UNIVERSIDADE D
COIMBRA

Diogo Emanuel Ribeiro Cruz

# DESIGN AND DEVELOPMENT OF LABORATORY INFRASTRUCTURE FOR 5G LAN NETWORK ENVIRONMENTS

**Dissertation in the context of the Master in Informatics Engineering, specialization in Software Engineering, advised by Professor Vasco Nuno Sousa Simões Pereira and Professor Tiago José dos Santos Martins da Cruz presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.**

September of 2023

This page is intentionally left blank.

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

DEPARTMENT OF INFORMATICS ENGINEERING

Diogo Emanuel Ribeiro Cruz

# Design and Development of Laboratory Infrastructure for 5G LAN Network Environments

Dissertation in the context of the Master in Informatics Engineering, specialization in Software Engineering, advised by Professor Vasco Nuno Sousa Simões Pereira and Professor Tiago José dos Santos Martins da Cruz presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

September of 2023

This page is intentionally left blank.

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Diogo Emanuel Ribeiro Cruz

# Conceção e Desenvolvimento de Infraestrutura Laboratorial para Ambientes de Rede 5G LAN

Dissertação no âmbito do Mestrado em Engenharia Informática, especialização em Engenharia de Software, orientada pelo Professor Vasco Nuno Sousa Simões Pereira e o Professor Tiago José dos Santos Martins da Cruz e apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Setembro de 2023

This page is intentionally left blank.

# Acknowledgements

First and foremost, I would like to thank my thesis advisors, Professor Vasco Pereira and Professor Tiago Cruz, for their invaluable guidance, support, and incentive throughout the course of this dissertation. Their expertise and direction were crucial to the success of this work.

I would also like to express my gratitude to Jorge Proença for always showing avaliabilty to help, especially for all of the valuable corrections and suggestions he provided.

Finally, I would like to thank my family and friends for their support and motivation throughout this journey. Without their love and encouragement, this would not have been possible.

This work was funded by the project POWER (grant number POCI - 01 - 0247 - FEDER - 070365) [1], co-financed by the European Regional Development Fund (FEDER), through Portugal 2020 (PT2020), and by the Competitiveness and Internationalization Operational Programme (COMPETE 2020). It was also partially supported by the Smart5Grid project (co-funded by FEDER - Competitiveness, and Internationalisation Operational Program (COMPETE 2020), Portugal 2020 framework) [2].

This page is intentionally left blank.

# Abstract

With the advent of 5G technology, a range of application scenarios and use cases emerges, facilitated by its specific characteristics. Research in this domain is a priority to understand and fully exploit the possibilities offered by the ecosystem and associated services. However, gaining access to production infrastructures and real-world scenarios is complex, as they are considered essential services for which availability is a priority, being typically managed by telecommunications operators. Similarly, establishing a 5G network with the corresponding radio infrastructure for laboratory purposes requires licenses and spectrum usage authorizations, often slow and challenging to obtain.

Therefore, the purpose of this thesis is to conceive and implement a high-fidelity 5G test environment for 5G Local Area Networks (LANs) use cases, providing the key elements for an Industrial Internet of Things (IoT) ecosystem over 5G. The goal is to create a hybrid laboratory environment, comprising a combination of real, emulated, or virtualized components, facilitating Research and Development (R&D) activities focused on exploring use cases, integrating components, and managing infrastructure.

This environment has undergone an experimental validation process to assess its functional characteristics and performance, which has confirmed its capabilities for the intended purposes. The validation effort has also helped identify strategic action points for future development and improvement activities.

# Keywords

5G, 5G LANs, Laboratory Testbeds, Industrial IoT, Infrastructure Management, Virtualized Services

This page is intentionally left blank.

# Resumo

Com o advento da tecnologia 5G, emerge igualmente um conjunto de cenários de aplicação e casos de uso viabilizados pelas suas características específicas, cuja investigação se reveste como uma prioridade, no sentido de compreender e melhor explorar todas as possibilidades oferecidas pelo ecossistema e serviços associados. Contudo, o acesso a infraestruturas de produção e cenários reais reveste-se de um considerável grau de complexidade, visto serem considerados serviços essenciais cuja disponibilidade é prioritária, sendo a sua responsabilidade operacional assegurada pelos operadores de telecomunicações. Do mesmo modo, a criação de uma rede 5G com o correspondente estabelecimento de uma infraestrutura rádio para fins laboratoriais está pendente de licenciamentos e autorizações de utilização de espectro, frequentemente morosas e difíceis de obter.

Assim sendo, o propósito desta tese é o de conceber e implementar um ambiente de testes 5G de alta-fidelidade para casos de uso de Redes de Área Locais (LANs) 5G, que proporcione os elementos principais de um ecossistema de Internet das Coisas (IoT) industrial sobre 5G. Deste modo, pretendeu-se disponibilizar um ambiente laboratorial híbrido, composto por um misto de componentes reais e emulados ou virtualizados, que permita o desenvolvimento de atividades de Inovação e Desenvolvimento (I&D) focadas na exploração de casos de uso, integração de componentes e gestão de infraestrutura.

Este ambiente foi ainda sujeito a um processo de validação experimental, para aferir as suas características funcionais e em termos de performance. Este esforço de validação permitiu também identificar pontos-chave para futuros desenvolvimentos da plataforma.

# Palavras-Chave

5G, 5G LANs, Testbeds Laboratoriais, IoT Industrial, Gestão de Infrastrutura, Serviços Virtualizados

This page is intentionally left blank.

# Contents

# Acronyms

**3GPP** 3rd Generation Partnership Project.

**5GENESIS** 5th Generation End-to-end Network, Experimentation, System Integration, and Showcasing.

**5G-VINNI** 5G Verticals INNovation Infrastructure.

**5G-EVE** 5G European Validation platform for Extensive trials.

**5G-PPP** 5G Infrastructure Public Private Partnership.

**5GC** 5G Core.

**AF** Application Function.

**AI** Artificial Intelligence.

**AKA** Authentication and Key Agreement.

**AMF** Access and Mobility Management Function.

**API** Application Programming Interface.

**AR** Augmented Reality.

**ARP** Address Resolution Protocol.

**ARPF** Authentication Credential Repository and Processing Function.

**AUSF** Authentication Server Function.

**BSF** Binding Support Function.

**CBRS** Citizens Broadband Radio Service.

**CHF** Charging Function.

**CIoT** Cellular Internet of Things.

**CISUC** Centre for Informatics and Systems of the University of Coimbra.

**CNCF** Cloud Native Computing Foundation.

**CNs** Core Networks.

**CSCF** Call State Control Function.

**DB** Database.

**DHCP** Dynamic Host Configuration Protocol.

**DN** Data Network.

**DNS** Domain Name System.

**DoD** Department of Defense.

**DPDK** Data Plane Development Kit.

**DSS** Dynamic Spectrum Sharing.

**DSSS** Direct Sequence Spread Spectrum.

**ENDC** E-UTRAN New Radio – Dual Connectivity.

**EU** European Union.

**E-UTRAN** Evolved Universal Terrestrial Radio Access Network.

**eMBB** Enhanced Mobile Broadband.

**EPC** Evolved Packet Core.

**EPS** Evolved Packet System.

**FBD** Function Block Diagram.

**FHSS** Frequency-Hopping Spread Spectrum.

**FR** Functional Requirement.

**gNB** gNodeB.

**gNB-DUs** gNB-Distributed Units.

**gNB-CU** gNB-Central Unit.

**GUI** Graphical User Interface.

**HA** High Availability.

**HSS** Home Subscriber Server.

**I/O** Input/Output.

**ICT** Information and Communications Technology.

**IIoT** Industrial Internet of Things.

**IL** Instruction List.

**IMS** IP Multimedia Subsystem.

**IMSI** International Mobile Subscriber Identity.

**IMT-2020** International Mobile Telecommunications - 2020.

**IoT** Internet of Things.

**IP** Internet Protocol.

**IT** Information Technology.

**ITU** International Telecommunication Union.

**K8s** Kubernetes.

**KPIs** Key Performance Indicators.

**LAN** Local Area Network.

**LANs** Local Area Networks.

**LD** Ladder Logic.

**LTE** Long Term Evolution.

**LXDE** Lightweight X11 Desktop Environment.

**MCC** Mobile Country Code.

**MIMO** Multiple-Input Multiple-Output.

**ML** Machine Learning.

**MME** Mobility Management Entity.

**mMTC** Massive Machine-Type Communications.

**MNC** Mobile Network Code.

**MSPs** Managed Service Providers.

**NEF** Network Exposure Function.

**NFR** Non-Functional Requirement.

**NFs** Network Functions.

**NFV** Network Function Virtualization.

**NG** Network Gateway.

**NG-RAN** Next Generation Radio Access Network.

**NR** New Radio.

**NRF** Network Repository Function.

**NSA** Non-Standalone.

**NSSAI** Network Slice Selection Assistance Information.

**NSSF** Network Slice Selection Function.

**NTN** Non-Terrestrial Networks.

**PCF** Policy Control Function.

**PCRF** Policy and Charging Rules Function.

**PDN** Public Data Network.

**PGW-U** Packet Gateway User Function.

**PGW-C** Packet Gateway Control Function.

**PLC** Programmable Logic Controller.

**PLCs** Programmable Logic Controllers.

**QoS** Quality of Service.

**RAN** Radio Access Network.

**RKE2** Rancher Kubernetes Engine 2.

**SA** StandAlone.

**SAE** System Architecture Evolution.

**SBA** Service-Based Architecture.

**SCADA** Supervisory Control and Data Acquisition.

**SCTP** Stream Control Transmission Protocol.

**SD** Slice Differentiator.

**SFC** Sequential Function Chart.

**SGW-U** Serving Gateway User Function.

**SGW-C** Serving Gateway Control Function.

**SMF** Service Mobility Function.

**SMF** Session Management Function.

**SSL** Secure Sockets Layer.

**ST** Structured Text.

**TC** Technical Constraint.

**TCP** Transmission Control Protocol.

**TLS** Transport Layer Security.

**TSG** Technical Specification Group.

**UDM** Unified Data Management.

**UDP** User Datagram Protocol.

**UDR** Unified Data Repository.

**UE** User Equipment.

**UEs** User Equipments.

**UI** User Interface.

**UPF** User Plane Function.

**URLLC** Ultra Reliable Low Latency Communications.

**US** United States.

**USIM** Universal Subscriber Identity Module.

**VM** Virtual Machine.

**VNC** Virtual Network Computing.

**VPN** Virtual Private Network.

**VPP** Vector Packet Processing.

**VR** Virtual Reality.

**Xn-C** Xn control plane.

This page is intentionally left blank.

# List of Figures

This page is intentionally left blank.

# List of Tables

This page is intentionally left blank.

# Chapter 1

# Introduction

In this chapter, we will present the motivations that drove the creation of this thesis and the context of the research work. Furthermore, the objectives and contributions produced to the projects, and the work plan pursued in the first and second semesters. Finally, it presents the risks involved in completing this thesis and the thesis structure.

## 1.1 Motivations

This thesis emerges in response to the necessity of having an advanced laboratory infrastructure for 5G-based Local Area Networks scenarios, showcasing high performance, ultra-low latency, high reliability, and support for Massive Machine-Type Communications (mMTC). Such a laboratory, embodying these crucial attributes, stands as an indispensable asset, positioned to serve as a foundation for both the POWER and Smart5Grid projects. By encompassing these foundational traits, this testbed contributes to innovation, with a specific focus on domains like 5G private networks, and Industrial Internet of Things (IIoT).

## 1.2 Context

The 5th generation of mobile network technology, commonly designated by 5G, was conceived to connect virtually everyone and everything, including machines, objects, and all sorts of devices. It was designed to be faster, more scalable, more energy efficient, and more reliable, providing massive network capacity and increased availability when compared to previous generations, while being able to cope with ultra-low latency requirements.

In the present landscape, 5G networks already empower solutions that link devices together, providing the capacity and means to seamlessly connect a massive number of gadgets and sensors, with diverse data rate requirements, low latency, and low power consumption. Within the domain of IIoT, 5G plays a pivotal role in changing the industrial landscape by optimizing processes, minimizing

downtime, enabling predictive maintenance, and delivering high-performance networks easily and at reduced costs. In the context of smart cities, it underpins sophisticated traffic management systems, advanced lighting solutions, and real-time environmental monitoring. This adaptability is amplified by the 5G Service Architecture (SA), which is based on a microservice concept, dividing its core through multiple functions, that can open doors for flexible horizontal scaling, and therefore optimal performance in adverse workloads. Furthermore, the 3rd Generation Partnership Project (3GPP) specifications encompass specific support for verticals utilizing slicing and 5G Local Area Networks (LANs), paving the way for a paradigm shift in terms of the relationship between service, telecom, and operational infrastructure tenants.

Taking into consideration the 5G potential arises Project POWER and Project Smart5Grid. In Project POWER *"Empowering a digital future"* [1], the emphasis is on crafting solutions for commercialization within enterprises, leveraging cloud technologies, cognitive capabilities, and strategic technology vectors such as private 5G networks, Edge/Cloud computing, data-driven technologies, and Artificial Intelligence (AI). Conversely, Project Smart5Grid [2] is oriented towards devising a framework adaptable for smart energy networks, seeking to enhance the operations of energy systems by harnessing the potential of private 5G networks.

## 1.3 Objectives and Contributions

The primary goal of this thesis is to establish a laboratory environment capable of supporting 5G LAN technologies and solutions. It will encompass resource orchestration, service virtualization using containers, and a functional 5G core. This objective is fueled by the creation of an IIoT use case that leverages 5G LANs, involving physical and virtualized 5G components. Furthermore, the laboratory is prepared for potential federation with partner infrastructures via an Virtual Private Network (VPN) communication solution. Crucially, the overarching vision of this testbed extends beyond its primary objective, serving as a foundational base for not only the POWER and Smart5Grid projects but also future initiatives yet to unfold. Finally, documentation and validation of the scenario are critical segments of this thesis.

During the realization of the goals and as part of the Centre for Informatics and Systems of the University of Coimbra (CISUC) ecosystem, a set of contributions were made that must be noted:

- The testbed containing 5G LAN solutions and incorporating the IIoT use case scenario [3].
- One article published [4] as part of the Smart5Grid and POWER projects.
- Presentation titled "SP3 - Future Operations: 5G Private Networks" in the workshop of the project POWER on February 8th, 2023, in Coimbra.

- Presentation titled "Designing an Industrial IoT 5G testbed" at the 33rd Seminar of Mobile Communications Thematic Network [5] on February 10th, 2023, at the NOVA School of Science and Technology in Lisbon.

- Presentation titled "Design and Development of Laboratory Infrastructure for 5G LAN Network Environments" in the workshop of the project POWER titled "5G and Beyond" [6] on June 20th, 2023, in Coimbra.

- Conducted a live demo of the testbed as part of the Smart5Grid project on July 20th, 2023, remotely.

The testbed is housed within the CISUC data center and can be replicated to cloud or other on-premises centers.

## 1.4 Work Plan

Throughout the 1st semester, our work plan centered on studying, analyzing, and designing a system-specific use case. We also conducted an in-depth assessment of the current State of the Art and relevant prior research. The main activities undertaken during this period included:

- Acquire background knowledge

- Study State of the Art

- Design the use case based on 5G LANs and IIoT nodes

- Research technologies based on the use case

- Build the testbed consisting primarily of 5GC, virtual PLCs, virtual UEs, and NG-RAN

- Write dissertation document

The following figure 1.1 shows the work that was concluded during the first semester:

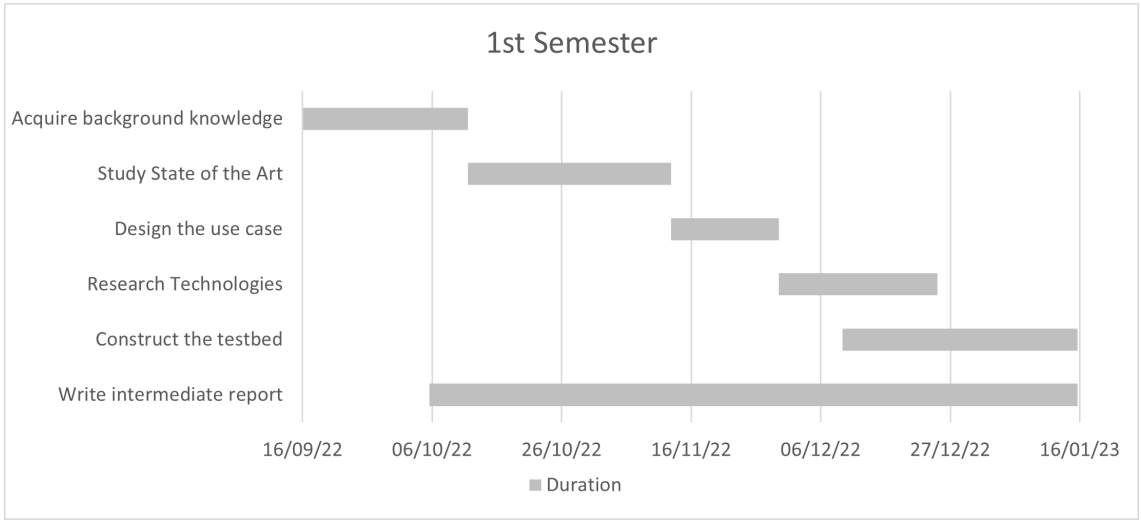


Figure 1.1: Work Plan 1st Semester

In the second semester, the primary focus was finishing developing the testbed, accompanied by comprehensive validation tests for the entire environment. The main tasks completed during this time were:

- Develop the testbed
  - Integrate the RaspberryPI serving as UE
  - Integrate the physical Programmable Logic Controller (PLC)
- Conduct validation tests for the use case requirements
- Compose the final dissertation document

The following figure 1.2 shows the work accomplished for the second semester:

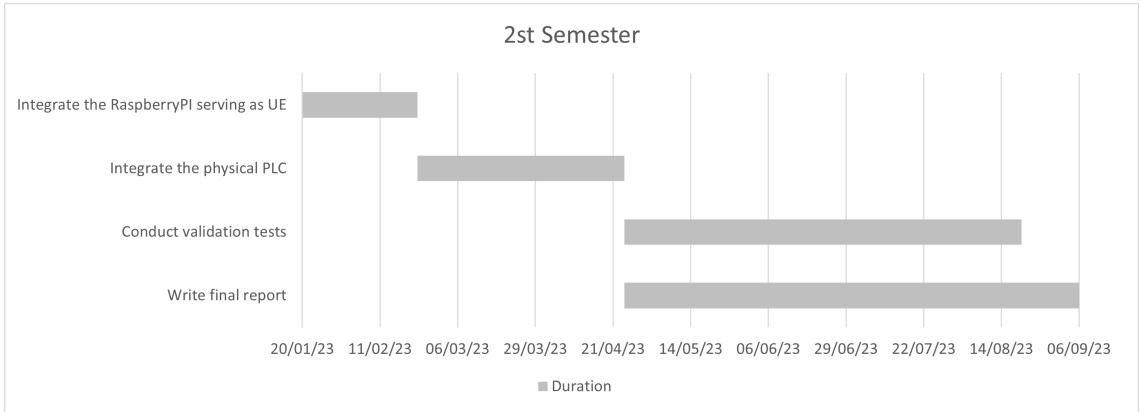


Figure 1.2: Work Plan 2st Semester

## 1.5 Risks

Risk is the combination of the likelihood that the event will occur and its potential impact on the project. Events with a high likelihood and high impact are considered serious risks.

The table 1.1 presents a review of the criteria for evaluating risks.

| **Likelihood** | **Impact** | **Severity (Likelihood x Impact)** |
|---|---|---|
| 0 - Unlikely | 0 - Insignificant | 0 - Insignificant (0 x 0) |
| 1 - Possible | 1 - Minor | 1 - Minor (1 x 1) |
| 2 - Likely | 2 - Moderate | 2 - Moderate (1 x 2 or 2 x 1) |
| 3 - Almost certain | 3 - Major | 3 - Major (1 x 3 or 3 x 1) |
| **Status** | | 4 - Severe (2 x 2) |
| • Achieved: The risk has occurred<br>• Not Achieved: The risk has not occurred<br>• Pending: The risk has the potential to occur in the future | | 6 - Critical (3 x 2 or 2 x 3)<br>9 - Catastrophic (3 x 3) |

Table 1.1: Risk Criteria.

The table 1.2 presents events that could negatively impact the thesis if they occur. It includes a description of each risk, its severity, a plan for managing it, and its current status.

| ID | Risk Description | Likelihood | Impact | Severity | Contingency Plan | Status |
|---|---|---|---|---|---|---|
| 1 | **Absence of a Physical Next Generation Radio Access Network (NG-RAN):** This risk involves the potential unavailability of a physical NG-RAN infrastructure for the testbed. | 2 | 2 | **4** | Implement software emulation to replicate NG-RAN behavior, enabling testing of user equipment interactions and network performance despite the lack of a physical NG-RAN. | Achieved |
| 2 | **Risk of an incomplete definition of the use case:** This risk emerges if the project goals or requirements, underlying this thesis, shift during its execution, causing an unclear use case definition. | 1 | 3 | **3** | As a backup strategy, regularly review and update project goals, adopt flexible development, involve stakeholders early, create a communication plan, develop a fallback plan, and assign a dedicated project manager. | Not Achieved |
| 3 | **Risk of the software not having all the required functionalities:** It refers to the potential situation where the set software solution may lack certain features that are necessary to meet the project objectives and requirements. | 1 | 3 | **3** | Leverage open-source alternatives and community contributions to address potential gaps in functionality. Otherwise, consider in-house development to meet the required functionalities. | Not Achieved |

Table 1.2: Risks that may affect the completion of this thesis.

## 1.6 Document Structure

The document structure is organized as follows. In chapter 2, we provide background knowledge on the core technologies important to this thesis. In chapter

3 (State of the Art), we present a system overview of 5G, including a detailed explanation of its architecture and an analysis of 5G Verticals and 5G LANs with support to IIoT. In chapter 4, we outline the use case and its requirements. Chapter 5 elaborates on the testbed created from the use case, detailing its technologies, architecture, and implementation. Chapter 6 is dedicated to validating the requirements of the use case. Finally, Chapter 7 summarizes key points of this thesis and suggests future work.

This page is intentionally left blank.

# Chapter 2

# Background Knowledge

This chapter aims to present an overview of the essential technologies and concepts that are relevant to the research discussed in the following chapters. The purpose of this overview is to provide readers with the necessary foundational knowledge to comprehensively grasp the analysis of these technologies distributions, as well as the subsequent sections outlining the use case and testbed within the thesis.

## 2.1 Kubernetes

Kubernetes (K8s) [7] is an open-source platform for automating the deployment, scaling, and management of containerized applications. It was originally created by Google and is now maintained by the Cloud Native Computing Foundation (CNCF). Kubernetes is designed to run across a cluster of machines, possibly even across multiple data centers. It allows you to schedule and run containerized applications on those machines, as well as to scale those applications up or down and roll out new features or updates with minimal downtime.

One of the key features of Kubernetes is its use of "pods" to run multiple containers together. Pods are the smallest deployable units in Kubernetes and can be used to host applications or perform specific tasks such as running a database. Pods can be managed and scaled independently, and can also be replicated to provide high availability for the applications that they host.

Kubernetes also provides a number of other features to help the management of containerized applications, including:

- Service discovery and load balancing: It can automatically expose the services running in your pods to the outside world, and can also load balance traffic between them.
- Configuration management: Allows you to manage the configuration of your applications through the use of "config maps" and "secrets".

- Persistent storage: Permits you to attach persistent storage to your pods so that your applications can retain data even if the pods are terminated or moved to different machines.

- Health checking: Monitors the health of your applications and take corrective action if necessary, such as restarting or replacing unhealthy pods.

Overall, K8s provides a powerful and flexible platform for deploying, scaling, and managing containerized applications at scale.

## 2.2 Programmable Logic Controller

A Programmable Logic Controller (PLC) [8] is a type of computer that is specifically designed for use in industrial control systems. It is used to automate a wide variety of industrial processes, such as factory automation, oil and gas refining, and power generation.

PLCs are ruggedized and built to withstand harsh industrial environments, and they are typically more reliable and durable than traditional computers. They are also designed to be easily programmable and reconfigurable so that they can be adapted to control different processes as needed.

One of the key features of PLCs is their use of Input/Output (I/O) modules to interface with the physical world. These modules can be used to read data from sensors, such as temperature or pressure sensors, and to control actuators, such as motors or valves. The PLC can then use this data to make decisions and take actions based on a set of pre-programmed instructions.

PLCs are typically programmed using one of the five languages defined in the IEC 61131-3 standard [9]: Ladder Logic (LD), Function Block Diagram (FBD), Instruction List (IL), Structured Text (ST), and Sequential Function Chart (SFC). These languages are designed to be easy to use and to closely mimic the logic of physical control systems, making it simple for engineers and technicians to program and troubleshoot PLCs.

Overall, PLCs are an essential component of many industrial control systems, and they play a crucial role in automating and optimizing a wide variety of processes across many different industries.

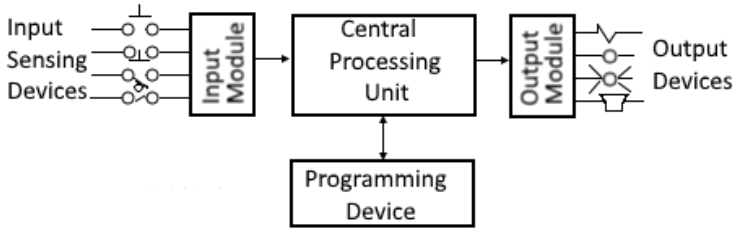


Figure 2.1: Programmable Logic Controller Diagram. Source: [10]

The figure 2.1 shows how the PLC operates. The input module maps the physical input sensing devices, then it passes in a program with certain functionalities designed by the user. Finally, the output maps register to the physical output devices and loop again all these processes.

## 2.3 Modbus

Modbus [11] is a serial communication protocol that was originally developed in 1979 for use with PLCs. It is used to establish communication between different devices, such as PLCs, computers, and other industrial control equipment.

Modbus is a client-server protocol, which means that one device (the client) sends a request to another device (the server) and the server sends a response. Modbus can be used over a variety of physical communication media, including RS-232 [12], RS-485 [13], and Ethernet.

Modbus does not provide any encryption or secure communication on its own. If security is a concern, it is typically implemented at a higher level, such as by using a secure communication protocol like Secure Sockets Layer (SSL)/Transport Layer Security (TLS) to encrypt the data transmitted over the network. Alternatively, Modbus packets can be encapsulated within another protocol, such as Transmission Control Protocol (TCP)/Internet Protocol (IP), which does provide some level of security.

It is important to note that the lack of built-in security in Modbus can make it vulnerable to man-in-the-middle attacks [14] and other types of cyber threats. As a result, it is important to carefully consider the security requirements of your system and to take appropriate steps to protect against potential threats.

One of the essential features of Modbus is its use of function codes to specify the type of action that the client is requesting. For example, a function code might be used to request that the server send back the current value of a particular register, or to request that the server write a new value to a register.

Modbus is a widely used protocol in the industrial automation and control industry, and it is supported by many different types of devices. It is known for its simplicity and ease of use, and it is often used as a "language" that allows different devices to communicate with each other.

Overall, Modbus is a valuable tool for enabling communication between different devices in industrial control systems, and it continues to be widely used today.

## 2.4 Industrial Internet of Things

The Internet of Things (IoT) [15] refers to the growing network of physical objects that are connected to the internet and can communicate with each other and with people. These objects, which can include everyday household appliances, indus-

trial equipment, and vehicles, are equipped with sensors and other technologies that allow them to collect and exchange data. The IoT allows for the remote monitoring and control of these objects, and it has the potential to transform many different industries by enabling the automation of processes and the collection of vast amounts of data.

