Implementation and Evaluation of a 5G network with End-to-End connectivity using srsRAN

Bachelorarbeit zur Erlangung des Bachelor-Grades
Bachelor of Science im Studiengang Technische Informatik
an der Fakultät für Informations-, Medien- und Elektrotechnik
der Technischen Hochschule Köln

vorgelegt von:	Pascal Sthamer
Matrikel-Nr.:	11134814
Adresse:	Escher Str. 8a
	50767 Köln
	pascal.sthamer@smail.th-koeln.de
eingereicht bei:	Prof. Dr. Andreas Grebe
Zweitgutachter:	B.Sc. Jonas Dieterich

Köln, 03. Juni 2024

Contents

Acronyms

3GPP 3rd Generation Partnership Project

AMF Access and Mobility Management Function

AUSF Authentication Server Function

BS Base Station

BSF Binding Support Function

COTS Commercial off-the-shelf

CPRI Common Public Radio Interface

CQI Channel Quality Indicator

CU Central Unit

DLT Data Link Type

DU Distributed Unit

E2SM E2 Service Model

EI Enrichment Information

eMBB enhanced Mobile broadband

FCAPS Fault, Configuration, Accounting, Performance and Security

FFT Fast Fourier Transform

GPRS General Packet Radio System

IMEI International Mobile Equipment Identity

IMSI International Mobile Subscriber Identity

ITU International Telecommunication Union

KPI Key Performance Indicator

KPM Key Performance Measurement

MAC Medium Access Control

MCC Mobile Country Code

MCS Modulation and Coding Scheme

MIMO multiple-input multiple-output

mMTC massive machine-type communication

MNC Mobile Network Code

MSIN Mobile Subscriber Identification Number

NAS Non-access stratum

Near-RT RIC near-real-time RIC

NFV network function virtualisation

Non-RT RIC non-real-time RIC

NR New Radio

NRF NF Repository Function

NSA Non-Standalone

NSSAI Network Slice Selection Assistance Information

NSSF Network Slice Selection Function

O-CU O-RAN Central Unit

O-CU-CP O-RAN Central Unit - Control Plane

O-CU-UP O-RAN Central Unit - User Plane

O-DU O-RAN Distributed Unit

O-RAN Open RAN

O-RAN SC O-RAN Software Community

O-RU O-RAN Radio Unit

OFH Open Fronthaul

PCF Policy and Charging Function

PDCP Packet Data Convergence Protocol

PDU protocol data unit

PLMN Public Land Mobile Network

QoS Quality of Service

RAI RAN Analytics Information

RAN Radio Access Network

RDP Remote Desktop Protocol

RF Radio Frequency

RIC RAN Intelligent Controller

RLC Radio Link Control

RRC Radio Resource Control

RSRP Reference Signal Received Power

RU Radio Unit

SA Standalone

SCP Service Communication Proxy

SCTP Stream Control Transmission Protocol

SD Slice Differentiator

SDAP Service Data Adaptation Protocol

SDL Shared Data Layer

SDN software-defined networking

SDU Service Data Unit

SIM Subscriber Identity Module

SMF Session Management Function

SMO Service Management and Orchestration Framework

SNR Signal to Noise Ratio

SRS Software Radio Systems

SST Slice/Service Type

UDM	Unified Data Management
UDR	Unified Data Repository
UE	User Equipment
UPF	User Plane Function
uRLLC	ultra-reliable low-latency communication

List of Figures

List of Tables

1 Introduction

5G, the fifth generation of wireless technology, represents a monumental leap forward in the realm of communications. It promises gigabit speeds, ultra-low latency, and massive connectivity. Besides the traditional use case of multimedia applications, 5G aims to enable new applications and services, ranging from autonomous vehicles and remote surgery to immersive augmented reality experiences.

With 5G came a paradigm shift towards an open Radio Access Network (RAN). Unlike traditional proprietary systems, Open RAN (O-RAN) embraces open interfaces and standardized protocols. It introduces a disaggregated approach to network architecture and decouples hardware and software components, fostering interoperability and innovation. Driven by the need for agility, vendor neutrality, and accelerated innovation cycles, network operators and vendors increasingly turn to open RAN solutions for their network infrastructure. According to the O-RAN Alliance, 32 mobile network operators have committed to the open RAN, and there exist more than 300 O-RAN companies in total [1]. This transition departs from the traditional vendor lock-in, enabling collaboration and community-driven development. O-RAN enables researchers worldwide to participate in RAN development without access to costly proprietary solutions. Companies like Software Radio Systems (SRS) started to develop open-source RAN solutions based on O-RAN specifications.

1.1 Goals

This work implements a 5G network based on open-source software. In addition to a 5G core network, this includes a RAN and a simulated User Equipment (UE) based on the srsRAN software to achieve End-to-End connectivity. End-to-End connectivity means a UE can reach the internet over a simulated radio connection with packets sent over the RAN into the 5G core network, which forwards them to the internet. This work aims to provide a functional 5G RAN deployment, which other researchers can use in the future. Potential research areas that benefit from a test network like this include RAN security research, training machine learning models for network management, and xApp development. This work also evaluates the current state of open-source O-RAN solutions and compares two RAN Intelligent Controllers (RICs): FlexRIC and the O-RAN Software Community (O-RAN SC) RIC.

The following goals are defined for the test network implemented in this work:

1. *End-to-End connectivity* - A simulated UE should have internet connectivity via the simulated RAN.
2. *RAN Intelligent Controller* - A near-real-time RIC (Near-RT RIC) should be part of the test network, and it should be possible to read RAN Key Performance Measurements (KPMs) from xApps.
3. *Easy and portable deployment* - The test network should run on general-purpose hardware on a Linux operating system.
4. *Visibility* - RAN metrics, logs, and packet captures should be exposed to gain insights into the test network.
5. *Well documented* - The test network architecture and deployment should be properly explained so that future researchers can use it as a foundation.
6. *Open-source O-RAN compatible software* - No commercial software solutions.
7. *Reproducible deployments* - Upstream dependency updates should not lead to different or broken deployments.
8. *Extendability and integration* - It should be possible to upgrade existing and integrate new RAN components in the future.
9. *External Near-RT RIC and core* - It should be possible to integrate an externally deployed Near-RT RIC and 5G core into the test network.

1.2 Project partners

The University of Applied Sciences (TH) Cologne research group for data networks (DN.Lab) provided the server hardware for this work. The DN.Lab operates a 5G core network and a O-RAN SC Near-RT RIC for the 5G-FORAN project. The project is a collaboration between TH Cologne and PROCYDE GmbH, supported by the Federal Office for Information Security. It aims to address security challenges in O-RANs. The project is divided into two sub-projects: 5G-FORAN-ATTACK, which simulates possible attack scenarios, and 5G-FORAN-DFIR, which tries to detect and mitigate attacks. This work contributes to the 5G-FORAN project by providing a more complete 5G network deployment. The deployment is meant to be extended with a gNodeB and simulated UEs based on the results of this work. This enables more extensive security research by making it possible to perform attacks from a UE, perform attacks against gNodeB components and UEs, and analyze how attacks targeting the 5G core network impact RAN components and end users.

1.3 Structure

Chapter 2 explains 5G fundamentals. Chapter 3 explains the concept behind O-RAN, while chapter 4 is about RICs. Chapter 5 introduces software components and fundamentals used to build the test network. Chapter 6 is about the practical part of this work. It describes the design of the test network and explains deployed network services. Chapter 7 explains how UE traffic flows through the network. Expanding the test network by RICs is described in chapter 8. Chapter 9 concludes this work and gives an outlook into the future development of srsRAN and O-RAN in general.

2 5G Fundamentals

The requirements for mobile networks are rising. There is a need for increased bandwidth, lower latency, and the possibility to connect up to one million devices per square kilometer. 5G aims to fulfill these requirements. Three main use cases have been defined by the International Telecommunication Union (ITU) for 5G [2]:

- *enhanced Mobile broadband (eMBB)* - minimum user experienced data rate of 100 Mbit/s downlink, 50 Mbit/s uplink
- *massive machine-type communication (mMTC)* - ability to connect at least 1 000 000 devices per km^2
- *ultra-reliable low-latency communication (uRLLC)* - 99.999% probability of transmitting a layer 2 protocol data unit (PDU) of 32 bytes successfully within 1ms

The test network implemented in this work is not meant to fulfill any of the beforementioned goals.

2.1 Standardization

Standards are technical specifications that are

- accessible to the public,
- developed through cooperation and consensus of all involved parties
- based on well-established scientific and technical results,
- aimed at wide support and social welfare,
- supported by a regional, national, or international institution [3].

Standards are essential in mobile networks, where devices of different manufacturers should be compatible across different network operators worldwide. They enable global roaming and ensure quality. The complex standardization process involves regional and global regulatory bodies, national administrators, and industry forums. The 3rd Generation Partnership Project (3GPP) is a worldwide standardization body for the standardization of mobile radio systems. It was founded in 1999 and developed standards like GSM/EDGE and LTE. Seven telecommunication standard development organizations worldwide are united under the 3GPP: ARIB, atis, CCSA, ETSI, tsdsi, TTA, and TTC. Regarding 5G, the 3GPP and ITU are the two essential organizations that define 5G. ITU is a specialized agency of the United Nations for information and communication technologies. It is responsible for defining 5G requirements and spectrum regulation globally. Work on 5G specifications is still ongoing. Release 15 is the first 5G release and was approved in March 2017. Releases 16 and 17 followed in June 2018 and December 2019. Release 18 is expected to be approved in June 2024 [4]. This work is based on 3GPP Release 17.

2.2 New Radio

5G New Radio (NR) is the global standard for 5G wireless technology. It is designed to offer faster download and upload speeds, lower latency, and support more connected devices than previous generations of wireless technology. 5G NR introduces new technologies such as massive multiple-input multiple-output (MIMO), network-based sensing, machine learning-based digital signal processing, and the usage of diverse frequency bands. These technologies enable higher data rates, greater capacity, lower latency, and more efficient spectrum utilization and power usage. Additionally, 5G NR has the potential to enable new use cases such as self-driving cars, smart cities, and remote healthcare. A wide range of advanced technologies have been introduced that enable higher data rates, greater capacity, lower latency, and more efficient spectrum utilization and power usage.

2.3 Mobile protocol stack

This section provides an overview of the 5G protocol stack. Understanding the individual layers is crucial to understanding how the 5G Base Station (BS) is split into multiple components, as explained in chapter 3.3. A 5G BS is referred to as gNodeB. Figure 1 contains a visual representation of the mobile protocol stack.

The Radio Frequency (RF) layer is responsible for the transmission and reception of wireless signals over the air interface between the UE and the BS. It is responsible for handling the electromagnetic waves that carry the data transmitted between the UE and the BS. The RF layer consists of several components, including the antennas, the amplifiers, and the transmitters and receivers. The antennas transmit and receive electromagnetic waves, while the amplifiers amplify the signals to ensure they can travel long distances. The transmitters and receivers are responsible for the modulation and demodulation of the signals to ensure that the data can be transmitted and received correctly. [5]

The physical layer is responsible for transmitting and receiving data over the air interface between the UE and the BS. It provides modulation and coding schemes to convert digital data into analog signals that can be transmitted over the air. The physical layer also handles functions such as channel coding, modulation, and demodulation, as well as the management of radio resources. [5] The RF and physical layers are emulated in this work as described in chapter 5.6.

The Medium Access Control (MAC) layer controls access to the network and provides the necessary control mechanisms for accessing the network. It is responsible for the allocation of radio resources, the scheduling of transmissions, and the management of the power control mechanisms. The MAC layer also handles error control, flow control, and congestion control functions. [5]

The Radio Link Control (RLC) layer is responsible for several functions, including segmentation and reassembly of data packets, error correction, and flow control. It is responsible for breaking down the data packets into smaller segments to enable efficient transmission over the air interface and then reassembling them at the receiving end. The RLC layer also provides error correction mechanisms to detect and correct any errors that may occur during transmission, ensuring that the data is transmitted reliably and without loss. In addition, the RLC layer provides flow control mechanisms to ensure that the data is transmitted at an appropriate rate for the receiving end. This helps to prevent congestion and ensure that the network is used efficiently. [5]

The Packet Data Convergence Protocol (PDCP) layer is responsible for the compression and decompression of the IP packets transmitted between the UE and the core network. It handles header compression and decompression functions, encryption and decryption, packet reordering, and duplication detection. The PDCP layer also provides the necessary mechanisms to differentiate the different types of traffic, such as voice, video, and data. [5]

The Service Data Adaptation Protocol (SDAP) layer maps data flows between the UE and the core network. It manages the Quality of Service (QoS) parameters, including prioritization, scheduling, and allocating resources to different flows. The SDAP layer also handles functions such as the header compression and decompression and the multiplexing and demultiplexing of data flows. [5]

The Radio Resource Control (RRC) layer is responsible for the establishment, maintenance, and release of radio connections between the UE and the BS. It handles functions such as the configuration of radio bearers, the management of handovers, and the management of the radio resource allocation. The RRC layer is also responsible for the signaling messages that control the RAN. [5]

The Non-access stratum (NAS) layer enables authentication, security, and idle procedures such as paging. It operates between the Access and Mobility Management Function (AMF) and the UE. [5]

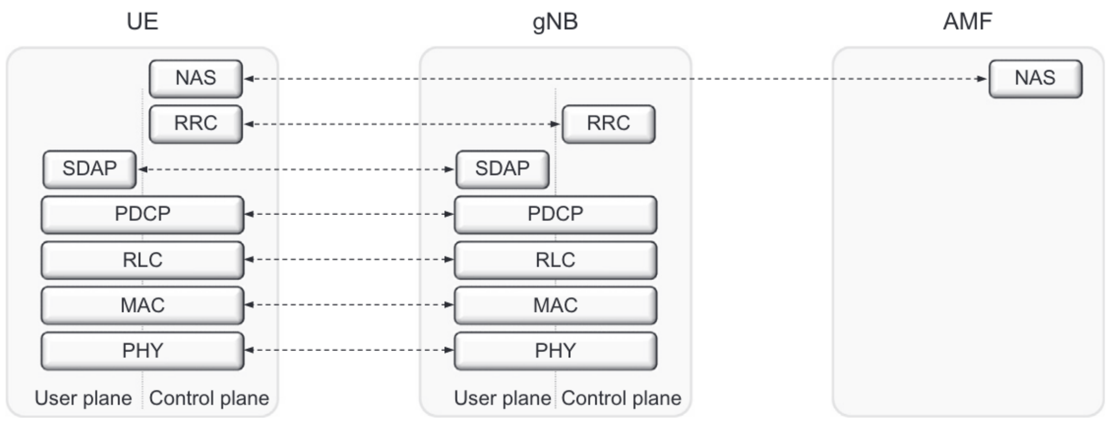


Figure 1 5G mobile protocol stack [5]

2.4 New Generation RAN

In addition to the improved radio protocol, 5G introduces a new core architecture. The 5G core network is responsible for End-to-End connection setup, packet forwarding, mobility management, authentication, charging, and more. 5G uses software-defined networking (SDN) to abstract physical networking resources. The control and user plane are separated to allow independent scaling and evolution. 5G introduces network function virtualisation (NFV): on-demand, individually scalable network functions, which run on Commercial off-the-shelf (COTS) hardware. Each network function is responsible for a different 5G service. Chapter 6.2 gives an overview of different 5G network services.

