were fulfilled. The Orchestrator and the Visualization modules are not described here as they rely on existing solutions that already fulfill their requirements.

The Traffic Generator module has three functional requirements; **TGFR1** (generating traffic adhering to real-world scenarios), **TGFR2** (generating traffic based on parameters for PPS, duration, hosts, and packet size), and **TGFR3** (customizing packet fields). The satisfaction of **TGFR1** and **TGFFR3** are described in Section 4.1. Based on the results for the effect questions for Traffic Generation Alternative 1 and 2, we argue that **TGFR2** is fulfilled as well. The non-functional requirement of the Traffic Generator is **TGNFR1** (sustaining stable packet transmission over a prolonged period) not met on an intra *ms* scale for Traffic Generation Alternative 1. However, for larger scales, it is arguably satisfied. Based on the results from the effect question for Traffic Generation Alternative 2, the deviations from the original capture file are very small. Thus, we argue that **TGNFR1** is met by Traffic Generation Alternative 2. Since Traffic Generation Alternative 3 is only a specification, the fulfillment depends on the implementation.

The Packet Matcher module has five functional requirements; **PMFR1** (capturing packets on at least two interfaces), **PMFR2** (filter packets), **PMFR3** (match packets), PMFR4 (store matched packets), and PMFR5 (provide updates at fixed intervals during a trial). **PMFR1** and **PMFR2** are validated together with the effect question for the Packet Capturer submodule. The results were as expected, and we consider these fulfilled. **PMFR3** and **PMFR4** are tested with the effect questions for the Packet Matching. These results were also as expected and the requirements

are considered satisfied. Finally, **PMFR5** is not explicitly validated but has been informally validated through usage of the tool. The non-functional requirement for the Packet Matcher, **PMNFR1** (process 1 million packets in less than 60 seconds), was found to be satisfied when the packet loss was less than or equal to 2%, but not at 15%, as discussed in Section 6.1.2. Therefore, we consider **PMNFR1** to be met with the first implementation of the sliding window under some conditions. The second implementation of the sliding window introduced performance gain, but we have not validated its processing time.

The Packet Analyzer module has three functional requirements; **PAFR1** (calculate pre-defined per-packet KPIs), **PAFR2** (calculate pre-defined aggregate KPIs), and **PAFR3** (write the results to persistent storage). **PAFR1** and **PAFR2** were both validated in the effect question for the Packet Analyzer, and the results found that both requirements were met. Furthermore, **PAFR3** was informally validated as the results for the effect questions were fetched from the persistent storage. The non-functional requirement, **PANFR1** (process 1 million packets in less than 60 seconds), is validated.

Since we have now concluded all the module-level requirements, we can conclude the tool-level requirements as well. **TFR1** (generating traffic that emulates real-world scenarios) is fulfilled through the Traffic Generator module. **TFR2** (capturing network traffic) is fulfilled through the Packet Matcher module. **TFR3** (calculate pre-defined KPIs based on network traffic) is fulfilled through the Packet Matcher and the Packet Analyzer. **TFR4** (visualize the results of a trial) is satisfied through the Visualization module. **TFR5** (providing the possibility for custom analysis) is fulfilled by making output files of the Packet Matcher and Packet Analyzer accessible. **TFR6** (executing trials automatically) is satisfied by the Orchestrator module. Finally, the Packet Matcher and the Visualization modules fulfill **TFR7** (displaying the status of the executing trial).

7.2 Case Study Comparison

This section discusses the case study comparison, starting with a discussion about the configuration of the trials, followed by a discussion about what the results of the case study mean in the context of the Industry 4.0 use cases it tests. Finally, the insights we gained from the case study are discussed.

7.2.1 Setup of Trials

Before discussing the comparison of the networks, we will comment on the methodology used.

We initially intended to perform more trials, but the setup at the Open-source network was unstable. The gNB tended to crash, and the connection between the Software Defined Radio (SDR) and the 5G gateway often broke. Because of this, several trials had to be re-run. To reduce the impact of this burden, we limited the case study to three trials. Furthermore, we reduced the duration of the longest trial from 2 hours to 30 minutes.

Based on the results of our case studies, we should have increased the duration of the third trial to at least 3 minutes. This increase could potentially have made it clearer if some of the observations were caused by transient factors. On the other hand, storage capacity was a limitation, and we wanted to balance the capture size files with the length of our trials.

To reduce the impact of performance variations, we performed five repetitions of each trial in each network. We then used averages of the five repetitions. This was beneficial as some of the repetitions produced quite different results from the others.

Finally, the network in the B5G Lab was subject to multiple concurrent experiments from other parties. Consequently, we could not verify the present network load caused by others, potentially increasing the stochasticity of the performance. Thus, the results produced on this network potentially represent a lower performance bound.

7.2.2 Comparison of Case Study Results

The results of the case study in the previous chapter highlight the different performance factors of the tested networks. However, it is unclear to what extent the networks satisfy the Industry 4.0 use cases that the trials represent.