The Industrial Internet of Things (IIoT) is a specific application of the IoT in industrial settings, such as manufacturing, retail, healthcare, and other businesses. IIoT devices, ranging from sensors to equipment, provide businesses with real-time, detailed data that can be used to improve business processes and increase efficiency. These devices can be used to optimize supply chain management, logistics, human resources, and reduce production costs and increase revenue streams.

Some examples of IIoT applications in different verticals include:

- Manufacturing: In manufacturing can be used for predictive maintenance to reduce unplanned downtime, and for the wearables to improve operator safety.
- Automotive: Sensor-driven analytics and robotics can increase efficiency in automotive maintenance and manufacturing.
- Logistics and transportation: Devices can assist in supply chain management, including inventory management, supplier relationship management, fleet management, and scheduled maintenance.
- Retail: It can be used for automation and human-machine collaboration in retail.

Overall, IIoT has the potential to revolutionize many different industries and transform the way we live and work. However, they also raise concerns about data privacy and security, as well as the potential for widespread job automation.

## 2.5 Chapter Wrap-up

In conclusion, this chapter provided an overview of several key technologies that are commonly used in industrial automation and control systems. These technologies have been chosen to underpin our work due to their unparalleled relevance and utility. Their alignment with our thesis objectives underscores their significance in meeting our intended outcomes. We covered Kubernetes, a container orchestration platform that is used to automate the deployment, scaling, and management of containerized applications. We covered the concept of PLCs, which are specialized computers that are used to automate a wide variety of industrial processes. We also discussed the Modbus communication protocol, which is used to establish communication between different devices in industrial control systems. Finally, we explored the IIoT, which refers to the use of smart devices in industrial settings to increase efficiency and optimize business processes.

# Chapter 3

# State of the Art

In the following sections, we will explore the system overview of 5G, including its architecture based on the Service-Based Architecture (SBA) framework. We will also delve into the topic of 5G verticals and 5G Local Area Networks with support for the Industrial Internet of Things (IIoT).

## 3.1 5G System Overview

International Mobile Telecommunications - 2020 (IMT-2020) is a set of standards that specified a collection of requirements for future 5G networks that have been developed by the International Telecommunication Union (ITU) in 2015. The 3rd Generation Partnership Project (3GPP) is responsible for developing the technical specifications that implement the IMT-2020 standards in mobile communication networks. In 2018, 3GPP released its first pack of technical specifications for standalone 5G networks, known as Release 15. This release included the full batch of 3GPP specifications for 5G. This phase introduces a new radio transmission technique and other important concepts, such as improved reliability, increased modularity, and faster response times.

Release 16, which was functionally frozen in December 2020, specifies the second phase of 5G deployment. This phase builds upon the foundations established in Release 15 and introduces new features and capabilities to further improve the performance of 5G networks. Some of the key features of Release 16 include:

- Enhanced Mobile Broadband (eMBB): This feature aims to improve the quality of mobile broadband services by increasing the data rates and capacity of 5G networks.

- Ultra Reliable Low Latency Communications (URLLC): This feature aims to provide extremely low latency and high reliability for applications that require real-time communication, such as remote surgery or autonomous driving.

- Massive Machine-Type Communications (mMTC): This feature aims to support a large number of devices with low data rate requirements, such as sensors and other IoT devices.

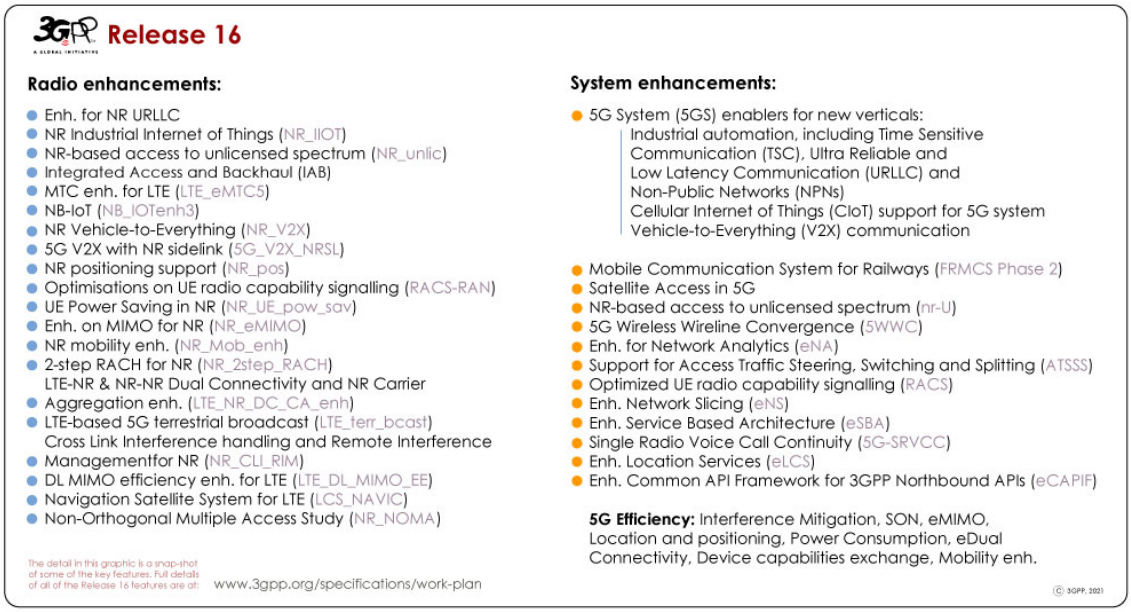


Figure 3.1: Release 16 Description. Summary of Rel-16 Work Items. Source: [16]

Release 16 presented enhancements to the 5G core network, including support for network slicing, which allows for the deployment and use of multiple Core Networks (CNs) simultaneously, each one specialized in providing a specific set of services and/or subscribers. This allows for greater flexibility and simplifies maintenance in the network deployment. It also introduced Network Function Virtualization (NFV), which allows for the communication of all Network Functions (NFs) through a common interface and the ability to locate them anywhere. This adds to the flexibility of the network. Additionally, introduced edge computing, which brings computational power closer to the end-user to reduce the response time of the network for applications that require low latency, such as virtual reality, autonomous driving, and factories of the future.

Release 17, which was functionally frozen in March 2022, brought further enhancements for 5G networks. [17] Some of the main features are:

- Enhanced support for non-public networks: Focuses on improving security, scalability, and customization options for private 5G networks deployed by enterprises and industries.

- Internet of Things (IoT) over Non-Terrestrial Networks (NTN): It enables seamless integration of IoT devices with non-terrestrial infrastructure, opening up possibilities for remote sensing, precision agriculture, and disaster response.

- Further enhancements on Multiple-Input Multiple-Output (MIMO) for New Radio (NR): Optimizes MIMO technology in 5G networks, improving performance, capacity, and coverage for higher data rates and better network reliability.

These advancements expand the capabilities of 5G, enabling customized private networks, seamless connectivity in remote environments, and improved performance. Release 17 reinforces 5G position as a versatile technology driving innovation across verticals.

Release 18 of the 3GPP specification are expected to introduce new features and capabilities. [18] Some of the essential attributes that are expected to be included according to the Technical Specification Group (TSG) meetings are:

- Support for new frequency bands and technologies: It is expected to be used for 5G in many regions around the world. They may also introduce support for new technologies such as Dynamic Spectrum Sharing (DSS), which allows for the sharing of spectrum between different wireless technologies.
- Improvements to the 5G core network: It includes support for new network architectures and protocols that will improve the efficiency and performance of 5G networks.
- Enhancements to the 5G radio access network: It includes enhancements to the 5G NR air interface and support for new spectrum bands.
- Improvements to interworking between different networks: Enhancements to the protocols that enable interworking between different networks, such as the 5G System Architecture Evolution (SAE) and the 5G Non-Standalone (NSA) architecture.

It should be noted that the exact features and capabilities of 3GPP Release 18 are still being developed and may change prior to the finalization of the specifications.

Next, we will examine the architecture of 5G and explain the various components that make up this complex system. We will also discuss the roles and capabilities of these components, and how they contribute to the overall performance of 5G networks.

## 3.2 5G Architecture

As previously referred, the 5G architecture is a Service-Based Architecture (SBA). A SBA is a design approach for building systems in which the architecture elements are defined in terms of NFs rather than by traditional network entities. In an SBA, any given NF offers its services to all other authorized NFs and/or to any "consumers" that are permitted to use these provided services. This approach

offers modularity and reusability, as it allows for the creation of flexible and scalable systems that can be easily modified or expanded as needed to support new features and services. [19]

In the context of 5G, the 3GPP has adopted an SBA framework for the 5G Core (5GC). The 5GC is the part of the 5G network that controls the communication between the User Equipment (UE), itself composed of a Mobile Station and a Universal Subscriber Identity Module (USIM), and the data network. It is formed of several NFs that work together to provide various services to the UE. These NFs can be grouped into two main categories: the User Plane, which handles the transport of user data, and the Control Plane, which handles the signaling and control of the network.

**Some of the 5G network components are:**

- The Home Subscriber Server (HSS) is the main subscriber database used within the IP Multimedia Subsystem (IMS), providing details of subscribers to other entities within the network. The IMS enables users to be granted or denied access to other services based on their status. [20]

- The Mobility Management Entity (MME) is a crucial component of the Evolved Packet Core (EPC) for Long Term Evolution (LTE). It manages mobility sessions for the LTE network and supports subscriber authentication, roaming, and handovers to other networks. [21]

- The Policy and Charging Rules Function (PCRF) is a network node that determines how traffic flows over the network for individual subscribers. Using input from other nodes in the network, the PCRF creates rules that control various aspects of traffic flow, such as Quality of Service (QoS) requirements, restrictions, throttling, blocking, and billing. These rules are applied at a granular level, allowing for precise control over individual subscribers data flows in the network. [22]

- The Packet Gateway Control Function (PGW-C) is a component of the Service Mobility Function (SMF). When control and user plane separation is in place, the PGW-C controls the functions performed by the assigned Packet Gateway User Function (PGW-U). When a subscriber establishes an Evolved Packet System (EPS) bearer to a given Public Data Network (PDN), the PGW-C selects and controls the point of attachment to that PDN for the duration of the EPS bearer, regardless of any mobility procedures. The PGW-C is responsible for managing resources for bearer resources, binding bearers to subscribers, managing subscriber IP addresses, and supporting mobility. [23]

- The PGW-U is a component of the User Plane Function (UPF). When control and user plane separation is in place, the PGW-U serves as the user data plane ingress and egress point to the EPC. When a subscriber establishes an EPS bearer to a given PDN, the PGW-U, under the control of the PGW-C, serves as the point of attachment to that PDN for the duration of the EPS bearer, regardless of any mobility procedures. The PGW-U is responsible

for inspecting packets to ensure that they have the appropriate service level applied. [24]

- The Serving Gateway Control Function (SGW-C) controls the functions performed by the assigned Serving Gateway User Function (SGW-U) when control and user plane separation is in place. Each subscriber is served by a single SGW-C and can have multiple SGW-Us selected for multiple PDN connections. When the subscriber moves around the Evolved Universal Terrestrial Radio Access Network (E-UTRAN), their point of attachment to the EPC remains fixed at the SGW-U under the control of the SGW-C, unless the network decides that an SGW-U relocation is required. The SGW-C also has additional responsibilities, such as lawful interception of subscriber traffic and triggering downlink data buffering while the subscriber is paged. The SGW-C also manages packet gateway pause and charging policies based on implemented policies, such as failed paging, abnormal radio link release, and dropped packets/bytes at the SGW-U. [25]

- The SGW-U is the user data plane ingress and egress point of the E-UTRAN side of the EPC when control and user plane separation is in place. When the subscriber moves around the E-UTRAN, their point of attachment to the EPC remains fixed at the SGW-U, unless the network decides that an SGW-U relocation is required. A single subscriber may be supported by multiple SGW-Us if they have connectivity to multiple PDNs. The SGW-U also has additional responsibilities, such as lawful interception of subscriber traffic, inter-operator accounting, and downlink data buffering while the subscriber is paged. [26]

- The Binding Support Function (BSF) is a key component of the 3GPP SBA for 5G Core networks. The BSF enables other NFs, such as an IMS Call State Control Function (CSCF) or Network Exposure Function (NEF), to determine which Policy Control Function (PCF) holds the needed policy and accounting information for each active mobile device data session. This allows the NFs to access and apply the appropriate policies and accounting rules for each session, ensuring that traffic is properly controlled and managed in the network. [27]

- The Network Slice Selection Function (NSSF) is a component of the 3GPP 5G architecture that helps the Access and Mobility Management Function (AMF) select the appropriate network slice instances to serve a particular device. The NSSF determines the allowed Network Slice Selection Assistance Information (NSSAI) that is supplied to the device, and may also be used to allocate an appropriate AMF if the current AMF is not able to support all network slice instances for a given device. The NSSF plays a key role in ensuring that devices are able to access the network slices that are best suited to their needs and requirements. [28]

- The Unified Data Repository (UDR) is a centralized repository of subscriber information that can be used by multiple network functions. For example, the 5G Unified Data Management (UDM) can use the UDR to store and

retrieve subscription data, while the PCF can use the UDR to store and retrieve policy-related data. From a Cellular Internet of Things (CIoT) perspective, the NEF may use the UDR to store subscriber-related data that is permitted to be exposed to third-party applications. The UDR provides a single, unified source of data that can be accessed and used by various network functions, enabling efficient and effective management of subscriber information. [29]

Next, we will analyze some of the key functions within the 5G architecture. These functions play vital roles in enabling the capabilities and features of the 5G network.

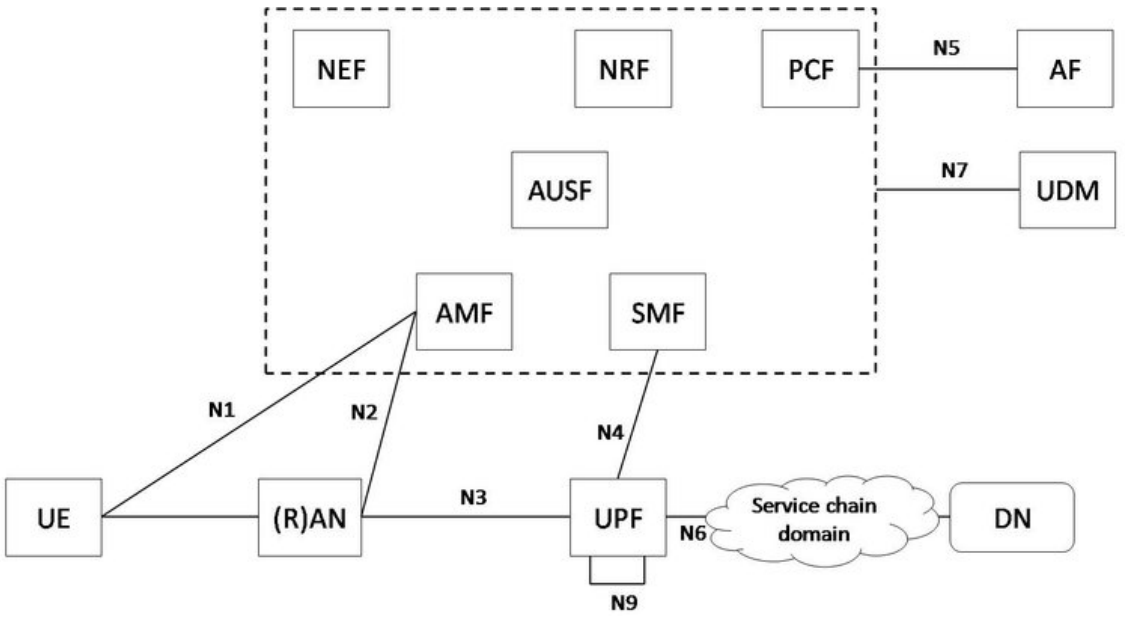


Figure 3.2: 3GPP proposed function architecture and reference points for 5G networks. Source: [30]

**The functions according to figure 3.2 of 5G are as follows:**

- The AMF main tasks include registration management, connection management, reachability management, mobility management, and various functions related to security, access management, and authorization. The AMF plays a key role in ensuring that subscribers can access the network and move around it seamlessly while maintaining secure and authorized access to network resources. [31]
- The Authentication Server Function (AUSF) is a component of the 3GPP 5G architecture. The AUSF is responsible for facilitating security processes in a 5G network. It plays a key role in ensuring that only authorized users are able to access network resources, protecting against unauthorized access and potential security threats. [32]
- The Network Repository Function (NRF) is a key component of the 3GPP SBA for 5G Core networks. The NRF acts as a central services discovery

broker for all NFs in the 5G Core. This allows NFs to discover and access the services they need in order to perform their functions and enables the network to efficiently allocate resources and manage traffic flow. The NRF plays a critical role in ensuring that the 5G Core network functions smoothly and efficiently. [33]

- The PCF is a network component that governs the control plane functions through policy rules and the user plane functions through policy enforcement. The PCF works closely with the Charging Function (CHF) for usage monitoring, allowing operators to manage and govern network behavior. Through the PCF, operators can define and enforce policies that control how traffic flows over the network, allowing for precise control over network behavior and resource allocation. This enables operators to optimize network performance and ensure that subscribers receive the best possible service. [34]

- The Session Management Function (SMF) is a key component of the 3GPP 5G architecture and the next generation core. The SMF is responsible for managing subscriber sessions, including establishing, modifying, and releasing them. This enables subscribers to access network resources and use various services, such as data and voice communications. The SMF plays a critical role in ensuring that subscribers can connect to the network and use its services seamlessly and efficiently. [35]

- The UDM supports the Authentication Credential Repository and Processing Function (ARPF) and stores the long-term security credentials used in authentication for Authentication and Key Agreement (AKA). In addition to storing security credentials, the UDM also stores subscription information for subscribers. This allows the UDM to play a key role in ensuring that subscribers are able to securely and reliably access network resources and services. [36]

- The UPF is a function of the 3GPP 5G architecture. The UPF performs functions similar to those of the Serving Gateway and Packet Gateway in a 4G LTE system. It supports various features and capabilities to facilitate user plane operation, such as packet routing and forwarding, interconnection to the data network, policy enforcement, and data buffering. The UPF plays a critical role in ensuring that user plane traffic is properly managed and routed in the 5G network, enabling subscribers to access network resources and use various services. [37]

- The NEF enables the secure exposure of services and capabilities offered by 3GPP network functions. This function can be used for external or internal exposure/re-exposure, for example by third parties. [38]

- Data Network (DN) refers to services provided by the service provider, Internet access, or services offered by third parties. [39]

- The Application Function (AF) is responsible for controlling the application(s) and may also be involved in the user plane. [19]

- The Network Gateway (NG) is the reference point between the access and core networks and consists of multiple interfaces like N2 and N3. [19]

In 5G, there are two modes of operation: StandAlone (SA) and NSA. The SA mode uses only 5G components, while the NSA mode can use components from 4G/LTE or the EPC in addition to 5G components. The SA architecture utilizes the 5GC and the Next Generation Radio Access Network (NG-RAN) to provide end-to-end connectivity to users. On the other hand, the NSA mode relies on the EPC or 4G/LTE infrastructure to provide connectivity and uses the 5GC and RAN for additional functionality and improved performance. Both modes offer various features and benefits, and the choice of which mode to use depends on the specific requirements and goals of the deployment.

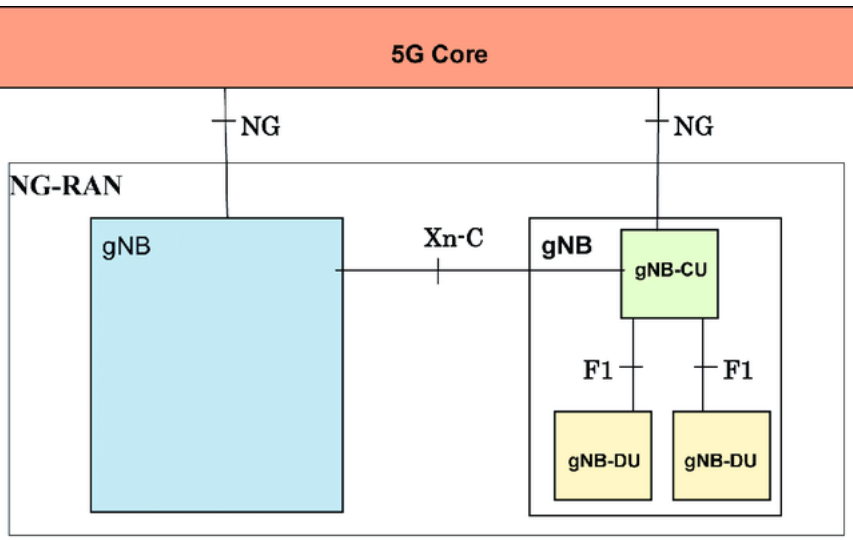


Figure 3.3: Architecture 5G base station gNB. Source: [40]

The other major component that forms the 5G, besides the 5GC in the SA mode, is the NG-RAN. The NG-RAN is the component of the network that provides the wireless connection between the UE and the 5GC as shown in figure 3.3. The NG-RAN is responsible for transmitting and receiving radio signals to and from the UE and for performing radio resource management tasks such as allocation of radio resources and interference management.

The main entity of the NG-RAN is the gNodeB (gNB), which is the radio transmitter and is responsible for transmitting and receiving radio signals to and from the UE. The gNB may be further divided into a gNB-Central Unit (gNB-CU) and one or more gNB-Distributed Units (gNB-DUs) linked by the F1 interface. The F1 interface supports signaling exchange and data transmission between the endpoints, separates Radio Network Layer and Transport Network Layer, and enables the exchange of UE-associated and non-UE-associated signaling.

The NG-RAN is connected to the 5GC via the NG interface, and it uses the 5GC to control the communication between the UE and the data network. The NG-RAN and the 5GC work together to provide various services to the UE, including voice

and data communication, location services, and access to the internet and other data networks.

Finally, the Xn control plane (Xn-C) interface is defined between two NG-RAN nodes. The transport network layer built on this interface is based on the Stream Control Transmission Protocol (SCTP) running on top of IP.

## 3.3 5G Verticals

A 5G vertical refers to a particular industry or market segment that a 5G network is designed to serve. Different industries may have specific requirements for connectivity or capabilities, and telecommunications companies may offer specialized solutions for each vertical to meet these needs. By segmenting the market into different verticals, companies can tailor their products and services to the specific needs of each industry. The main 5G verticals [41] are:

- Automotive: Promoting self-driving vehicles and other transportation innovations, improving safety and reducing congestion.
- Manufacturing: Boosting the use of robotics and automation in manufacturing, improving efficiency and reducing costs.
- Media: Allowing higher-quality streaming of video and other media, as well as the use of virtual and augmented reality in the media industry.
- Energy: Facilitating the use of the IoT in the energy industry, allowing for the remote monitoring and control of power grids and other energy infrastructure.
- eHealth: Allowing remote surgery, telemedicine, and other healthcare services, permitting doctors to treat patients remotely and providing access to healthcare in underserved areas.
- Public Safety: Using faster response times for emergency services, as well as the use of drones and other technologies for search and rescue operations.
- Smart Cities: Enabling the use of the IoT in city infrastructure, such as smart traffic systems and smart lighting, as well as the use of sensors and other technologies to improve city services and livability.

One major contributor that supports the growth and advancement of these industries is the 5G Infrastructure Public Private Partnership (5G-PPP). It is a partnership between the European Union (EU) and the European Information and Communications Technology (ICT) industry, with the goal of promoting the development of 5G technology and ensuring that Europe remains competitive in the global market for telecommunications services. [42]

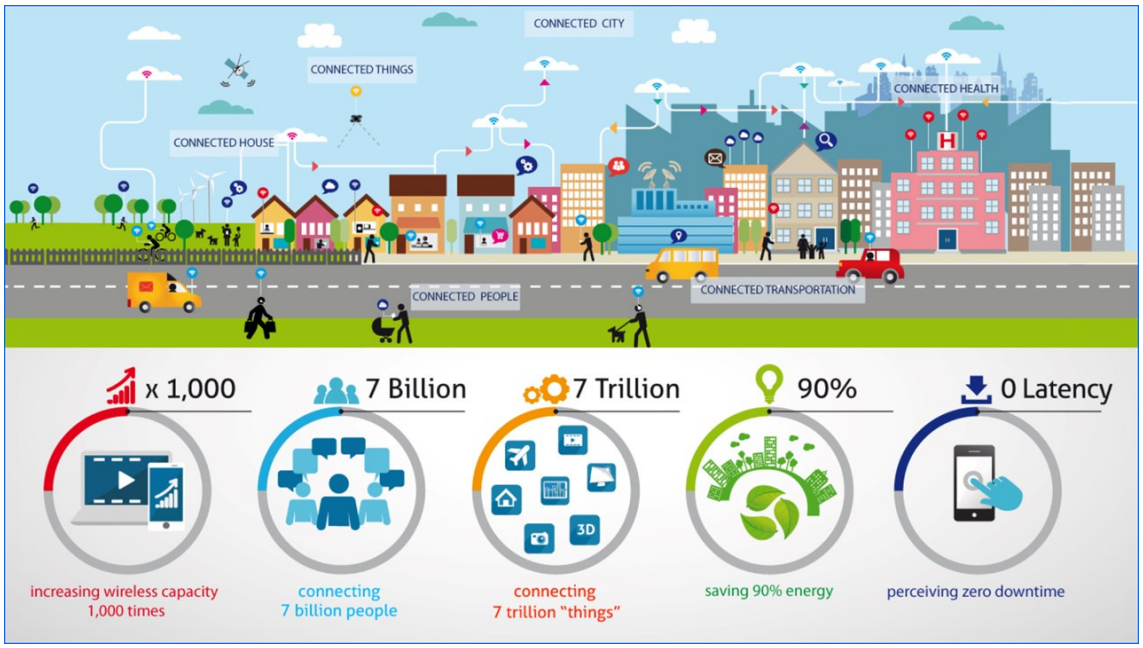


Figure 3.4: 5G PPP goals. Source: [42]

The main objectives are:

- Develop and demonstrate innovative technologies and solutions for 5G networks, including new radio technologies, network architecture, and applications.
- Foster the deployment of 5G networks and services across Europe, by supporting the development of business models and regulatory frameworks that facilitate the deployment of 5G infrastructure.
- Enhance the competitiveness of the European telecommunications industry, by supporting the development of new products and services in the verticals industry.

The 5G European Validation platform for Extensive trials (5G-EVE), 5G Verticals INNovation Infrastructure (5G-VINNI), and 5th Generation End-to-end Network, Experimentation, System Integration, and Showcasing (5GENESIS) infrastructure projects, that belong to the 5G-PPP Phase 3 Part 1 [43], are focused on supporting the deployment and evolution of 5G networks in Europe. These initiatives aim to establish facilities for testing and validating 5G technologies and services and to encourage the participation of vertical industries in the design and evaluation of 5G services. They also seek to demonstrate the value of 5G solutions to the 5G community and promote the adoption of these solutions. The projects will utilize a range of 5G technologies and techniques, such as NFV, network slicing, and automated testing, to verify 5G network Key Performance Indicators (KPIs) under various combinations of technologies and network conditions. They will also provide interfaces and tools for advanced testing and analysis and will strive to create sustainable and accessible facilities for 5G experimentation beyond the duration of the projects.

The 5G-PPP Phase 3 Part 3 [44], initiative aims to conduct advanced 5G validation trials across multiple vertical industries, with the goal of demonstrating the benefits and capabilities of 5G technologies for a wide range of use cases. The initiative includes eight projects, each with its own specific focus and purpose. In terms of projects emphasizing IIoT, we have three prominent examples:

- 5G-SOLUTIONS is a project that aims to demonstrate the benefits of 5G technologies for a range of innovative use cases across five vertical industries: factories of the future, smart energy, smart cities, smart ports, and media and entertainment. It will validate more than 140 performance indicators for 20 use cases and provide technological enablers to facilitate the automation of field trials. The project aims to demonstrate the potential of 5G for improving the efficiency and effectiveness of various industries and sectors.