Network slicing allows the provision of multiple virtual networks on top of a shared physical network. This allows to provide different functionalities and resources per customer. Network slicing is supported using the Network Slice Selection Assistance Information (NSSAI) header. It is split into the Slice/Service Type (SST) and Slice Differentiator (SD). The SST refers to one of the three main use cases of 5G: eMBB, uRLLC, and mMTC. The SD is used to differentiate slices of the same type.

Many existing network providers decided to deploy 5G Non-Standalone (NSA) as a bridge technology. They use 5G Base Stations but keep the existing 4G core network. This allows an easier and more cost-effective migration towards 5G. NSA does not support NSSAI, which results in network slicing not being supported. In Germany, most providers started with 5G NSA but are now rolling out 5G Standalone (SA) for their customers. Teléfonica and Vodafone have already deployed 5G SA for non-business customers and refer to it as 5G+ in their marketing strategy [6], [7]. Deutsche Telekom is rolling out 5G SA for non-business customers this year [8]. Because NSA only serves as a bridge technology and is missing important 5G features, this work focuses on 5G SA.

2.5 GTP Tunnelling

Like previous generations, 5G uses the General Packet Radio System (GPRS) Tunnelling Protocol (GTP). It is used at multiple interfaces in a 5G system that transport user data, including F1-u, Xn, N3, N9, and N19 interfaces. GTP encapsulates IP packets sent either in the downlink direction from the external network to the UE or in the uplink direction from the UE to the external network. [9]

3 Open Radio Access Network

Today, the RAN mainly consists of proprietary hardware and software from a limited number of vendors. Network operators see these solutions as black-box solutions that implement every layer of the cellular protocol stack, which comes at the cost of limited reconfigurability and vendor lock-in. [10] This is starting to change as new specifications standardize RAN technology and enable interoperability between different vendors.

3.1 O-RAN Alliance

A significant player towards the shift to an open RAN is the O-RAN Alliance. The O-RAN Alliance "aims to drive the mobile industry towards an ecosystem of innovative, multi-vendor, interoperable, and autonomous RAN, with reduced cost, improved performance, and greater agility" [1]. It was founded in 2018 by leading mobile network operators, such as AT&T, China Mobile, and Deutsche Telekom. Eleven technical working groups are working on different topics, such as overall architecture, RIC, interface specification, cloudification, and security. They provide standardizations, white papers, and resources for the open RAN. The standards published by the O-RAN Alliance aim to be compatible with 3GPP standards. [1] Based on O-RAN specifications, multiple open-source solutions from third parties have emerged that provide O-RAN compatible implementations. In addition to that, the O-RAN SC is working on reference implementations for the specifications published by the O-RAN Alliance.

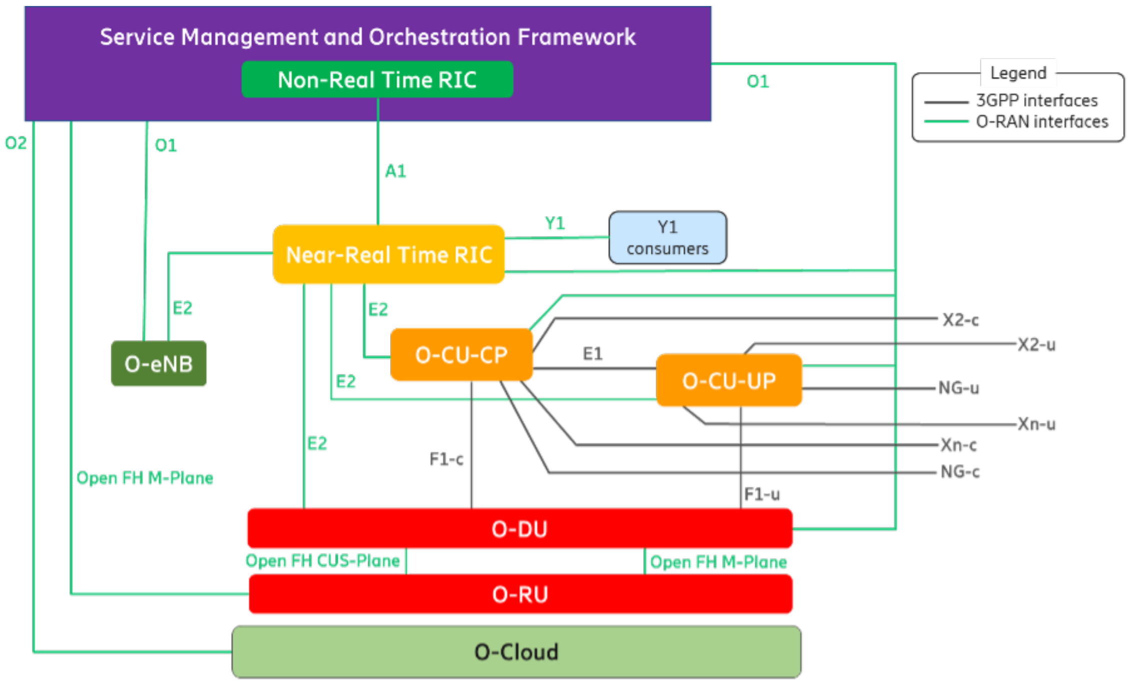


Figure 2 O-RAN Architecture Overview [11]

3.2 O-RAN Architecture Overview

O-RAN Work Group 1 defines the overall architecture of O-RAN. Figure 2 gives an overview of the O-RAN architecture. The BS is split into Central Unit (CU), Distributed Unit (DU), and Radio Unit (RU) as specified by 3GPP [12]. In the context of O-RAN, the RU is referred to as O-RAN Radio Unit (O-RU), the DU is referred to as O-RAN Distributed Unit (O-DU), and the CU is further divided into O-RAN Central Unit - Control Plane (O-CU-CP) and O-RAN Central Unit - User Plane (O-CU-UP). This split is further described in chapter 3.3. 3GPP interfaces connect the O-CU-CP, O-CU-UP, and O-DU to a 5G Core network, as specified in TS 23.501 [13]. In addition to that, the O-RAN E2 interface connects them to the Near-RT RIC. The components to which the Near-RT RIC connects via the E2 interface are called E2 nodes.

The Near-RT RIC enables near real-time control over connected E2 nodes. E2 nodes collect KPMs, such as throughput, channel quality, or amount of connected UEs. The Near-RT RIC hosts multiple xApps, which can subscribe to these KPMs and perform various actions based on available data and configuration, including network slicing and scheduling. [11] RAN analytics services are exposed via the Y1 interface to authorized consumers [14].

The Service Management and Orchestration Framework (SMO) hosts the non-real-time RIC (Non-RT RIC). It provides policy-based guidance and enrichment information to the Near-RT RICs. Similar to the Near-RT RIC, it hosts rApps that leverage functionality of the SMO and Non-RT RIC to perform radio resource management, data analytics, and provide enrichment information. [15]

The O-Cloud refers to a cloud computing platform that provides the infrastructure to host the O-RAN architecture. It provides necessary software components, including but not limited to a virtual machine hypervisor, operating systems, and a container runtime. The O2 interface exposes infrastructure management capabilities. This includes discovering existing infrastructure, deploying new RAN components on demand, scaling them as needed, and managing configuration. This allows O-RAN to manage its own deployment. The O-eNB can optionally be deployed for LTE support, which is out of the scope of this work. [11]

Figure 3 shows how the architecture of O-RAN differs from traditional approaches. O-RAN uses a tiered architecture. RICs are hosted in a regional O-Cloud, while O-CU-UP, O-CU-CP, and O-DU are hosted in an O-Cloud on the edge, which is physically more close to the cell sites. This is required because a high-bandwidth and low-latency connection is required between the O-DU and O-RU. The cell sites contain the O-RU, the only component that cannot be virtualized in cloud architecture.

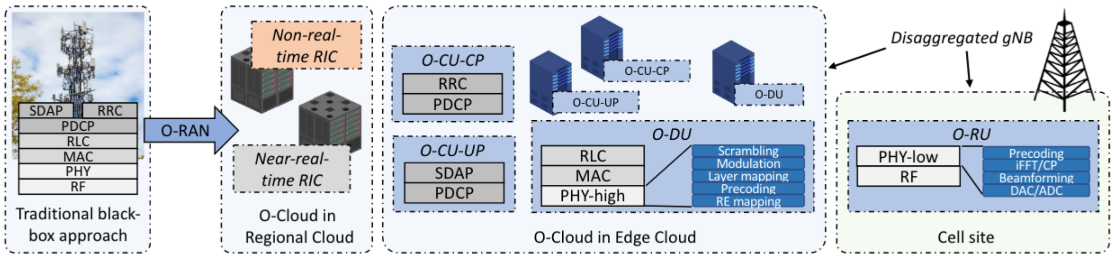


Figure 3 Evolution of traditional black-box base station solutions towards an open RAN [10]

3.3 Disaggregated gNodeB architecture

The 5G BS is a logical component called gNodeB. To accommodate increased complexity, the gNodeB is split into different functional units. Instead of implementing the whole mobile protocol stack on the cell site, higher-layer functionality is moved to a virtualized cloud environment. This makes it easier and more cost-efficient to maintain cell sites [10]. Figure 3 illustrates the disaggregated gNodeB architecture. The O-RAN Central Unit (O-CU) implements the higher level of the protocol stack and is split into the control plane and the user plane. The control plane implements the RRC and PDCP layers. The user plane implements the SDAP and PDCP layers. The DU implements the RLC and MAC layer. The physical layer is split using Open Fronthaul (OFH) option 7.2 as explained in chapter 3.3.1.

A cell site is connected to a single O-DU, and each O-DU is managed by a single O-CU. A single O-CU can manage multiple O-DUs, and a single O-DU can be connected to multiple cell sites, creating a tree-like network topology.

Figure 4 shows the disaggregated gNodeB architecture in more detail. It shows the different components and the mobile protocol stack layers they implement. Internal interfaces interconnect the gNodeB components, and external interfaces that connect the gNodeB to external network services are visible. Compared to figure 3, the O-DU is further split into O-DU-high and O-DU-low. O-DU-high contains the RLC and MAC functionality while O-DU-low is only responsible for the physical layer. The following sections describe the gNodeB internal interfaces and the NG interface. The external O2 and E2 interfaces are explained in chapter 4.