Table 7.1 summarizes the share of packets with a OWD lower than or equal to the given threshold in trial one for each network. Lost packets are assumed to have an infinite OWD. We consider reliability as the percentage of packets having a OWD within a given threshold. This trial represents the condition monitoring for safety applications from Table 2.1 from [33]. A reliability of 99.9% and an OWD between $5ms$ and $10ms$ is required for this use case. As the table shows, neither of the networks satisfy these requirements. 99.9% of the packets in the Open-source network has a OWD less than or equal to $35ms$. On the other hand, the network in the B5G Lab delivers 99.9% of packets within $15ms$. This shows that the B5G lab network can almost meet the requirements of the use case, while the Open-source network is a way off.

The second trial represents the motion control use case from Table 2.1 from [33]. This use case requires a reliability of 99.9999% and OWD between $0.5ms$ and $2ms$.

Network	Threshold	50%	80%	90%	95%	99%	99.9%
OS	5ms						
	10ms						
	15ms						
	25ms						
	35ms						
B5G	5ms						
	10ms						
	15ms						

Table 7.1: Threshold compliance of OWD for the Open-source and B5G Lab networks in the first trial.

Table 7.2 shows the extent to which the networks comply with the requirements of the use case. It is evident that neither of the networks provides the required performance. The Open-source network can only provide 99.9999% reliability for a threshold of $850ms$, while the network in the B5G Lab provides the required reliability within a threshold of $35ms$. However, Table 7.2 does not account for the duration of the executed trial, which was 10 minutes. Realistically, this is too short to make any valid inferences regarding the use cases. Nevertheless, they still serve as an illustration.

Network	Threshold	50%	90%	99%	99.9%	99.999%	99.9999%
OS	5ms						
	35ms						
	75ms						
	100ms						
	850ms						
B5G	5ms						
	15ms						
	35ms						

Table 7.2: Threshold compliance of OWD for the Open-source and B5G Lab networks in the second trial.

The third trial represents the AR/VR use case from Table 2.1 from [33], which requires 99.9% and a OWD less than $10ms$. As shown in Table 7.3, neither network satisfies these requirements. Regardless of the requirements for OWD, the Open-source network cannot meet the required reliability due to the substantial packet loss. The B5G Lab network can also not meet the requirement of 99.9% due to packet loss. However, most of the packet loss occurs in the first $400ms$ of the trial. Therefore, after the first $400ms$ the network can seemingly meet the required reliability with a

threshold of 25ms.

Network	Threshold	50%	80%	90%	95%	99%	99.9%
OS	5ms						
	1,000ms						
	1,250ms						
	1,500ms						
	1,750ms						
	3,000ms						
	3,150ms						
B5G	5ms						
	10ms						
	15ms						
	20ms						
	25ms						

Table 7.3: Threshold compliance of OWD for the Open-source and B5G Lab networks in the third trial.

The tables show that neither of the networks satisfies the requirements of the use cases the trials are based on. However, the network in the B5G Lab is consistently closer to compliance than the open-source network. It must be considered that the networks used in the case study were not necessarily customized to satisfy these use cases. They were simply used as placeholders in order to compare a set of networks. Moreover, the same can be said for comparing the individual networks. The network in the B5G Lab is based on equipment from Nokia, while the Open-source network is based on open-source software running on commercial off-the-shelf hardware and an SDR. Nevertheless, the purpose of the case study was to utilize the benchmarking framework for two real-world private 5G networks. Therefore, no effort was made to ensure that networks were equivalent.

7.2.3 Insights gained from case study

After performing the case study, we discovered two phenomena worth addressing. These can potentially impact the reproducibility of the benchmarks made by the tool.

As discussed in Section 6.2 and 6.3, nearly all repetitions of trials in both networks experienced packet loss during the first 400ms, which we hypothesize is due to the 5G gateway being in an RRC Idle state. Therefore, the dropped packets might be due to how the Teltonika gateway handles queueing of packets while awaiting a data bearer. Regardless of whether this hypothesis is true, this highlighted that

the benchmarking tool does not ensure a consistent starting RRC state for the 5G gateway. This shortcoming can impact the reproducibility of the benchmarks. For instance, if the 5G gateway had consistently begun in an RRC Active state in one network while in an Idle state in another network, this effect would suggest differences in the networks. Therefore, the RRC state of the 5G gateway should be handled explicitly by the benchmarking framework.

The initial peak in OWD shown in the Open-source network in Section 6.2 might, as discussed, be the result of low initial resource allocation. On the one hand, one might argue that because this effect is transient, a warm-up phase should be added to remove it. On the other hand, the initial performance might provide useful insights into a transitional phase of a network. This could be an interesting characteristic for the analysis of delay-sensitive applications. Therefore, the tool should not offer a warmup phase to hide this effect. To ensure that the benchmark also highlights the performance when the network is in a steady state, trials should have a sufficiently long duration to account for this.