- 5G-TOURS is a project that aims to demonstrate the benefits of 5G technologies for real users. It will deploy full end-to-end trials for thirteen representative use cases in the areas of e-health, media and broadcast, and mobility. The project will focus on providing efficient and reliable close-to-commercial services that improve the quality of life for citizens and tourists and represent an important business opportunity. The 5G-TOURS network system will integrate strategic components of the ecosystem, including network infrastructure, terminals and end devices, vertical solutions enabled by 5G, and vertical customers receiving the services. The project will evaluate the viability of the use cases through technical performance analysis, economic impact analysis, and customer satisfaction.

- 5G-HEART is a project that aims to demonstrate the benefits of 5G technologies for healthcare, transport, and food production through the validation of 5G KPIs. It will focus on use cases such as colon cancer screening, vital-sign patches, 5G paramedic services in healthcare, autonomous / assisted / remote driving and vehicle data services in transport, and the transformation of the aquaculture sector through 5G in food production. The project will host innovations such as slicing as a service and resource orchestration and will conduct trials at sites in several locations. It involves major vertical players, research / academic institutions, and small and medium-sized enterprises and aims to improve healthcare, public safety, farm management, and business models in the 5G market.

## 3.4 5G Local Area Networks

A 5G Local Area Network (LAN) is a type of private network conceived for dedicated wireless connectivity within a specific area, typically an industrial enterprise [45]. It integrates with an organization existing Information Technology (IT) infrastructure to provide high-speed, predictable wireless connectivity with deterministic performance and latency for mission-critical digital initiatives across

the enterprise. 5G LANs differ from commercial 5G services and Managed Service Providers (MSPs) in that they offer more control and flexibility to the enterprise, such as priority scheduling, resource allocation, and security. Differently from Ethernet, private 5G LANs do not need costly and bulky wired equipment and allow the ability to connect large numbers of devices in a dynamic environment such as Industry 4.0 without the need for specialized cellular network expertise.

Compared with private LTE networks, private 5G networks offer significant advantages in both the radio domain and system architecture. In the radio domain, private 5G networks provide spectrum flexibility, enabling optimal utilization of available frequency bands, multi-Gbps peak data rates for high-speed data transmission, ultra-low latency for applications demanding real-time responsiveness, massive connectivity to accommodate a large number of devices, and ultra-high reliability to ensure mission-critical communications.
At the system level, private 5G networks introduce innovative features that enhance their suitability for various use cases. Vertical network slicing enables the creation of isolated virtual networks, tailored to specific applications or industries, optimizing resource allocation and ensuring consistent performance. Improved security measures, including network isolation, data protection, and device/user authentication, guarantee the confidentiality and integrity of sensitive data, a vital aspect for industries like manufacturing and healthcare. Moreover, private 5G networks support private edge computing, allowing computation and data processing to occur closer to the data source, reducing latency and enabling faster response times.

Private 5G networks simplify complexities related to interference management, localization, and tracking. Within an industrial setting, the simultaneous transmission of signals from controllers to actuators often results in signal interference. This interference adversely impacts both reliability and latency. Consequently, effective interference management assumes paramount significance for private 5G networks. Three distinct techniques for effectively managing interference are:

- The multiple Access technique avoids the reuse of some radio resources by multiple nodes/users using parameters such as time, frequency, code, and spatial resources in a contention-free manner.
- The spread spectrum technique includes Direct Sequence Spread Spectrum (DSSS) and Frequency-Hopping Spread Spectrum (FHSS) to reduce interference. DSSS is used for low-medium narrow-band interference and makes the transmitted signal wider in bandwidth than the information bandwidth. After applying the decoding at the receiver, the information bandwidth is restored while the interference is substantially reduced. By contrast, FHSS reduces the likelihood of colliding with other transmissions via frequency hopping and is preferred for severe interference environments.
- Transmission power control involves dynamic adjustment of the transmit power for managing co-channel interference, reducing energy consumption, and increasing spectral efficiency while ensuring successful communication and maintaining a given QoS.

Private 5G networks effectively address the challenge of accurate localization and tracking by gathering local statistics. They utilize detailed information about the wireless environment within specific areas, to enhance precision in localization and tracking algorithms. Leveraging the advanced attributes of 5G NR, including multiple antennas and wide bandwidth, these networks also employ Radio Frequency-based techniques such as trilateration and triangulation to calculate object positions. The integration of local statistics optimizes these techniques, mitigating signal attenuation, multipath effects, and environmental factors. Moreover, scene analysis techniques, backed by comprehensive local statistics, establish databases of signal fingerprints for accurate location matching. By integrating edge computing capabilities, private 5G networks ensure faster processing of localization data, resulting in reduced latency and improved accuracy in determining object positions for real-time applications involving interactions between the digital and physical domains [45].

The Stand-Alone Deployment represents a type of private network where both the 5GC and Radio Access Network (RAN) are kept private. Illustrated in Figure 3.5, this setup operates independently without reliance on a public network. To access public services, we have a firewall that intercepts the traffic going from outside to inside and vice versa, which we can set up as needed. This model finds relevance in industries and sectors that prioritize high levels of customization, security, and control over their network environment. It is particularly applicable in settings where data privacy, low latency, and tailored network services are essential, such as in industrial automation, manufacturing, healthcare, and critical infrastructure sectors. Importantly, this model is also relevant to our thesis as we explore the deployment and integration of 5G Local Area Networks (LANs) and IIoT use cases.

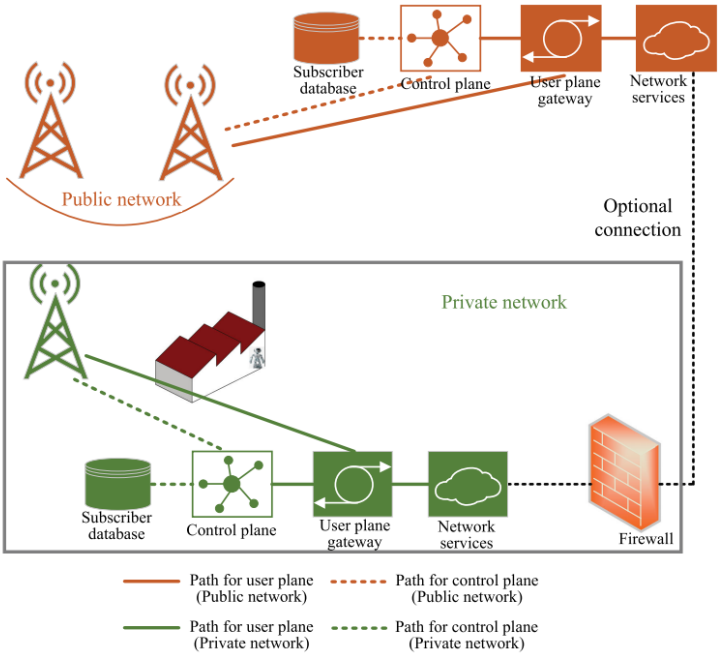


Figure 3.5: Deployment as an isolated network. Source: [45]

Private 5G networks are secure, and fast and can provide reliable voice and data

services across several verticals. Numerous use cases can be supported by these networks. To illustrate their versatility, we explored a selection of examples and live demonstrations.

The manufacturing vertical has the widest range of use cases [46]. Some examples are:

- Production lines can undergo enhancements through multiple routes, encompassing improved flexibility, heightened reliability, minimized latency, and enhanced performance. Given the intricate interplay of control systems and field devices such as sensors, actuators, and robotics within these lines, the demand for robust connectivity and time-sensitive networks is imperative. The establishment of private 5G networks emerges as a critical solution, offering rapid, high-speed, and ultra-low latency connectivity. Leveraging 5G private networks empowers production lines to swiftly adapt to the introduction of new products or the prompt identification and removal of defective items.
- Private 5G technology paves the way for sophisticated logistics management. From finished products to components, assemblies, and supplies, spanning production facilities, supply chains, and warehouses, the integration of cost-effective tracking devices becomes feasible. Through the utilization of the IoT and IIoT technologies, critical information regarding the whereabouts, condition, and environment throughout warehousing, distribution, and circulation processes can be harnessed. This data serves as a foundation for intelligent decision support systems, ultimately enhancing the quality of logistics services while simultaneously reducing costs and resource consumption associated with logistics operations.

Exploring the manufacturing success story of Ericsson and the United States (US) Department of Defense (DoD), respectively:

- Ericsson deployment of a smart factory in Lewisville, Texas, USA [47] using private 5G networks stands out as a beacon of Industry 4.0 progress, recognized by the World Economic Forum. Harnessing the speed and security of 5G connectivity, the facility has pioneered 25 diverse use cases. Notable applications include energy monitoring and management, Augmented Reality (AR) for remote assistance, and Machine Learning (ML)-driven visual inspection.
  For energy monitoring, the factory tracks energy consumption and remotely controls appliances in real-time. AR aids the maintenance team with virtual guidance from global experts for efficient troubleshooting, while ML-powered visual inspection accelerates the process with high-resolution cameras and advanced algorithms.
  This 5G-enabled smart factory boasts 200 operational robots, leading to a remarkable 120% boost in output per employee and a substantial 65% reduction in manual material handling compared to conventional factories. This implementation underscores the factory dedication to efficiency, productivity, and innovative manufacturing practices.

- The US DoD has pioneered a smart warehouse initiative at the Marine Corps Logistics Base situated in Albany, Georgia. Collaborating with Cisco and other technology firms, they harnessed the capabilities of a private 5G wireless network [48]. This advanced network infrastructure utilizes the Citizens Broadband Radio Service (CBRS) and millimeter wave spectrum to power its operations.
  This groundbreaking framework has enabled the deployment of a multitude of intelligent applications. These encompass robotics, barcode scanning, holographic technologies, and AR/Virtual Reality (VR). The integration of these cutting-edge applications has ushered in a new era of operational modernization and heightened efficiency across various facets, including storage, inventory management, maintenance, and auditing. The smart warehouse initiative stands as a testament to the transformative potential of private 5G networks in revolutionizing military logistics and operational processes.

Finally, some healthcare vertical use cases [49]:

- Private 5G technology holds the potential to revolutionize healthcare across multiple dimensions. An illustrative example is its capacity to eliminate the need for cumbersome wires, enabling swift and dependable transmission of sizable medical image files to specialists for assessment. The deployment of high-speed 5G wireless networks contributes to the expansion of the telemedicine arena. This facilitates video consultations between patients at their homes and medical professionals in hospitals, bolstering the accessibility of healthcare services.
  Through the integration of IoT devices, patients' vital signs and health conditions can be collected and relayed in real-time, streamlining rapid medical decision-making by doctors. Within the healthcare sector, Artificial Intelligence (AI) plays a pivotal role in disease recognition and treatment planning. Private 5G networks play a critical role in supporting real-time machine learning, which demands substantial data resources.
  Furthermore, 5G-enabled AR and VR capabilities offer the exciting potential to train medical students through immersive experiences, allowing them to practice surgical procedures within virtual environments. The amalgamation of private 5G technology and healthcare promises a transformative shift in the delivery, accessibility, and advancement of medical services and education.

## 3.5 State of the Art Conclusions

In this chapter, we have provided an all-encompassing overview of 5G technology and its architecture, thereby shedding light on the primary components and their functionalities. This comprehensive insight amplifies our understanding of the underlying mechanisms governing the operation of the 5G network.

The latest milestone of 3GPP, Release 17, brought enhancements such as IoT over NTN, advancements in spectrum utilization, data rates, and overall capacity.

We have delved into the concept of 5G verticals. These delineate distinct industry or market segments that 5G networks are tailored to accommodate, showcasing 5G versatility and ability to cater to specific sector demands. Notably, the pivotal role played by the 5G-PPP in steering the development and widespread deployment of 5G networks and services across Europe is emphasized. Especially, the utilization of 5G testbeds in phase 3 part 1, involving three projects, is highlighted as a significant aspect.

Furthermore, a fundamental concept introduced is that of 5G LANs private cellular networks expressly designed for enterprise applications. Highlighting their features of high performance, mMTC, ultra-low latency, reliability, and customization, we uncover their pivotal role in revolutionizing connectivity within enterprise contexts. The profound synergy between 5G LANs and the IIoT becomes evident, catalyzing transformative advancements. This chapter serves as a gateway to understanding the potential of these technologies, particularly within sectors such as manufacturing and healthcare, where the fusion of seamless connectivity and innovation has the power to reshape industries.

# Chapter 4

# Use Case

We opted to create an Industrial Internet of Things (IIoT) use case scenario, as it encompasses extensive requirements that demand the capabilities of 5G private Local Area Networks (LANs). Also, it is aligned with the objectives of both projects within which this thesis is being conducted. To begin, we will introduce the use case to provide context, followed by listing the requirements to establish a thorough understanding of the essential criteria. Finally, we will present the proposed architecture for the use case.

## 4.1 Definition

This use case directs to a practical scenario that demonstrates the application of 5G network technology within an industrial setting to enable seamless communication, optimize data exchange, and efficient management of interconnected devices and systems. Involves the integration of 5G LANs to provide high-speed, low-latency connectivity, and reliable connectivity in a controlled environment, connecting various PLCs. The PLCs employ the Modbus protocol for communication and operate in a master/slave device topology, with the master accessing registers from the slaves. The overarching objective of this use case is to thoroughly assess the advanced capabilities of private 5G networks within an industrial environment.

## 4.2 Requirements

In this section, we will explore the requirements to achieve the realization of the use case.

Firstly, the descriptions for the terms "must," "should," "could," and "won't" are provided, along with the definitions of functional and non-functional requirements.

| Term | Description |
|---|---|
| Must | Represents a requirement or a technical constraint that is essential and critical to the success of the project. It must be implemented without any exceptions or compromises. |
| Should | Represents a requirement that is highly desirable and recommended for implementation. It is important for achieving the project goals and delivering a satisfactory solution, but there may be justifiable reasons for not fulfilling it in certain situations. |
| Could | Represents a requirement that is optional but desirable. It may enhance the functionality or performance of the system if implemented, but its exclusion would not significantly impact the core objectives. |
| Won't | Represents a requirement that has been deliberately excluded or deemed unnecessary for the project. It is a conscious decision not to pursue or address this requirement, based on factors such as time, resources, or project constraints. |
| **Type** | **Description** |
| Technical Constraint (TC) | Are specific limitations and conditions within a project that arise from the technical aspects of the work. These constraints primarily revolve around the tools, technologies, systems, and methodologies that will be employed during project planning, development, and execution. |
| Functional Requirement (FR) | Represents a requirement that describes the specific behavior or functionality that the system or product must exhibit to meet the needs of the users or stakeholders. It focuses on what the system should do and how it should behave. |
| Non-Functional Requirement (NFR) | Represents a requirement that describes the attributes, characteristics, or qualities of the system or product. It focuses on aspects such as performance, reliability, security, usability, scalability, and other system-wide qualities. |

Table 4.1: Requisites Criteria.

In the following table, the requirements are assigned unique IDs and categorized based on their type (functional or non-functional), priority, and a brief description

is provided for each requirement.

| ID | Name | Type | Priority | Description |
|---|---|---|---|---|
| TC1 | Integrate a 5G core | TC | Must | Integrate a 5G core network into the system to provide the necessary infrastructure and functionalities for managing and routing data between the devices and services within the network. |
| TC2 | Integrate an NG-RAN | TC | Must | Incorporate a Next Generation Radio Access Network (NG-RAN) into the system to enable connectivity between the devices and the 5G core network. |
| F1 | Efficient and Optimized Resource Management | FR | Must | Efficiently manage computing resources within the environment, optimizing resource allocation to minimize waste and maximize performance. |
| F2 | Container Management Through a Graphical Interface | FR | Could | Provide a user-friendly graphical interface for managing containers, simplifying administrative tasks, and monitoring containerized applications. |
| F3 | Ability to Integrate Additional Resources with Minimal Downtime | FR | Should | Seamlessly integrate new resources into the system with minimal downtime, emphasizing high availability and scalability. |

*Continues on the next page*

| ID | Name | Type | Priority | Description |
|---|---|---|---|---|
| F4 | Automatic Management of public IP Addressing for Containers | FR | Should | Implement automated management of public Internet Protocol (IP) addressing for containers to enable external access, prevent conflicts, and streamline network configurations. |
| F5 | Automatic Traffic Management for Containers via a public Domain Name System (DNS) | FR | Could | Implement automatic traffic management for containers using a public DNS, ensuring efficient routing and load distribution. |
| F6 | Persistence of Container Volumes in Case of Termination | FR | Should | Ensure that container volumes are persisted, even if containers are terminated, to prevent data loss and maintain application state. |
| F7 | Creation of Load Balancers Between Containers and Their Replicas Based on CPU Usage | FR | Could | Establish load balancing mechanisms that distribute network traffic intelligently among container replicas based on real-time CPU utilization. |
| F8 | PLCs Execute Simple Processes with Distributed Models, Including Horizontal and Vertical Communication | FR | Must | Programmable Logic Controllers (PLCs) should execute simple processes using a distributed model, allowing for both horizontal (between similar devices) and vertical (between different levels of devices) communication. |

*Continues on the next page*

| ID | Name | Type | Priority | Description |
|---|---|---|---|---|
| F9 | Access to Virtual PLCs Through a Graphical User Interface (GUI) | FR | Could | Provide access to virtual PLCs via a GUI, allowing users to interact with and manage these virtual devices conveniently. |
| F10 | Integration of Containerized PLCs with the Existing 5G Core | FR | Must | Seamlessly integrate containerized PLC instances with the current 5G core infrastructure, ensuring compatibility and efficient operation. |
| F11 | Traffic from PLCs Passes Through the Existing 5G Core | FR | Must | Ensure that data traffic generated by PLCs is routed through the existing 5G core network for communication and processing. |
| F12 | Ability to Read PLC Registers from Any Mobile Station Connected to the 5G Core | FR | Should | Enable the reading of PLC registers from any mobile station that is connected to the 5G core network. |
| F13 | Simulation of Data Layer for the Existing 5G Core | FR | Could | Enable the simulation of the data layer for the existing 5G core network, allowing testing and validation of data handling mechanisms. |
| F14 | Establish a Scenario with a Physical Programmable Logic Controller (PLC) | FR | Must | Create a scenario that includes a physical PLC, allowing for interaction and testing with real-world equipment |

*Continues on the next page*

| ID | Name | Type | Priority | Description |
|---|---|---|---|---|
| NF1 | Performance | NFR | Must | Ensure low latency and high throughput for real-time communication on the 5G network. |
| NF2 | Scalability | NFR | Won't | Support a scalable infrastructure capable of accommodating increasing workload on the 5G network. |

Table 4.2: Requirements to implement the use case.

## 4.3 Use Case Architecture

Derived from the initial use case definition and our specific requirements, an architecture was created to support all the needed features. At its core, a 5G network serves as the primary connectivity infrastructure, offering high-speed and low-latency communication. To integrate PLCs into this network, an NG-RAN serves as an intermediary, facilitating data exchange between devices via the 5G Core network.

PLCs function as controllers, overseeing processes using the Modbus protocol to enable effective communication. To integrate PLC devices with the 5G core, User Equipment (UE) is employed. UEs act as access points to the 5G network. To facilitate the integration between UEs and PLCs, we can use containers that provide network interfaces connected to the 5G core.

Efficient resource utilization is realized through container orchestration, an infrastructure that automates key tasks such as provisioning, deployment, scaling, and management of containerized applications, optimizing resource usage.

To simplify the management of these containerized applications, a graphical interface can be seamlessly integrated, reducing dependency on traditional command-line interfaces and making the entire system more user-friendly.

Moreover, the incorporation of high-availability mechanisms minimizes downtime and ensures that additional resources can be seamlessly integrated into the infrastructure as needed. Persistent volume technology plays a crucial role in safeguarding critical data, offering data preservation even in the event of container failures, thereby ensuring data integrity and availability.

Furthermore, automatic traffic management, employing both IP and DNS, optimizes resource allocation and guarantees uninterrupted connections to containers. Container replication, driven by CPU utilization metrics and efficient load

balancing of network traffic, further enhances resource efficiency and network performance, ensuring that our system operates at peak efficiency.

For simulating industrial environments, a master-slave PLC setup is established, with simple programs running to mimic industrial operations. This setup includes a physical PLC acting as a slave device to exploit real-world industrial scenarios.

The scalability and flexibility offered by the container orchestrator, combined with the ultra-high data rates and ultra-low latency of the 5G network, empower our solution to efficiently accommodate new PLCs, maintaining optimal performance and adaptability. Figure 4.1 depicts this use case scenario.

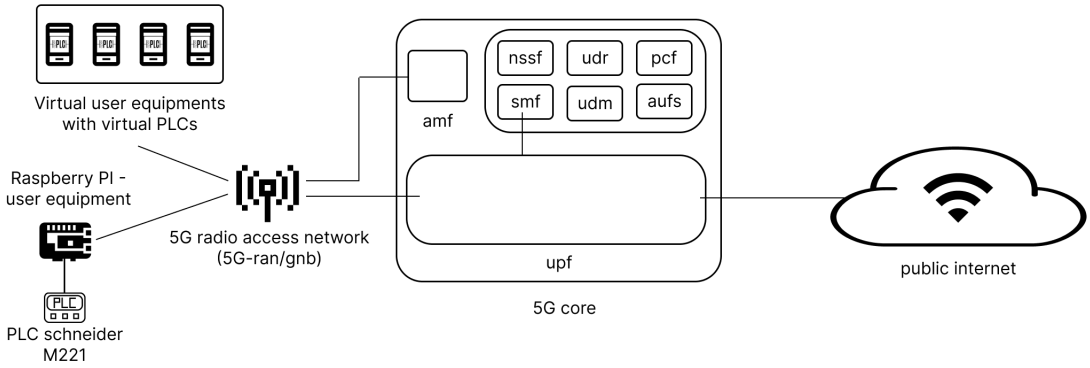


Figure 4.1: IIoT use case scenario. Communication between PLC devices using the 5G network. Adapted from: [50]

This page is intentionally left blank.

# Chapter 5

# Testbed Scenario

In the upcoming sections, we will explore the technologies incorporated into our testbed scenario, selected by members of the Centre for Informatics and Systems of the University of Coimbra (CISUC) laboratory who are actively involved in the Power and Smart5Grid projects. Subsequently, in the architecture section, we will analyze how these technologies seamlessly integrate to construct our testbed. Finally, we will provide detailed insights into the practical implementation of these technologies, including configuration steps complemented in the appendix, and address the scalability requirement. Important to highlight, that this testbed fulfills our specified use case scenario.

## 5.1 Technologies

These technologies were selected based on their utilization by the CISUC laboratory members, who have accumulated knowledge and experience with them. Although no thorough comparison was conducted with other technologies, these chosen technologies demonstrated their suitability for fulfilling our objectives and requirements. Furthermore, the open-source nature of these technologies offers the advantage of customization to our specific needs, potentially resulting in cost savings and reduced vendor lock-in.

### 5.1.1 Resource Managament

This subsection aims to elaborate on the specific technologies employed for resource management. As mentioned earlier, Kubernetes excels in orchestrating containers efficiently, utilizing the resources at our disposal. We will delve into a Kubernetes distribution utilized in our data center, and subsequently explore a server designed to enhance usability through a graphical user interface for the management of the Kubernetes cluster.

#### Rancher Next Generation Distribution

The Rancher Kubernetes Engine 2 (RKE2) [51] is an open-source distribution designed for the deployment and management of Kubernetes (K8s) clusters. Its attributes encompass the seamless integration of resources into clusters, catering to users of varying skill levels. Additionally, RKE2 places a significant emphasis on security [52], guaranteeing data protection and overall cluster integrity. Moreover, RKE2 presence in real-world production environments [53] serves as a testament to its reliability and adaptability for critical use cases, showcasing its prowess in thriving within demanding scenarios.

#### Rancher Server

The Rancher server [54] streamlines Kubernetes cluster management, containers, and applications through a graphical user interface, enhancing user interaction and simplifying deployment and management processes. Additionally, the Rancher server offers resource oversight capabilities, including metrics, logs, and events viewing for individual pods, along with deployment management and scaling.

Moreover, Rancher Server exhibits versatility by managing clusters across diverse platforms such as on-premises, cloud, and hybrid environments. It facilitates multi-cluster management, unifying control, and observation of various clusters within a single interface. Lastly, its integration capabilities extend to third-party services, promoting compatibility with other tools or services.

### 5.1.2 Cluster functionalities

In this subsection, we explore the core cluster functionalities that encompass dynamic storage, ensure high cluster availability, enable auto-scalability based on a diverse set of up to 30 metrics, and present a controller for handling external service traffic.

#### OpenEBS

OpenEBS is a technology that allows dynamic storage of persistent volumes [55], making it a good fit for organizations with varying storage needs. It creates persistent volumes which are important for applications that need to retain data across restarts or failures. The automatic linking of a created volume with a specified application, such as a Database (DB), simplifies the process of setting up and configuring storage. Additionally, allows for flexibility in terms of how data is stored, whether that is distributed among Virtual machines (not advisable) or in a single Virtual Machine (VM). Also, it can be integrated into a Kubernetes-based environment as it is built to work natively with Kubernetes.

#### Kube-vip

Kube-Vip [56] enhances Kubernetes clusters by providing them with a virtual IP and load balancing for the control plane, securing the creation of highly available clusters. Moreover, it facilitates the setup of LoadBalancer-type Kubernetes Services without the need for external hardware or software dependencies. This technology employs a leader-based mechanism within the cluster servers. In the event of the leader failure, Kube-VIP efficiently designates a new leader within approximately 10 seconds, ensuring high availability.

#### KEDA

KEDA [57] is a Kubernetes-based event-driven autoscaler that allows you to scale any container in your cluster based on the number of events that need to be processed. It is a lightweight, single-purpose component that can be integrated into your Kubernetes cluster without replacing or duplicating existing components such as the Horizontal Pod Autoscaler. With KEDA, you can selectively scale certain services based on events while maintaining the stability and security of your other services. It also offers a variety of metrics, including over 30 resource-based metrics, for configuring your scaled objects. KEDA automatically creates a horizontal autoscaler for you once a scaled object has been defined.

#### Traefik

Traefik is an implementation of an Ingress Controller for Kubernetes [58], which orchestrates external access to services by directing incoming traffic according to specified rules, managing domain names, and Secure Sockets Layer (SSL)/Transport Layer Security (TLS) termination. Originally a lightweight reverse proxy, Traefik has evolved to integrate with Kubernetes clusters while remaining compatible with Docker and other interfaces.

The choice for this *ingress controller* was also because the default *nginx-ingress-controller* of the RKE2 cluster has some security flaws [59].

### 5.1.3 5G Network

In this subsection, we delve into the implementations of 5G Core (5GC) and New Radio (NR) distributions within our testbed, alongside a data plane simulator prepared to benchmark our 5G network solution.

#### Open5gs and UERANSIM

Open5GS [60] is an open-source implementation of the 5GC and Evolved Packet Core (EPC) in the C programming language. It is the core network for NR/Long Term Evolution (LTE) and is capable of operating in both StandAlone (SA) and

Non-Standalone (NSA) modes. Currently, it supports 3rd Generation Partnership Project (3GPP) Release 17 with providing 5G Core (AMF, SMF+PGW-c, UPF+PGW-u, PCF, UDR, UDM, AUSF, NRF) network functions and EPC (MME, SGW-c, SGW-u, HSS, and PCRF) network functions. In comparison to other cores (Free5GC and Magma), it is not the most capable [61], but it aligns well with our specific demands. The documentation and community support associated with Open5GS has proven to be valuable assets, even considering the challenges presented by our project.

UERANSIM [62], is the next generation of open-source 5G User Equipment (UE) and Next Generation Radio Access Network (NG-RAN) (gNodeB) implementation. It can be considered as a 5G mobile phone and a base station in basic terms, both running as separate virtualized instances.

The author in [50] showed how to integrate the 5G main components and test the 5G system.

**Simu5g, X11vnc and LXDE**

Simu5G is the evolution of the popular SimuLTE 4G network simulator that incorporates 5G New Radio access. Based on the OMNeT++ framework, it is written in C++ and is fully customizable with a simple pluggable interface. One can also develop new modules implementing new algorithms and protocols.