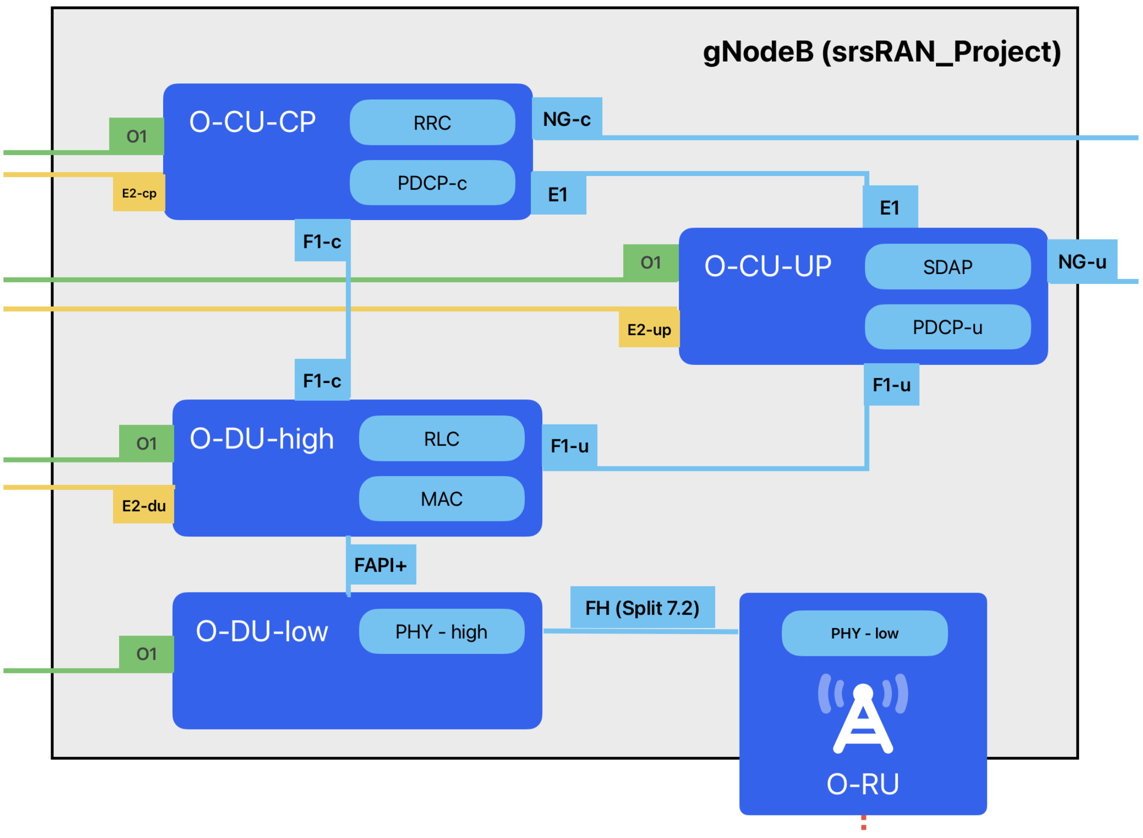


Figure 4 Disaggregated gNodeB architecture

3.3.1 Open Fronthaul interface

The OFH interface standardizes the communication between the DU and RU. The interface allows for advanced coordination and optimization techniques between the DUs and RUs, enhancing network efficiency, capacity, and coverage, thus improving the overall performance of the 5G network. The standardization process is primarily carried out by the 3GPP and O-RAN Work Group 4: The Open Fronthaul Interfaces Work Group. The interface is divided into two main planes: Control, User, Synchronization (CUS) plane, and Management (M) plane.

The OFH initially inherited aspects from Common Public Radio Interface (CPRI) and its more flexible and bandwidth-efficient version, eCPRI. These protocols define the connectivity for passing CUS data. OFH standardizes split points, indicating where the division of baseband processing tasks occurs between the DU and RU. Each of these split options presents a tradeoff between complexity, performance, and the cost of the transport network required to connect the DU and RU. Higher-level splits tend to reduce the bandwidth requirements on the fronthaul network but may increase the processing workload on the CUs. Conversely, lower-level splits demand more from the fronthaul regarding bandwidth and latency. However, they can offer better performance in terms of network efficiency and capability to support advanced features like beamforming and MIMO. There exist many different split options. This work uses split option 7.2, recommended by O-RAN. In split option 7.2 shown in figure 3, the physical layer is split into PHY-high implemented on the DU and PHY-low implemented on the RU. The DU is responsible for frequency-domain functionalities, including scrambling, modulation, layer mapping, and precoding. The RU only performs time-domain functionalities like Fast Fourier Transform (FFT), cyclic prefix operations, and beamforming, which makes it inexpensive and easy to deploy. [16]

The OFH M-plane supports NETCONF/YANG-based configuration of O-RUs. As shown in figure 2, it allows an SMO and O-DU to communicate with O-RUs. The OFH M-plane supports management features like startup installation and enables Fault, Configuration, Accounting, Performance and Security (FCAPS) functionality. [17]

3.3.2 F1 interface

The F1 interface connects the O-DU to the O-CU-CP and O-CU-UP. Specifically, the F1-u interface connects to the user plane, while the F1-c connects to the control plane. The application protocol F1AP is implemented on top of Stream Control Transmission Protocol (SCTP). Its main functions are UE context management, RRC message transfer, warning message transmission, system information, and paging. [18]

3.3.3 E1 interface

The E1 interface is a point-to-point interface between CU user plane and control plane. The application protocol is called E1AP and uses the SCTP protocol on top of IP. The interface exchanges signaling information between the endpoints and separates the control and data plane. [19], [20]

3.3.4 FAPI+ interface

The FAPI+ interface connects the O-DU-high and O-DU-low. It is responsible for communication between the physical and MAC layer. It specifies the interface for configuration, control, and data plane services between the layers. Splitting the O-DU into high and low has the advantage that different 5G PHY implementations from different vendors can be used.

3.3.5 NG interface

The NG interface connects the gNodeB to the 5G core network, specifically to the AMF. It is split into two subinterfaces.

The NG-c interface connects the O-CU-CP to the AMF. It is responsible for signaling and controlling information and uses the NGAP protocol. It establishes and manages user sessions, mobility management, and authentication. [21] The NG-c interface is also referred to as the N2 interface.

The NG-u interface connects the O-CU-UP to the AMF. It carries the user data traffic from the gNodeB to the 5G core. A GTP tunnel between the gNodeB and the User Plane Function (UPF) is established per session to forward user data. The interface between the gNodeB and the UPF is called the N3 interface. The gNodeB is not connected directly to the UPF. All traffic flows through the AMF.

4 RAN Intelligent Controller

The O-RAN Alliance introduces RICs, which are programmable, logical controllers. Data pipelines stream and aggregate KPMs from all RAN components. Together with contextual information about the network infrastructure, this grants a centralized view of the network. Network operators can deploy RIC applications, which can automatically optimize the network. This includes network and RAN slicing, load balancing, handovers, and scheduling policies. There are two types of RICs: the Non-RT RIC and the Near-RT RIC. [10] They are explained in more detail in this chapter.

4.1 Near-RT RIC

The Near-RT RIC enables network control in near-real-time, typically in the range of 10ms to 1s. It is deployed at the edge and terminates the O1, A1, Y1, and E2 interface (see figure 2). It is connected to a single Non-RT RIC via the A1 interface and to the SMO framework via the O1 interface. Multiple Y1 consumers can be connected via the Y1 interface. A single Near-RT RIC is connected to multiple E2 nodes via the E2 interface. The Near-RT RIC is programmable using xApps.

Figure 5 shows the architecture of a Near-RT RIC. It provides an internal messaging infrastructure for communication across the Near-RT RIC and xApps. It connects interface terminations, platform services, and xApps with each other. Subscription management provides functionality for xApps to perform E2 subscriptions. It controls which xApps can access which E2 messages and deduplicates multiple identical subscription requests. The security subsystem is meant to detect malicious xApps and prevent the leakage of sensitive RAN information. The conflict mitigation gets active when multiple xApps try to apply different, conflicting configurations. A Shared Data Layer (SDL) stores information on the RAN, e.g., a list of registered E2 nodes and connected users. [22]

The following sections describe the function of each interface, as well as xApps.

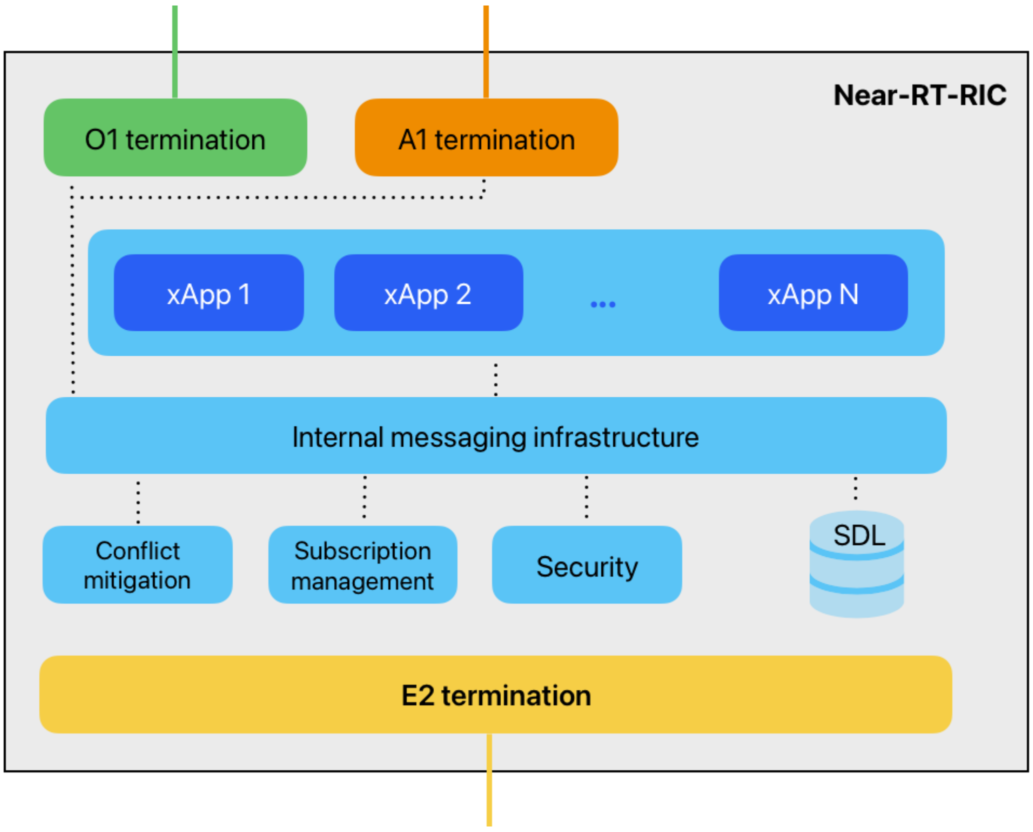


Figure 5 Near-RT RIC architecture

4.1.1 E2 interface

The E2 interface, defined by the O-RAN Alliance, connects the Near-RT RIC with the disaggregated E2 node: O-CU-CP, O-CU-UP, and O-DU. It is a point-to-point interface for exchanging information between the Near-RT RIC and the E2 node. The transport protocol used by the E2 interface is SCTP. The application protocol is called E2AP. For an E2 node, the E2 connection is optional, meaning the gNodeB should continue to work even when the E2 connection fails or is not configured. The O-RAN alliance defines E2 Service Models (E2SMs). These are function-specific protocols that are implemented on top of E2AP. An E2SM defines contracts between xApps and E2 nodes. One example is the E2SM-KPM, which defines the KPMs an E2 node might expose. [23]

4.1.2 E2 setup process

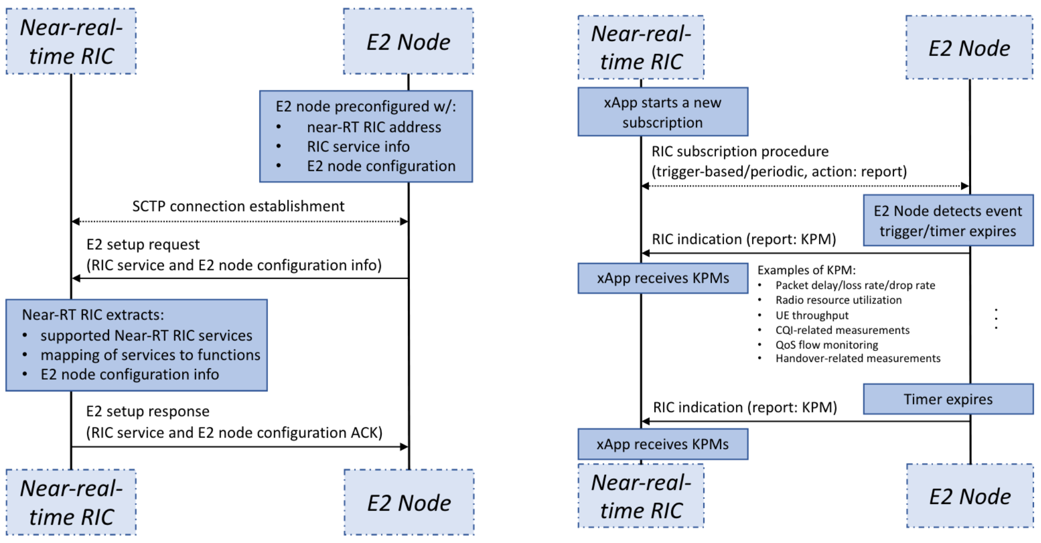
For an E2 node to be able to connect to the Near-RT RIC, it needs to be preconfigured with the Near-RT RIC address, information about which RIC services are supported (e.g., E2SM-KPM to expose RAN KPMs), and E2 information, such as the gNodeB ID. The E2 node then establishes a SCTP connection with the Near-RT RIC. After establishing the connection, it sends an E2SetupReqest message containing the preconfigured list of supported services and E2 information. The Near-RT

RIC answers with an E2SetupResponse, which acknowledges the configuration and includes a list of accepted services. At this point, the E2 connection is established.

4.1.3 E2 subscription process

Once the E2 setup process has been completed, xApps can initiate subscriptions via the Near-RT RIC. The subscription specifies an action that should be performed and a trigger that defines when the action should be performed. One possible action is *report*, which indicates that the E2 node should send information like KPMs to the Near-RT RIC. The trigger can be a specific event or a timer. Once the specified trigger is detected by the E2 Node (e.g., a timer expires), a RIC indication message is sent to the Near-RT RIC containing the requested information. This process allows xApps to receive the information they need when they need it.

Figure 6 shows the setup process on the left and the subscription process on the right.