The discussed phenomena highlight an insight into the start of the produced benchmarks. Ensuring a consistent RRC state for the beginning of trials would remove a shortcoming of the framework itself while adding a warm-up phase would hide network behavior. Therefore, we argue that the RRC state of the 5G gateway should be addressed, but the warm-up phase should not. This concludes the insights gained from the final step of the second iteration of the feedback loop.

7.3 Fulfillment of Research Questions

This section discusses the research questions presented in the introduction in light of the results and discussions of the thesis. The research questions are discussed in order.

7.3.1 Benchmarking and Reproducability

When considering RQ1 (”How can a system capable of performing reproducible benchmarking of private 5G networks be designed?”), we decided to split the question into two aspects. The first aspect is how benchmarking private 5G networks can be performed, and the second is how this can be done reproducibly.

Previously, we defined benchmarking as *an empirical experiment that generates data that enables comparing candidates with respect to a certain (set of) performance measures*, inspired by [8]. One needs a way to generate and capture network traffic to conduct an empirical experiment that generates data for benchmarking. Then, analyzing this data enables calculating a set of performance measures. Finally, a way

to communicate the performance measures is necessary to enable comparisons. In our framework, these functions are fulfilled by the Traffic Generator, Packet Matcher, Packet Analyzer, and Visualization module. The combination of these features provides the foundation to perform benchmarking of private 5G networks.

To support reproducibility in these benchmarks we introduced another module, the Orchestrator. By automating the execution of trials, this module ensures that the same steps are executed in the same order every time. Furthermore, it can also perform tasks ensuring that the internal state of the tool is consistent for all trials. For example, the Trex server used for Traffic Generation Alternative 1 is started and stopped at the beginning and end of each trial.

As discussed in Section 7.2.3, the framework does not actively ensure a consistent RRC state for the 5G gateway. This enables the potential for an inconsistent initial RRC state between trials, which, if this is the case, would reduce the reproducibility of the benchmarking framework.

Furthermore, the realization of the high-level tested architecture can impact the reproducibility. When the L2 bridge approach is used, the expected offered traffic load can differ from the actual one. If the L2 bridge drops packets, the trial can potentially be run with a lower load than intended. The offered traffic may thus not be representative of the configured trial. Utilizing port mirroring to the measurement machine instead of sending the traffic through the measurement machine reduces this risk.

Furthermore, the framework does not document or measure the environment during a repetition. Factors such as the background traffic and signal-to-noise ratio are not documented and can differ between repetitions. These factors can potentially have a non-negligible impact on the KPIs and should be documented as meta-information for a trial. This would enable discovering when two trials or repetitions are not comparable due to external factors.

We believe that our benchmarking framework outlines how reproducible benchmarking of private 5G networks can be performed. It has functionality for generation, capturing, and analyzing network traffic required for benchmarking. Furthermore, the Orchestrator module supports reproducibility. However, some factors negatively impact reproducibility, such as the potential for an inconsistent initial RRC state, usage of the L2 bridge, and not considering external factors.

7.3.2 The Grey-Box Approach

To answer RQ2 ("How can the requirements for integration with the NUT and the granularity of information provided by the benchmark be balanced?"), we decided to

utilize a grey-box approach.

The grey-box approach reduced the need for integration with the NUT, leading to an overall higher portability of the benchmarking framework. This effect was noticeable when switching the NUT during the case study.

However, since the KPIs are based on the performance between the two measurement points, we do not know the performance of individual network components. Thus, the framework can not produce the information to be able to pinpoint where potential performance bottlenecks occur. This was noticed in both the B5G Lab and the Open-source network. For instance, during the third trial on the Open-source network, where it was not possible to identify the reason(s) for the poor performance. The data did not separate the performance of the 5GC, gNB, or the SDR.

Utilizing the available capture points as in [20] would have led to a greater granularity in the produced data. In our case, this would only have incurred a minor altering of the testbed architecture and would leverage the level of network integration greater than our current architecture. This approach would have enabled the separation of performance of the RAN and the 5GC.

7.3.3 Support for Industry 4.0 Use Cases

When considering RQ3 ("To what extent can such a system support benchmarking of a defined set of common Industry 4.0 use cases?"), the framework is limited by three factors.

Firstly, the benchmarking framework neither generates nor calculates KPIs for downlink traffic. The high-level testbed architecture does not include hosts responding to the traffic. Thus, bidirectional traffic cannot be generated. Because no downlink traffic is generated, the tool does not have the necessary data to calculate downlink KPIs. Furthermore, due to downlink traffic never being part of the produced results, we did not write the logic to separate uplink and downlink traffic, thus necessitating a minor rewrite of the Packet Matcher module. Depending on the configuration of the private 5G network, such as the uplink/downlink ratio in the radio network, the performance of the uplink and downlink can significantly differ. This means it does not provide the necessary information in use cases that are concerned with the performance in both directions.