The idea behind Simu5G [63] is to let researchers simulate and benchmark their 5G network solutions on an easy-to-use framework. It borrows the concept of modularity from OMNeT++ thus it is easy to extend. Moreover, it can be integrated with other modules from the INET Framework. It offers support to optimization tools (e.g. optimization solvers such as CPLEX).
Simu5G is compatible with SimuLTE and allows one to simulate network scenarios where 4G and 5G coexist, in both SA and E-UTRAN New Radio – Dual Connectivity (ENDC) deployments. Furthermore, it inherits SimuLTE compatibility with other OMNeT++-based libraries, for instance, Veins for vehicular mobility. The simulation of the 5G data plane using Simu5G has not been undertaken within the testbed, but it remains a potential route for future exploration and development.

We employ X11VNC as our VNC Server. It allows one to view remotely and interact with real X displays (i.e., a display corresponding to a physical monitor, keyboard, and mouse) with any VNC viewer. While it is not developed any longer by its original author Karl Runge, LibVNC and the GitHub community have taken over the development.
X11vnc does not create an extra display (or X desktop) for remote control. Instead, it shows in real-time the existing X11 display, unlike Xvnc, part of TigerVNC, which is an alternative VNC server available in the official repositories.
Also, note that x11vnc is not shipped with a client viewer. Any VNC viewer should do the job and be compatible with the x11vnc server while not necessarily using all its functionalities. [64]

The Lightweight X11 Desktop Environment (LXDE) is an extremely fast performing and energy saving desktop environment. Maintained by an international community of developers, it comes with a beautiful interface, multi-language support, standard keyboard shortcuts, and additional features like tabbed file browsing. LXDE uses less CPU and less RAM than other environments. It is specially designed for cloud computers with low hardware specifications, such as netbooks, mobile devices (e.g. MIDs), or older computers. [65]

### 5.1.4 Virtual Programmable Logic Controllers software

In this subsection, we address the software employed for the management and communication of virtual Programmable Logic Controllers (PLCs).

#### OpenPLC

Openplc [66] is the first fully functional standardized open-source Programmable Logic Controller (PLC), both in software and hardware. It is mainly used in industrial and home automation, the Internet of Things (IoT), and Supervisory Control and Data Acquisition (SCADA) research, as it carries built-in security features. It also does not require any additional licensing fees and allows for frequent updates and patches. The OpenPLC Project consists of two parts: Runtime and Editor. A Runtime is a portable software designed to run from the smallest of all microcontrollers (Arduino-compatible) to powerful servers in the cloud.
The OpenPLC Editor is the software that runs on your computer and, as mentioned, is used to create your PLC programs. It is very simple to use and supports all languages that are compatible with PLC.

The OpenPLC communicates with other PLCs using a protocol called Modbus that runs over port 502 using Transmission Control Protocol (TCP)/Internet Protocol (IP). When PLCs intercommunicate, one is designated as the master and the other as the slave. The master synchronizes with the slaves by reading and writing specific registers that are configured by the user, such as output registers, input registers, and other types of registers. This allows the PLCs to exchange data and coordinate their actions to control and monitor industrial processes.

OpenPLC offers a web User Interface (UI) where you can insert a program to conduct logical operations on the registers. Alternatively, you can add slave devices to synchronize specific registers between two or more instances. This allows you to tailor the system to your needs and requirements.

## 5.2 Final Architecture

The foundation of this testbed is established upon a Kubernetes cluster, which automates resource management based on predefined rules. Kubernetes is structured with master nodes and worker nodes, representing virtual machines. Mas-

ter nodes serve to integrate new nodes and schedule pods, which are one or multiple containers, onto the worker nodes. These worker nodes leverage their processing power, memory, and disk space to manage the containers. For production-grade Kubernetes cluster distribution, the technology RKE2 is employed. Within the cluster, a Rancher Server, equipped with a graphical interface, is deployed to administer it.

We have the flexibility to manually scale our pods or utilize KEDA, which is integrated into the cluster. KEDA offers automated pod scaling based on a range of 30 metrics. Specifically, in our setup, KEDA manages the scaling of our Rancher server pods by dynamically adjusting their CPU resource allocation as needed, and expanding or reducing resources when necessary.

For ensuring dynamic and persistent volumes for the MongoDB database within the 5G core, OpenEBS was implemented, facilitating automated management of this process. This means that even in the event of the 5G core database encountering issues, the data remains preserved on a node, contributing to data reliability and availability.

To ensure high availability within the cluster, Kube-VIP is implemented. The cluster operates in an Address Resolution Protocol (ARP) mode, wherein a leader (master node) is elected to inherit the virtual IP. The leader role encompasses the smooth integration of new worker nodes without disruptions. The ARP mode ensures a new leader election within seconds in the event of leader failure. This virtual IP serves as the connection point for expanding the cluster with more nodes. Kube-VIP also incorporates a "Load Balancing" mechanism, evenly distributing the workload across all pods and their replicas. This load-balancing property is linked to a set of "public" IPs for external accessibility of the cluster. These public IPs and all the testbed accessibility require a Virtual Private Network (VPN) connection to the data center of CISUC.

There are two ways to manage the cluster: through a VM that is a master node using the Kubectl, the Kubernetes command line, or by connecting to an endpoint on the rancher server as detailed in Appendix B. Figure 5.1 shows what we talked about so far.

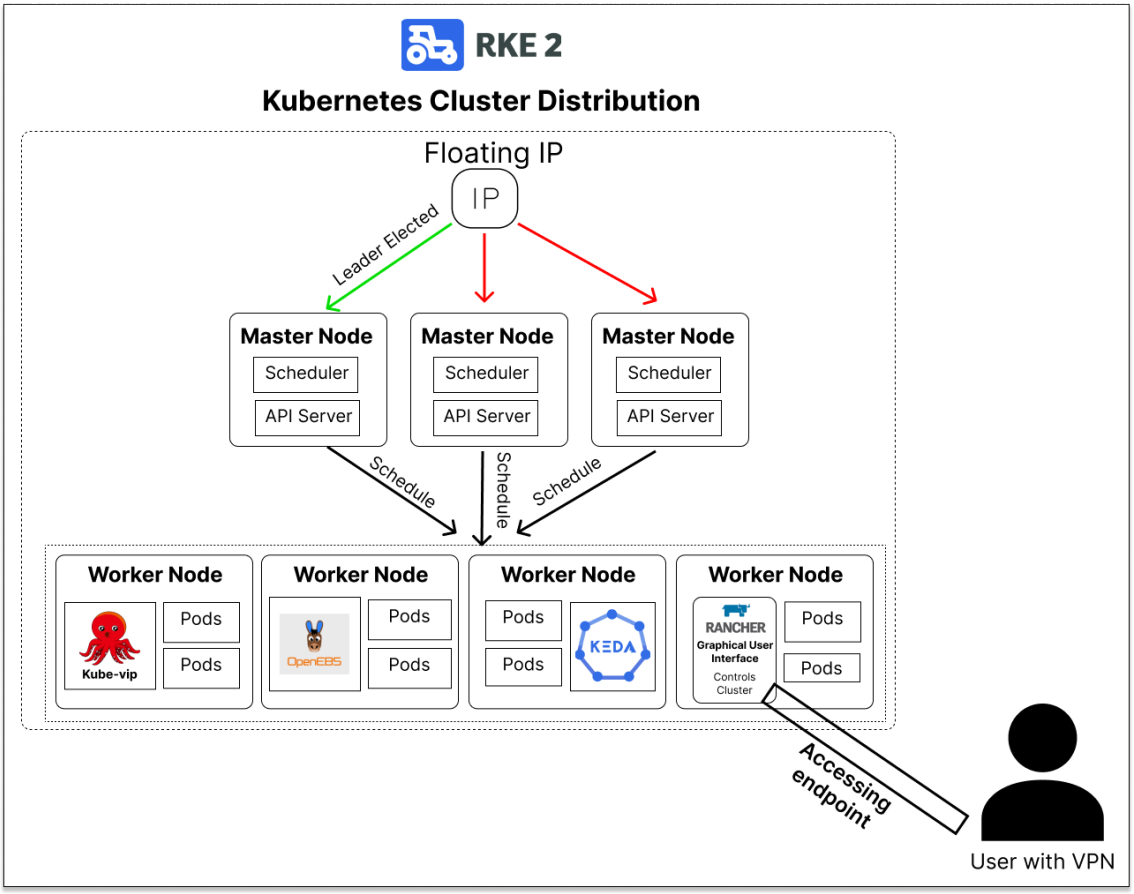


Figure 5.1: Kubernetes cluster diagram with an endpoint in rancher server.

In Kubernetes clusters, services provide a consistent and load-balanced endpoint with static IP addresses to access pods, masking the dynamic nature of pod IPs and facilitating smooth internal communication within the cluster. By default, services are configured with the "ClusterIP" type, allowing connections only within the cluster. Changing the type to "Load Balancing" enables external access by IP address. Services efficiently distribute work among pods and replicas, while offering the flexibility to target one or multiple running pods.

We have integrated an ingress controller using Traefik, facilitating the routing of incoming traffic to designated service endpoints based on their assigned names. This approach enables access to services not only through IP addresses but also via names. The diagram in Figure 5.2 illustrates the utilization of an ingress controller for three services. Although we are not currently using ingress for our services, this functionality is active within the cluster.

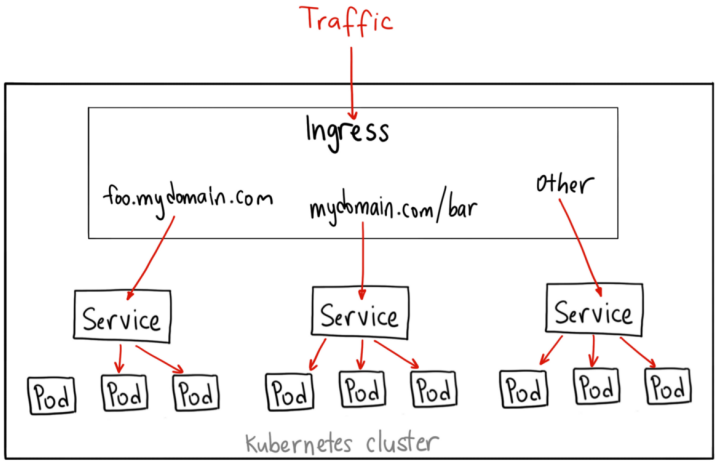


Figure 5.2: Diagram of Kubernetes services using an ingress controller. Source: [67]

Simu5G is normally utilized to assess a 5G network through traffic simulation in the 5G core data plane. This tool offers a graphical interface and is set up within a container containing a Virtual Network Computing (VNC) Server (X11VNC) for graphical access and an LXDE environment for a desktop experience. Despite Simu5G capability to simulate our 5G network data plane with customizable parameters, we did not utilize it for testing in our configuration.

In implementing the 5G core, the Open5GS distribution was employed, comprising a collection of functions deployed within the cluster. To enable user connectivity to the 5G network, UERANSIM, featuring both a NG-RAN function and user equipment function, are deployed within the cluster. Notably, these technologies are generally hosted in separate pods, with the exception of OpenPLC, which requires the user equipment from UERANSIM and is thus co-deployed alongside it. OpenPLC serves as the selected software for managing communications among virtual PLCs. The user equipment delivered by UERANSIM functions as an access point facilitating communication between the PLCs within the 5G network.

Finally, we introduced a physical PLC to simulate real-time scenarios and integrated a Raspberry Pi with user equipment from UERANSIM. This configuration facilitated port forwarding to the physical PLC, enabling access to the 5G network and communication with other PLCs.

Figure 5.3 illustrates the technologies used by adapting the use case scenario.

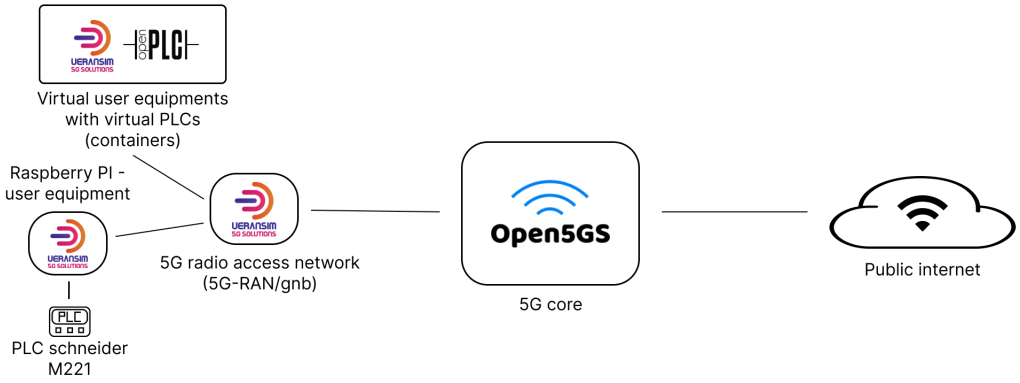


Figure 5.3: Use case scenario with the testbed technologies. Adapted from: [50]

## 5.3 Implementation

Firstly we started by deploying a Kubernetes cluster with 1 master node and to 2 worker nodes. To check its functionality, and how it worked in practical terms. Then we realized that for a cluster to have high availability it has to have at least 3 master nodes. Because having an odd number of master nodes can help with leader selection in the case of machine or zone failure, by preventing ties in the voting process. This is crucial because, in case of a tie, the cluster might experience delays or disruptions in leader selection, potentially leading to reduced availability or inconsistent management decisions.

To provide us with high availability on the cluster and load-balancing services we choose to deploy the Kube-VIP with the RKE2 cluster distribution. To better understand the steps to install the RKE2 we used the information on the official website of the rancher [68] and the tutorial step-by-step of the author in [69]. We needed to adapt the interface, the virtual IP, and the range of IPs for the load-balancing service to our virtual machine and network. In Appendix A.1 we have a detailed explanation of the network topology of our cluster.

There are two ways in which master nodes can operate within a cluster. The default approach involves master nodes managing control plane functions [70] like the scheduler and Application Programming Interface (API) server, in addition to handling pods from various technologies. Alternatively, the approach we employ to distribute load more efficiently involves master nodes solely managing control plane functions, while all other workloads are offloaded to the worker nodes. In Kubernetes, a "taint" refers to the configuration of a node to repel a specific set of pods. To learn about tainting master nodes, please refer to Appendix A.2.1.

To deploy the worker nodes, we followed a tutorial step by step on how to integrate them into the cluster [71]. For our cluster and all the technologies inside it to have enough resources we used 4 worker nodes. Figure 5.1 details the cluster resources used in this testbed. The CPU model utilized is an Intel(R) Xeon(R) Gold 5120 CPU @ 2.20GHz, boasting 56 processors. The hypervisor employed in this setup is VMware ESXi 6.7.0.

| Role | VCPU(s | RAM | Disk |
|---|---|---|---|
| Kubernetes master | 2 | 8 GB | 48 GB |
| Kubernetes master | 2 | 8 GB | 48 GB |
| Kubernetes master | 2 | 8 GB | 48 GB |
| Kubernetes worker | 2 | 8 GB | 48 GB |
| Kubernetes worker | 2 | 8 GB | 48 GB |
| Kubernetes worker | 2 | 8 GB | 48 GB |
| Kubernetes worker | 4 | 8 GB | 48 GB |

Table 5.1: Cluster Resources.

As prerequisites for various technologies used in our setup, we installed Helm, a Kubernetes package manager, directly within the Kubernetes cluster [72]. Helm enables us to efficiently deploy charts, simplifying the management of complex configurations and applications.

After the cluster was operational we deployed the rancher server to manage the cluster through a graphical interface. We can see the detailed installation in Appendix A.2.2.

Then, we proceeded to deploy OpenEBS to facilitate dynamic and persistent volume management for Open5GS MongoDB. Detailed instructions for the installation can be found in Appendix A.2.3.

For the Open5GS implementation, we have opted for an openverso chart, which offers a comprehensive set of 5G functions necessary for both SA and NSA deployment modes [73]. Charts essentially package the required components and offer customizable parameters, simplifying the deployment of technologies within Kubernetes clusters. After evaluating various chart options, we decided to employ openverso charts, which emerged as the most complete and stable option among the alternatives we considered. As a result, we have selected them for deploying both open5GS and UERANSIM.

We deployed Open5GS, creating a dedicated namespace that serves not only the 5G core functions but also the user equipment and radio simulation, ensuring seamless communication between these components. A feature of Open5GS that we utilized is framed routing for user equipment. This functionality allows connectivity from an external N6 network (N6 being the interface between the Data Network (DN) and the User Plane Function (UPF)) to IP networks located within a UE, essentially acting as a router. In our case, we employed framed routing to establish a sub-network among all of our user equipments. Its primary purpose

is to enable devices to utilize the 5G network without requiring a dedicated mobile station to be installed. To implement framed routing, we conducted manual configuration within the UPF container and the open5GS database. For a comprehensive step-by-step guide to configuring open5Gs, refer to Appendix A.2.4.

For deploying the NG-RAN, we opted for the openverso chart, leveraging the synchronized configurations with open5gs, as both components belong to the same chart. Detailed installation steps for the gNodeB (gNB) can be found in Appendix A.2.5.

We integrated KEDA into the cluster with its dedicated namespace. Then, we modified the rancher configurations by allocating a 100mCPU reservation for each pod. This reservation ensures that when scaling pods, Kubernetes selects a worker node with at least 100mCPUs available and assigns it to the pod. Subsequently, we created an auto scaler using KEDA, which dynamically scales the number of pods from a minimum of 1 to a maximum of 5 based on the average CPU usage, targeting an 80% threshold per pod. Detailed installation and configuration steps for KEDA can be found in Appendix A.2.6.

To efficiently manage Simu5G, a desktop environment and a VNC server were essential, enabling a graphical interface for Simu5G. We created an image encompassing these three components and exposed a service to facilitate external access to Simu5G. For more in-depth details, check out Appendix A.2.7.

For the deployment of the OpenPLC software within the 5G network, we created an entry point script responsible for configuring the network interface to utilize the 5G network and initiating the OpenPLC server. Following this, a user equipment configuration file was generated, offering customization based on variables provided during image deployment. Subsequently, we constructed an image encompassing OpenPLC and UERANSIM, which was then pushed to a private repository. To access this image from the cluster, a corresponding secret was established. Finally, the image was deployed within the cluster, exposing the web user interface and Modbus port for external access. For a more detailed explanation, go to Appendix A.2.8.

A Raspberry Pi was integrated into the 5G network, acting as a gateway for the physical PLC. The process began with configuring the VLAN to align with that of the cluster, along with setting up a private network interface for communication with the physical PLC. Following this, we compiled and installed the UERANSIM software, and configured the user equipment with a unique identifier matching the subscription in open5GS. Furthermore, routes for the 5G network interface were configured, alongside the setup for forwarding PLC packets to the physical PLC. For comprehensive step-by-step details, view Appendix A.2.9.

Our physical PLC (Schneider M221) is equipped with an integrated switch that is interconnected and synchronized with the VLAN switch of the cluster. This arrangement is essential for enabling communication between the Raspberry Pi and the physical PLC. Subsequently, we installed the software, as outlined in the appendix A.2.10, and utilized that software to configure the IP address of the PLC to match the sub-network of the Raspberry Pi.

We have implemented a hub-and-spoke scenario, which involves designating one master device responsible for polling registers from various slave devices. To establish this setup, we initially selected a virtual PLC to serve as the master device. Subsequently, we configured all the slave devices with a program designed to perform logical operations on two registers named "slave1" and "parvo." If the value of "slave1" exceeds 10, "parvo" is set to 0. Conversely, if "slave1" is equal to or less than 10, "parvo" becomes 1. The initial value of "slave1" is set to 11. After running all slave devices with this program (excluding the physical PLC), we accessed the web UI of the master device to configure the monitoring of all the slaves by polling the registers used in the program on the virtual PLCs and the discrete input on the physical PLC. For detailed steps and the program logic, consult Appendix A.2.11. Figure 5.4 illustrates a hub-and-spoke example with one master device and two slave devices.

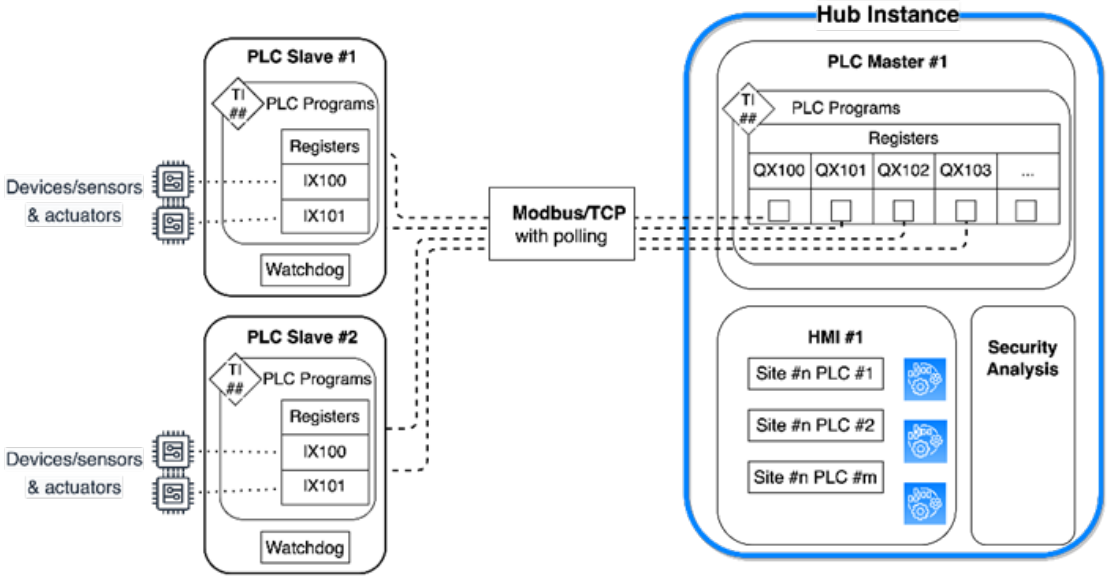


Figure 5.4: Hub and Spoke scenario example with PLCs.

The scalability of the 5G network was a consideration, but due to its complexity and being outside the scope of our objectives, we chose not to pursue it in this implementation. Instead, we engaged with the openverso community contributors to explore the possibility of implementing this feature in the future.
The key functions that require scalability are the AMF, UPF, and SMF, as they handle a significant portion of user traffic. Initially, there is an auto scaler restricting these functions to only one pod, which needs to be removed to allow for scaling. To ensure efficient workload distribution, CPU reservations need to be defined in the cluster for these functions. Determining the appropriate reservations requires a thorough study and analysis of the traffic patterns for these functions.
Downscaling these functions is a challenge due to active user sessions. Further investigation is needed to determine if the sessions can be redirected to a single pod during downscaling to avoid disruption. As a recommendation, the addition of Prometheus, a metric scraper, and the utilization of KEDA for autoscaling based on these metrics could enhance the scalability and resource management of the 5G functions.

## 5.4 Chapter Wrap-up

In this chapter, we have explored the technologies, their interconnection, and their implementation in our testbed scenario. We began by selecting technologies based on the expertise of the CISUC laboratory members, their suitability for our objectives, and their open-source nature, which allows for customization.

We discussed resource management using Kubernetes and the Rancher Server, emphasizing the importance of high availability in our cluster. Cluster functionalities were addressed, highlighting the role of OpenEBS for dynamic storage, Kube-VIP for high availability, KEDA for autoscaling, and Traefik as an Ingress Controller. These components contribute to efficient workload distribution and external service traffic management within our cluster.

In the context of our 5G network, we adopted Open5GS for the 5G core, and UERANSIM for user equipment and radio simulation. Simu5G, X11VNC, and LXDE were integrated to facilitate benchmarking and experimentation within a 5G data plane simulation. We make use of OpenPLC for virtual PLCs alongside a physical Schneider M221 PLC. To facilitate communication between the physical PLC and the 5G network, we have integrated a Raspberry Pi as a communication gateway. This arrangement enables effective communication through the 5G network and satisfies the specific requirements of our hub-and-spoke scenario.

Finally, we have highlighted scalability considerations for future work. As a recommendation for enhancing scalability, we propose the utilization of Prometheus and KEDA to automate the scaling of 5G functions.

This page is intentionally left blank.

# Chapter 6

# Validation of the Use Case

In this chapter, we will address both functional and non-functional requirements. We will begin by presenting our strategy for evaluating the functional requirements and then discuss whether each requirement has been successfully validated, along with the corresponding evaluation procedure. Subsequently, we will turn our attention to the non-functional requirement, which is performance. We will outline our strategy for evaluating performance and provide details about the implementation and results, with an explanation of those results.

It's important to highlight that the technical constraints are validated through the utilization of open5GS for the 5G Core (5GC) and UERANSIM for the Next Generation Radio Access Network (NG-RAN).

## 6.1 Functional Evaluation

To validate the functional requirements, we have devised a strategy that includes various tools and procedures within our 5G network environment. This strategy encompasses the following key elements:

- Traffic Monitoring: We will use tcpdump to capture and analyze traffic on a 5G network interface. This allows us to observe data flows.
- Ping Tests: We will perform ping tests to assess network connectivity between various network components.
- Rancher Web User Interface (UI) Assessment: We will access the Rancher web interface to perform pod deletions, allowing us to observe container management in action.
- Ensuring access to Programmable Logic Controllers (PLCs) from all the user equipments: Using the terminal-based mbpoll tool [74], we will evaluate the ability to read PLC registers of master and slave devices from all the user equipments.

- High Availability Testing: We will disconnect the leader master node and ping the virtual IP to verify the high availability of the cluster.
- Address Linking Examination: We will create a service with a LoadBalancing type to examine if automatic address linking is working correctly.
- Volume Persistence: As part of our volume persistence testing, we will terminate the database pod of open5GS and verify if data persists after the automatic creation of a new one.
- OpenPLC Web UI: We will access the web-based user interface of OpenPLC to make changes to the "slave1" register and confirm that the corresponding "parvo" register on virtual PLCs accurately reflects these changes.
- CPU-Utilization-Driven Auto-Scaling Test: We will increase the traffic load, through a traffic generator called Packet Sender [75], on the Rancher server to verify if the auto-scaler dynamically increases the number of replicas in response to heightened CPU utilization.
- Register Polling: We will check the registers on the master device to ensure it is correctly polling all slave devices using the Modbus client. The Modbus client Graphical User Interface (GUI) installation details can be found in Appendix B.6.

Next, we will address all the functional requirements, presenting validation results along with explanations of the validation process. The requirements will be presented in the format (ID - Name: Validation Status). Our approach leveraged the evaluation strategy to verify these requisites.

**F1 - Efficient and Optimized Resource Management: Validated**
Within the Rancher server, in the "nodes" category, we are able to view the CPU, RAM, and the percentage of pods allocated to each node. In the "pods" category, we can observe the node to which each pod was assigned. To assess resource management, we systematically deleted pods and verified that the allocation process correctly assigned pods to nodes with the most available resources.

**F2 - Container Management Through a Graphical Interface: Validated**
Within the Rancher server, we not only have the capability to delete pods but also to create new ones, allocate them to specific nodes, and make edits as needed. This requirement was successfully validated by deleting a pod through the graphical interface.

**F3 - Ability to Integrate Additional Resources with Minimal Downtime: Validated**
To validate this requirement, we deliberately disconnected the leader node and observed the behavior of the virtual IP. It successfully reestablished connectivity with another master node within approximately 10 seconds. As mentioned earlier, the virtual IP serves as the endpoint for connecting additional resources to the cluster.

**F4 - Automatic Management of public IP Addressing for Containers: Validated**
We configured the Rancher server as a 'LoadBalancer' type and proceeded to

verify its IP address. Subsequently, we attempted to access it from a computer with a Virtual Private Network (VPN) connection to the lab, located outside the cluster. Our successful connection confirmed the validation of this requirement.