Near-real-time RIC
E2 Node
E2 node preconfigured w/:
• near-RT RIC address
• RIC service info
• E2 node configuration
SCTP connection establishment
E2 setup request
(RIC service and E2 node configuration info)
Near-RT RIC extracts:
• supported Near-RT RIC services
• mapping of services to functions
• E2 node configuration info
E2 setup response
(RIC service and E2 node configuration ACK)
Near-real-time RIC
E2 Node
Near-real-time RIC
E2 Node
xApp starts a new subscription
RIC subscription procedure
(trigger-based/periodic, action: report)
E2 Node detects event trigger/timer expires
RIC indication (report: KPM)
xApp receives KPMs
Examples of KPM:
• Packet delay/loss rate/drop rate
• Radio resource utilization
• UE throughput
• CQI-related measurements
• QoS flow monitoring
• Handover-related measurements
Timer expires
RIC indication (report: KPM)
xApp receives KPMs
Near-real-time RIC
E2 Node

(a) Procedure for the setup of an E2 session between the near-RT RIC and an E2 node. The procedure is initiated by the E2 node which interacts with the near-RT RIC.

(b) Procedures related to the streaming of KPMs from the E2 node to the near-RT RIC. The subscription procedure is started by an xApp on the near-RT RIC, which then receives the reports.

Figure 6 E2 Setup and Subscription process [10]

4.1.4 xApps

An xApp is an application designed to run on a Near-RT RIC. It may consist of multiple microservices and utilizes the APIs and internal messaging infrastructure that the Near-RT RIC provides. Third parties can provide xApps, which extend RAN functionality. xApps receive data from the RAN via E2 subscribptions, such as KPMs, user, and cell info. The xApp could store this data in an external database for future analysis. Example use cases are to present the data in dashboards to analyze RAN health or to train Machine-Learning models that predict radio traffic.

An xApp can also use the data it receives from the RAN to compute control actions sent back to the RAN via the E2 interface for radio resource management. A sample use case would be to enable additional cells during peak traffic times. [22]

xApps enable closed-loop control of the RAN in the time scale of 10ms to 1s [10].

4.1.5 Y1 interface

The Near-RT RIC implements the Y1 interface. It exposes a RESTful HTTP API. Data is serialized using JSON. Multiple authorized Y1 consumers can connect to the Y1 interface to receive RAN Analytics Information (RAI). There are two services for retrieving RAI: [14]

1. *Y1_RAI_Subscription* - Consumers create a subcription for a specific RAI type. The information can be further filtered, e.g., by specific UEs or network slices. Consumers specify a target address to which the Near-RT RIC should send the information and notification criteria, indicating the timing or condition when the information is sent. Whenever the notification criteria are met, the Near-RT RIC sends the information to the target address. Upon registration, the consumer receives a subscription ID, which can later be used to modify or delete the subscription.

2. *Y1_RAI_Query* - Similar to Y1_RAI_Subscription, consumers can specify a RAI type and filter. The requested information will be returned immediately if available. When the consumer wants updated information, it needs to query again - no subscription gets created.

The Y1 interface is interesting but is yet to be implemented in the Near-RT RICs evaluated in this work.

4.2 Non-RT RIC and SMO Framework

The goal of the SMO is to be an intelligent automation platform for the RAN that can improve network performance and minimize RAN operational costs. Figure 7 shows the architecture of the SMO Framework. The SMO hosts the Non-RT RIC. They share an internal messaging infrastructure and terminate the A1, O1, and O2 interfaces. Multiple Near-RT RICs are connected to the SMO and receive policies from it. The Non-RT RIC is programmable using rApps. It enables closed-loop control of the RAN with timescales larger than 1s (non-real-time). [10], [17]

The Non-RT RIC and SMO Framework are described to understand the concept of O-RAN and to get an insight into the complete architecture. While providing additional functionality and management capabilities, they are not required for a network to function. A Near-RT RIC works without connecting to a Non-RT RIC. The SMO Framework and Non-RT RIC are out of scope for the test network implemented in this work.

The following sections describe the function of each interface, as well as rApps.

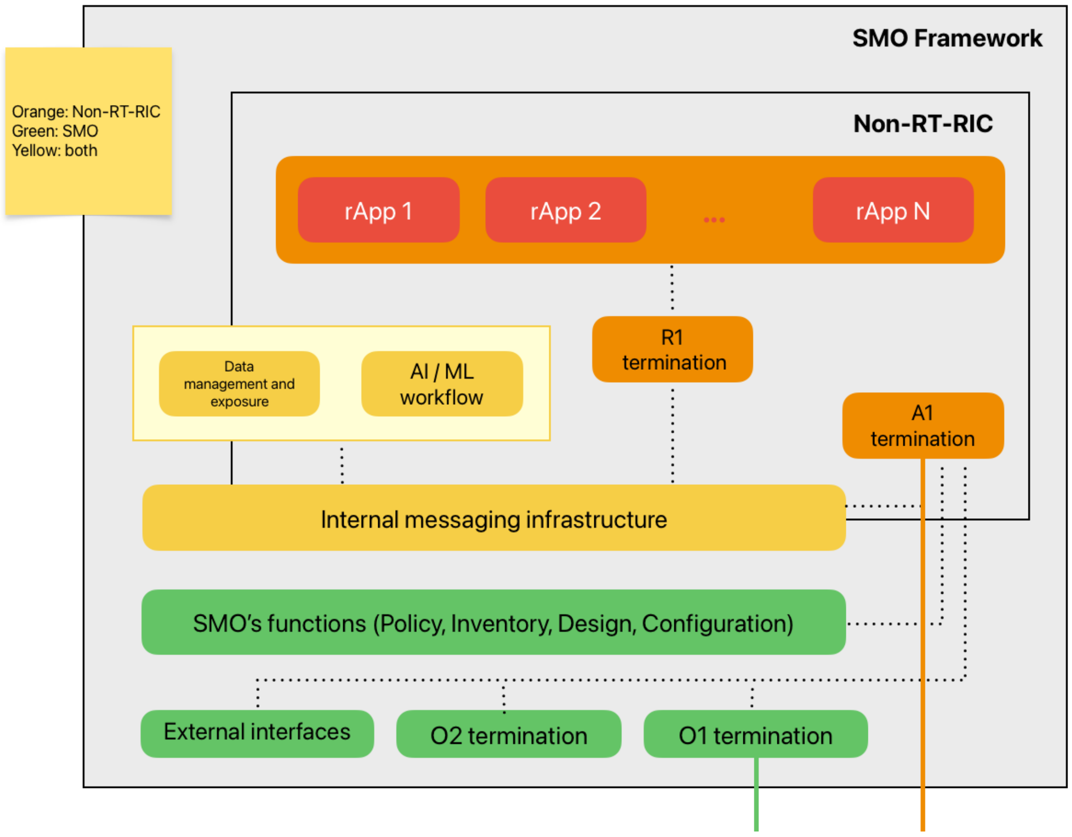


Figure 7 SMO Framework architecture

4.2.1 A1 interface

The A1 interface connects the Non-RT RIC with the Near-RT RIC. It enables radio resource management based on A1 policies. Both endpoints of the A1 interface expose a RESTful HTTP API that uses JSON payloads [10]. The Non-RT RIC deploys A1 policies to the Near-RT RIC, that define high-level optimization goals, called the *RAN Intent*. The RAN Intent defines QoS targets, or Key Performance Indicator (KPI) goals. The policy syntax is based on a JSON schema and can refer to a group of UEs or even to a single specific UE. There are various policies, each serving a different use case, such as QoS targets, traffic steering configuration, or throughput maximization. Finally, the Non-RT RIC can provide A1 Enrichment Information (EI) - additional information that comes from sources outside the RAN, e.g., traffic forecasts based on events like a football game. This information is exposed via the A1 interface and can be used by xApps in the Near-RT RIC. [24]

4.2.2 O1 interface

The O1 interface connects the SMO Framework to the Near-RT RIC and the elements of the gNodeB (O-CU-CP, O-CU-UP, O-DU). It enables FCAPS functions. Provisioning is done using the NETCONF protocol. Configuration changes are propagated using a RESTful HTTP API with data modeled using YANG. [25]

4.2.3 O2 interface

The O2 interface connects the SMO with the O-Cloud. It enables the SMO to manage cloudified network functions. This includes deployment, termination, and scaling of network functions, as well as FCAPS functionality for abstract infrastructure resources. The O-Cloud and O2 interfaces are built on top of existing technology like Kubernetes or Open Stack. [26]

4.2.4 rApps

Like xApps, rApps are modular applications that provide additional services to RAN operation. They run on the Non-RT RIC and leverage the functionality provided by the R1 interface. Compared to xApps, rApps operate on a time scale usually greater than 1s. rApps perform tasks like defining A1 policies, generating A1 Enrichment Information, or recommending configuration that may be applied over the O1 and O2 interfaces. [10]

5 Software components and fundamentals

This chapter describes the test network's software components and fundamentals before discussing its design details in the next chapter.

5.1 Containers

A container can be seen as a lightweight virtual machine that runs container images. Container images package software into standardized units, which can easily be distributed and deployed. Compared to virtual machines, containers share the system kernel of the host, which makes them more lightweight and efficient while providing security measurements to protect the host operating system.

Virtualization is a great focus of O-RAN. Every component (except the O-RU) can be deployed as a container or virtual machine running on cloud infrastructure. A set of tools is needed to work with containers, which allows the building of container images, provides a runtime engine to run the containers, and provides a CLI or API to deploy and manage containers.

The Docker platform is used for the test network to build, run, and manage containers. It was chosen over other container platforms, such as Podman or LXC, due to its maturity, widespread usage, and rich feature set.

5.2 Container Images

A Dockerfile contains instructions on how to build a container image. Appendix B shows an example Dockerfile to build the FlexRIC Near-RT RIC. A container image is usually based on another image, like Ubuntu Jammy. In several steps, a Dockerfile describes which steps must be run to build the image. This usually involves installing dependencies via a package manager, cloning source code, and building the program for the target operating system. The CMD line defines what is executed when a container is started using the image.

In some cases, it was not possible to use container images that are already published to a registry, like the O-RAN SC container registry [27] or Docker Hub [28]. Either because changes were required to the images or because they were not published. The /docker directory in the git repository (appendix A) contains modified Dockerfiles from upstream repositories like Open5GS, srsRAN, or FlexRIC. For most images, the source code cloned to build the image has been fixed to a specific commit. This avoids inconsistent builds and possible regressions when new commits get added to the upstream repository. The tradeoff is that updates need to be implemented manually. Table 1 gives an overview of custom-built docker images used to run the test network.

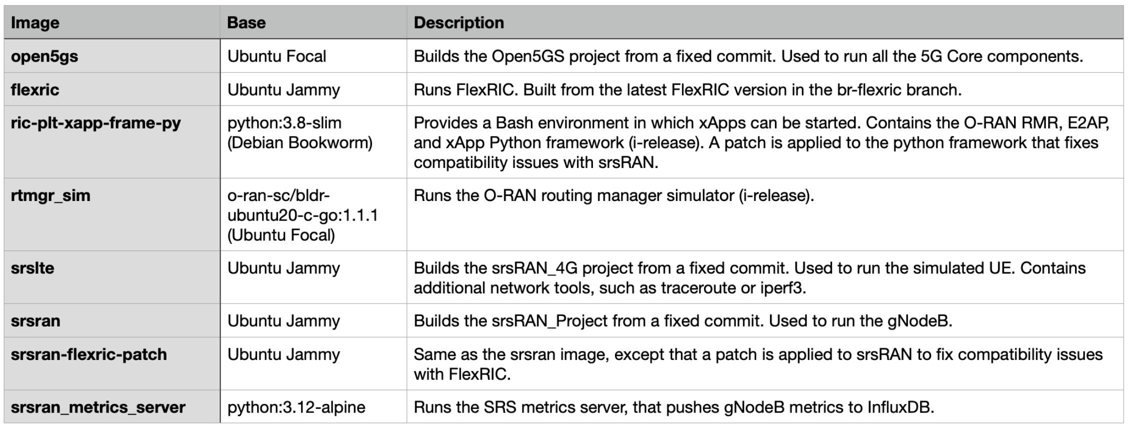


Image	Base	Description
open5gs	Ubuntu Focal	Builds the Open5GS project from a fixed commit. Used to run all the 5G Core components.
flexric	Ubuntu Jammy	Runs FlexRIC. Built from the latest FlexRIC version in the br-flexric branch.
ric-plt-xapp-frame-py	python:3.8-slim (Debian Bookworm)	Provides a Bash environment in which xApps can be started. Contains the O-RAN RMR, E2AP, and xApp Python framework (i-release). A patch is applied to the python framework that fixes compatibility issues with srsRAN.
rtmgr_sim	o-ran-sc/bldr-ubuntu20-c-go:1.1.1 (Ubuntu Focal)	Runs the O-RAN routing manager simulator (i-release).
srslte	Ubuntu Jammy	Builds the srsRAN_4G project from a fixed commit. Used to run the simulated UE. Contains additional network tools, such as traceroute or iperf3.
srsran	Ubuntu Jammy	Builds the srsRAN_Project from a fixed commit. Used to run the gNodeB.
srsran-flexric-patch	Ubuntu Jammy	Same as the srsran image, except that a patch is applied to srsRAN to fix compatibility issues with FlexRIC.
srsran_metrics_server	python:3.12-alpine	Runs the SRS metrics server, that pushes gNodeB metrics to InfluxDB.

Table 1 Custom container images used to run the test network

5.3 Docker Compose

Docker provides a tool called Docker Compose, which defines multi-container applications with a single YAML configuration file. Appendix C shows a sample Docker Compose file, which deploys the time-series database InfluxDB, the monitoring solution Prometheus, and Grafana for dashboards. At the top of the file, the Compose syntax version is specified. The second line defines the name of the Compose application. By default, all containers that are part of this application are prefixed by this name.