Secondly, the method for traffic generation from Alternative 1 and 2 limits the extent to which the system can support benchmarking of Industry 4.0 use cases. All the traffic is currently generated from a single device. Based on the requirements discussed in [33], most Industry 4.0 use cases require multiple devices. Having only one device generate all the traffic might not provide traffic patterns representative of

the actual use cases. For instance, having a single device only requires a single data bearer, which might impact the scheduling of devices.

The benchmarking tool matches UDP packets based on an identifier in the first four bytes of the payload. This causes issues when packets are large enough to require fragmentation into multiple datagrams. In that case, only the first packet will have the guaranteed unique identifier, while all the other packets will contain the padding bytes. This results in arbitrary matching of the fragments lacking an identifier. To mitigate the challenges caused by fragmentation, the matching could have been done on the IPv4 Identification header field. This approach provides multiple improvements, such as being agnostic of the transport protocol, and enables simpler matching of fragmented packets. Fragmented packets could be handled by matching on a combination of the Identification field, More Fragments flag, and the Fragmentation offset field in the IPv4 header. However, most of the requirements for payload size of the use cases in [33] are at most 250B. These use cases should not cause fragmentation and, therefore, not be an issue for the benchmarking framework. However, the tool cannot support use cases such as mobile robotics (video operations), where the requirements for payload sizes range from 15kB to 250 kB [33].

In conclusion, the benchmarking framework can support benchmarking of industry 4.0 use cases with some caveats. It does not consider the downlink performance. Furthermore, the traffic patterns generated can approximate realistic patterns depending on the use case and traffic generation alternative used. Specifically, the payload size cannot cause fragmentation, and the number of hosts generating traffic cannot be more than one. However, based on the case study, we argue that the benchmarking framework satisfies many aspects, such as calculating several of the relevant uplink KPIs and emulating aspects of the traffic patterns, such as throughput.

7.3.4 Comparing Network Performance

To answer RQ4 (”How can the framework be used to compare 5G implementations?”), we conducted a case study using the benchmarking framework on two different networks and compared the results.

Based on our case study, assessing the KPIs of the two networks enables a systematic comparison of the performance between the networks. Furthermore, since the trial execution is automated with parameters describing the trial, the framework can easily produce data for a multitude of scenarios. This enables the comparison of networks under varying conditions.

The choice of trial parameters, such as duration and packet rate, is important for the comparison. The networks can behave differently depending on the load and trial duration. For instance, the OWD in the Open-source network increased dramatically

as the packet rate reached 30,000PPS, while the network in the B5G Lab remained at the same level. However, for the first trial, the difference in performance between the Open-source and the B5G Lab networks was smaller. Thus, it is important to keep in mind that the results are only representative of the specific parameters used. If the framework is used to compare networks, the parameters should reflect the use cases to be tested.

To produce comparable results, the network integration for different networks must be as similar as possible. For our framework, this means that the measurement points must be similar across networks. If the traffic measurement is done after it has been processed by the core in one network and before it has been processed by the core in the other, the resulting KPIs will not be comparable.

The depth of the network comparison is limited to the available set of KPIs, and the granularity offered by the grey-box approach. For instance, the set of KPIs does not consider KPIs targeting the physical interface. Thus, the framework is currently not suited for comparing the radio networks of 5G implementations. Furthermore, as discussed for RQ2, the benchmarking framework does not provide information about the performance of individual components in the networks. Therefore, it can only be used to compare the performance of the networks as a whole. Adding more KPIs, such as round trip time, signal-to-noise ratio, bandwidth, and bit error rate, could provide a more thorough foundation for comparing networks. Moreover, separating the performance of the RAN and the 5GC, as discussed for RQ2, would enable a finer granularity when comparing networks.

The case study of the B5G Lab and Open-source networks shows an example of how the framework can be used to compare 5G implementations. However, some considerations should be taken when utilizing the framework. Firstly, the trial parameters must be representative of the desired use case(s). Furthermore, the comparison is limited to the available KPIs and to the granularity of the network as a whole. Nevertheless, useful insights can be drawn from the produced results as the benchmarking framework stands.

Chapter 8
Conclusion and Future Work

8.1 Conclusion

In this thesis, we have designed and implemented a framework for benchmarking private 5G networks. We performed validation on several of the modules of the framework. Based on the insights gained from the validation, we improved the Packet Matcher module to mitigate potential issues caused by the sliding window. Once the adjustments were made, we used the framework to benchmark the performance of two private 5G networks. Finally, we compared their performance based on the results produced by the benchmarking framework. As the thesis concludes, we are left with a benchmarking framework consisting of a validated software tool and a high-level testbed architecture, and a case study showing the performance of two real-world private 5G networks. Since the thesis was based on the research questions we will now revisit each question for a brief conclusion.