**F5 - Automatic Traffic Management for Containers via a public Domain Name System (DNS): Not Validated**
Traefik has been successfully implemented within the cluster. However, validation could not be conducted due to the unavailability of a public DNS for testing purposes. We recommend exploring this aspect as part of future work.

**F6 - Persistence of Container Volumes in Case of Termination: Validated**
We initially created subscribers using the open5gs web UI. Subsequently, we deleted the MongoDB database pod, allowing a new replica to be instantiated. Upon revisiting the open5gs web UI, we confirmed that the subscribers were still present. This successful outcome validates this requirement.

**F7 - Creation of Load Balancers Between Containers and Their Replicas Based on CPU Usage: Validated**
We initially allocated 100mCPU usage for each of the Rancher server containers. Subsequently, we created a scaled object with a utilization threshold set at 80% average usage per container. When a container exceeded the 80% threshold, it could scale up to a maximum of 5 replica containers. To test this, we installed a traffic generator, Packet Sender [75], which stressed the Rancher server's endpoint, thereby activating the auto-scaler. This validation was successful, as the containers scaled up with increased traffic and downscaled when the traffic decreased.

**F8 - PLCs Execute Simple Processes with Distributed Models, Including Horizontal and Vertical Communication**
Following the implementation of the hub-and-spoke scenario, we conducted an assessment using the Modbus client, as described in detail in appendix B.6. Through this client, we continuously monitored the registers that The functionality of logical programs was validated: when a register's value exceeded 10, another register was set to 1, and conversely, when a register's value was lower than or equal to 10, another register was set to 0. Furthermore, we verified that the master device effectively polled specific registers from both virtual PLC slaves and the physical PLC slave, thus validating this requirement.

**F9 - Access to Virtual PLCs Through a GUI: Validated**
The validation process for this requirement involved the deployment of openPLC for virtual PLCs, which provided a web-based UI. Subsequently, we created load balancer services and accessed these virtual PLCs through the web.

**F10 - Integration of Containerized PLCs with the Existing 5G Core: Validated**
After deploying images that included OpenPLC with UERANSIM and exposing OpenPLC as a web UI in all these deployments, we accessed the web UI of the master device. Next, we added the slave device to the configuration. We then ran a tcpdump in the slave device container to confirm that the registers were being polled. This confirmation was based on the observed traffic by the IP address of the User Plane Function (UPF) function on port 502, the Modbus protocol port. Subsequently, we repeated the process on the master device and verified that it

was effectively sending traffic from the 5G network interface to the slave device.

**F11 - Traffic from PLCs Passes Through the Existing 5G Core: Validated**
We employed tcpdump within all containerized PLCs and the Raspberry Pi device to verify that PLC traffic was indeed traversing through the 5G network interface. This confirmation was based on the utilization of port 502, which is the designated port for Modbus protocol traffic, and the IP addresses corresponding to the 5G network.

**F12 - Ability to Read PLC Registers from Any Mobile Station Connected to the 5G Core: Validated**
We installed the mbpoll package on the User Equipments (UEs) and verified that we could access all PLCs from any mobile station. This was achieved through the Modbus terminal client mbpoll, allowing us to read registers with ease.

**F13 - Simulation of Data Layer for the Existing 5G Core: Not Validated**
As this functionality wasn't the core focus of our requirements but rather an additional feature, it was assigned a lower priority, and we didn't have sufficient time to test it. However, it has been implemented within the Kubernetes cluster and can be accessed through a Virtual Network Computing (VNC) viewer, allowing the execution of Simu5G via a graphical interface, as outlined in the tutorial provided in Appendix B.2. We recommend considering validation of this for future work.

**F14 - Establish a Scenario with a Physical PLC: Validated**
After integrating the Raspberry Pi to forward packets to and from the physical PLC, we added the physical PLC as a slave to the virtual master PLC and monitored the polling of its physical input.

## 6.2 Performance Evaluation

For our performance evaluation, we have employed IPERF [76], a versatile tool designed for active measurements of maximum attainable bandwidth in Internet Protocol (IP) networks. IPERF provides the flexibility to fine-tune various parameters related to timing, buffers, and protocols (Transmission Control Protocol (TCP), User Datagram Protocol (UDP), Stream Control Transmission Protocol (SCTP) with support for both IPv4 and IPv6). In each test, IPERF provides essential metrics such as bandwidth, packet loss, and other relevant parameters.

To evaluate the performance of our 5G network, we have devised two types of tests: maximum-rate tests and fixed-rate tests for our 5G core network. For better context on how to perform these tests we utilize the information given by the author in [77].

The maximum-rate tests will involve a single IPERF client and one IPERF server communicating over a single port through protocol TCP. This test is designed to assess the peak capabilities of our 5G core network. We will use only one session because we want the maximum peak rate without concurrency and configure the window to take the maximum link possible. To choose what is the best window

size (8K,16K,64K,128K,256K, and 400K) we will run some preliminary tests, and based on the highest measured bandwidth we will use that window size. We do not go higher than the 400K window because the TCP buffer of the container is restrained to that max window. After that when we run the actual tests we will check the standard deviation, the 95 percent confidence interval, the maximum, the minimum, and the average values of the bit rate.

In the fixed-rate tests, we will deploy 5, 20, and 50 IPERF clients, each communicating with an external IPERF server via different ports through protocol UDP. These clients will be actively operating within the 5G network, allowing us to gauge network performance under varying workloads. We will conduct tests using bitrates of 1Mbps, 5Mbps, and 10Mbps, and analyze the following metrics: standard deviation, 95 percent confidence interval, maximum, and average values of lost packets, as well as the lost packet percentage and average bytes value.

We will start doing 5 tests for each and if the standard deviation is contained it means it's already distributed and there doesn't exist a significant variance so we can stop there. In contrast, if the standard deviation is significantly different from each other we will increase the tests to 10 and recheck the variance. If the standard deviation is still significantly different we will increase it to 30. According to the central theorem, if you take sufficiently large samples from a population, the samples' means will be normally distributed, even if the population is not normally distributed. According to the law of the big number, if the Sample Size is bigger or equal to 30, the shape of the Sample Mean Distribution will be Normally distributed, regardless what the shape of the Population Distribution.

For each one of those tests, we will take 5 minutes. We consider 5 minutes to be sufficient to stabilize communication in controlled environments. We will check the CPU and Memory usage for an existing bottleneck in the 5G network.

To facilitate our tests, we deployed a virtual machine with access to the cluster's VLAN. This virtual machine is equipped with 5 VCPUs and 8GB of memory. The specific IP address for this machine can be found in Appendix A.1. Its primary purpose is to function as our IPERF server, designed to receive communications from various IPERF clients within the cluster, all of which utilize the 5G network and connect through different ports.

To begin, we will provide an overview of the peak rate tests, discussing the results and potential performance enhancements. Following that, we will delve into the fixed rate tests and their outcomes.

### 6.2.1 Peak rate Tests

We initiated our testing phase with the peak rate assessments, aiming to verify the maximum capabilities of our 5G network. To begin, we conducted an evaluation of the optimal window size, a parameter that defines the amount of data transferred before requiring a new communication establishment. Our investigation revealed that window size significantly influences network performance. Through measurements conducted across various window sizes, we observed an

average throughput of 278 Mbits/s using a 400 KiloBytes window size. These findings are summarized in Table 6.1 for reference.

| Window Size (KBytes) | Average Measured Bandwidth (Mbits/s) |
|---|---|
| 8 | 44.3 |
| 16 | 89.2 |
| 64 | 142 |
| 128 | 250 |
| 256 | 263 |
| 400 | 278 |

Table 6.1: Best performance related to window size in TCP.

After selecting the optimal window size, we proceeded to deploy a user equipment and bind the IPERF client to the 5G network interface. Subsequently, we conducted a series of five tests, each lasting for five minutes, and obtained the following results.

| Minimum (Mbits/s) | Average (Mbits/s) | Maximum (Mbits/s) | Standard Deviation | Confidence Interval 95% |
|---|---|---|---|---|
| 202.42 | 307.59 | 394.88 | 4.45 | $\pm$ 0.22 |

Table 6.2: Peak rate test results.

To obtain these results, we configured the IPERF server to output the results to JSON files. We generated a total of five files, each corresponding to one test run. Subsequently, we utilized the script provided in the appendix A.2.12 to merge all these files into a single CSV file containing all the required information (fields detailed in the appendix).

To calculate the standard deviation, minimum value, maximum value, and average value of the bit rate, we imported the data into Google Sheets and employed its built-in formulas to derive these results effectively.

We can check that we had some peaks in the 5 tests but in 95% of the cases, the connections were stable. The data was not significantly spread as demonstrated

by the standard deviation value. We reached a maximum peak of 395 Mbits per second while utilizing an average of 1217 mCPUs for the User Equipments (UEs), 1368 mCPUs for the gNodeB (gNB), and 606 mCPUs for the UPF. We also assessed the node availability, and there are still unallocated VCPUs. Memory usage remained relatively stable, so it was not a concern.

The reason we achieved a maximum peak of approximately 395 Mbits per second is primarily due to resource limitations within our cluster, particularly being CPU-bound. We initially investigated network policies that could potentially restrict network traffic but found none. As mentioned in discussions by users in [78], the open5GS throughput is determined by the single-core CPU master frequency, essentially limited by single-thread performance. This limitation can be mitigated by employing a more powerful CPU. We tested this hypothesis by adding more VCPUs to the master node, bringing the total to 8 VCPUs, which corresponds to a physical core. With this configuration, running UPF, NG-RAN, and UE inside that node, we achieved an average throughput of 410 Mbits per second, providing empirical support for this theory.

Furthermore, we explored the possibility of configuring the user equipment container kernel to optimize for high throughput. However, there are potential limitations in the tun interface and gnodeB of UERANSIM. An alternative approach involves leveraging Vector Packet Processing (VPP)-UPF [79] with Data Plane Development Kit (DPDK) [80] to enhance UPF capabilities. This configuration allows for the processing of multiple packets simultaneously with low latency, thereby accelerating packet processing. The author in [81] has published a step-by-step tutorial on configuring VPP-UPF with DPDK in a virtual environment, and it is compatible with open5GS. This could be considered for future work to further optimize network performance.

Additionally, it's worth noting that the author in [82] achieved an average throughput in the open5GS core of 314.5 Mb per second, demonstrating results similar to our findings.

### 6.2.2 Fixed rate Tests

For the fixed-rate tests, we set up a scenario. This involved the automation of subscriber deployment to the open5GS database, the deployment of multiple user equipments within a single container, and the creation and execution of a script to generate various server processes allocated to different ports within the Virtual Machine (VM) designated for performance tests. These tests were saved in JSON files for later conversion. Additionally, we devised and executed a script that associated multiple IPERF clients with various 5G network interfaces, each running simultaneously. While these scripts were running, we monitored the CPU and memory usage through the master node. In all the cases memory usage remained relatively stable, so it was not a concern. Finally, to consolidate the results, we developed and ran a script to merge all the JSON files into a single CSV file, with the necessary parameters specified. The detailed steps for each of these procedures can be found in appendix A.2.13.

We limited our testing to just five trials for each case due to the lack of disproportionate variance in the results. It did not justify the need for additional testing rounds.

At a 1Mbps data transfer rate, detailed in table 6.3, we observed no packet losses, a common outcome for lower bit rates. Lower bit rates tend to experience fewer packet losses due to their reduced data volume, a behavior consistent with standard network communication. Lower data rates typically encounter less congestion and fewer data packet drops, contributing to this favorable result. Consequently, our analysis focused solely on the volume of bytes transferred. Additionally, we conducted CPU usage monitoring for 5, 20, and 50 user equipments, yielding average values of 202 mCPUs, 436 mCPUs, and 1048 mCPUs, respectively, for the UEs container. For the UPF, the averages were 39 mCPUs, 111 mCPUs, and 213 mCPUs, and for the gNB, the averages were 81 mCPUs, 147 mCPUs, and 291 mCPUs.

We noted that as the number of UEs increased, the data load was more dispersed, a logical consequence of having more UEs. Despite this increased data load, the average number of bytes transferred remained constant across all scenarios. This uniformity suggests that the network adeptly managed the amplified data demands introduced by additional UEs, maintaining a consistent average data transfer rate.

This stability in the average bytes transferred serves as a positive indicator of the network's scalability. It demonstrates the network's ability to gracefully accommodate higher UE loads without compromising the quality of service.

| **Number of UEs** | **Minimum (Bytes)** | **Average (Bytes)** | **Maximum (Bytes)** | **Standard Deviation** | **Confidence Interval 95%** |
|---|---|---|---|---|---|
| 5 | 122668 | 125364 | 126712 | 606.9399 | $\pm$ 13.7352 |
| 20 | 118624 | 125364 | 128060 | 629.5915 | $\pm$ 7.1233 |
| 50 | 121320 | 125364 | 128060 | 631.7676 | $\pm$ 14.2970 |

Table 6.3: Table displaying test results with a fixed rate of 1Mbps for bytes values.

For the 5Mbps data rate, the 5G network demonstrated its capability to handle a maximum of 30 UEs. After we ran the tests with 50 UEs, only 30 user equipment sessions remained. Also, there were no packet losses observed when the UE count was 5, which is why these results were excluded from Table 6.4. Additionally, we conducted CPU usage monitoring for 5, 20, and 30 user equipments, yielding average values of 327 mCPUs, 1184 mCPUs, and 1525 mCPUs, respectively, for the UEs container. For the UPF, the averages were 140 mCPUs, 266 mCPUs, and 267 mCPUs, and for the gNB, the averages were 202 mCPUs, 525 mCPUs, and 535 mCPUs.

Regarding the packet loss tests in table 6.4, with 20 UEs, the network maintained minimal packet loss, registering a maximum of 23 lost packets (4.95%). However, this scenario changed when the UE count was raised to 30. The network experienced an increase in packet loss, reaching a maximum of 70 lost packets (15.18%). This shift implies that the network forms more congestion and data packet drops as the UE count escalates.

In the context of Table 6.5, the analysis of data transfer rates revealed that, with 30 UEs, the standard deviation experienced a significant surge, reaching 15921.6195. The widened confidence interval for bytes transferred with 30 UEs indicates a broader potential range of data transfer values, aligning with the increased standard deviation observed under the higher UE count. However, the network handled the increased data load introduced by additional UEs, as evidenced by the consistent average bytes transferred.

| **Number of UEs** | **Minimum and percentage** | **Average and percentage** | **Maximum and percentage** | **Standard Deviation** | **Confidence Interval 95%** |
|---|---|---|---|---|---|
| 20 | 0 ; 0% | 0 ; 0% | 23 ; 4.95% | 1.6751 | ± 0.0189 |
| 30 | 0 ; 0% | 1 ; 0.21% | 70 ; 15.18% | 4.9210 | ± 0.1070 |

Table 6.4: Table displaying test results with a fixed rate of 5Mbps for lost packets values.

| **Number of UEs** | **Minimum (Bytes)** | **Average (Bytes)** | **Maximum (Bytes)** | **Standard Deviation** | **Confidence Interval 95%** |
|---|---|---|---|---|---|
| 5 | 590424 | 625472 | 659172 | 1367.9250 | ± 30.9214 |
| 20 | 594468 | 625472 | 644344 | 2870.0725 | ± 32.4336 |
| 30 | 478540 | 624124 | 767012 | 15921.6195 | ± 346.1334 |

Table 6.5: Table displaying test results with a fixed rate of 5Mbps for bytes values.

At a data rate of 10Mbps, the 5G network demonstrated its capacity to manage a maximum of 14 UEs. After conducting tests with 50 UEs, only 14 user equipment sessions were sustained. Examining packet loss in Table 6.6, we observe a slightly more noticeable impact and data variance concerning UE count on packet loss metrics. However, the differences were not considerable. Additionally, we conducted CPU usage monitoring for 5, and 14 user equipments, yielding average values of 434 mCPUs, and 910 mCPUs, respectively, for the UEs container.

For the UPF, the averages were 219 mCPUs and 251 mCPUs, and for the gNB, the averages were 278 mCPUs and 396 mCPUs.

Turning to byte transfer, as displayed in Table 6.7, we encountered a notable increase in variance as the UE count grew. Despite this variance, the average bytes transferred remained consistent. This consistency signifies that the network maintained stable communication, even though there were occasional maximum peaks when the UE count was 14.

| Number of UEs | Minimum and percentage | Average and percentage | Maximum and percentage | Standard Deviation | Confidence Interval 95% |
|---|---|---|---|---|---|
| 5 | 0 ; 0% | 0 ; 0% | 67 ; 7.23% | 1.3830 | ± 0.0313 |
| 14 | 0 ; 0% | 0 ; 0% | 76 ; 8.20% | 6.4995 | ± 0.1241 |

Table 6.6: Table displaying test results with a fixed rate of 10Mbps for lost packets values.

| Number of UEs | Minimum (Bytes) | Average (Bytes) | Maximum (Bytes) | Standard Deviation | Confidence Interval 95% |
|---|---|---|---|---|---|
| 5 | 1159280 | 1249596 | 1281948 | 3344.1072 | ± 75.6778 |
| 14 | 1110752 | 1249596 | 1343956 | 12845.3687 | ± 245.3352 |

Table 6.7: Table displaying test results with a fixed rate of 10Mbps for bytes values.

The testbed provides feasible support for low-data-rate Industrial Internet of Things (IIoT) scenarios with multiple nodes. However, it does have limitations when dealing with scenarios that require a high number of UEs and high data rates per UE. Nevertheless, in scenarios with a moderate number of UEs, it remains possible to construct viable use cases.

# Chapter 7

# Conclusions and Future Work

The primary goal of this work was to establish a laboratory environment tailored to support 5G Local Area Networks (LANs) technologies and solutions. This encompassed the implementation of resource orchestration, service virtualization via containers, the integration of a functional 5G Core (5GC), and the deployment of an Next Generation Radio Access Network (NG-RAN). To practically demonstrate the potential of this environment, we designed an Industrial Internet of Things (IIoT) use case. The first step was to acquire background knowledge of relevant domains, including Kubernetes (K8s), IIoT, Programmable Logic Controllers (PLCs), and the Modbus communication protocol. Subsequently, we delved into the realm of 5G technology, to comprehend its architecture and examine its applications across various industry verticals in the context of IIoT. We also conducted an in-depth analysis of 5G LANs, and their applications in different industry verticals, with a particular emphasis on their role in manufacturing and healthcare, and their relevance in the context of IIoT.

The subsequent step involved designing the use case along with its associated requirements and creating a high-level architecture to seamlessly integrate all the components. This architecture comprised several key elements, starting with a container orchestrator serving as the foundational infrastructure for efficient container management. Additionally, it featured essential 5G network components, including 5G functions and an NG-RAN. Within the industrial context, PLCs were employed, communicating via the Modbus protocol and executing a straightforward logic program within a master-to-slave topology. To mimic real-world scenarios, a physical PLC was also incorporated. Crucially, all data traffic for these PLCs was routed through the 5G network.

We then proceeded to translate the defined use case and its prerequisites into a tangible testbed scenario. Initially, we scrutinized various technologies to identify if they aligned with our requirements. The selection process was made by the collective expertise of the Centre for Informatics and Systems of the University of Coimbra (CISUC) laboratory team, who had prior experience with these technologies.

Subsequently, we established a comprehensive architectural framework to interconnect these chosen components. This architecture encompassed several inte-

gral components, including Rancher Kubernetes Engine 2 (RKE2), a distribution of K8s, utilized for container orchestration. For the NG-RAN and user equipment provisioning, we employed UERANSIM. Open5GS was deployed as the core element responsible for handling the 5GC, while OpenPLC emulated the PLCs. For persistent volume management, OpenEBS was employed, while Kube-VIP ensured high availability of the RKE2 through a virtual IP, and also served for automatically assigning public IPs to services. To dynamically scale containers based on CPU metrics, we integrated KEDA into the setup. Simu5G was for simulating the data plane of the 5G network. Lastly, Traefik was for automating traffic management via Domain Name System (DNS).

Additionally, we incorporated a Raspberry Pi into the configuration, tasked with serving as a traffic forwarder and a 5G network gateway for the physical PLC.

In the concluding phase of establishing the testbed scenario, we provided a comprehensive account of the implementation process, complete with detailed steps and configuration files, which can be referenced in the appendix. It is worth noting that during this implementation, we contemplated the possibility of adding auto scalability to the 5G network. However, given the intricacies involved and the fact that it fell beyond the scope of our primary objectives, we opted not to pursue its implementation.

Following the successful implementation of the testbed scenario, our next objective was to validate both the functional and performance requirements of the use case. In terms of functional requirements, we validated the core ones and only two extra functionalities were not validated, due to time restrictions and as they were not deemed impactful in achieving our core objectives.

Moving on to performance validation, we conducted peak rate tests, which yielded a maximum throughput of 395 Mbits per second. However, it's important to note that this peak rate was constrained by the limited resources available in our environment. Additionally, we identified a limitation inherent in Open5GS, where throughput is primarily determined by the single-core CPU master frequency, essentially being limited by single-thread performance.

Subsequently, we conducted fixed-rate tests and arrived at the conclusion that our testbed demonstrates robust support for low-data-rate IIoT scenarios involving multiple nodes. Nonetheless, it's crucial to acknowledge that the testbed does exhibit certain limitations when dealing with scenarios demanding a high number of User Equipments (UEs) and high data rates per UE. Nevertheless, in scenarios featuring a moderate number of UEs, the testbed proves capable of constructing viable use cases.

As this research work is an integral part of the POWER and Smart5Grid projects, it serves as the foundational building block for the development of larger and more advanced testbeds, fully harnessing the potential of 5G LANs. In terms of future work, there are several promising routes to explore.

Firstly, we can continue to enhance our testbed by validating the remaining two functional requirements. This includes validation of Simu5G for simulating the 5G network's data plane and evaluating Traefik with a public DNS to optimize

traffic management.

Secondly, addressing non-functional requirements, we can focus on implementing scalability measures. This involves a comprehensive study of the Access and Mobility Management Function (AMF), User Plane Function (UPF), and Service Mobility Function (SMF) functions, exploring strategies for replicating these critical components as they handle a significant portion of user traffic. Additionally, we can leverage Prometheus, a metric scraper, in combination with KEDA for auto-scaling based on metrics, ensuring efficient resource allocation.

Lastly, we have the opportunity to enhance performance through the adoption of technologies like Vector Packet Processing (VPP) and Data Plane Development Kit (DPDK), which accelerate packet processing within the UPF. These improvements can further optimize data transfer rates and overall network capacity.

This page is intentionally left blank.

# References


[1] CISUC. *The POWER project*. Aug. 2023. URL: https://www.cisuc.uc.pt/en/projects/power.

[2] CISUC. *The Smart5Grid project*. Aug. 2023. URL: https://www.cisuc.uc.pt/en/projects/smart5grid-automated-5g-networks-and-services-for-smart-grids.

[3] Diogo Cruz. *Repository composed by UERANSIM v2.2.6 and OpenPLC V3 to run in a Kubernetes environment*. Dec. 2022. URL: https://github.com/DiogoCruz40/UERANSIM_with_OpenPLC.

[4] Diogo Cruz et al. "Designing a high-fidelity testbed for 5G-based industrial IOT". In: *Proceedings of the 22nd European Conference on Cyber Warfare and Security (ECCWS 2023), Athens, Greece* (June 2023). DOI: 10.34190/eccws.22.1.1204.

[5] *33o Seminary of Mobile Communications Thematic Network*. Feb. 2023. URL: https://rtcm.inesctec.pt/33o-seminario/.

[6] CISUC. Aug. 2023. URL: https://www.cisuc.uc.pt/en/workshop-on-5g-and-beyond.

[7] *Kubernetes Documentation*. Dec. 2022. URL: https://kubernetes.io/docs/home/.

[8] *PLC*. Jan. 2023. URL: https://en.wikipedia.org/wiki/Programmable_logic_controller.

[9] Ramakrishnan Ramanathan. "The IEC 61131-3 programming languages features for industrial control systems". In: *2014 World Automation Congress (WAC)*. Jan. 2023, pp. 598–603. DOI: 10.1109/WAC.2014.6936062.

[10] *PLC Diagram*. Jan. 2023. URL: https://www.daenotes.com/electronics/industrial-electronics/PLC-programable-logic-control.

[11] *The Modbus Official Site*. Dec. 2022. URL: https://modbus.org/.

[12] *RS232 Serial Communication Protocol: Basics, Working & Specifications*. Jan. 2023. URL: https://circuitdigest.com/article/rs232-serial-communication-protocol-basics-specifications.

[13] *RS-485 Serial Interface Explained*. Jan. 2023. URL: https://www.cuidevices.com/blog/rs-485-serial-interface-explained.

[14] *Man in the middle (MitM) attack*. Jan. 2023. URL: https://www.imperva.com/learn/application-security/man-in-the-middle-attack-mitm/.

[15] *What Are Iot and IIoT?* Jan. 2023. URL: https://aws.amazon.com/what-is/iot/?nc1=h_ls.

[16] *3GPP - Release 16*. Jan. 2023. URL: https://www.3gpp.org/specifications-technologies/releases/release-16.

[17] *3GPP - Release 17*. June 2023. URL: https://www.3gpp.org/specifications-technologies/releases/release-17.

[18] *3GPP - Release 18*. June 2023. URL: https://www.3gpp.org/specifications-technologies/releases/release-18.

[19] *3GPP, A 5G analysis*. Dec. 2022. URL: https://www.3gpp.org/technologies/5g-system-overview.

[20] *What is home subscriber server (HSS)?* Dec. 2022. URL: https://www.dialogic.com/glossary/home-subscriber-server-hss.

[21] *Mobility Management Entity(MME)*. Dec. 2022. URL: https://www.iplook.com/products/epc-mme.

[22] *Policy and charging rules function(pcrf)*. Dec. 2022. URL: https://www.iplook.com/products/epc-pcrf.

[23] *PGW-C - PDN Gateway Control Plane Function*. Nov. 2022. URL: https://www.mpirical.com/glossary/pgw-c-pdn-gateway-control-plane-function.

[24] *PGW-u - PDN gateway user plane function*. Nov. 2022. URL: https://www.mpirical.com/glossary/pgw-u-pdn-gateway-user-plane-function.

[25] *SGW-C - serving Gateway Control Plane Function*. Nov. 2022. URL: https://www.mpirical.com/glossary/sgw-c-serving-gateway-control-plane-function.

[26] *SGW-u - serving Gateway User Plane Function*. Nov. 2022. URL: https://www.mpirical.com/glossary/sgw-u-serving-gateway-user-plane-function.

[27] *5G binding support function (BSF) data sheet page*. July 2022. URL: https://titaniumplatform.com/5g-binding-supportfunction-bsf-data-sheet-page/.

[28] *NSSF - network slice selection function*. Nov. 2022. URL: https://www.mpirical.com/glossary/nssf-network-slice-selection-function.

[29] *UDR - Unified Data Repository*. Nov. 2022. URL: https://www.mpirical.com/glossary/udr-unified-data-repository.

[30] *3GPP proposed architecture and reference points for 5G networks*. Dec. 2022. URL: https://www.researchgate.net/figure/The-3GPP-5G-architecture-adapted-from-6_fig1_327635348.

[31] *AMF - access and mobility management function*. Nov. 2022. URL: https://www.mpirical.com/glossary/amf-access-and-mobility-management-function.

[32] *Ausf - authentication server function*. Nov. 2022. URL: https://www.mpirical.com/glossary/ausf-authentication-server-function.

[33] Derek Cheung. *5G core network repository function (NRF)*. Nov. 2022. URL: https://derekcheung.medium.com/5g-core-part-5-network-repository-function-nrf-5dd65afc6f12.

[34] *PCF - policy control function*. Nov. 2022. URL: https://www.mpirical.com/glossary/pcf-policy-control-function.

[35] *SMF - session management function*. Nov. 2022. URL: https://www.mpirical.com/glossary/smf-session-management-function.

[36] *UDM - Unified Data Management*. Nov. 2022. URL: https://www.mpirical.com/glossary/udm-unified-data-management.

[37] *UPF - user plane function*. Nov. 2022. URL: https://www.mpirical.com/glossary/upf-user-plane-function.