A Compose application consists of one or more services. A service refers to one or more containers running the same image with the same configuration. Services can be scaled up or down to multiple replicas as needed. By default, only a single replica is deployed. All services are configured under the services map in the Compose file. The key refers to the name of the service, in this case, influxdb, prometheus, and grafana. The value is a map of container options, including the image, mounted volumes, ports, environment, and network configuration. The container name can optionally be overwritten, as shown in the example for the influxdb service. Otherwise, it would default to shared_influxdb.

In addition to services, a Compose application can define Docker networks and volumes. As shown in the example, environment variables can be used to substitute configuration values of the Compose application, such as the network name or subnet. This allows different configurations on different host systems without changing the Compose file.

When a Compose application is started using *docker compose up*, all the networks, volumes, and containers get created with a single command. The command *docker compose down* can be used to remove the application and all of its components. The ability to quickly deploy multi-container applications on a single host system makes

Docker Compose an excellent choice for implementing the test network. Deployments are portable to different host systems just by sharing the Compose file.

5.4 Open5GS

L. Mamushiane, A. Lysko, H. Kobo, and J. Mwangama compared projects focused on 3GPP-compliant 5G Core Network implementations [29]. Based on their work, Open5GS was chosen in favor of alternatives like Magma, Free5GC, or OpenAirInterface 5GC. Open5GS is an open-source 4G / 5G Core implementation, following the 3GPP Release-17 specification. It is implemented using the C-language and is available on GitHub [30]. The project is funded by sponsors, e.g., on OpenCollective [31]. Open5GS supports both 5G NSA and 5G SA. 5G NSA is a bridge technology for providers to reuse their existing 4G core network. 5G NSA cores are already getting replaced by 5G SA cores. Therefore, a 5G SA core network is used in this work. This enables easier deployment and allows future researchers to test 5G features not supported by NSA, such as network slicing.

5.5 srsRAN

srsRAN is open source 4G and 5G RAN software developed by SRS. *srsRAN_Project* is an O-RAN compatible 5G CU / DU solution available on GitHub [32]. *srsRAN_4G* is the predecessor of *srsRAN_Project*, which implements a 4G radio suite [33]. Even though 4G is not used in this work, the *srsRAN_4G* project is still relevant because it implements a simulated UE called srsUE that can connect to 5G networks.

srsRAN was chosen in favor of OpenAirInterface due to its capability of emulating a radio channel and not requiring actual radio hardware, excellent documentation, and community support, and easier deployment [29].

srsRAN employs the disaggregated gNodeB architecture explained in chapter 3.3. However, starting the CU and DU individually is currently impossible. The gNodeB is started using a single binary, which contains both components. A CU / DU split deployment is on the roadmap [34]. The O1 interface is currently not implemented by SRS and is therefore not further evaluated.

5.6 ZeroMQ

ZeroMQ is a high-performance asynchronous messaging library. It is open-source and provides APIs in various languages. ZeroMQ allows the implementation of common messaging patterns over a variety of transports. Compared to other messaging solutions, it does not require a message broker and, therefore, does not introduce additional latency. This makes it possible to use ZeroMQ to simulate a point-to-point radio channel. [35]

To simulate a radio channel using zeroMQ, 2 TCP connections are established between the communication partners. The simulated radio channel is limited to 2 communication partners. This is contrary to the nature of a radio channel. Chapter 6.5.2 explains how this limitation can be circumvented to some degree. Both communication partners (in this case, a gNodeB on one side and a UE on the other side) have two ports: one for transmitting data (TX) and one for receiving data (RX). Figure 8 shows the underlying zeroMQ connections for the simulated radio channel. There is one TCP connection from gNodeB to UE and a second one from UE to gNodeB. The startup procedure is identical for the gNodeB and the UE. On startup, a zeroMQ socket is started using zmq_bind() for the TX port. This will listen on the provided address using the TCP protocol. At the same time, zmq_connect() is called for the RX port to connect to the TX port of the communication partner. Socket connections would typically fail at this point unless both communication partners start simultaneously and the socket of the communication partner is already available when zmq_connect() gets called. zeroMQ differs in this case from traditional socket connections. When connecting to another socket, the other socket is not required to be already running. Instead, messages are stored in a message queue, and once the connection is established, all messages from the queue start to transmit. This makes it possible to start the UE and gNodeB after one another, even if they require a duplex point-to-point connection. [36]

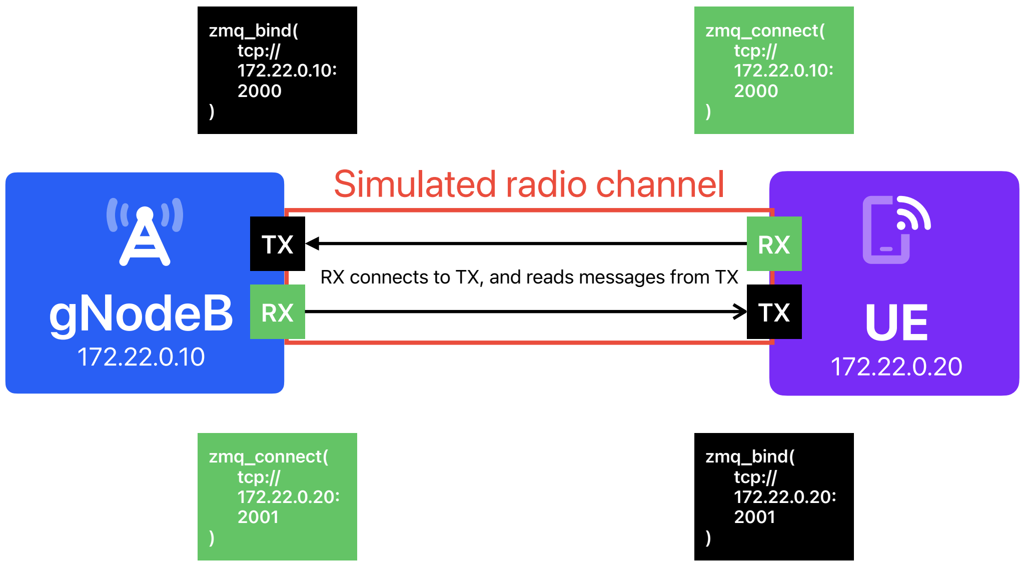

zmq_bind(tcp:// 172.22.0.10: 2000)
zmq_connect(tcp:// 172.22.0.10: 2000)
Simulated radio channel
gNodeB
172.22.0.10
TX
RX
RX connects to TX, and reads messages from TX
RX
TX
UE
172.22.0.20
zmq_connect(tcp:// 172.22.0.20: 2001)
zmq_bind(tcp:// 172.22.0.20: 2001)

Figure 8 Simulated radio channel using duplex zeroMQ point-to-point connection

Figures 9 and 10 show how the simulated radio channel is configured in the srsRAN gNodeB and the simulated srsUE. The configuration matches the diagram shown in figure 8. Both SRS projects have the concept of radio drivers. A radio driver is responsible for receiving and transmitting packets over a physical link. SRS supports the USRP Hardware Driver for using actual radio hardware and zeroMQ for simulated channels. The device driver is set to zmq for the gNodeB and the UE. It is required to build srsRAN using the *-DENABLE_ZEROMQ=ON* flag to use the zeroMQ driver. *libzmq* must be installed on the system during build- and runtime. This is ensured using the srslte and srsran docker images (see table 1). The zmq driver is configured using multiple arguments:

- *tx_port* - The TCP address and port to listen on for the TX port. zmq_bind() is called with the given address.
- *rx_port* - The TCP address and port for the TX port of the communication partner. zmq_connnect() is called with the given address.
- *id* - an optional parameter only used by the UE. It avoids duplicate connections when accidentally starting the srsUE twice.
- *base_srate* - The sampling rate of the radio channel in MHz. It must match the value of *srate* configured in the gNodeB.

```
ru_sdr:
  device_driver: zmq
  device_args:
   tx_port=tcp://172.22.0.10:2000,rx_port=tcp://172.22.0.20:2001,
   base_srate=23.04e6
   srate: 23.04
```

Figure 9 gNodeB zeroMQ driver configuration

```
device_name = zmq
device_args =
   tx_port=tcp://172.22.0.20:2001,rx_port=tcp://172.22.0.10:2000,
   id=ue,base_srate=23.04e6
```

Figure 10 srsUE zeroMQ driver configuration

6 Test Network Design

This chapter is about the design of the test network. It explains the deployed services and how they are configured to fulfill the goals set for the test network. Appendix D contains a diagram showing the architecture of the test network.

6.1 Hardware requirements

The server that runs the test network has the following requirements:

1. At least 8 CPU threads
2. At least 16 GB of memory
3. At least 40 GB of storage for dependencies and logs
4. Sufficient bandwidth to fetch multiple GBs of git repositories and dependencies
5. Ability to run a recent Linux operating system
6. Ability to run Docker - requires CPU support for virtualization

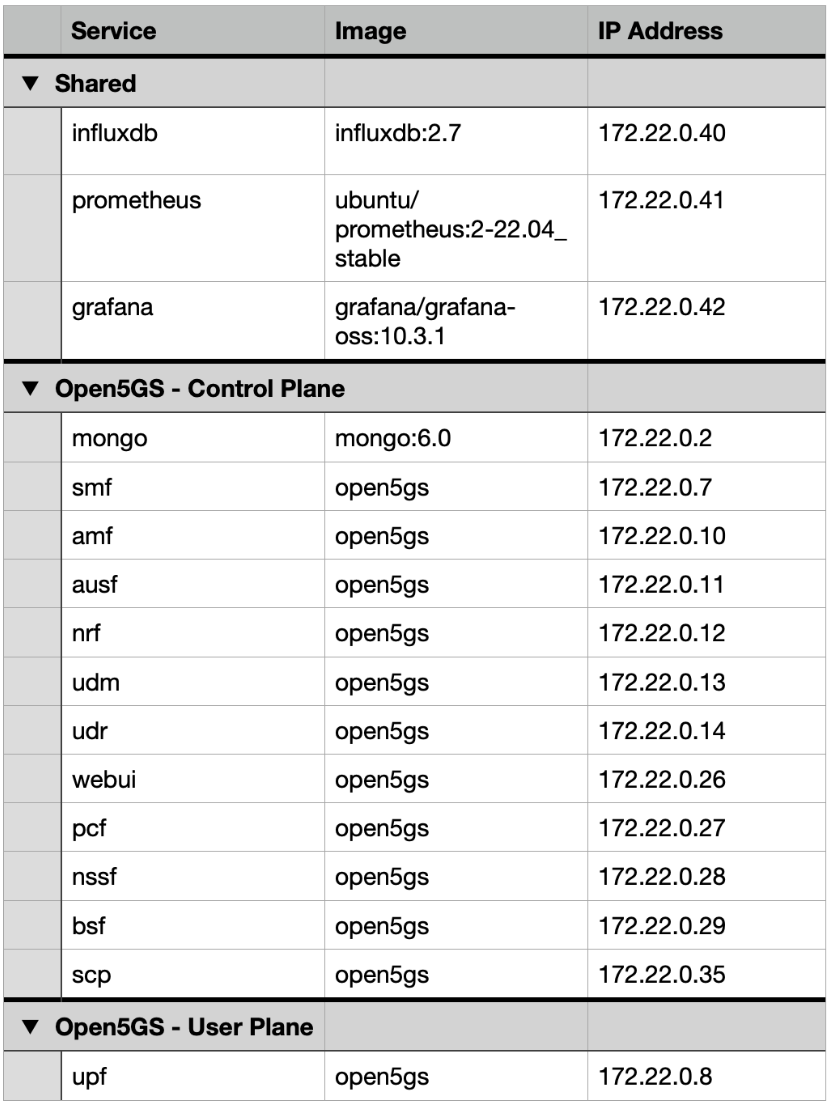
The DN.Lab provided a virtual machine with 8 CPU threads, 16 GB of memory, 80 GB of storage, and Ubuntu 22.04 LTS installed. Because the radio channel is emulated, no additional radio hardware or UE is required.

6.2 Network services

As explained earlier, a 5G network consists of multiple parts, e.g., a 5G control plane and user plane, monitoring infrastructure, RICs, and the cell sites. In a production environment, there would be separate subnets for different network parts. Routing and firewalls must be configured to ensure packets are properly forwarded, and security policies are applied. Building a secure 5G network is out of the scope of this work. For ease of use, all services are deployed to the same subnet. The test network uses the 172.22.0.0/24 address range. Table 2 shows the services deployed in the test network along with their container image and IP address. Note that some IP addresses are skipped. This is because they belong to unused components used in a 5G NSA core. The goal was not to create a gap-free subnetting strategy that uses as few addresses as possible. Because this is a private IPv4 address space, IPv4 address scarcity is ignored.

Services are grouped into different applications. Each application is defined by a docker-compose.yaml file in the git repository (appendix A) and can be started individually. See the README.md file in the repository for details. The applications that make up the test network are:

- *Shared* - Contains services consumed by the other applications, e.g., databases and monitoring services. They are defined in the 0-shared folder.
- *Open5GS* - Contains the 5G SA core network. Defined in the 1-core folder.
- *srsRAN* - Contains the gNodeB's and simulated UEs. Defined in the 3-srsran folder. See chapter 6.5 for details.
- *FlexRIC* and *O-RAN SC RIC* - Discussed in chapter 8.