Firstly, for RQ1 ("How can the requirements for integration with the NUT and the granularity of information provided by the benchmark be balanced?"), we argue that our contribution captures how such a system can be designed. The results of our case study show how it could perform reproducible benchmarks of two different networks. Currently, only uplink performance is considered, but the design of the framework can be extended to include downlink performance. Furthermore, some factors can impact the reproducibility, which we did not have time to address. Examples include not accounting for background traffic and the initial RRC state of the 5G gateway. Aside from this, we believe the framework shows how a system capable of performing reproducible benchmarking of private 5G networks can be designed.

Secondly, for RQ2 ("How can integration with the network under test and level of detail in benchmark output be balanced?"), we went for a grey-box approach. This required little integration with the NUT, which made the framework portable and easy to set up in our experience. However, we argue that by requiring a bit more integration into the network, the framework could have extracted more

useful information. The authors of [20] illustrated this, enabling the separation of the performance of the RAN and the 5GC. Augmenting the KPIs for the overall performance with KPIs for both the RAN and the 5GC enables more fine-grained analysis. We believe this is a better balance of integration with the NUT and the granularity of the provided information.

For RQ3 ("To what extent can such a system support benchmarking a defined set of common Industry 4.0 use cases?"), we argue the framework can support several aspects of common Industry 4.0 use cases as defined in [33]. However, there are some limitations related to the traffic generation and lack of downlink KPIs.

Finally, for RQ4 ("How can the framework be used to compare 5G implementations?"), we argue that the framework can be used to compare the uplink performance of 5G networks. By assessing the calculated KPIs the performance of networks can be compared. Utilizing comparable measurement setups and carefully choosing representative trial parameters of the desired use case(s) ensures that the KPIs calculated are comparable between networks.

In conclusion, the thesis explored the design and implementation of a benchmarking framework for private 5G networks. This, combined with the validation and usage of the framework provided insights into all the research questions. The framework shows promise but requires further work to fulfill its potential. Due to time limitations, we were not able to address the shortcomings identified after the case study and during the evaluation of the research questions.

8.2 Future Work

As mentioned, there are several aspects of the benchmarking framework that can be improved or extended in future work. The main categories are extending the framework to consider downlink KPIs or separation of KPIs from the RAN and 5GC, making the tool produce results in real-time, and improving the traffic patterns.

- **Separate the performance of the RAN and core**: Enchancing the benchmarking framework by separating the performance of the RAN and the 5GC. This could give increased insight into the main components of the network while requiring only a bit more integration to the NUT.
- **Extend the framework to consider downlink KPIs**: Extending the functionality of the framework to also calculate KPIs for the downlink performance would enable the benchmarking to both uplink and downlink. This could give a more complete view of the performance of a network.
- **Make the tool able to run in real-time**: Currently, the tool does not calculate the KPIs until after a trial is completed. Therefore, it is not well suited for monitoring. Enabling the tool to function in real-time would increase

its applicability as a monitoring tool.

- **Improved Traffic Patterns**: To improve the extent to which the framework can be used to benchmark common Industry 4.0 use cases, it would be beneficial to provide easy access to traffic patterns based on common Industry 4.0 use cases. It is currently possible to replay a pcap using Traffic Generation Alternative 2, but having this in a simple and scripted manner, such as Traffic Generation Alternative 1, would make this more usable. Furthermore, extending the traffic generations to allow for stateful protocols would also open up more use cases for the framework.

By focusing on these areas of future work, the benchmarking framework can perform benchmarks with more detail regarding downlink performance and the separation of RAN and core performance. Furthermore, improving the traffic generation could enable the framework to be used for more use cases. Finally, making the tool produce results in real-time would open up a new avenue of possibility as a performance monitoring tool.

References

[1] S. Fuglesang and C. Lewin, «A Methodology for Benchmarking a Private 5G Network Exemplified with Industry 4.0 Use Cases», Department of Information Security, Communication NTNU – Norwegian University of Science, and Technology, Project report in TTM4502, Dec. 2023.

[2] ITU-R, «IMT Vision – Framework and Overall Objectives of the Future Development of IMT for 2020 and Beyond», ITU-R, Tech. Rep. Recommendation ITU-R M.2083-0 (09/2015), 2015.

[3] A. Aijaz, «Private 5G: The Future of Industrial Wireless», *IEEE Industrial Electronics Magazine*, vol. 14, no. 4, pp. 136–145, 2020.

[4] M. Wen, Q. Li, *et al.*, «Private 5G Networks: Concepts, Architectures, and Research Landscape», *IEEE Journal of Selected Topics in Signal Processing*, vol. 16, no. 1, pp. 7–25, 2021.

[5] United Nations, *Goal 8: Promote sustained, inclusive and sustainable economic growth, full and productive employment and decent work for all*, Accessed: 2024-05-28, 2015. [Online]. Available: https://sdgs.un.org/goals/goal8#targets_and_indicators.

[6] S. Eswaran and P. Honnavalli, «Private 5G Networks: a Survey on Enabling Technologies, Deployment Models, Use Cases and Research Directions», *Telecommunication Systems*, vol. 82, Nov. 2022.