[38] *NEF - Network Exposure Function*. Jan. 2023. URL: https://www.mpirical.com/glossary/nef-network-exposure-function.

[39] *DN - Data Network*. Jan. 2023. URL: https://www.mpirical.com/glossary/dn-data-network.

[40] *Architecture 5G base station gNB*. Dec. 2022. URL: https://www.researchgate.net/figure/Architecture-5G-base-station-gNB_fig5_339059845.

[41] *5G And Verticals*. Jan. 2023. URL: https://5g-ppp.eu/verticals/.

[42] *5G PPP Website*. Jan. 2023. URL: https://5g-ppp.eu/.

[43] *5G PPP Phase 3, Part 1: Infrastructure Projects*. Jan. 2023. URL: https://5g-ppp.eu/5g-ppp-phase-3-1-projects/.

[44] *5G PPP Phase 3, Part 3: Advanced 5G validation trials across multiple vertical industries*. Jan. 2023. URL: https://5g-ppp.eu/5g-ppp-phase-3-3-projects/.

[45] Miaowen Wen et al. "Private 5G Networks: Concepts, Architectures, and Research Landscape". In: *IEEE Journal of Selected Topics in Signal Processing* 16.1 (Aug. 2023), pp. 7–25. DOI: 10.1109/JSTSP.2021.3137669.

[46] GSMA. *5G Private and Dedicated Networks for Industry 4.0*. Aug. 2023. URL: https://www.gsma.com/iot/resources/5g-private-npn-industry40/.

[47] Ericsson. *Ericsson's USA 5G Smart Factory*. Aug. 2023. URL: https://www.ericsson.com/en/about-us/company-facts/ericsson-worldwide/united-states/5g-smart-factory.

[48] US DoD. *U.S. Department of Defense's 5G Smart Warehouses*. Aug. 2023. URL: https://www.defense.gov/News/Releases/Release/Article/2650242/department-of-defense-successfully-demonstrates-a-5g-network-for-smart-warehous/.

[49] Accedian. *Private 5G answers Healthcare's urgent call for digital ... - accedian*. Aug. 2023. URL: https://accedian.com/wp-content/uploads/2022/03/Accedian_private-5G_answers-urgent-call-digital-transformation_whitepaper.pdf.

[50] $\lambda$.eranga. *Deploying 5G core network with Open5GS and UERANSIM*. Feb. 2022. URL: https://medium.com/rahasak/5g-core-network-setup-with-open5gs-and-ueransim-cd0e77025fd7.

[51] SUSE Rancher. *Rancher's next-generation Kubernetes distribution*. Dec. 2022. URL: https://docs.rke2.io/.

[52] SUSE Communities. *When to use K3s and RKE2*. Aug. 2023. URL: https://www.suse.com/c/rancher_blog/when-to-use-k3s-and-rke2/.

[53] Don Poole. *Rancher government solutions announces fully supported no-code deployment of rancher on AWS GovCloud*. Aug. 2023. URL: https://finance.yahoo.com/news/rancher-government-solutions-announces-fully-091700764.html.

[54] *Rancher Server*. Dec. 2021. URL: https://rancher.com/docs/rancher/v2.6/en/.

[55] *OpenEBS documentation: Openebs docs*. Apr. 2022. URL: https://openebs.io/docs/.

[56] *Kube-vip*. Apr. 2022. URL: https://kube-vip.chipzoller.dev/docs/.

[57] Kedaorg. *Keda*. Apr. 2022. URL: https://keda.sh/.

[58] Will Ho. *Kubernetes Ingress Controllers: Why I chose Traefik*. Apr. 2022. URL: https://ikarus.sg/why-traefik-ingress-controller/.

[59] *Nginx ingress - insecure*. Feb. 2022. URL: https://gist.github.com/oskapt/18cae046c99c60a3cb2eeaa72a2ad1cc.

[60] Sukchan Lee. *Open5gs - Documentation*. Jan. 2022. URL: https://open5gs.org/open5gs/docs/.

[61] Francisco Neto et al. "Analysis for Comparison of Framework for 5G Core Implementation". In: Aug. 2023, pp. 1–5. DOI: 10.1109/ICISCT52966.2021.9670414.

[62] Aligungr. *Ueransim - Github*. Feb. 2022. URL: https://github.com/aligungr/UERANSIM.

[63] Simu5G is the result of a joint research project carried out by Intel Corporation and the Computer Networking Group of the University of Pisa. *SIMU5G*. Aug. 2022. URL: http://simu5g.org/.

[64] *Virtual Network Computing - x11*. Aug. 2022. URL: https://tigervnc.org/doc/Xvnc.html.

[65] *Desktop Environment - lxde*. Aug. 2022. URL: http://www.lxde.org/.

[66] *Open-source PLC software*. Dec. 2022. URL: https://openplcproject.com/.

[67] Suleiman Abualrob. *Kubernetes: Expose pod externally bypass service load balancing*. Feb. 2022. URL: https://medium.com/@suleimanabualrob/kubernetes-expose-pod-externally-bypass-service-load-balancing-bf89038afee2.

[68] *Setting up a high-availability RKE2 Kubernetes cluster for rancher*. Aug. 2023. URL: https://ranchermanager.docs.rancher.com/v2.6/how-to-guides/new-user-guides/kubernetes-cluster-setup/rke2-for-rancher.

[69] Adrian Goins. *Configuration Step by step of RKE2 with Kube-VIP*. Aug. 2023. URL: https://gitlab.com/monachus/channel/-/tree/master/resources/2021-09-07-ha-rke2-kube-vip-rancher.

[70] *Control Plane components of Kubernetes*. Aug. 2023. URL: https://kubernetes.io/docs/concepts/overview/components/.

[71] *Installation step by step of Kubernetes worker nodes*. Aug. 2023. URL: https://docs.rke2.io/install/quickstart/#linux-agent-worker-node-installation.

[72] *Installing Helm*. Aug. 2023. URL: https://helm.sh/docs/intro/install/.

[73] *Openverso Charts for Kubernetes*. Aug. 2023. URL: https://github.com/Gradiant/openverso-charts.

[74] *Mbpoll - Usage and Installation*. Aug. 2023. URL: https://manpages.ubuntu.com/manpages/lunar/man1/mbpoll.1.html.

[75] Jon Dugan and et al. *Packet Sender - Send and receive TCP, UDP, SSL. HTTP Requests*. Aug. 2023. URL: https://packetsender.com/.

[76] Jon Dugan and et al. *What is IPERF?* Aug. 2023. URL: https://iperf.fr/.

[77] Tom Fentom. *Using iPerf to Baseline Network Performance*. Aug. 2023. URL: https://www.controlup.com/resources/blog/entry/using-iperf-to-baseline-network-performance/.

[78] *Open5gs Maximum Throughput only 500Mbps - Discussion*. Aug. 2023. URL: https://github.com/open5gs/open5gs/discussions/2208.

[79] Ke-liang DU et al. "High-Performance UPF Prototype Based on VPP". In: *Journal of Beijing University of Posts and Telecommunications* 44.6, 89 (Aug. 2023), pp. 89–95.

[80] Haoran Zhang, Zikang Chen, and Yang Yuan. "High-Performance UPF Design Based on DPDK". In: *2021 IEEE 21st International Conference on Communication Technology (ICCT)*. Aug. 2023, pp. 349–354. DOI: 10.1109/ICCT52962.2021.9657903.

[81] *Installation step by step of VPP-UPF with DPDK*. Aug. 2023. URL: https://github.com/s5uishida/install_vpp_upf_dpdk.

[82] Gabriel Lando. *Uma avaliação de desempenho de implementações open source em software de núcleos de rede 5G*. Sept. 2023. URL: https://lume.ufrgs.br/bitstream/handle/10183/254479/001161325.pdf?sequence=1.

[83] *Taint master nodes to only handle control plane functions*. Aug. 2023. URL: https://stackoverflow.com/questions/55191980/remove-node-role-kubernetes-io-masternoschedule-taint.

[84] *Taint on Kubernetes*. Aug. 2023. URL: https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/.

[85] Rancher. *Install/upgrade rancher on a kubernetes cluster*. Aug. 2023. URL: https://rancher.com/docs/rancher/v2.6/en/installation/install-rancher-on-k8s/.

[86] *Rancher Technical Items*. Aug. 2023. URL: https://ranchermanager.docs.rancher.com/v2.6/faq/technical-items.

[87] *Mongodb compass installation*. Aug. 2023. URL: https://www.mongodb.com/try/download/compass.

[88] s5uishida. *S5uishida Open5gs 5GC and UERANSIM UE / ran sample configuration - framed routing*. Aug. 2023. URL: https://github.com/s5uishida/open5gs_5gc_ueransim_framed_routing_sample_config.

[89] *Scale applications based on CPU metrics - KEDA*. Aug. 2023. URL: https://keda.sh/docs/2.10/scalers/cpu/.

[90] *Simu5G image in docker hub*. Aug. 2023. URL: https://hub.docker.com/r/diogocruz40/simu5g.

[91] thiagoralves. *OpenPLC GitHub repository*. Aug. 2023. URL: https://github.com/thiagoralves/OpenPLC_v3.

[92] thiagoralves. *Installation of EcoStruxure Machine Expert Basic*. Aug. 2023. URL: https://www.se.com/in/en/download/document/Machine_Expert_Basic_V1_2_SP1/.

[93] *Open5gs-dbctl script to update database*. Sept. 2023. URL: https://github.com/open5gs/open5gs/blob/main/misc/db/open5gs-dbctl.

[94] *Download VNC Viewer*. Aug. 2022. URL: https://www.realvnc.com/pt/connect/download/viewer/.

[95] CharlesPandian. *Simulation of 5G networks under omnet++ and Simu5g simulator*. Aug. 2022. URL: https://www.projectguideline.com/simulation-of-5g-networks-under-omnet-and-simu5g-simulator/.

[96] *Modbus Tools*. Dec. 2022. URL: https://www.modbustools.com/modbus_poll.html.

[97] *OpenPLC project - Modbus addressing*. Aug. 2023. URL: https://openplcproject.com/docs/2-5-modbus-addressing/.

# Appendices

This page is intentionally left blank.

# Appendix A

# Network Topology and Configurations

This appendix covers the network topology involving the Kubernetes cluster, Raspberry Pi, Virtual Machine (VM) for performance tests and physical PLC. It also outlines the configurations used to implement the testbed scenario and validate the use case requirements.

## A.1 Network topology

In our network setup, we leverage Virtual LAN 223 via a virtual switch, as illustrated in Figure A.1. This arrangement is established within the Centre for Informatics and Systems of the University of Coimbra (CISUC) data center and is managed by a type 1 hypervisor (VMware ESXi). This hypervisor takes care of resource allocation, management of Virtual Machines, and network configurations.

At the moment there are 7 VMs in this group, three of them correspond to the master nodes and have fixed IP addresses, **172.27.223.12**, **172.27.223.13** and **172.27.223.14**. These are responsible for managing the cluster, and resources and assigning tasks to the agent nodes (workers).

Agent nodes (workers) have dynamic IP addresses in the range of **172.27.223.200 - 172.27.223.253** because they don't need to have static IPs, as they can be dynamically scaled up or down. These connect to the cluster from the virtual IP that makes the master nodes in High Availability (HA) the **172.27.223.50**.

The Raspberry Pi is assigned two static IP addresses: one from the VLAN 223 (**172.27.223.20**), and another private IP (**192.168.10.99**) used for connecting to the physical Programmable Logic Controller (PLC). The physical PLC, on the other hand, has only one IP address, which is the private IP (**192.168.10.100**) used to connect to the Raspberry Pi.

VM 8 is dedicated to conducting tests on the 5G network. It operates outside the

cluster and functions as an IPERF server [76] for performing active bandwidth measurements on IP addresses. This VM is assigned the VLAN 223 IP address of **172.27.223.228**, which is dynamically assigned by the Dynamic Host Configuration Protocol (DHCP) server.

The range of **172.27.223.51 - 172.27.223.60** is being used to host services automatically through the load balancing type.

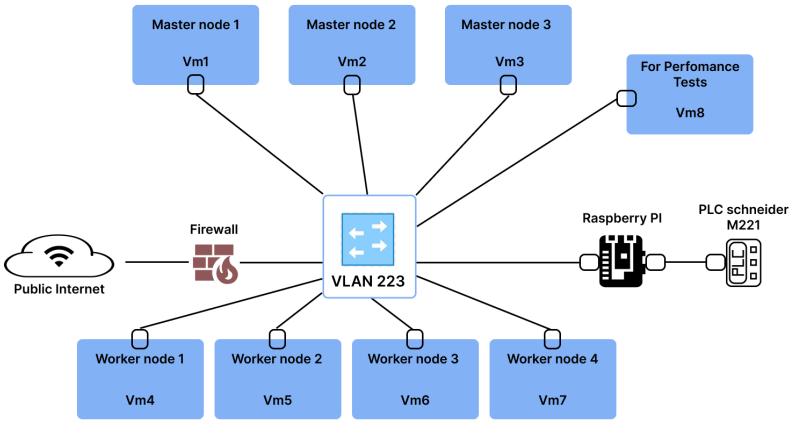


Figure A.1: Network topology diagram showing the connections between our Kubernetes cluster, an external Raspberry Pi, an external VM for performance tests, the physical PLC, and the public internet.

## A.2 Configurations

This section provides comprehensive installations and configurations that complement the Implementation section of the testbed. If you require assistance accessing any of the interfaces, you can refer to Appendix B. Links and credentials, many of which are default, are listed in Appendix C.

### A.2.1 Tainting a master node

To apply tainting to master nodes, the following command is employed [83]:

```
$ kubectl taint nodes <node-name>
 ↪  node-role.kubernetes.io/master=true:NoSchedule
```

This command effectively limits the utilization of master nodes to control plane functions exclusively [84].

### A.2.2 Installing Rancher server

For the installation of the Rancher server, we followed the steps outlined by the authors in [85], making some necessary adjustments. To deploy the Rancher

server with a personalized hostname and password, the following command can be utilized:

```
$ helm upgrade --install rancher rancher-stable/rancher \
--namespace cattle-system \
--set hostname=localhost \
--set bootstrapPassword=admin12345678
```

If you encounter issues with the password, you can reset it by executing the following commands [86]:

```
$ KUBECONFIG=./kube_config_cluster.yml
$ kubectl --kubeconfig $KUBECONFIG -n cattle-system exec
 ↪  $(kubectl --kubeconfig $KUBECONFIG -n cattle-system get pods
 ↪  -l app=rancher --no-headers | head -1 | awk '{ print $1 }')
 ↪  -c rancher -- reset-password
```

Subsequently, modify the Rancher server service to the "LoadBalancer" type to enable external access and proceed to locate the public IP address:

```
$ kubectl edit svc rancher -n cattle-system
$ kubectl get svc -n cattle-system
```

### A.2.3 Installing OpenEBS

The installation of OpenEBS is a straightforward process involving a few commands. The following instructions will add the OpenEBS official repository, create a dedicated namespace named "openEBS," and proceed to install OpenEBS within that designated namespace in the cluster:

```
$ helm repo add openebs https://openebs.github.io/charts
$ helm repo update
$ helm install openebs --namespace openebs openebs/openebs
 ↪  --create-namespace
```

### A.2.4 Installing and configuring Open5GS

The subsequent steps will involve adding the openverso repository, establishing a dedicated namespace labeled "open5GS," and then deploying the open5GS within this specifically designated namespace in the cluster:

```
$ kubectl create ns open5gs
$ helm repo add openverso
 ↪  https://gradiant.github.io/openverso-charts/
```

```
$ helm repo update
$ helm install open5gs openverso/open5gs -n open5gs
```

In addition, we leverage the capabilities of openEBS to create persistent volumes for the Open5GS database. In the Rancher Server, navigate to the Storage category and access the PersistenceVolumeClaims folder. Here, create a persistence volume claim for the Open5GS MongoDB using the class "openebs-hostpath". Driving forward, go to the Deployments folder within the Workload category. Edit the configuration of "open5gs-mongodb" by clicking the 3 vertical dots at the right corner of the screen. Under the storage tab, modify the persistence volume claim to match the one created earlier. Saving these changes will promptly generate a persistence volume for the Open5GS database, ensuring proper functionality of its functions.

To integrate user equipment with the 5G Core (5GC), you need to add subscribers through the web user interface provided by Open5GS. Access the Deployments folder in the Rancher Server and edit the "open5gs-webui" configuration. Add a port of type "LoadBalancer" with the private container port and listening port set to 3000. Moving on, navigate to the Service Discovery category and access the Services folder. Locate "open5gs-webui-loadbalancer" and edit the configuration to match the selector of pods. Due to a bug in creating a service type load balancer by Rancher Server, adjust the selectors as follows:

```
app.kubernetes.io/instance=open5gs
app.kubernetes.io/name=webui
```

These changes link the service to the web user interface pod. Once completed, find the IP address of the web user interface in the Services folder. It will be listed in the "target" column of "open5gs-webui-loadbalancer". Click on it to access the interface. Further details regarding credentials can be found in Appendix C of the Open5GS web user interface documentation.

Next, replicate the steps to add a load balancer service for the MongoDB of Open5GS. There are minor differences; specifically, the private container port and listening port should be set to 27017, and the selector name must be "mongodb". For accessing the database, use software like MongoDB Compass [87].

Before proceeding to add subscribers to the 5GC, it is essential to configure the UPF function to enable framed routing on user equipment. To enable the framed routing we followed a tutorial from s5uishida [88]. Start by navigating to the Storage category in Rancher Server and access the ConfigMaps folder. Locate the "open5gs-upf-entrypoint" configuration and append the following command below the IP address of the "ogstun" interface, with the specified sub-networks of the framed routes:

```
ip route add <Framed route sub-network>/24 dev ogstun;
```

After applying this configuration, go to the Deployments folder and redeploy "open5gs-upf" to ensure that the changes take effect.

To add subscribers, access the Open5GS web user interface as outlined in Appendix B.3. The International Mobile Subscriber Identity (IMSI) is formed by the Mobile Country Code (MCC), Mobile Network Code (MNC), and the user equipment identifier. The MCC uses the first 3 numbers of the IMSI, followed by 2 numbers for MNC, and the remaining 10 numbers for the identifier. Add the IMSI for each subscriber.

Subsequently, use MongoDB Compass to connect to the database and locate the Open5GS subscribers document. The document structure, shown below, includes configurations for each subscriber, including IMSI, Slice Differentiator (SD), and "ipv4_framed_routes". Manually add SD and the necessary Internet Protocol (IP) routes to recognize subscribers within the slice and enable framed routing, respectively.

```
{
  "_id": {
    "$oid": "637b8c4a829ed200017f05e6"
  },
  "imsi": "999700000000001",
  "subscribed_rau_tau_timer": 12,
  "network_access_mode": 0,
  "subscriber_status": 0,
  "access_restriction_data": 32,
  "slice": [
    {
      "sst": 1,
 -->  "sd":"0x111111",
      "default_indicator": true,
      "_id": {
        "$oid": "637b8c4a829ed200017f05e7"
      },
      "session": [
        {
          "name": "internet",
          "type": 3,
          "_id": {
            "$oid": "637b8c4a829ed200017f05e8"
          },
          "pcc_rule": [],
          "ambr": {
            "uplink": {
              "value": 1,
              "unit": 3
            },
            "downlink": {
              "value": 1,
              "unit": 3
            }
```

```
          },
          "qos": {
            "index": 9,
            "arp": {
              "priority_level": 8,
              "pre_emption_capability": 1,
              "pre_emption_vulnerability": 1
            }
          },
-->         "ipv4_framed_routes": [
-->           "<IP Address within the sub-net>/32"
-->         ]
        }
      ]
    }
  ],
  "ambr": {
    "uplink": {
      "value": 1,
      "unit": 3
    },
    "downlink": {
      "value": 1,
      "unit": 3
    }
  },
  "security": {
    "k": "465B5CE8 B199B49F AA5F0A2E E238A6BC",
    "amf": "8000",
    "op": null,
    "opc": "E8ED289D EBA952E4 283B54E8 8E6183CA",
    "sqn": {
      "$numberLong": "129"
    }
  },
  "purge_flag": [],
  "mme_realm": [],
  "mme_host": [],
  "imeisv": [
    "4370816125816151"
  ],
  "msisdn": [],
  "schema_version": 1,
  "__v": 0
}
```

Repeat this process for each User Equipment (UE) subscriber.

### A.2.5 Installing the New Radio (gnb) from UERANSIM

The upcoming steps entail adding the openverso repository and deploying the gNodeB (gNB) within the identical namespace as the 5G Core (5GC).

```
$ helm repo add openverso
 ↪  https://gradiant.github.io/openverso-charts/
$ helm repo update
$ helm install ueransim-gnb openverso/ueransim-gnb -n open5gs
```

Since this is a component of the openverso chart, similar to open5GS, no additional configurations were required. After deployment, the radio will automatically establish a connection to the core.

### A.2.6 Installing and configuring KEDA

The following instructions will add the KEDA official repository, create a dedicated namespace named "keda," and proceed to install KEDA within that designated namespace in the cluster:

```
$ helm repo add kedacore https://kedacore.github.io/charts
$ helm repo update
$ helm install keda kedacore/keda --namespace keda
 ↪  --create-namespace
```

To use the capabilities of KEDA in scaling through CPU metrics we first need to edit the deployment of the component we want to scale [89]. For example the rancher server, we go to the Deployments folder and search for "rancher", we edit the config and go to the resources tab. In there we add the CPU reservation to 100mCPUs, basically, this is requesting to the schedulers a worker node in were can provide 1/10 core CPU for running the container.

Subsequently, we generate a scaled object tailored to the rancher deployment. This entails specifying the namespace, assigning a name to this scaled object, indicating the trigger type, which in this case is CPU utilization, setting the minimum and maximum replicas for the pods, and defining the average CPU utilization between pods as 80%. KEDA will automatically construct a Horizontal Pod Autoscaler to manage the pods according to these defined parameters. Following is an illustration of the configuration file for the scaled object in the context of rancher.

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: memory-scaledobject-rancher
  namespace: cattle-system
spec:
```

```
  scaleTargetRef:
    name: rancher
  minReplicaCount: 1
  maxReplicaCount: 5
  triggers:
  - type: cpu
    metadata:
      type: Utilization
      value: "80"
```

### A.2.7 Installing Simu5G

The following Dockerfile was built by integrating a Virtual Network Computing (VNC) server and LXDE to deliver a functional desktop interface with Simu5G. In this image, specific configurations were established, including setting a password for the VNC server, specifying its resolution, and compiling and installing Simu5G. The exposure of port 5900 facilitates external access through a service. This image was developed using Docker and is publicly accessible on Docker Hub [90].

```
# Dockerfile
FROM dorowu/ubuntu-desktop-lxde-vnc as env-build
VOLUME [ "/dev/shm" ]
ENV VNC_PASSWORD=power
ENV RESOLUTION=1920x1080
RUN \
wget -q -O - https://dl.google.com/linux/linux_signing_key.pub |
 ↪  sudo apt-key add - && \
apt-get update -y  && apt-get upgrade -y  && \
DEBIAN_FRONTEND=noninteractive TZ=Etc/UTC apt-get install
 ↪  --no-install-recommends --yes \
gcc g++ gdb bison flex make \
git python3 python3-pip python3-dev openscenegraph
 ↪  libopenscenegraph-dev curl \
openmpi-bin libopenmpi-dev \
gdal-bin libgdal-dev minizip rocksdb-tools duktape cmake \
default-jre default-jdk openjfx \
swig doxygen graphviz libpcap-dev tcl libqt5svg5
 ↪  libqt5opengl5-dev \
qtbase5-dev qtchooser qt5-qmake qtbase5-dev-tools \
ffmpeg && \
apt-get clean && \
python3 -m pip install --upgrade pip && \
pip install posix-ipc numpy scipy pandas matplotlib  && \
rm -rf /var/lib/apt/lists/*
```

```
from env-build as omnetpp-build
shell ["/bin/bash", "-c"]
run cd / && curl -L
 ↪  https://github.com/omnetpp/omnetpp/releases/download/
 ↪  omnetpp-6.0pre11/omnetpp-6.0pre11-src-linux.tgz|tar -zxv &&\
    cd /omnetpp-6.0pre11 && \
    source ./setenv -f && ./configure WITH_OSGEARTH=no
     ↪  PREFER_CLANG=no && make

from env-build as inet-build
copy --from=omnetpp-build /omnetpp-6.0pre11 /omnetpp-6.0pre11
shell ["/bin/bash", "-c"]
run cd /omnetpp-6.0pre11 && source ./setenv -f && \
    curl -L
     ↪  https://github.com/inet-framework/inet/releases/download/
     ↪  v4.3.2/inet-4.3.2-src.tgz | tar -zxv -C
     ↪  /omnetpp-6.0pre11/samples && \
    cd /omnetpp-6.0pre11/samples/inet4.3 && \
    source ./setenv -f && make makefiles && make

from env-build as simu5g-build
copy --from=inet-build /omnetpp-6.0pre11 /omnetpp-6.0pre11
shell ["/bin/bash", "-c"]
run cd /omnetpp-6.0pre11 && source ./setenv -f && cd
 ↪  /omnetpp-6.0pre11/samples/inet4.3 && source ./setenv -f && \
    curl -L https://github.com/Unipisa/Simu5G/archive/refs/tags/
     ↪  v1.2.0.tar.gz | tar -zxv -C /omnetpp-6.0pre11/samples &&
     ↪  \
    cd /omnetpp-6.0pre11/samples/Simu5G-1.2.0 && \
    source ./setenv -f && make makefiles && make

from env-build
copy --from=simu5g-build /omnetpp-6.0pre11 /omnetpp-6.0pre11
run chown -hR 1000 /omnetpp-6.0pre11 && apt-get update && apt-get
 ↪  install nano
EXPOSE 5900
cmd ["/bin/bash"]
```

Upon constructing and uploading this image to Docker Hub, the subsequent step involved navigating to the Rancher server "Deployments" folder. Within this segment, a deployment named "simu5g" was established utilizing the previous image within the "open5gs" namespace. Subsequently, an external access service of the LoadBalancer type was exposed, with the private container port set to 5900 and the listening port matching. Furthermore, the selectors for this service were adjusted to align with the intended configuration.

```
workload.user.cattle.io/workloadselector=apps.deployment-open5gs
 ↪  -simu5g
```

In Appendix 5.1.3, you will find detailed instructions on how to access the Simu5G graphical user interface (GUI).

### A.2.8 Installing and configuring OpenPLC with UEs

The following script file will serve as the entry point for the image we are constructing, which combines OpenPLC and User Equipment (UERANSIM). This script will undertake the task of initializing the user equipment by establishing a connection with the 5G core. Subsequently, it will direct all traffic associated with the 5G network towards the "uesimtun0" interface. Furthermore, it will incorporate a framed-route IP and initiate the OpenPLC server. We followed a tutorial provided by s5uishida [88] to enable framed routing in the UE.