	Service	Image	IP Address
▼ Shared			
	influxdb	influxdb:2.7	172.22.0.40
	prometheus	ubuntu/prometheus:2-22.04_stable	172.22.0.41
	grafana	grafana/grafana-oss:10.3.1	172.22.0.42
▼ Open5GS - Control Plane			
	mongo	mongo:6.0	172.22.0.2
	smf	open5gs	172.22.0.7
	amf	open5gs	172.22.0.10
	ausf	open5gs	172.22.0.11
	nrf	open5gs	172.22.0.12
	udm	open5gs	172.22.0.13
	udr	open5gs	172.22.0.14
	webui	open5gs	172.22.0.26
	pcf	open5gs	172.22.0.27
	nssf	open5gs	172.22.0.28
	bsf	open5gs	172.22.0.29
	scp	open5gs	172.22.0.35
▼ Open5GS - User Plane			
	upf	open5gs	172.22.0.8

Table 2 Monitoring and Open5GS deployment

The shared monitoring services are explained in more detail in chapter 6.6. The details of a 5G SA core network are not the focus of this work. Therefore, the services and interfaces connecting them are not described in detail. Figure 11 shows the test network's 5G SA architecture. The following list gives an overview of the deployed 5G core components and their function:

- *mongo* - MongoDB is a document-oriented NoSQL database. It is used to store subscriber information.
- *smf* - The Session Management Function manages data sessions and controls the data plane functions.
- *amf* - The Access and Mobility Management Function handles user authentication, mobility management, and session establishment.
- *ausf* - The Authentication Server Function validates user identities and provides authentication credentials.
- *nrf* - The NF Repository Function stores network function profiles and assists in the discovery and selection of network functions.
- *udm* - The Unified Data Management provides a centralized repository for subscriber data, including subscription information and authentication credentials.
- *udr* - The Unified Data Repository stores user-related data for services and applications, including user profiles and policies. It reads from the MongoDB.
- *pcf* - The Policy and Charging Function enforces network polcies related to QoS, charging, and access control.
- *nssf* - The Network Slice Selection Function selects and allocates network slices based on service requirements and network conditions.
- *bsf* - The Binding Support Function manages the binding between user plane and control plane entities. It helps to maintain continuity during UE handover.
- *scp* - The Service Communication Proxy enables indirect communication between 5G network functions.
- *upf* - The User Plane Function forwards user data packets between the gNodeB and the internet.
- *webui* - Provides a web interface for managing subscriber information.

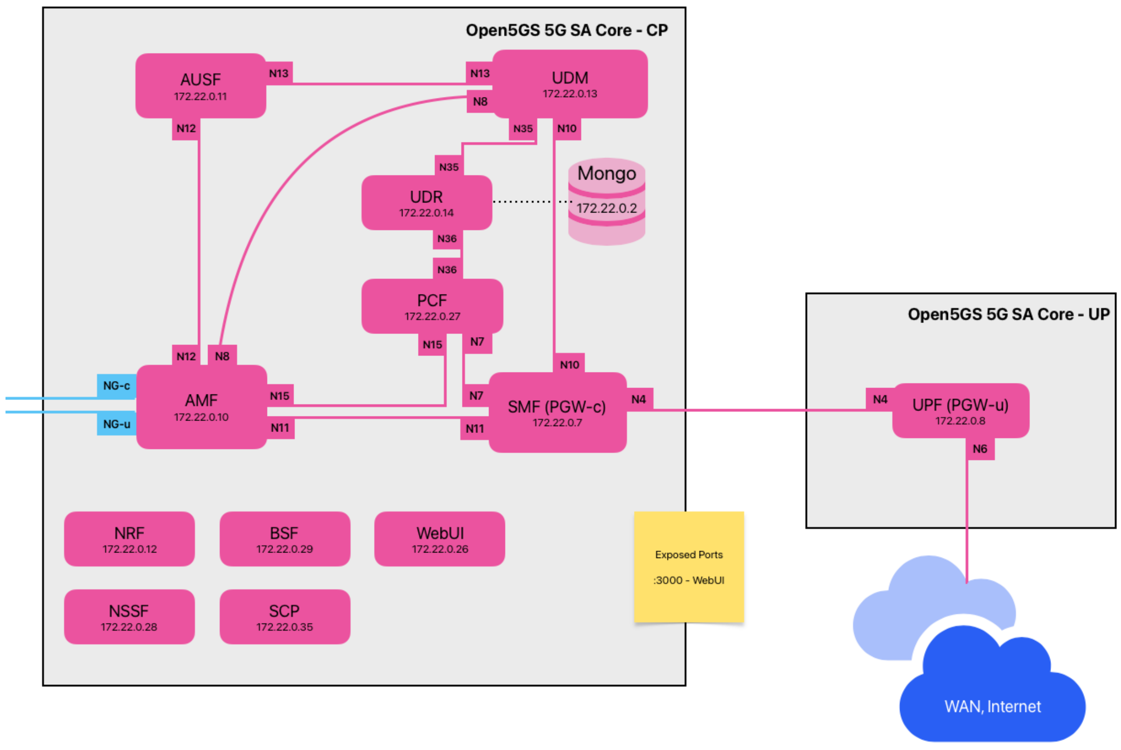


Figure 11 5G SA core architecture

6.3 gNodeB configuration

A srsRAN gNodeB is configured using a YAML configuration file. Figure 12 gives an overview of the gNodeB configuration file. The gNodeB connects to the 5G core component AMF via the NG interfaces. Therefore, the AMF IP address needs to be configured in the gNodeB. In addition to that, every gNodeB has an id and a RAN node name assigned to it. These are used to identify the gNodeB in the 5G core. Furthermore, the identifiers can be used by xApps or rApps to filter information for a single gNodeB. The cell configuration sets frequency, bandwidth, Public Land Mobile Network (PLMN), and other essential cell parameters. It is possible to span multiple cells using a single gNodeB, which makes sense when using tri-sector or similar antennas. For the test network, each gNodeB creates a single cell. The RU configuration of a gNodeB has already been explained in chapter 5.6. The E2 configuration is explained in chapter 8. The pcap, log, and metrics configurations are explained in the following sections of this chapter.

```
gnb_id: 1 # Sets the numerical ID associated with the gNB.
ran_node_name: gnb1 # Sets the text ID associated with the gNB.

amf:
  addr: 172.22.0.10 # The address or hostname of the AMF.
  bind_addr: 172.22.0.101 # A local IP that the gNB binds to for
    traffic from the AMF.

cell_cfg: # Cell configuration. A single cell is used here.
  dl_arfcn: 368500 # ARFCN of the downlink carrier (center frequency).
  band: 3 # The NR band.
  channel_bandwidth_MHz: 20 # Bandwith in MHz.
  common_scs: 15 # Subcarrier spacing in kHz used for data.
  plmn: "00101" # PLMN broadcasted by the gNB.
  tac: 1 # Tracking area code (needs to match the core configuration).
  pci: 1 # Physical cell ID.

ru_sdr: # Explained in the zeroMQ chapter
e2: # Explained in the RIC chapter
log: # Explained in the Logs chapter
pcap: # Explained in the Packet captures chapter
metrics: # Explained in the Monitoring chapter
```

Figure 12 gNodeB configuration overview

For full configuration, see the gnb_zmq.yml files in the /3-srsran/*/config/gnb-*/ subdirectories of the git repository (appendix A).

6.4 UE and subscriber configuration

When connecting to a 5G network, the UE must authenticate with the 5G core network. Therefore, each simulated UE is preconfigured with Subscriber Identity Module (SIM) credentials. This includes:

- *Operator Code* - A 128-bit key that identifies a mobile network operator. The test network uses 11111111111111111111111111111111.
- *Subscriber Authentication Key* - A randomly generated 128-bit subscriber key.
- *International Mobile Subscriber Identity (IMSI)* - A 15 digit number, which uniquely identifies every SIM card. It is composed of the Mobile Country Code (MCC), Mobile Network Code (MNC), and a Mobile Subscriber Identification Number (MSIN). The MCC and MNC make up the PLMN. The test network uses the international test PLMN 001/01. Therefore, the IMSI needs to start with 00101, followed by the MSIN, which is a unique number for every SIM card.

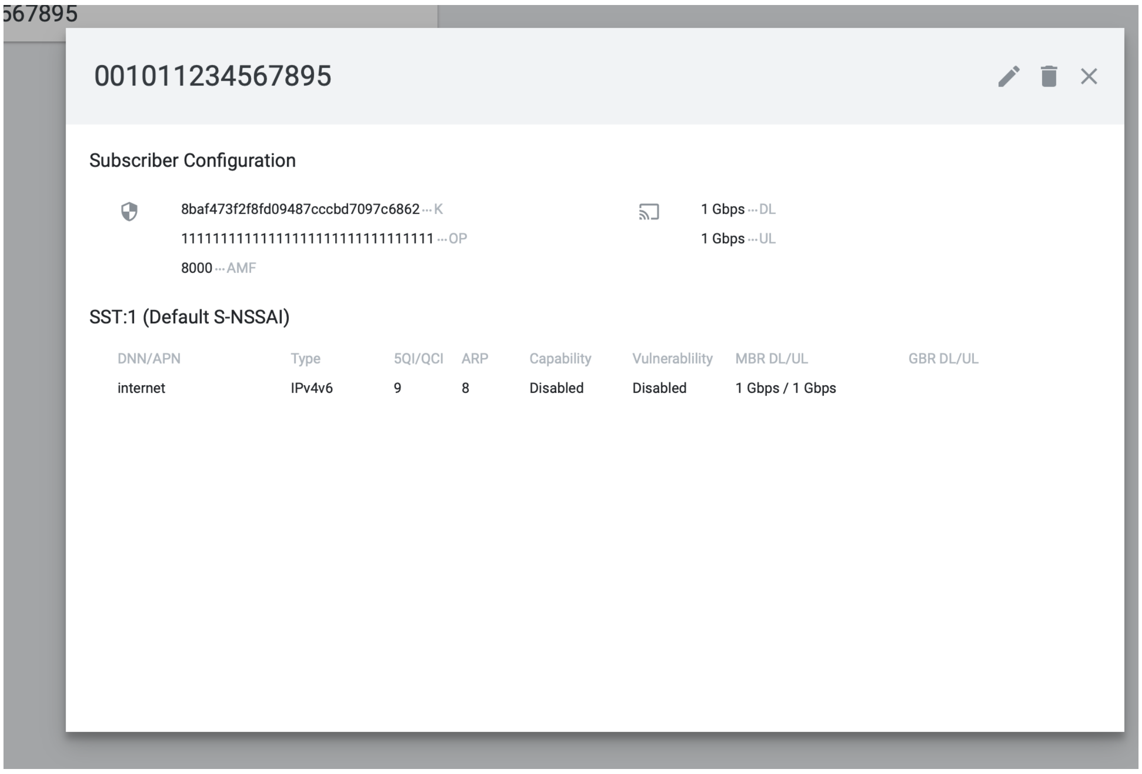
- *International Mobile Equipment Identity (IMEI)* - Usually unique 15 digit numeric identifier for mobile phones. The simulated UEs use 353490069873319. This default value is provided by SRS and refers to an LG Nexus 5.

SIM credentials are configured in the srsUE main configuration file under the usim section (see figure 13). The UE credentials used in the test network can be found in the .env file.

```
[usim]
mode = soft
algo = mileage
op   = UE_OP
k    = UE_KI
imsi = UE_IMSI
imei = 353490069873319
```

Figure 13 srsUE configuration overview
For full configuration, see the *.conf files in the /3-srsran/*/config/ue-*/ subdirectories of the git repository (appendix A).

The 5G core has a subscriber database that contains every UE that is allowed to connect to the network, along with configuration like QoS values or bandwidth limits. For a UE to successfully connect to the network, it must be added to the subscriber database. Open5GS provides a web interface to manage the subscriber database. Figure 14 shows the subscriber configuration for a UE. The IMSI, Subscriber Authentication Key (K), and Operator Code (OP) need to match the values configured for the simulated UE. Details on accessing the Web interface and adding the subscriber data are available in the README.md file (appendix A).



567895

001011234567895

Subscriber Configuration

8baf473f2f8fd09487cccbd7097c6862 ··· K
11111111111111111111111111111111 ··· OP
8000 ··· AMF

1 Gbps ··· DL
1 Gbps ··· UL

SST:1 (Default S-NSSAI)

DNN/APN	Type	5QI/QCI	ARP	Capability	Vulnerablility	MBR DL/UL	GBR DL/UL
internet	IPv4v6	9	8	Disabled	Disabled	1 Gbps / 1 Gbps	

Figure 14 Open5GS Subscriber Configuration

6.5 Deployment scenarios

The test network provides five different standalone deployment scenarios for srsRAN. Each scenario deploys a gNodeB and one or more UEs with a unique configuration. Each scenario uses different IDs for UEs and gNodeB's, which allows multiple scenarios to co-exist and be deployed simultaneously using a shared 5G core network. The following subchapters describe each deployment scenario. Instructions on how to start individual deployment scenarios are available in the README.md file (appendix A).

After registration with the gNodeB, an additional IP address from the 192.168.100.0/24 subnet is dynamically assigned to a UE. This IP address is used for communication over the simulated radio channel. Docker services to which this applies are marked with an asterisk (*) in the following tables. The addresses listed in the tables are used for communication over the docker network.

6.5.1 Single UE

Service	Image	IP Address
gnb-1	srsran	172.22.0.51
metrics_server-1	srsran_metrics_server	172.22.0.101
ue-1*	srslte	172.22.0.111

Table 3 srsRAN deployment scenario 1 - Single UE

This is the most straightforward deployment scenario. A gNodeB is deployed without a connection to a Near-RT RIC. A single UE is connected to the gNodeB. Table 3 shows the docker services used in this deployment scenario.