[7] United Nations, *Goal 9: Build resilient infrastructure, promote inclusive and sustainable industrialization and foster innovation*, Accessed: 2024-05-28, 2015. [Online]. Available: https://sdgs.un.org/goals/goal9#targets_and_indicators.

[8] T. Hothorn, F. Leisch, *et al.*, «The Design and Analysis of Benchmark Experiments», *Journal of Computational and Graphical Statistics*, vol. 14, no. 3, pp. 675–699, 2005.

[9] S. O. Bradner and J. McQuaid, *Benchmarking Methodology for Network Interconnect Devices*, RFC 2544, Mar. 1999. [Online]. Available: https://www.rfc-editor.org/info/rfc2544.

[10] V. Bajpai, A. Brunstrom, *et al.*, «The Dagstuhl Beginners Guide to Reproducibility for Experimental Networking Research», *ACM SIGCOMM Computer Communication Review*, vol. 49, no. 1, pp. 24–30, 2019.

[11] K. Schwaber and J. Sutherland, *The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game*, This publication is offered for license under the Attribution Share-Alike license of Creative Commons, accessible at https://creativecommons.org/licenses/by-sa/4.0/legalcode., Nov. 2020. [Online]. Available: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf.

[12] R. J. Wieringa, *Design Science Methodology for Information Systems and Software Engineering*. Springer, 2014.

[13] G. Almes, S. Kalidindi, *et al.*, *A One-Way Delay Metric for IP Performance Metrics (IPPM)*, RFC 7679, Jan. 2016. [Online]. Available: https://www.rfc-editor.org/info/rfc7679.

[14] C. M. Demichelis and P. Chimento, *IP Packet Delay Variation Metric for IP Performance Metrics (IPPM)*, RFC 3393, Nov. 2002. [Online]. Available: https://www.rfc-editor.org/info/rfc3393.

[15] A. Lamberti, *How to Measure Jitter & Keep Your Network Jitterbug Free*. [Online]. Available: https://obkio.com/blog/how-to-measure-jitter/#how-do-you-measure-jitter (last visited: Jun. 5, 2024).

[16] H. Schulzrinne, S. L. Casner, *et al.*, *RTP: A Transport Protocol for Real-Time Applications*, RFC 3550, Jul. 2003. [Online]. Available: https://www.rfc-editor.org/info/rfc3550.

[17] M. Clouqueur and W. D. Grover, «Availability Analysis of Span-Restorable Mesh Networks», *IEEE journal on selected areas in communications*, vol. 20, no. 4, pp. 810–821, 2002.

[18] S. O. Bradner, *Benchmarking Terminology for Network Interconnection Devices*, RFC 1242, Jul. 1991. [Online]. Available: https://www.rfc-editor.org/info/rfc1242.

[19] J. Kurose and K. Ross, *Computer Networking: A Top-Down Approach, Global Edition*, English, 7th. Pearson, 2017.

[20] J. Rischke, P. Sossalla, *et al.*, «5G Campus Networks: A First Measurement Study», *IEEE Access*, vol. 9, pp. 121 786–121 803, 2021.

[21] D. Xu, A. Zhou, *et al.*, «Understanding Operational 5G: A First Measurement Study on its Coverage, Performance and Energy Consumption», in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 479–494.

[22] P. Emmerich, S. Gallenmüller, *et al.*, «MoonGen: A Scriptable High-Speed Packet Generator», in *Proceedings of the 2015 Internet Measurement Conference*, ser. IMC '15, Tokyo, Japan: Association for Computing Machinery, 2015, pp. 275–287. [Online]. Available: https://doi.org/10.1145/2815675.2815692.

[23] X. Foukas, G. Patounas, *et al.*, «Network Slicing in 5G: Survey and Challenges», *IEEE Communications Magazine*, vol. 55, no. 5, pp. 94–100, 2017.

[24] A. Anand, G. de Veciana, and S. Shakkottai, «Joint Scheduling of URLLC and eMBB Traffic in 5G Wireless Networks», *IEEE/ACM Transactions on Networking*, vol. 28, no. 2, pp. 477–490, 2020.

[25] M. Sauter, *From GSM to LTE-Advanced Pro and 5G*. John Wiley and Sons Ltd, 2021.

[26] 3GPP, «5G; System Architecture for the 5G System (5GS); (3GPP TS 23.501 version 16.6.0 Release 16)», 3GPP, Tech. Rep. ETSI TS 123 501 V16.6.0 (2020-10), 2020.

[27] 3GPP, «5G; Study on Scenarios and Requirements for Next Generation Access Technologies (3GPP TR 38.913 version 14.2.0 Release 14)», 3GPP, Tech. Rep. ETSI TR 138 913 V14.2.0 (2017-05), 2017.

[28] OpenAirInterface Organization, *Openairinterface*. [Online]. Available: https://openairinterface.org/ (last visited: May 22, 2024).

[29] Open5GS, *Open5gs*. [Online]. Available: https://open5gs.org/ (last visited: Jun. 5, 2024).