```
# entrypoint.sh

#!/bin/bash

set -e

_term() {
    case "$command" in
    ue)
        echo "Deleting ue: nr-ue -c ue.yaml"
        for x in $(./usr/local/bin/nr-cli -d); do
            ./usr/local/bin/nr-cli $x --exec "deregister
             ↪  switch-off"
        done
        echo "UEs switched off"
        sleep 5
        ;;
    *)
        echo "It isn't necessary to perform any cleanup"
        ;;
    esac
}

if [ $# -lt 1 ]
then
        echo "Usage : $0 [gnb|ue]"
        exit
fi

sh -c "echo 1 > /proc/sys/net/ipv4/ip_forward" $@
```

```
(bash -c "sleep 10; ip route add 10.45.0.0/24 dev uesimtun0" &)
(bash -c "sleep 10; ip addr add <Frame route IP>/24 dev uesimtun0
 ↪  &)
( "/workdir/start_openplc.sh" & )
command=$1
trap _term SIGTERM
shift

case "$command" in

ue)
    GNB_IP=${GNB_IP:-"$(host -4 $GNB_HOSTNAME |awk
 ↪  '/has.*address/{print $NF; exit}')"}
    export GNB_IP
    echo "GNB_IP: $GNB_IP"
    envsubst < /etc/ueransim/ue.yaml > ue.yaml
    echo "Launching ue: nr-ue -c ue.yaml"
    nr-ue -c ue.yaml $@ &
    child=$!
    wait "$child"
    ;;
gnb)
    N2_BIND_IP=${N2_BIND_IP:-"$(ip addr show ${N2_IFACE}  | grep
 ↪  -o 'inet [[:digit:]]\{1,3\}\.
 ↪  [[:digit:]]\{1,3\}\.[[:digit:]]\{1,3\}\.[[:digit:]]\{1,3\}'|
 ↪  cut -c 6-)"}
    N3_BIND_IP=${N3_BIND_IP:-"$(ip addr show ${N3_IFACE} | grep
 ↪  -o 'inet [[:digit:]]\{1,3\}\.
 ↪  [[:digit:]]\{1,3\}\.[[:digit:]]\{1,3\}\.[[:digit:]]\{1,3\}'|
 ↪  cut -c 6-)"}
    RADIO_BIND_IP=${RADIO_BIND_IP:-"$(ip addr show ${RADIO_IFACE}
 ↪  | grep -o 'inet
 ↪  [[:digit:]]\{1,3\}\.[[:digit:]]\{1,3\}\.[[:digit:]]\{1,3\}\.
 ↪  [[:digit:]]\{1,3\}'| cut -c 6-)"}
    AMF_IP=${AMF_IP:-"$(host -4 $AMF_HOSTNAME |awk
 ↪  '/has.*address/{print $NF; exit}')"}
    export N2_BIND_IP N3_BIND_IP RADIO_BIND_IP AMF_IP
    echo "N2_BIND_IP: $N2_BIND_IP"
    echo "N3_BIND_IP: $N3_BIND_IP"
    echo "RADIO_BIND_IP: $RADIO_BIND_IP"
    echo "AMF_IP: $AMF_IP"
    envsubst < /etc/ueransim/gnb.yaml > gnb.yaml
    echo "Launching gnb: nr-gnb -c gnb.yaml"
    nr-gnb -c gnb.yaml $@
    ;;
*) echo "unknown component $1 is not a component (gnb or ue).
 ↪  Running $@ as command"
```

```
    $@
    ;;
esac
```

The succeeding file contains the configuration for the user equipment. All these variables will be replaced when deploying within the Kubernetes cluster.

```
# IMSI number of the UE. IMSI = [MCC|MNC|MSISDN] (In total 15 or
↪  16 digits)
supi: 'imsi-${MCC}${MNC}${MSISDN}'
# Mobile Country Code value
mcc: ${MCC}
# Mobile Network Code value (2 or 3 digits)
mnc: ${MNC}

# Permanent subscription key
key: '${KEY}'
# Operator code (OP or OPC) of the UE
op: '${OP}'
# This value specifies the OP type and it can be either 'OP' or
↪  'OPC'
opType: '${OP_TYPE}'
# Authentication Management Field (AMF) value
amf: '8000'
# IMEI number of the device. It is used if no SUPI is provided
imei: '356938035643803'
# IMEISV number of the device. It is used if no SUPI and IMEI is
↪  provided
imeiSv: '4370816125816151'

# List of gNB IP addresses for Radio Link Simulation
gnbSearchList:
  - ${GNB_IP}

  # UAC Access Identities Configuration
uacAic:
  mps: false
  mcs: false

# UAC Access Control Class
uacAcc:
  normalClass: 0
  class11: false
  class12: false
  class13: false
  class14: false
  class15: false
```

```
# Initial PDU sessions to be established
sessions:
  - type: 'IPv4'
    apn: ${APN}
    slice:
      sst: ${SST}
      sd: ${SD}
    emergency: false

# Configured NSSAI for this UE by HPLMN
configured-nssai:
  - sst: ${SST}
    sd: ${SD}

# Default Configured NSSAI for this UE
default-nssai:
  - sst: ${SST}
    sd: ${SD}

# Supported encryption algorithms by this UE
integrity:
  IA1: true
  IA2: true
  IA3: true

# Supported integrity algorithms by this UE
ciphering:
  EA1: true
  EA2: true
  EA3: true

# Integrity protection maximum data rate for user plane
integrityMaxRate:
  uplink: 'full'
  downlink: 'full'
```

Once you have created the entry point and user equipment files, you can proceed to build the image that executes the compilation and installation steps for UERANSIM, followed by the installation of OpenPLC. The OpenPLC repository, complete with the installation script, is available on thiagoralves GitHub page [91]. First, we will go through the configuration file, followed by the steps to build the image. Afterward, we will discuss how to push it to a private GitLab repository and create a Kubernetes secret, enabling its deployment within the cluster.

```
# Dockerfile
FROM ubuntu:latest as env-build
```

```
RUN apt update -y && apt install -y git && git clone
 ↪  https://github.com/aligungr/UERANSIM.git &&  apt install make
 ↪  -y && apt install gcc -y && \
 apt install g++ -y  && \
  apt install build-essential libssl-dev -y && apt-get install
   ↪  wget unzip zip -y && \
  wget
   ↪  https://github.com/Kitware/CMake/releases/download/v3.25.0/
   ↪  cmake-3.25.0.tar.gz &&  \
  tar zxvf cmake-3.25.0.tar.gz && cd cmake-3.25.0 && ./bootstrap
   ↪  && make && \
  make install && apt install libsctp-dev lksctp-tools iproute2
   ↪  -y && cd ../UERANSIM && make

FROM ubuntu:latest

COPY --from=env-build /UERANSIM/build/* /usr/local/bin/
RUN mkdir /etc/ueransim
COPY ue.yaml /etc/ueransim/ue.yaml
COPY entrypoint.sh /entrypoint.sh

RUN apt update -y -q && apt install -y -q  tcpdump iptables
 ↪  iputils-ping traceroute build-essential bind9utils
 ↪  bind9-doc   dnsutils    curl     gettext     iperf3
 ↪  libsctp-dev lksctp-tools iproute2  # buildkit

ENV N2_IFACE=eth0
ENV N3_IFACE=eth0
ENV RADIO_IFACE=eth0
ENV AMF_HOSTNAME=amf
ENV GNB_HOSTNAME=localhost

COPY . /workdir
WORKDIR /workdir
RUN ./install.sh docker

EXPOSE 8080
EXPOSE 502

ENTRYPOINT ["/entrypoint.sh"]
```

The subsequent commands will require the GitLab account credentials, which can be located in Appendix C. These commands streamline the connection to the GitLab account and the publication of the image in the designated repository.

```
# Run this commands on terminal
$ docker login registry.gitlab.com
$ docker build -t <your-private-repo>/ue-openplc:latest .
$ docker push <your-private-repo>/ue-openplc:latest
```

On the master node, executing the following command will generate the necessary secret within the Kubernetes cluster. It will require GitLab credentials. This procedure grants access to the private GitLab images within the cluster.

```
$ kubectl create secret docker-registry <secret-name>
 ↪  --docker-server=registry.gitlab.com
 ↪  --docker-username=<your-gitlab-username>
 ↪  --docker-password='<your-gitlab-access-token>'
```

Next, you will need to clone the "ueransim-ues" folder from the openverso charts [73] and proceed to edit the "values.yaml" file. This file contains the variables necessary for configuring the user equipment. The required changes include updating the "repository" to match the repository, ensuring that "pullSecrets" aligns with the name of the created secret, and editing "initialMSISDN" to correspond to a unique identifier for the UE, which must match the subscription in open5gs.

```
# source ueransim-ues/values.yaml

## @section Global parameters
## Global Docker image parameters
## Please, note that this will override the image parameters,
 ↪  including dependencies, configured to use the global value
## Current available global Docker image parameters:
 ↪  imageRegistry, imagePullSecrets and storageClass

## @param global.imageRegistry Global Docker image registry
## @param global.imagePullSecrets Global Docker registry secret
 ↪  names as an array
## @param global.storageClass Global StorageClass for Persistent
 ↪  Volume(s)
##
global:
  imageRegistry: ""
  ## E.g.
  ## imagePullSecrets:
  ##   - myRegistryKeySecretName
  ##
  imagePullSecrets: []
  storageClass: ""

## @section Common parameters
```

```
## @param kubeVersion Override Kubernetes version
##
kubeVersion: ""
## @param nameOverride String to partially override
 ↪  common.names.fullname
##
nameOverride: ""
## @param fullnameOverride String to fully override
 ↪  common.names.fullname
##
fullnameOverride: ""
## @param commonLabels Labels to add to all deployed objects
##
commonLabels: {}
## @param commonAnnotations Annotations to add to all deployed
 ↪  objects
##
commonAnnotations: {}
## @param clusterDomain Kubernetes cluster domain name
##
clusterDomain: cluster.local
## @param extraDeploy Array of extra objects to deploy with the
 ↪  release
##
extraDeploy: []

image:
  registry: registry.gitlab.com
  repository: <your-private-repo>
  tag: latest
  ## Specify a imagePullPolicy
  ## Defaults to 'Always' if image tag is 'latest', else set to
   ↪  'IfNotPresent'
  ## ref: http://kubernetes.io/docs/user-guide/images/
   ↪  #pre-pulling-images
  ##
  pullPolicy: Always
  ## Optionally specify an array of imagePullSecrets.
  ## Secrets must be manually created in the namespace.
  ## ref:
   ↪  https://kubernetes.io/docs/tasks/configure-pod-container/
   ↪  pull-image-private-registry/
  ## e.g:
  pullSecrets:
    - <your-secret-name>
  ##
  ##pullSecrets: []
```

```
  ## Enable debug mode
  ##
  debug: false

name: ueransim-ues

count: 1
## Change initialMSISDN if you want to have different UEs
initialMSISDN: '0000000001'
mcc: '999'
mnc: '70'
key: 465B5CE8B199B49FAA5F0A2EE238A6BC
op: E8ED289DEBA952E4283B54E88E6183CA
opType: OPC
gnb:
  hostname: ueransim-gnb
sst: 1
sd: "0x111111"
apnList:
  - type: 'IPv4'
    apn: 'internet'
    slice:
      sst: 1
      sd: "0x111111"
    emergency: false

## @param command Override default container command (useful when
 ↪  using custom images)
##
command: []
## @param args Override default container args (useful when using
 ↪  custom images)
##
args: []

resources:
  limits: {}
  requests: {}
podSecurityContext:
  enabled: false
containerSecurityContext:
  enabled: false
podLabels: {}
podAnnotations: {}
affinity: {}
nodeSelector: {}
tolerations: []
```

You can deploy the user equipment within the Kubernetes cluster using the following command:

```
$ helm install <ue-name> ./ueransim-ues -n open5gs
```

Then, proceed to create two services of type "LoadBalancer," exposing ports 8080 and 502 for the web interface and Modbus port, respectively. Ensure to adjust the selector in the Services folder to align with your user equipment name, as demonstrated next:

```
app.kubernetes.io/instance = <ue-name>
```

### A.2.9 Configuring the RaspberryPI

This Raspberry Pi will function as user equipment, responsible for forwarding packages to the physical PLC within the 5G network.

If the "/etc/netplan/" directory is missing, install the "netplan.io" package. Then, proceed to edit the configuration file within that directory. The provided configuration will establish a network interface in VLAN 223 and assign a private static IP for communication with the physical PLC.

```
# source /etc/netplan/50-cloud-init.yaml
network:
  version: 2
  renderer: networkd
  ethernets:
          eth0:
            dhcp4: no
            addresses: [<Private IP address (same sub-network as
             ↪ physical PLC)>/24]
vlan.223:
    id: 223
    link: eth0
    dhcp4: yes
```

Execute the subsequent commands to apply the network configurations to the Raspberry Pi.

```
$ sudo netplan generate
$ sudo netplan apply
```

Execute the following commands to compile and install UERANSIM on the Raspberry Pi.

```
$ sudo apt update
$ sudo apt upgrade
$ sudo apt install make
$ sudo apt install gcc
$ sudo apt install g++
$ sudo apt install libsctp-dev lksctp-tools
$ sudo apt install iproute2
$ sudo snap install cmake --classic
$ cd ~/UERANSIM
$ make
$ cp /UERANSIM/build/* /usr/local/bin/
```

Create a configuration file with the following content, ensuring to replace the last ten digits of the "supi" with the unique identifier of the user equipment in the 5G core.

```
# source ue.yaml

# IMSI number of the UE. IMSI = [MCC|MNC|MSISDN] (In total 15 or
 ↪  16 digits)

## Change if u want to have different UEs
supi: 'imsi-999700000000005'
mcc: '999'
mnc: '70'

# Permanent subscription key
key: '465B5CE8B199B49FAA5F0A2EE238A6BC'
# Operator code (OP or OPC) of the UE
op: 'E8ED289DEBA952E4283B54E88E6183CA'
# This value specifies the OP type and it can be either 'OP' or
 ↪  'OPC'
opType: 'OPC'
# Authentication Management Field (AMF) value
amf: '8000'
# IMEI number of the device. It is used if no SUPI is provided
imei: '356938035643803'
# IMEISV number of the device. It is used if no SUPI and IMEI is
 ↪  provided
imeiSv: '4370816125816151'

# List of gNB IP addresses for Radio Link Simulation
gnbSearchList:
  - ${GNB_IP}

  # UAC Access Identities Configuration
```

```
uacAic:
  mps: false
  mcs: false

# UAC Access Control Class
uacAcc:
  normalClass: 0
  class11: false
  class12: false
  class13: false
  class14: false
  class15: false

# Initial PDU sessions to be established
sessions:
  -
    apn: internet
    emergency: false
    slice:
      sd: "0x111111"
      sst: 1
    type: IPv4

# Configured NSSAI for this UE by HPLMN
configured-nssai:
  - sst: 1
    sd: 0x111111

# Default Configured NSSAI for this UE
default-nssai:
  - sst: 1
    sd: 0x111111

# Supported encryption algorithms by this UE
integrity:
  IA1: true
  IA2: true
  IA3: true

# Supported integrity algorithms by this UE
ciphering:
  EA1: true
  EA2: true
  EA3: true

# Integrity protection maximum data rate for user plane
integrityMaxRate:
```

```
  uplink: 'full'
  downlink: 'full'
```

The following commands are used to run the user equipment and configure packet forwarding to the physical PLC. Make sure to refer to section A.1 for the necessary IP addresses. You might need to install the "iptables" package to execute iptables commands. The steps involve starting the user equipment as a background process, enabling forwarding between network interfaces, adding the framed route IP, and creating a route to the "uesimtun0" interface used for communication in the 5G network. Additionally, the iptables commands are responsible for forwarding PLC packets from the 5G network to the physical PLC.

```
$ nr-ue -c ue.yaml &
$ sh -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
$ ip addr add <Frame route IP>/24 dev uesimtun0
$ ip route add 10.45.0.0/24 dev uesimtun0
$ iptables -t nat -A PREROUTING -i uesimtun0 -p tcp --dport 502
 ↪  -d <Frame route IP> -j DNAT --to <Private IP of physical
 ↪  PLC>:502
$ iptables -t nat -A POSTROUTING -o eth0 -p tcp --dport 502 -j
 ↪  SNAT --to-source <Private IP of RaspberryPI>
```

### A.2.10 Configuring the physical PLC (Schneider M221)

To access and configure the Schneider M221 physical PLC, you need to install the software "EcoStruxure Machine Expert - Basic" [92]. Connect an Ethernet cable from your computer to the physical PLC and switch your IP address to a static one within the same subnetwork as the physical PLC. By doing so, you will be able to configure, through the installed software, the physical PLC and modify its IP to match the desired sub-network. Ensure that the sub-network aligns with the private network of the Raspberry Pi.

To read the discrete input from the Schneider M221 physical PLC located at the other user stations, utilize the following command. When you poll the discrete input from the physical PLC, it will return 8 bits, even if most of the bits are 0. Remember to provide the framed route IP of the Raspberry Pi as part of the command.

```
$ mbpoll <Framed route IP of the Raspberry Pi> -t1 -r 801
```

### A.2.11 Configuring the hub-and-spoke scenario

Upon accessing the web interface of user equipment with openPLC installed, you will find a tab labeled "Programs." These programs represent the logical processes of the PLCs. The subsequent program is intended for implementation in the slave PLCs. Essentially, it encompasses two holding registers: one named "slave1,"

initially set to 11, and another named "parvo." When the value of "slave1" exceeds 10, the value of the "parvo" register becomes 1. Conversely, if "slave1" is less or equal to 10, the value of "parvo" is set to 0.

```
# source program.st

PROGRAM program0
  VAR
    SWITCH AT %QX0.0 : BOOL;
  END_VAR
  VAR
    setpoint : INT := 10;
    GT2_OUT : BOOL;
  END_VAR
  VAR
    SLAVE1 AT %QW150 : INT := 11;
    PARVO AT %QW151 : INT;
  END_VAR

  GT2_OUT := GT(SLAVE1, setpoint);
  SWITCH := GT2_OUT;
  IF SWITCH=TRUE THEN
    PARVO := 1;
  ELSE
    PARVO := 0;
  END_IF;

END_PROGRAM


CONFIGURATION Config0

  RESOURCE Res0 ON PLC
    TASK task0(INTERVAL := T#20ms,PRIORITY := 0);
    PROGRAM instance0 WITH task0 : program0;
  END_RESOURCE
END_CONFIGURATION
```

The previous program will be executed on all the slave devices except for the physical PLC, as it does not require one.

In the master device, you can execute the default blank program. Access the "Slave Devices" tab in the master PLC web interface and add the slaves with their framed-route IPs, the Modbus port (502), and for Holding Registers - Read (%IW100), set the start address as 150 and size as 2, as this is where the registers from the program are located. However, this does not apply to the physical PLC. For the physical PLC, set the framed-route IP of the Raspberry Pi, and the Modbus port, and for Discrete Inputs (%IX100.0), set the start address to 1 with

a size of 1. To read PLC registers, you can install a Modbus client as outlined in Appendix B.6.

### A.2.12 Configuring JSON to CSV Converter Script for Peak-Rate Tests

The following script is designed to recursively search subdirectories for IPERF data in JSON format and extract specific fields such as "sum_received_bytes," "sum_received_bps," "local_port," "start," "bytes_stream," and "bps_stream." It then appends the data from all the JSON files into a single CSV file. To run the script, you will need Python 3, the pip3 package manager, and the Pandas library installed on your system. When executing it, make sure to specify the directory you want to search for IPERF data as an argument in the command.

```
# source peakrate_jsontocsv.py

import pandas as pd
import json
import os
import sys

def process_file(filename):
    with open(filename, 'r') as f:
        data = json.load(f)

    # Extracting the required fields
    sum_received_bytes = data['end']['sum_received']['bytes']
    sum_received_bps =
     ↪  data['end']['sum_received']['bits_per_second']
    local_port = data['start']['connected'][0]['local_port']

    # Create a list to hold rows
    rows = []

    for interval in data['intervals']:
        start = interval['streams'][0]['start']
        bytes_stream = interval['streams'][0]['bytes']
        bps_stream = interval['streams'][0]['bits_per_second']

        rows.append([sum_received_bytes, sum_received_bps,
         ↪  local_port, start, bytes_stream, bps_stream])

    return pd.DataFrame(rows, columns=['sum_received_bytes',
     ↪  'sum_received_bps', 'local_port', 'start',
     ↪  'bytes_stream', 'bps_stream'])
```

```
# Process the files and concatenate them into one DataFrame
dfs = []

# Get the directory path from command-line arguments
if len(sys.argv) < 2:
    print("Please provide the directory path as an argument.")
    sys.exit()

base_directory = sys.argv[1]

# Walk through each directory and its subdirectories
for dirpath, dirnames, filenames in os.walk(base_directory):
    for file in filenames:
        if file.endswith('.json'):  # Process only JSON files
            dfs.append(process_file(os.path.join(dirpath, file)))

df = pd.concat(dfs, ignore_index=True)

# Print the DataFrame
print(df)

# Save the DataFrame to a CSV in the provided base directory
df.to_csv(os.path.join(base_directory, 'output.csv'),
 ↪  index=False)
```

### A.2.13 Configuring the Scenario for Fixed-Rate Tests

First, duplicate the original script named "open5gs-dbctl" from the open5gs repository, which can be found at [93]. Next, append the following instructions to the script. These instructions are designed to iteratively incorporate user equipment subscribers into the database. To execute these instructions, you should invoke the script with specific arguments. The first argument should be "add_ues_with_slice," followed by the starting and ending values for the IMSI range as the second and third arguments, respectively. The script will then iterate through this specified range, adding subscribers to the database.

```
if [ "$1" = "add_ues_with_slice" ]; then
    if [ "$#" -eq 3 ]; then
        IMSI_start=$2
        IMSI_finish=$3
        KI="465B5CE8 B199B49F AA5F0A2E E238A6BC"
        OPC="E8ED289D EBA952E4 283B54E8 8E6183CA"
        APN="internet"
        SST=1
        SD="0x111111"
```

```
        while [ $IMSI_start -le $IMSI_finish ]
        do
                mongosh --eval "db.subscribers.insertOne(
                    {
                        \"_id\": new ObjectId(),
                        \"schema_version\": NumberInt(1),
                        \"imsi\": \"$IMSI_start\",
                        \"msisdn\": [],
                        \"imeisv\": [],
                        \"mme_host\": [],
                        \"mm_realm\": [],
                        \"purge_flag\": [],
                        \"slice\":[
                        {
                            \"sst\": NumberInt($SST),
                            \"sd\": \"$SD\",
                            \"default_indicator\": true,
                            \"session\": [
                            {
                                \"name\" : \"$APN\",
                                \"type\" : NumberInt(3),
                                \"qos\" :
                                { \"index\": NumberInt(9),
                                    \"arp\":
                                    {
                                        \"priority_level\" :
↪   NumberInt(8),

↪   \"pre_emption_capability\": NumberInt(1),

↪   \"pre_emption_vulnerability\": NumberInt(2)
                                    }
                                },
                                \"ambr\":
                                {
                                    \"downlink\":
                                    {
                                        \"value\":
↪   NumberInt(1000000000),
                                        \"unit\": NumberInt(0)
                                    },
                                    \"uplink\":
                                    {
                                        \"value\":
↪   NumberInt(1000000000),
                                        \"unit\": NumberInt(0)
                                    }
```

```
                                },
                                \"pcc_rule\": [],
                                \"_id\": new ObjectId(),
                            }],
                            \"_id\": new ObjectId(),
                        }],
                        \"security\":
                        {
                            \"k\" : \"$KI\",
                            \"op\" : null,
                            \"opc\" : \"$OPC\",
                            \"amf\" : \"8000\",
                        },
                        \"ambr\" :
                        {
                            \"downlink\" : { \"value\":
↪   NumberInt(1000000000), \"unit\": NumberInt(0)},
                            \"uplink\" : { \"value\":
↪   NumberInt(1000000000), \"unit\": NumberInt(0)}
                        },
                        \"access_restriction_data\": 32,
                        \"network_access_mode\": 0,
                        \"subscribed_rau_tau_timer\": 12,
                        \"__v\": 0
                    }
                    );" $DB_URI
                IMSI_start=$(( IMSI_start+1 ))          #
                ↪  increments $IMSI_start
        done
        exit $?
    fi

    echo "open5gs-dbctl: incorrect number of args, format is
    ↪  \"open5gs-dbctl add_ue_with_slice imsi key opc apn sst
    ↪  sd\""
    exit 1
fi
```

Next, access the Rancher server and navigate to the "Deployment" category. Launch the shell of the "open5gs-populate" deployment. Within this shell, locate the "open5gs-dbctl" file located in the "usr/local/bin/" path, and replace it with your modified script that includes the instructions for iteratively adding user equipment subscribers to the database. Once you've made this replacement, you can execute the script from the command line by invoking "open5gs-dbctl" with the necessary arguments.

Next, you will need to clone the "ueransim-ues" folder from the openverso charts [73] and proceed to edit the "values.yaml" file. This file contains the variables necessary for configuring the user equipments within a single container. The re-

quired changes include updating the "count" to match the number of user equipments and editing "initialMSISDN" to correspond to the initial identifier of the first UE, which must match the subscriptions in open5gs database.

```
# source ueransim-ues/values.yaml
## @section Global parameters
## Global Docker image parameters
## Please, note that this will override the image parameters,
 ↪  including dependencies, configured to use the global value
## Current available global Docker image parameters:
 ↪  imageRegistry, imagePullSecrets and storageClass

## @param global.imageRegistry Global Docker image registry
## @param global.imagePullSecrets Global Docker registry secret
 ↪  names as an array
## @param global.storageClass Global StorageClass for Persistent
 ↪  Volume(s)
##
global:
  imageRegistry: ""
  ## E.g.
  ## imagePullSecrets:
  ##   - myRegistryKeySecretName
  ##
  imagePullSecrets: []
  storageClass: ""

## @section Common parameters

## @param kubeVersion Override Kubernetes version
##
kubeVersion: ""
## @param nameOverride String to partially override
 ↪  common.names.fullname
##
nameOverride: ""
## @param fullnameOverride String to fully override
 ↪  common.names.fullname
##
fullnameOverride: ""
## @param commonLabels Labels to add to all deployed objects
##
commonLabels: {}
## @param commonAnnotations Annotations to add to all deployed
 ↪  objects
##
commonAnnotations: {}
```

```
## @param clusterDomain Kubernetes cluster domain name
##
clusterDomain: cluster.local
## @param extraDeploy Array of extra objects to deploy with the
 ↪  release
##
extraDeploy: []

image:
  registry: docker.io
  repository: openverso/ueransim
  tag: 3.2.6
  ## Specify a imagePullPolicy
  ## Defaults to 'Always' if image tag is 'latest', else set to
   ↪  'IfNotPresent'
  ## ref:
   ↪  http://kubernetes.io/docs/user-guide/images/#pre-pulling-images
  ##
  pullPolicy: Always
  ## Optionally specify an array of imagePullSecrets.
  ## Secrets must be manually created in the namespace.
  ## e.g:
  ## pullSecrets:
  ##   - myRegistryKeySecretName
  ##
  pullSecrets: []
  ## Enable debug mode
  ##
  debug: false

name: ueransim-ues

count: <number-ues>
initialMSISDN: '0000000001' # Edit here too
mcc: '999'
mnc: '70'
key: 465B5CE8B199B49FAA5F0A2EE238A6BC
op: E8ED289DEBA952E4283B54E88E6183CA
opType: OPC
gnb:
  hostname: ueransim-gnb
sst: 1
sd: "0x111111"
apnList:
  - type: 'IPv4'
    apn: 'internet'
    slice:
```

```
      sst: 1
      sd: "0x111111"
    emergency: false

## @param command Override default container command (useful when
 ↪  using custom images)
##
command: []
## @param args Override default container args (useful when using
 ↪  custom images)
##
args: []

resources:
  limits: {}
  requests: {}
podSecurityContext:
  enabled: false
containerSecurityContext:
  enabled: false
podLabels: {}
podAnnotations: {}
affinity: {}
nodeSelector: {}
tolerations: []
```

You can deploy the user equipments within the Kubernetes cluster using the following command:

```
$ helm install <name> ./ueransim-ues -n open5gs
```

To simultaneously deploy all the IPERF servers within the virtual machine dedicated to performance tests, we have developed the subsequent script that leverages sub-processes to execute parallel operations. This script provides a real-time display of the number of running processes. However, before using it, it's essential to install the 'psutil' package.

The script requires two arguments: the first argument specifies the number of processes to create, and the second argument indicates the number of tests to run. For each server, a port is associated iteratively, and the script generates directories corresponding to user equipment identifiers. The first user equipment is associated with port 5001, and subsequent ports are assigned based on the number of user equipments.

IPERF will allocate test result files to the respective directories, each corresponding to a user equipment. When a test concludes, the associated process terminates. If additional tests remain to be conducted, a new batch of processes is immediately created. This approach guarantees an efficient and organized distribution of test files among the designated directories.