6.5.2 Multiple UEs

Service	Image	IP Address
gnb-2	srsran	172.22.0.52
metrics_server-2	srsran_metrics_server	172.22.0.102
ue-2-1*	srslte	172.22.0.121
ue-2-2*	srslte	172.22.0.122
ue-2-3*	srslte	172.22.0.122

Table 4 srsRAN deployment scenario 2 - Multiple UEs

A gNodeB without a connection to a Near-RT RIC is deployed in this deployment scenario. Multiple UEs are connected to the gNodeB. The simulated radio channel is shared between all UEs. Table 4 shows the docker services used in this deployment scenario. As stated earlier in chapter 5.6, the zeroMQ-based radio channel uses a duplex point-to-point connection. Out of the box, it is not possible to connect multiple UEs to a single gNodeB. To workaround this limitation, the software GNU Radio Companion is used. It is an open-source visual programming tool for signal processing. Flow graphs (.grc files) describe the flow of radio signals. The GNU Radio Companion cannot be launched in a headless server environment, requiring a graphical user interface. Therefore, this deployment scenario has an additional requirement for the host: a graphical user interface needs to be installed. gnome-remote-desktop has been installed on the test server. In addition to that, the xrdp package has been installed and configured, which allows the server to accept Remote Desktop Protocol (RDP) connections.

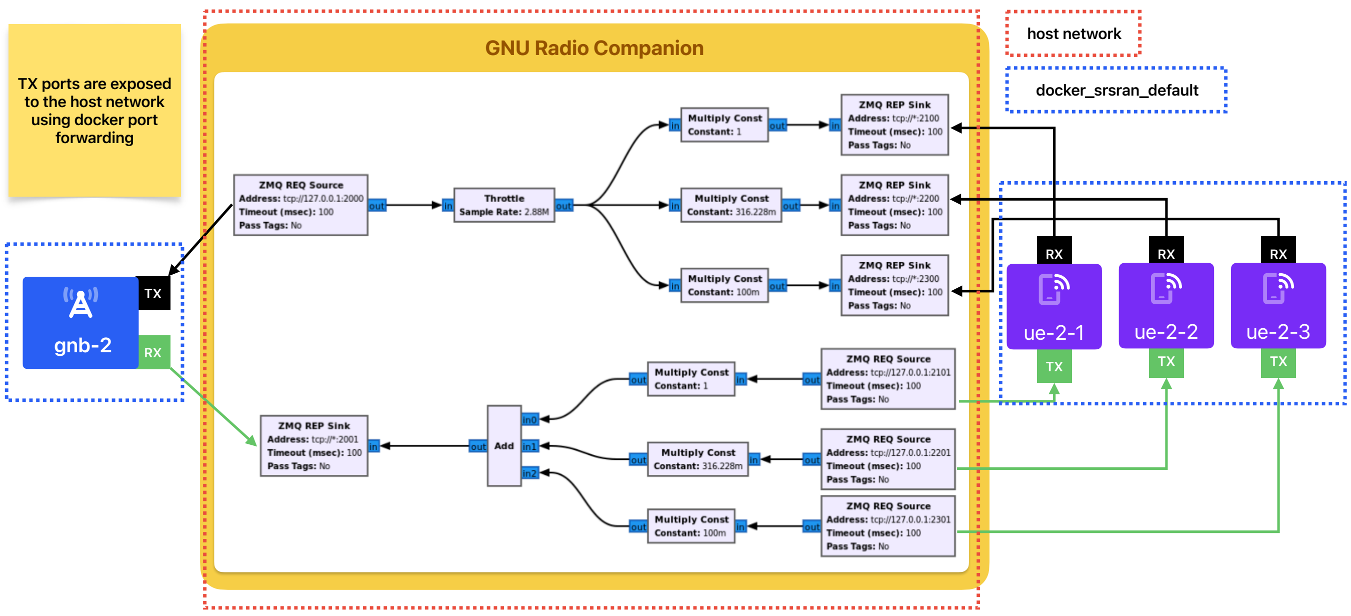
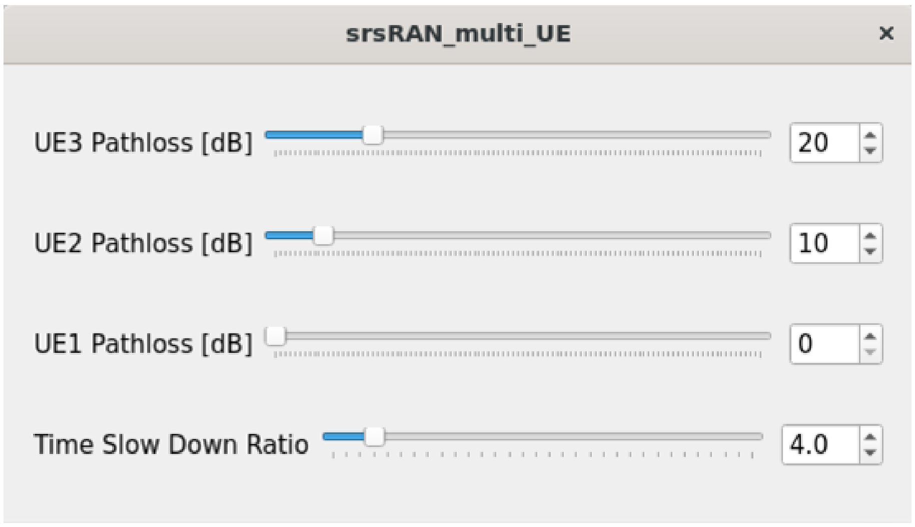

TX ports are exposed to the host network using docker port forwarding
GNU Radio Companion
host network
docker_srsran_default
gnb-2
TX
RX
ZMQ REQ Source Address: tcp://127.0.0.1:2000 Timeout (msec): 100 Pass Tags: No
Throttle Sample Rate: 2.88M
Multiply Const Constant: 1
Multiply Const Constant: 316.228m
Multiply Const Constant: 100m
ZMQ REP Sink Address: tcp://*:2100 Timeout (msec): 100 Pass Tags: No
ZMQ REP Sink Address: tcp://*:2200 Timeout (msec): 100 Pass Tags: No
ZMQ REP Sink Address: tcp://*:2300 Timeout (msec): 100 Pass Tags: No
ZMQ REQ Source Address: tcp://127.0.0.1:2101 Timeout (msec): 100 Pass Tags: No
ZMQ REQ Source Address: tcp://127.0.0.1:2201 Timeout (msec): 100 Pass Tags: No
ZMQ REQ Source Address: tcp://127.0.0.1:2301 Timeout (msec): 100 Pass Tags: No
Add
ZMQ REP Sink Address: tcp://*:2001 Timeout (msec): 100 Pass Tags: No
ue-2-1
ue-2-2
ue-2-3

Figure 15 Multiple UE architecture using GNU Radio Companion

Figure 15 shows the multi UE architecture. At the center of the figure is the GNU flow graph. It determines how zeroMQ connections are proxied and how signal processing is applied to multiplex and combine signals. A slightly modified version of the flow graph provided by srsRAN [37] is used. It is available in the git repository under 3-srsran/2-multi-ue/multi-ue-broker.grc (appendix A). The gNodeB and UE no longer connect directly. Instead, they connect to zeroMQ endpoints launched by the GNU Radio Companion. The radio broker receives the Downlink signal from the gNodeB and sends a copy to every connected UE. All Uplink signals from the connected UEs are aggregated, and the aggregated signal is sent to the gNodeB. gnb-2 and the UEs are inside the docker network, as indicated by the blue dotted rectangles. The GNU Radio Companion is running on the host network, as indicated by the red dotted rectangle. The black and green arrows outside the flow graph indicate which side initiates the connection. As explained in chapter 5.6, RX ports initiate a connection to the TX port of the communication partner. The ZMQ REP Sinks start a zeroMQ socket with zmq_bind() using the provided address on the host. In this case, * is the address to bind on all interfaces. The ZMQ REQ Sources initiate a zeroMQ connection using zmq_connect() to the TX ports of the gNodeB and UEs. The TX port addresses are exposed from the Docker network to the host network using Docker port forwarding. This makes it possible to use the loopback address 127.0.0.1 in the flow graph to connect to the containers running in the docker network.

After the flow has been started in GNU Radio Companion, a settings panel gets opened, which allows to change the path loss value for each UE (see figure 16). srsUE allows to enable tracing, which prints measurements for signal, uplink, and downlink every second. Tracing is enabled by attaching to a srsUE container using Docker, followed by typing t. The traces include a measurement for Reference Signal Received Power (RSRP). Increasing the path loss value for a UE results in a decreased RSRP, visible in the srsUE traces.


srsRAN_multi_UE

UE3 Pathloss [dB] 20

UE2 Pathloss [dB] 10

UE1 Pathloss [dB] 0

Time Slow Down Ratio 4.0

Figure 16 GNU Radio Companion settings panel

6.5.3 FlexRIC

Service	Image	IP Address
gnb-3	srsran-flexric-patch	172.22.0.53
metrics_server-3	srsran_metrics_server	172.22.0.103
ue-3*	srslte	172.22.0.131

Table 5 srsRAN deployment scenario 3 - Flexric

This deployment scenario is the same as scenario 1, except that the gNodeB is configured to connect to FlexRIC. Table 5 shows the docker services used in this deployment scenario. It requires FlexRIC to be deployed. Details around FlexRIC and this deployment scenario are available in chapter 8.2.

Due to an issue with FlexRIC, it crashes when decoding the RICIndication message from srsRAN. This is a known problem and has already been reported to the FlexRIC developers. Until the issue is fixed, it is required to use a patched version of srsRAN, which does not include the *granul_period* field that causes FlexRIC to crash [38]. Therefore, this deployment uses a patched image of srsRAN.

6.5.4 O-RAN SC Near-RT RIC

Service	Image	IP Address
gnb-4	srsran	172.22.0.54
metrics_server-4	srsran_metrics_server	172.22.0.104
ue-4*	srslte	172.22.0.141

Table 6 srsRAN deployment scenario 4 - O-RAN SC Near-RT RIC

This deployment scenario is the same as scenario 1, except that the gNodeB is configured to connect to the O-RAN SC Near-RT RIC. Table 6 shows the docker services used in this deployment scenario. It requires deploying the O-RAN SC Near-RT RIC. Details around the O-RAN SC Near-RT RIC and this deployment scenario are available in chapter 8.3.

6.5.5 External O-RAN SC Near-RT RIC

Service	Image	IP Address
gnb-5	srsran	172.22.0.55
metrics_server-5	srsran_metrics_server	172.22.0.105
ue-5*	srslte	172.22.0.151

Table 7 srsRAN deployment scenario 5 - External O-RAN SC Near-RT RIC

This deployment scenario is similar to scenario 4, except that the gNodeB is configured to connect to an external O-RAN SC Near-RT RIC running on another host. Table 7 shows the docker services used in this deployment scenario. Details are available in chapter 8.3.

6.6 Monitoring

A monitoring stack is deployed to gain insights into the test network. Grafana allows data visualization using dashboards and various panels, such as gauges, bar charts, or line charts. It connects to data sources like Prometheus or InfluxDB. Prometheus is an open-source monitoring and alerting toolkit. It collects and stores metrics as time series data, i.e., metrics information is stored with the timestamp at which it was recorded, alongside optional key-value pairs called labels. Like Prometheus, InfluxDB is a time-series database that allows for the storage, aggregation, and query of metrics.

Open5GS components AMF, SMF, PCF, and UPF expose Prometheus-compatible

HTTP endpoints that expose core network metrics. Prometheus is configured to scrape these endpoints every 5 seconds. The resulting metrics are stored in the Prometheus database.

The srsRAN project provides a metrics server that receives metrics data from a running gNodeB and sends it to InfluxDB. The TESTBED environment variable defines how the metrics server labels incoming data. Labeling the data differently depending on which gNodeB sends the data to the metrics server is currently impossible. It is, therefore, required to use a separate metrics server per gNodeB so that metrics data can be queried based on the gNodeB that produced them. The test network contains one metrics server per gNodeB, i.e., gnb-1 sends its metrics to metrics_server-1, while gnb-2 sends its metrics to metrics_server-2.

Figure 17 shows the monitoring architecture that is used for the test network implemented in this work. Grafana is configured with both InfluxDB and Prometheus data sources. Metrics servers publish srsRAN gNodeB metrics to InfluxDB and Prometheus scrapes metrics of Open5GS core components.