[30] S. Rao and R. Prasad, «Impact of 5g technologies on industry 4.0», *Wireless Personal Communications*, vol. 100, pp. 1–15, May 2018.

[31] L. D. Xu, E. L. Xu, and L. Li, «Industry 4.0: State of the Art and Future Trends», *International Journal of Production Research*, vol. 56, no. 8, pp. 2941–2962, 2018.

[32] S. Wang, J. Wan, *et al.*, «Towards Smart Factory for Industry 4.0: a Self-Organized Multi-Agent System with Big Data based Feedback and Coordination», eng, *Computer networks (Amsterdam, Netherlands : 1999)*, vol. 101, pp. 158–168, 2016.

[33] A. Mahmood, S. F. Abedin, *et al.*, «Factory 5G: A Review of Industry-Centric Features and Deployment Options», *IEEE Industrial Electronics Magazine*, vol. 16, no. 2, pp. 24–34, 2022.

[34] J. Dugan, S. Elliott, *et al.*, *iPerf 3 User Documentation*, n.d. [Online]. Available: https://iperf.fr/iperf-doc.php#3doc (last visited: Jun. 6, 2024).

[35] P. Emmerich, S. Gallenmüller, *et al.*, «Mind the Gap-A Comparison of Software Packet Generators», in *2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, IEEE, 2017, pp. 191–203.

[36] S. P, *Ostinato - features*. [Online]. Available: https://ostinato.org/features (last visited: May 9, 2024).

[37] P. Biondi, *Scapy - general documentation*. [Online]. Available: https://scapy.readthedocs.io/en/latest/ (last visited: May 9, 2024).

[38] S. Lange, A. Nguyen-Ngoc, *et al.*, «Performance Benchmarking of a Software-Based LTE SGW», in *2015 11th International Conference on Network and Service Management (CNSM)*, IEEE, 2015, pp. 378–383.

[39] TRex Team, *TRex Stateless Support*. [Online]. Available: https://trex-tgn.cisco.com/trex/doc/trex_stateless.html (last visited: May 9, 2024).

[40] TRex Team, *Trex advanced stateful support*. [Online]. Available: https://trex-tgn.cisco.com/trex/doc/trex_astf.html (last visited: May 18, 2024).

[41] Wireshark, *Tshark(1) manual page*. [Online]. Available: https://www.wireshark.org/docs/man-pages/tshark.html (last visited: Mar. 13, 2024).

[42] Tcpdump, *Tcpdump(1) man page*. [Online]. Available: https://www.tcpdump.org/manpages/tcpdump.1.html (last visited: Mar. 13, 2024).

[43] Wireshark, *Dumpcap(1) manual page*. [Online]. Available: https://www.wireshark.org/docs/man-pages/dumpcap.html (last visited: Mar. 13, 2024).

[44] Grafana labs, *Grafana data sources*. [Online]. Available: https://grafana.com/docs/grafana/latest/datasources/ (last visited: May 3, 2024).

[45] Grafana labs, *Json model*. [Online]. Available: https://grafana.com/docs/grafana/latest/dashboards/build-dashboards/view-dashboard-json-model/ (last visited: May 3, 2024).

[46] Grafana labs, *Variables*. [Online]. Available: https://grafana.com/docs/grafana/latest/dashboards/variables/ (last visited: May 3, 2024).

[47] Grafana labs, *Modify dashboard settings*. [Online]. Available: https://grafana.com/docs/grafana/latest/dashboards/build-dashboards/modify-dashboard-settings/ (last visited: May 3, 2024).

[48] Grafana labs, *Visualizations*. [Online]. Available: https://grafana.com/docs/grafana/latest/panels-visualizations/visualizations/ (last visited: May 3, 2024).

[49] Grafana labs, *Grafana documentation*. [Online]. Available: https://grafana.com/docs/grafana/latest/#grafana-documentation (last visited: May 3, 2024).

[50] Ansible, *Introduction to ansible*. [Online]. Available: https://docs.ansible.com/ansible/latest/getting_started/introduction.html (last visited: May 3, 2024).

[51] Ansible, *Introduction to ad hoc commands*. [Online]. Available: https://docs.ansible.com/ansible/latest/command_guide/intro_adhoc.html#why-use-ad-hoc-commands (last visited: May 3, 2024).

[52] Ansible, *Ansible playbooks*. [Online]. Available: https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_intro.html#playbook-syntax (last visited: May 3, 2024).

[53] Ansible, *How to build your inventory*. [Online]. Available: https://docs.ansible.com/ansible/latest/inventory_guide/intro_inventory.html (last visited: May 3, 2024).

[54] Ansible, *Using variables*. [Online]. Available: https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_variables.html (last visited: May 4, 2024).

[55] S. Gallenmüller, P. Emmerich, *et al.*, «Comparison of Frameworks for High-Performance Packet IO», in *2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2015, pp. 29–38.

[56] *Network bridge - ArchWiki*, English. [Online]. Available: https://wiki.archlinux.org/title/network_bridge (last visited: Apr. 12, 2024).