```
# source iperf-servers.py

import subprocess
import sys
import os
import psutil
import threading
import time

def get_next_test_number():
    ue1_dir = "ue-1"
    if not os.path.exists(ue1_dir):
        return 1
    files = os.listdir(ue1_dir)
    test_numbers = [int(file.replace("test", "").replace(".json",
     ↪  "")) for file in files if file.startswith("test") and
     ↪  file.endswith(".json")]
    if not test_numbers:
        return 1
    else:
        return max(test_numbers) + 1

def launch_iperf3_servers(num_processes):
    BASE_PORT = 5001
    test_number = get_next_test_number()
    processes = []
    for i in range(num_processes):
        port = BASE_PORT + i
        output_file = f"ue-{i+1}/test{test_number}.json"
        os.makedirs(f"ue-{i+1}", exist_ok=True)
        cmd = ["iperf3", "-s", "--port", str(port), "--json",
         ↪  "--logfile", output_file, "-1"]
        process = subprocess.Popen(cmd)
        processes.append(process)

    # Start the status display thread
    status_thread = threading.Thread(target=display_status,
     ↪  daemon=True)
    status_thread.start()

    for process in processes:
        process.wait()

    # Wait for the status display thread to finish
    status_thread.join()

def count_iperf3_processes():
```

```
    return sum(1 for proc in psutil.process_iter() if "iperf3" in
     ↪  proc.name())

def display_status():
    while count_iperf3_processes() > 0:
        num_processes = count_iperf3_processes()
        sys.stdout.write(f"\rNumber of iperf3 processes running:
         ↪  {num_processes}")
        sys.stdout.flush()
        time.sleep(1)

if __name__=='__main__':
    if len(sys.argv) < 3:
        print("Usage: python script_name.py <number_of_processes>
         ↪  <additional_times_to_run>")
        sys.exit()

    num_processes = int(sys.argv[1])
    additional_times_to_run = int(sys.argv[2])

    if num_processes not in [1, 5, 14, 15, 20, 30, 50]:
        print("Error: Number of processes must be one of 1, 5,
         ↪  14, 15, 20, 30 or 50.")
        sys.exit()

    # Run the script the specified additional number of times
    for _ in range(additional_times_to_run):
        launch_iperf3_servers(num_processes)
```

The next script streamlines the execution of multiple IPERF3 clients for network performance assessment. Users can tailor test parameters, including bitrate (first argument), the number of clients (second argument), and the number of test runs (third argument). Additionally, it offers real-time progress tracking of ongoing tests. The script systematically allocates 5G network interfaces to ports, ensuring consistency with specific user equipment. After the initial test and process completion, it incorporates a 5-minute pause before launching the subsequent batch of processes for further testing. This delay accounts for buffering and the connections associated with the interfaces.

```
# source iperf-clients.py

import sys
import time
import subprocess
import threading
import psutil # Required package installation

def get_uesimtun_ips():
```

```
    # List all uesimtun interfaces
    result = subprocess.check_output(["ip", "-4", "addr", "show",
    ↪  "type", "tun"]).decode("utf-8")
    ips = [line.strip() for line in result.split("\n") if "inet "
    ↪  in line and "uesimtun" in line]

    if not ips:
        raise ValueError("No active uesimtun interface found!")

    # Extract IPs
    ip_addresses = [ip_line.split(" ")[1].split("/")[0] for
    ↪  ip_line in ips]

    return ip_addresses

def count_iperf3_processes():
    return sum(1 for proc in psutil.process_iter() if "iperf3" in
    ↪  proc.name())

def display_status():
    while count_iperf3_processes() > 0:
        num_processes = count_iperf3_processes()
        sys.stdout.write(f"\rNumber of iperf3 clients running:
        ↪  {num_processes}")
        sys.stdout.flush()
        time.sleep(1)

def run_client(server_ip, start_port, bitrate, num_clients):
    ip_addresses = get_uesimtun_ips()
    processes = []

    if len(ip_addresses) < num_clients:
        raise ValueError(f"Only {len(ip_addresses)} uesimtun
        ↪  interfaces found, but {num_clients} clients
        ↪  requested!")

    for idx in range(num_clients):
        # Calculate port
        port = start_port + idx

        # Running the iperf3 client
        cmd = [
            "iperf3",
            "-B", ip_addresses[idx],
            "-c", server_ip,
            "-p", str(port),
            "-u",
```

```
            "-b", f"{bitrate}M",
            "-t", "300",
            "-4",
        ]

        process = subprocess.Popen(cmd)
        processes.append(process)

    # Start the status display thread
    status_thread = threading.Thread(target=display_status,
    ↪  daemon=True)
    status_thread.start()

    # Wait for all clients to finish
    for process in processes:
        process.wait()

    # Wait for the status display thread to finish
    status_thread.join()

    # Wait for 300 seconds before starting the next run
    time.sleep(300)

if __name__=='__main__':
    if len(sys.argv) < 4:
        print("Usage: python client_script.py <bitrate>
        ↪  <num_clients> <times_to_run>")
        sys.exit(1)

    bitrate = int(sys.argv[1])
    num_clients = int(sys.argv[2])
    times_to_run = int(sys.argv[3])

    if num_clients > 50:
        print("Maximum number of clients is 50.")
        sys.exit(1)

    # Run the client multiple times
    for _ in range(times_to_run):
        run_client("172.27.223.228", 5001, bitrate, num_clients)
```

The following command is employed to monitor the resource utilization of various 5G functions, including the gNB and the container that comprises the user equipments. It should be executed on one of the kubernetes master nodes.

```
$ watch kubectl top pods -n open5gs
```

The following script has been designed to perform a recursive search within the

user equipments directories to locate tests stored in JSON format. It extracts specific fields from these files, including "local_port," "start," "bytes_stream," "bps_stream," "jitter_ms," "lost_packets," and "lost_packet_percent." Subsequently, it consolidates the data from all these JSON files into a single CSV file. To execute this script, ensure that you have Python 3, the pip3 package manager, and the Pandas library installed on your system. When running the script, specify the directory you wish to search for IPERF data by providing it as an argument in the command. Additionally, the script initiates by verifying the JSON format of the files, and if any errors are detected, it omits those particular files from processing.

```
# source fixedrate_jsontocsv.py

import pandas as pd
import json
import os
import sys

def process_file(filename):
    try:
        with open(filename, 'r') as f:
            data = json.load(f)
    except json.JSONDecodeError:
        print(f"Skipping {filename} - Invalid JSON data")
        return pd.DataFrame(columns=['local_port', 'start',
        ↪  'bytes_stream', 'bps_stream', 'jitter_ms',
        ↪  'lost_packets', 'lost_packet_percent'])

    if data['intervals'] == []:
        return pd.DataFrame(columns=['local_port', 'start',
        ↪  'bytes_stream', 'bps_stream', 'jitter_ms',
        ↪  'lost_packets', 'lost_packet_percent'])


    # Extracting the required fields
   # sum_received_bytes = data['end']['sum_received']['bytes']
    local_port = data['start']['connected'][0]['local_port']

    # Create a list to hold rows
    rows = []

    for interval in data['intervals']:
        start = interval['streams'][0]['start']
        bytes_stream = interval['streams'][0]['bytes']
        bps_stream = interval['streams'][0]['bits_per_second']
        jitter_ms = interval['streams'][0]['jitter_ms']
        lost_packets = interval['streams'][0]['lost_packets']
```

```
        lost_packet_percent =
        ↪ interval['streams'][0]['lost_percent']

        rows.append([local_port, start, bytes_stream, bps_stream,
        ↪ jitter_ms, lost_packets, lost_packet_percent])

    return pd.DataFrame(rows, columns=['local_port', 'start',
    ↪ 'bytes_stream', 'bps_stream', 'jitter_ms',
    ↪ 'lost_packets', 'lost_packet_percent'])

# Process the files and concatenate them into one DataFrame
dfs = []

# Get the directory path from command-line arguments
if len(sys.argv) < 2:
    print("Please provide the directory path as an argument.")
    sys.exit()

base_directory = sys.argv[1]

# Walk through each directory and its subdirectories
for dirpath, dirnames, filenames in os.walk(base_directory):
    for file in filenames:
        if file.endswith('.json'):  # Process only JSON files
            dfs.append(process_file(os.path.join(dirpath, file)))

df = pd.concat(dfs, ignore_index=True)

# Print the DataFrame
print(df)

# Save the DataFrame to a CSV in the provided base directory
df.to_csv(os.path.join(base_directory, 'output.csv'),
↪ index=False)
```

# Appendix B

# Access Manual

In this appendix, we present an instruction manual on how to access our testbed. Initially, a user must have a Virtual Private Network (VPN) connection to the CISUC data center. The links and credentials for these technologies can be found in Appendix C.

## B.1 Accessing the Rancher Server for Cluster Management

To control the cluster from a Graphical User Interface (GUI), you must access the rancher server and enter your credentials. Once logged in, you can access the local cluster and manage nodes, pods, services, and other elements.

## B.2 To access Simu5G

Install a VNC viewer, for example, Real VNC [94]. Then enter the link and credentials given in Appendix C.

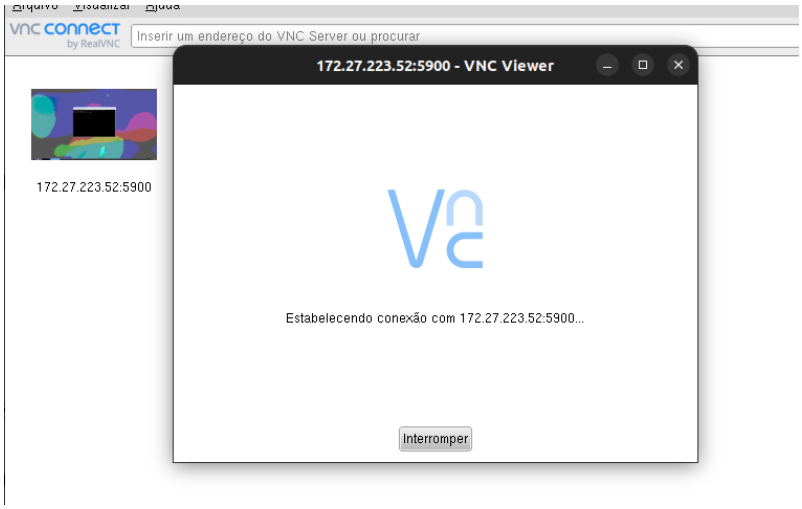


Figure B.1: VNC Viewer interface connecting to the Simu5g container.

Then run on the terminal:

```
$ /omnetpp-6.0pre11/bin/omnetpp
```

Finally, run omnetpp.ini that is inside */omnetpp-6.0pre11/samples/simu5g/simulations/demo*.

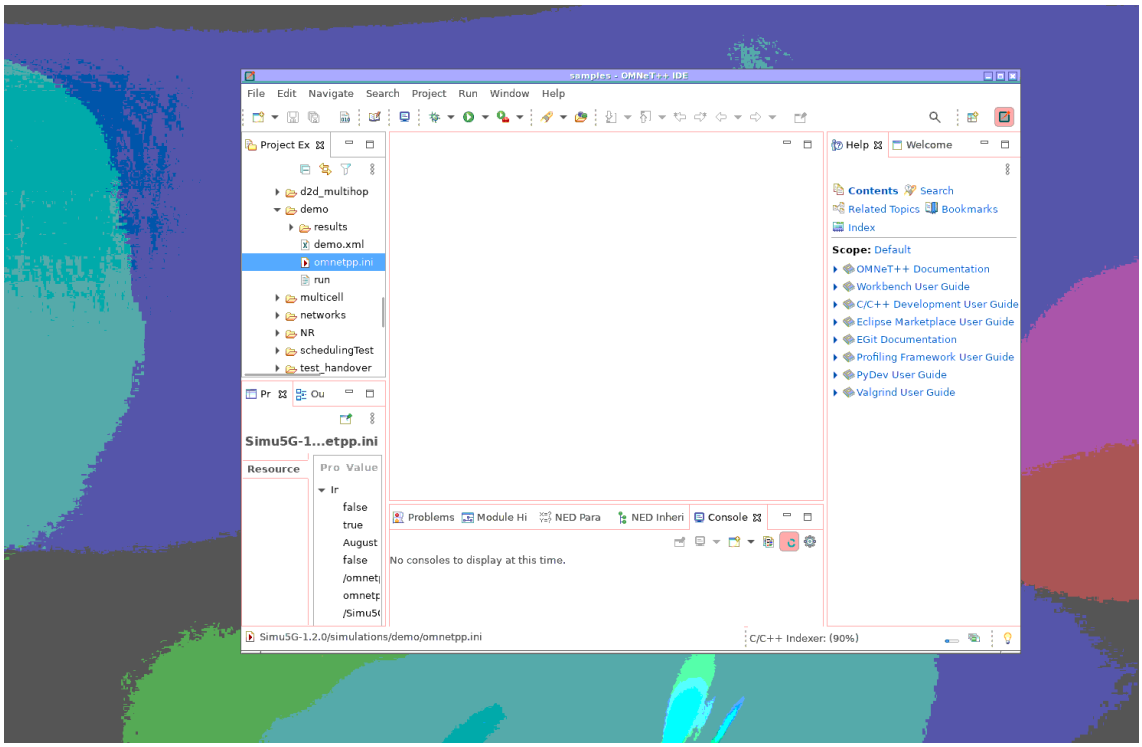


Figure B.2: Simu5g runtime interface for 5G network simulation.

For a detailed simulation of Simu5G look here [95].

## B.3 Accessing the Open5GS Web Interface

In Figure B.3, if you click on the button in the bottom right corner, you will be able to add more UE subscribers. This is necessary in order to connect the UE to the Access and Mobility Management Function (AMF) in the 5GC. By adding subscribers through this interface, you can ensure that the UE is properly connected to the 5G network.

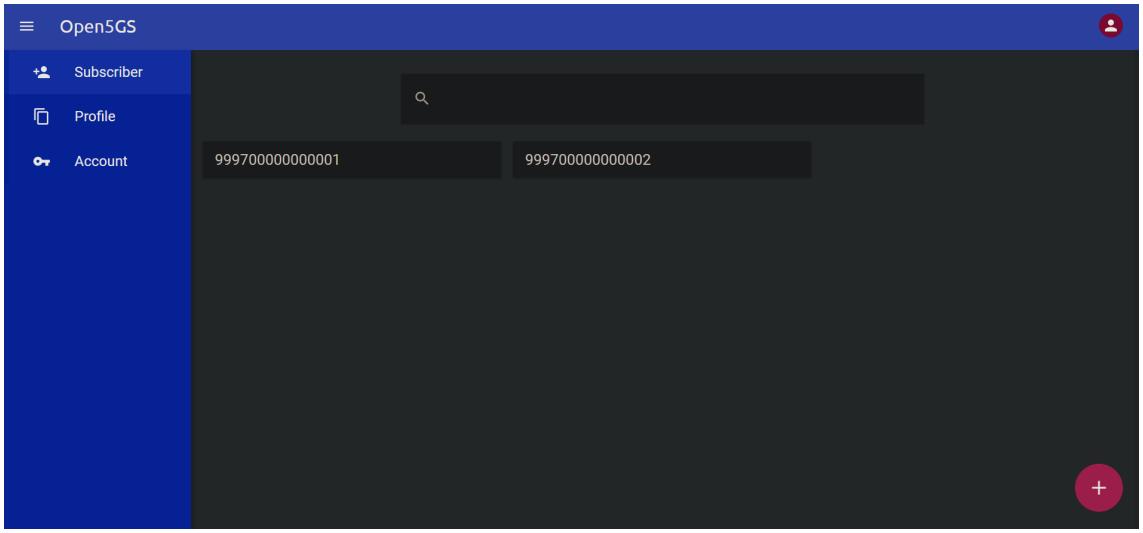


Figure B.3: Open5GS Web User Interface (UI).

## B.4 To access the UE Container

To access the pods, click on the "local" cluster and search for "Pods", or follow the link provided in Appendix C. In the pods, go to the "open5gs" namespace and click the three blue dots on the right. From there, select "Run Shell" to access the pods as shown in the figure below.

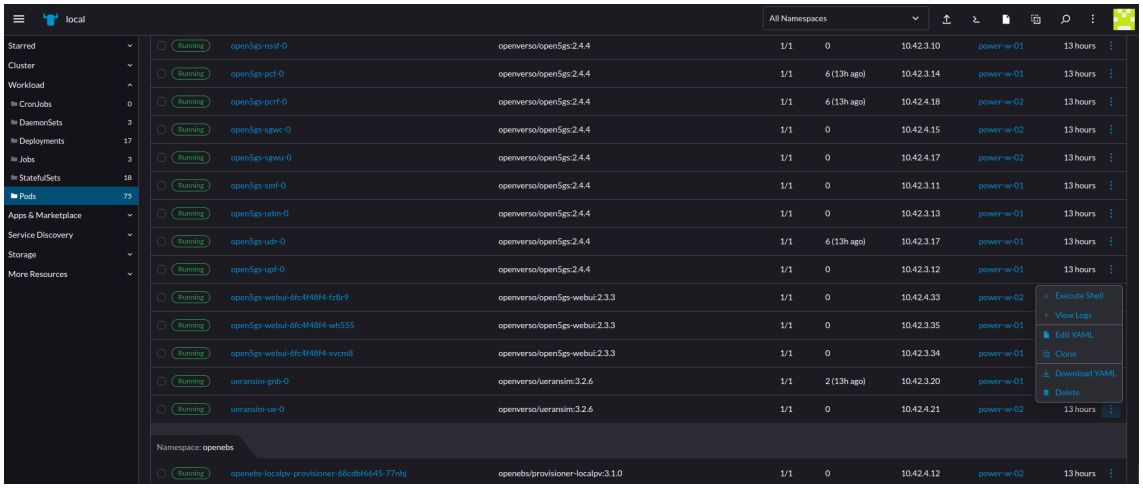


Figure B.4: UE pod.

And a terminal should appear as in the figure B.5:

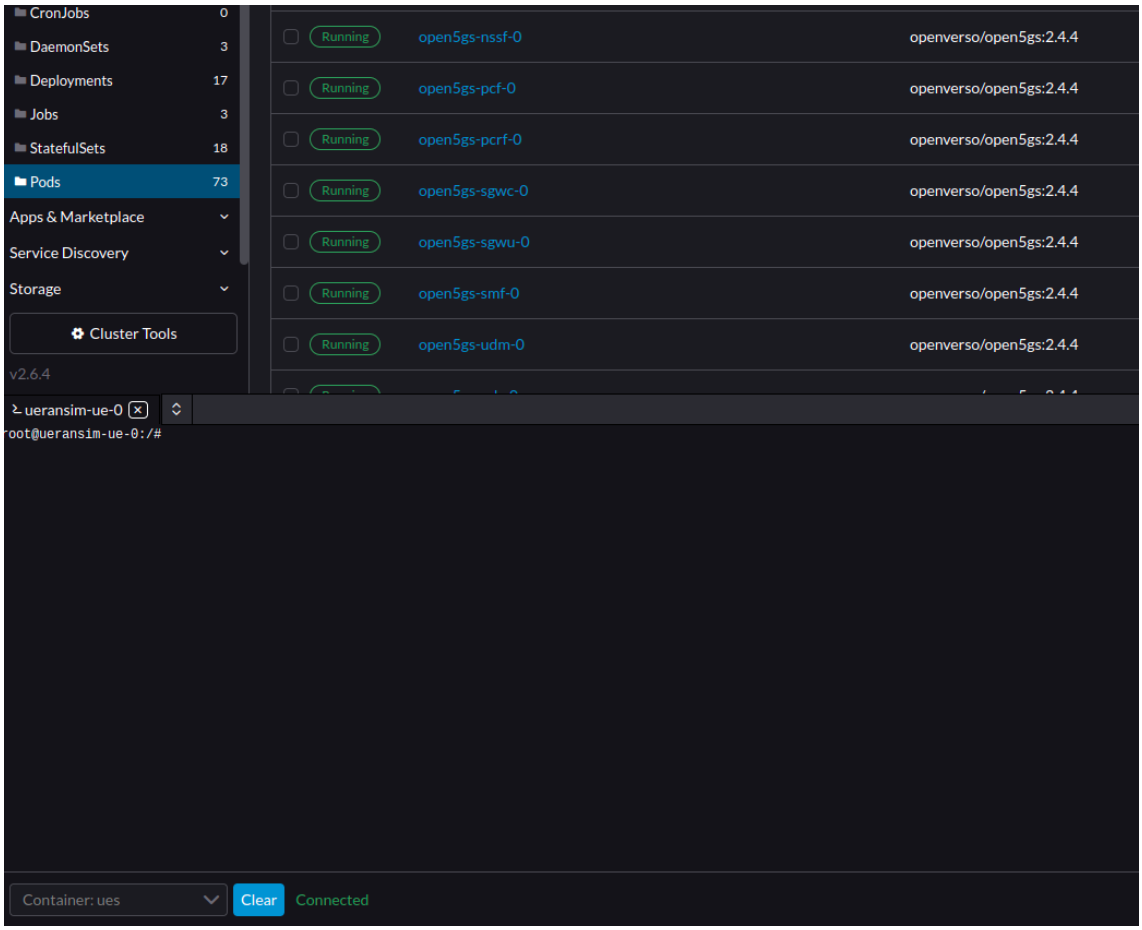


Figure B.5: UE container terminal.

This terminal will allow you to view the IP address of the "uesimtun0" network interface, which is connected to the 5GC.

## B.5 Accessing the OpenPLC Web Interface

Figure B.6 illustrates a master device synchronizing certain registers with a slave device. One important configuration step when adding a slave device is to enter the IP address of the user equipment corresponding to the "uesimtun0" network interface. This ensures that the master device can properly communicate with the slave device on the 5G network.

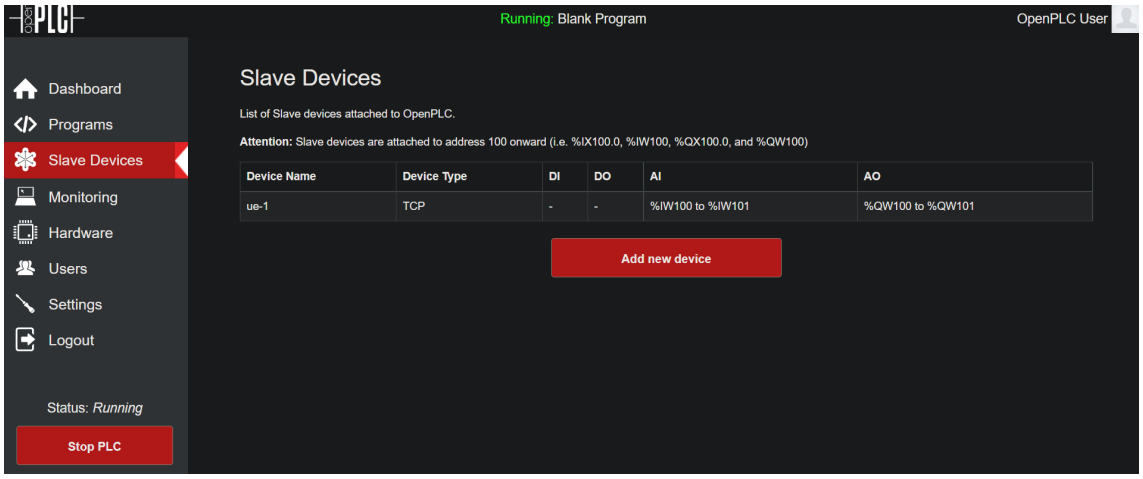


Figure B.6: Master OpenPLC device.

## B.6 To access the registers of PLC Instances

First, you will need to install a Modbus Transmission Control Protocol (TCP) client for example Modbus Poll [96]. It is important to note that the port you will need to use is 502. The links to the master and slave devices can be found in Appendix C.

When you modify the holding register at address 150 (%QW150) on a slave device, the corresponding change will be reflected at address 150 (%QW150) on the master device. In the case of discrete inputs, a change made to the register at address 1 in the slave device will be mirrored at address 801 [97] in the master device.

In Figure B.7, a reading scenario is depicted for holding registers (Function Code 3) in the master device, specifically for the ten registers starting from address 150.

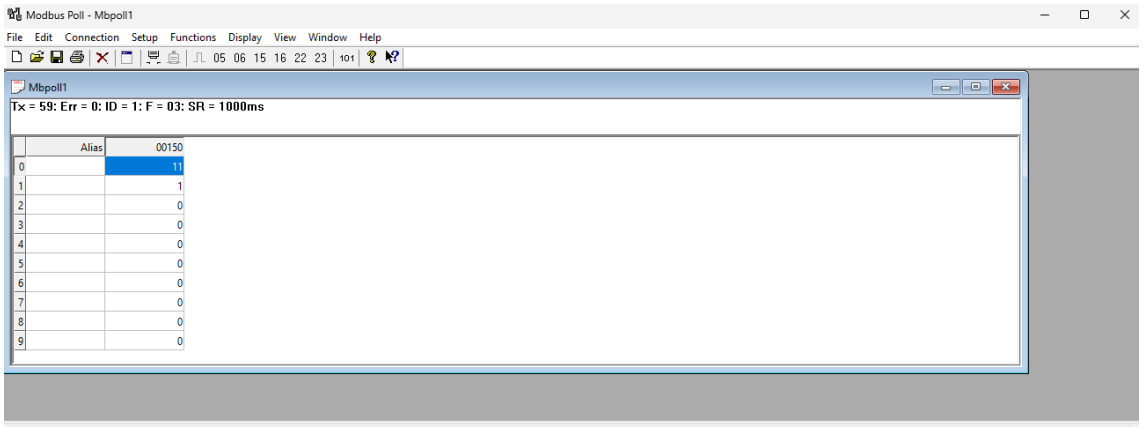


Figure B.7: Modbus poll read scenario for the slave device.

# Appendix C

# Links and credentials

This appendix presents the links and credentials for the testbed, which may be subject to change. It also provides links to the applications so that users can connect to them via a web browser or other tool.

| Title | Link | User | Password |
|---|---|---|---|
| Rancher Server | `https://172.27.223.51/dashboard/` | admin | admin12345678 |
| Simu5G | 172.27.223.52:5900 | - | power |
| Open5GS web UI | `http://172.27.223.53:3000/` | admin | 1423 |
| OpenPLC master UI | `http://172.27.223.54:8080/` | openplc | openplc |
| OpenPLC slave 1 UI | `http://172.27.223.55:8080/` | openplc | openplc |
| OpenPLC slave 2 UI | `http://172.27.223.56:8080/` | openplc | openplc |
| OpenPLC slave 3 UI | `http://172.27.223.57:8080/` | openplc | openplc |
| Raspberry PI terminal (user equipment) | `http://172.27.223.20` | power-pi | banana |

*Continues on the next page*

| Title | Link | User | Password |
|---|---|---|---|
| OpenPLC master Modbus (5G Network) | 192.168.20.100 with port 502 | - | - |
| OpenPLC master Modbus (External Access) | 172.27.223.54 with port 502 | - | - |
| OpenPLC slave 1 Modbus (5G Network) | 192.168.20.101 with port 502 | - | - |
| OpenPLC slave 1 Modbus (External Access) | 172.27.223.55 with port 502 | - | - |
| OpenPLC slave 2 Modbus (5G Network) | 192.168.20.103 with port 502 | - | - |
| OpenPLC slave 2 Modbus (External Access) | 172.27.223.56 with port 502 | - | - |
| OpenPLC slave 3 Modbus (5G Network) | 192.168.20.104 with port 502 | - | - |
| OpenPLC slave 3 Modbus (External Access) | 172.27.223.57 with port 502 | - | - |
| Physical PLC slave 4 Modbus (5G Network) | 192.168.20.105 with port 502 | - | - |

*Continues on the next page*

| **Title** | **Link** | **User** | **Password** |
|---|---|---|---|
| Gitlab account | https://gitlab.com | powerdei | **Password:** BananaGuest3# and **Token:** glpat-PJTpwaomxrKZbc4MBueE |

Table C.1: Login information for various systems.