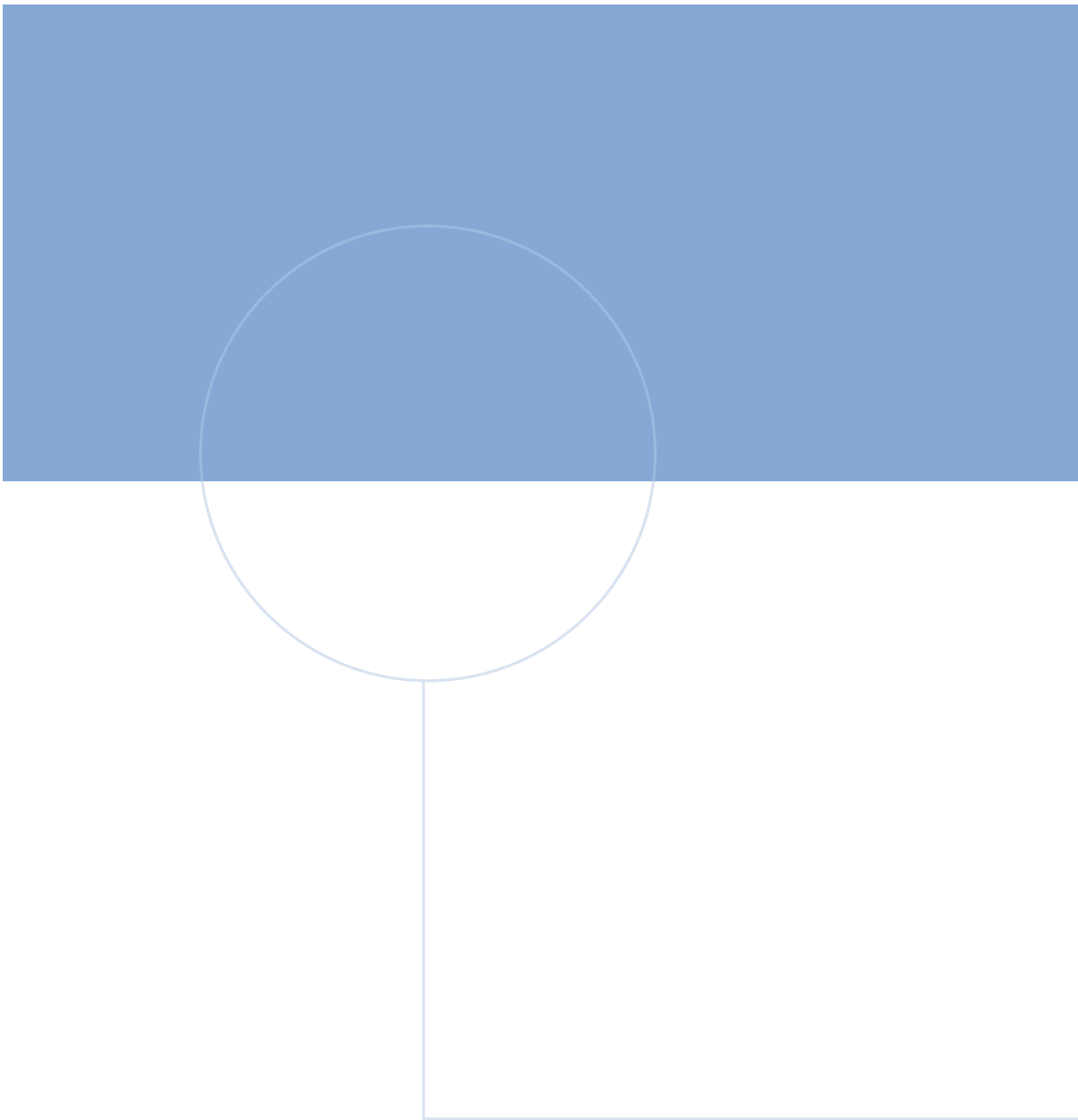
[57] S. Hemminger, «Network Emulation with NetEm», in *Linux conf au*, vol. 5, Apr. 2005, p. 8.

[58] A. Jurgelionis, J.-P. Laulajainen, *et al.*, «An Empirical Study of NetEm Network Emulation Functionalities», in *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*, 2011, pp. 1–6.

[59] F. Ludovici and H. P. Pfeifer, *tc-netem(8) - Linux manual page*. [Online]. Available: https://man7.org/linux/man-pages/man8/tc-netem.8.html (last visited: Apr. 25, 2024).

[60] T. Barbette, C. Soldani, and L. Mathy, «Fast Userspace Packet Processing», in *2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, IEEE, 2015, pp. 5–16.

[61] S. Lange, L. Linguaglossa, *et al.*, «Discrete-Time Modeling of NFV Accelerators That Exploit Batched Processing», in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, IEEE, 2019, pp. 64–72.



NTNU

Norwegian University of
Science and Technology