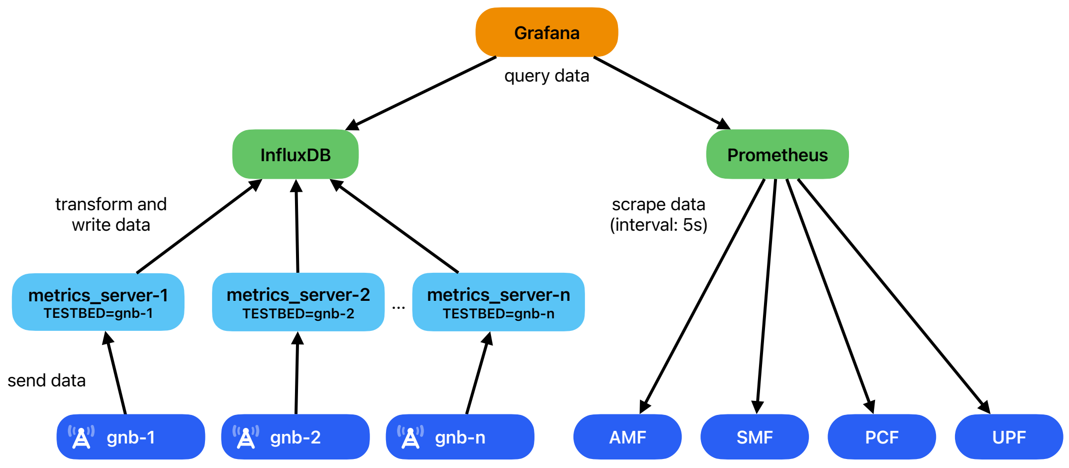


Figure 17 Monitoring architecture

A Grafana dashboard is deployed to visualize srsRAN gNodeB metrics. Figure 18 shows the dashboard, visualizing one minute of metrics data on gnb-1 after an iperf3 session. The data is fetched from InfluxDB. The gNodeB can be selected in the top-left corner. Values are available based on the TESTBED environment variable, configured in the metrics-server, that sends the data to InfluxDB. The dashboard shows the number of active UEs, uplink and downlink bitrate, Signal to Noise Ratio (SNR), Channel Quality Indicator (CQI), and Modulation and Coding Scheme (MCS). The SNR and CQI measure the overall quality of the radio channel. The higher the value, the better. The values are static due to the simulated radio channel. The MCS is affected by the quality of the radio channel. A better link quality allows for a higher MCS, meaning more complex modulation schemes can be used, resulting in higher data throughput.

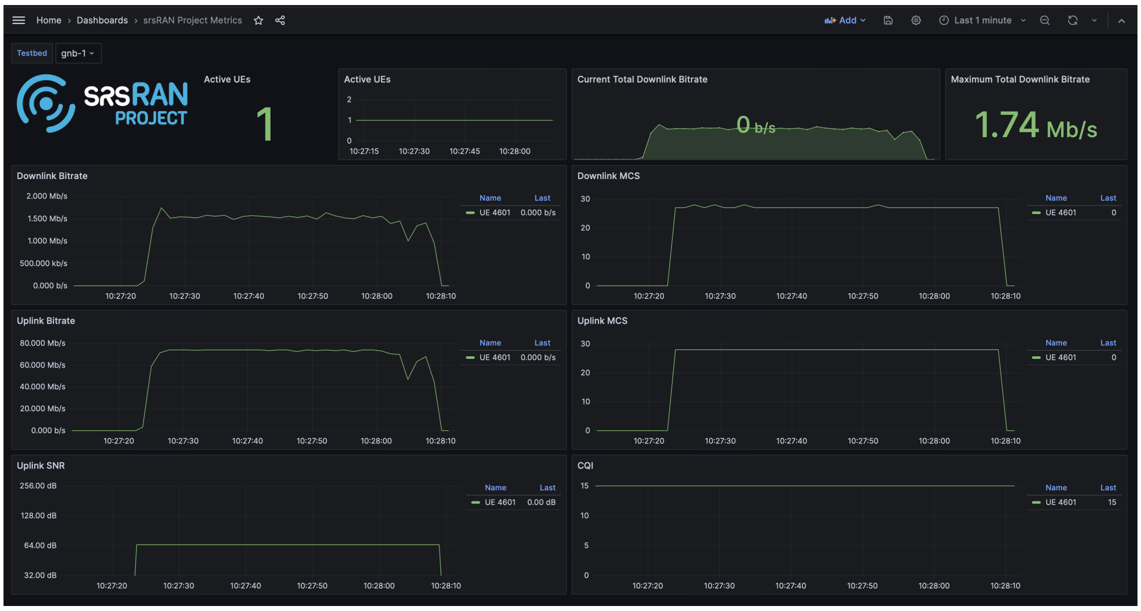


Figure 18 Metrics captured for an iperf3 session from simulated UE to host over virtualized radio. Duration: 45s, max throughput: 100Mbit/s

6.7 Logs

All deployed network services produce logs. Logs are essential to finding misconfigurations, analyzing performance bottlenecks, or detecting malicious RAN activity. Since all services are run inside containers, the *docker logs* command can be used to read logs. In addition to that, the srsRAN gNodeB and UE write logs to a file. Log files are made available to the host system via Docker bind mounts. There are different log levels (from lowest to highest level of detail): none, error, warning, info, and debug. A different log level can be assigned to each layer and component individually in the gNodeB and srsUE configuration files. This allows for analyzing a specific layer in detail. A non-excessive list of layers includes PHY, MAC, PDCP, RRC, or F1AP. Figures 19 and 20 show how logging is configured in srsRAN.

```
log:
  filename: /var/log/srsran/gnb.log # Path of the log file.
  all_level: info # Logging level applied to all layers.
  phy_level: error # Overwrite log level for a specific layer.
  cu_level: info # Overwrite log level for a specific component.
```

Figure 19 srsRAN gNodeB Log configuration

```
[log]
filename = /var/log/srslte/ue.log # Path of the log file.
all_level = info # Logging level applied to all layers.
phy_lib_level = none # Overwrite log level for a specific layer.
```

Figure 20 srsUE Log configuration

6.8 Packet captures

Packet captures allow traffic to be captured on a given link and saved to a file. Capture (pcap) files can later be analyzed using tools like Wireshark. They provide an in-depth view of the test network and allow an in-depth analysis of the protocols involved in a 5G network. In the context of the 5G-FORAN project, packet captures allow firewalls to be baselined with what is considered clean or regular traffic.

srsRAN can generate individual pcap files per network layer for UEs and gNodeBs. They are written to the filesystem after gracefully stopping the UE or gNodeB. Gracefully stopping requires attaching to the container. Detailed instructions are available in the README.md (appendix A). Previous packet captures are overwritten when stopping a gNodeB or UE.

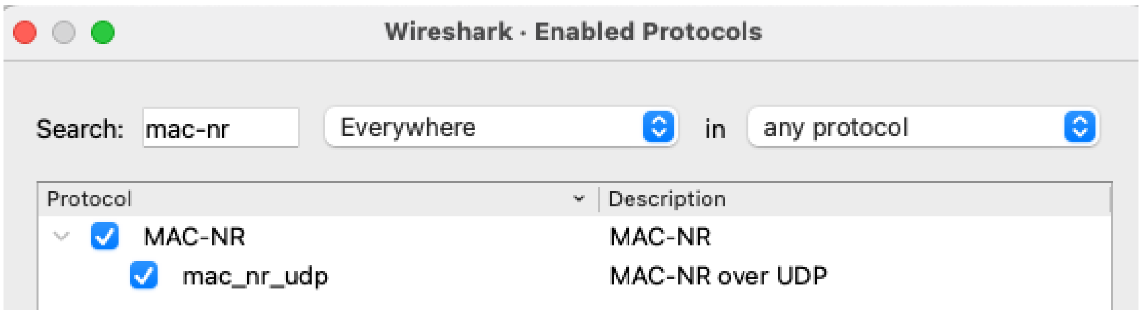
6.8.1 Data Link Types

The capture tool tags packets with a Data Link Type (DLT) when performing a packet capture. It tells tools like Wireshark how to interpret the incoming bytes of each packet. Most common network protocols have predefined DLTs. However, for 5G networking protocols, custom DLTs are used. The packet captures generated by srsRAN use the following custom DLT values:

- *DLT 148* - Used for NAS packets in UE packet captures.
- *DLT 149* - Used for MAC packets between UE and gNodeB. It uses the UDP protocol (MAC-NR over UDP).
- *DLT 152* - Used for NGAP packets between the gNodeB and the 5G core.
- *DLT 153* - Used for E1AP packets between the CU user plane and control plane.
- *DLT 154* - Used for F1AP packets between the CU and DU.
- *DLT 155* - Used for E2AP packets exchanged between the gNodeB and the Near-RT RIC.
- *DLT 156* - Used for GTP-u packets between the gNodeB and the 5G core.

6.8.2 Wireshark configuration

A recent version of Wireshark, such as 4.2 and newer, is required to parse the generated pcap files correctly. Wireshark must be configured to match the srsRAN custom DLT values to the correct protocol. Additional protocol-specific options need to be set. Figure 21 shows the configuration required for Wireshark to correctly interpret the packet captures generated by srsRAN.

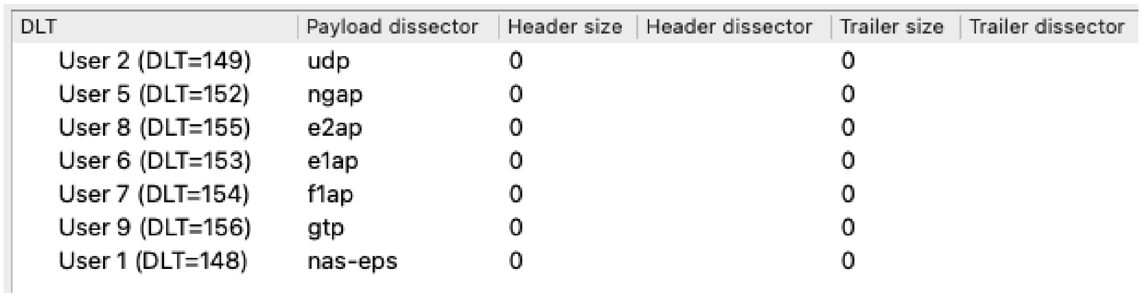


Wireshark · Enabled Protocols

Search: mac-nr   Everywhere   in   any protocol

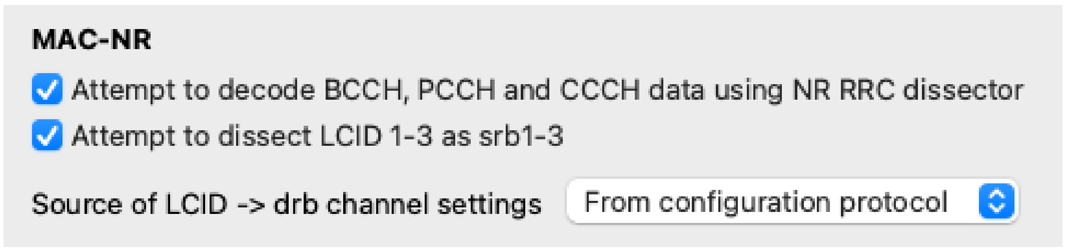
Protocol	Description
✓ MAC-NR	MAC-NR
✓ mac_nr_udp	MAC-NR over UDP

MAC-NR and MAC-NR over UDP are enabled in Analyze - Enabled Protocols



DLT	Payload dissector	Header size	Header dissector	Trailer size	Trailer dissector
User 2 (DLT=149)	udp	0		0	
User 5 (DLT=152)	ngap	0		0	
User 8 (DLT=155)	e2ap	0		0	
User 6 (DLT=153)	e1ap	0		0	
User 7 (DLT=154)	f1ap	0		0	
User 9 (DLT=156)	gtp	0		0	
User 1 (DLT=148)	nas-eps	0		0	

DLT mappings are configured in Preferences - Protocols - DLT_USER



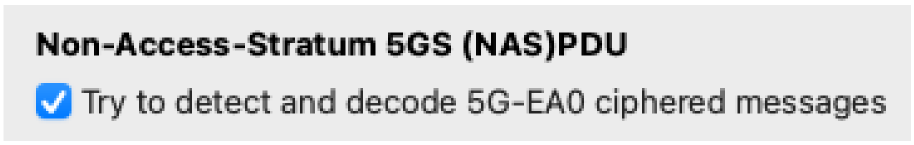
MAC-NR

✓ Attempt to decode BCCH, PCCH and CCCH data using NR RRC dissector

✓ Attempt to dissect LCID 1-3 as srb1-3

Source of LCID -> drb channel settings: From configuration protocol

MAC-NR configuration in Preferences - Protocols - MAC-NR



Non-Access-Stratum 5GS (NAS)PDU

✓ Try to detect and decode 5G-EA0 ciphered messages

NAS configuration in Preferences - Protocols - NAS-5GS

Figure 21 Wireshark configuration

6.8.3 srsRAN configuration

Packet captures can be enabled individually per layer. Different file names can be specified for each layer so that different layers can be analyzed individually. For srsUE the following layers are supported: mac for LTE packets, mac_nr for 5G NR packets, and nas for NAS packets. Because 4G LTE is unused, enabling packet captures for mac_nr and nas is sufficient. The gNodeB supports the following layers: mac for 5G NR packets, ngap for NGAP packets exchanged between gNodeB and AMF, e1ap for packets exchanged between the CU control and user plane, e2ap for packets exchanged between the gNodeB and Near-RT RIC, gtpu for the GTP tunnel established between gNodeB and UPF, and rlc for Radio Link Control. [39] The test network is configured to generate packet captures for all layers except rlc. RLC is only relevant with real radio hardware. The pcap files are accessible to the host filesystem using Docker volume binds.

```
[pcap]
enable = mac_nr,nas
mac_nr_filename = /mnt/pcap/ue1_mac_nr.pcap
nas_filename = /mnt/pcap/ue1_nas.pcap
```

Figure 22 srsUE pcap configuration

```
pcap:
  mac_enable: true
  mac_filename: /mnt/pcap/gnb1_mac.pcap
  ngap_enable: true
  ngap_filename: /mnt/pcap/gnb1_ngap.pcap
  f1ap_enable: true
  f1ap_filename: /mnt/pcap/gnb1_f1ap.pcap
  e1ap_enable: true
  e1ap_filename: /mnt/pcap/gnb1_e1ap.pcap
  e2ap_enable: true
  e2ap_filename: /mnt/pcap/gnb1_e2ap.pcap
  gtpu_enable: true
  gtpu_filename: /mnt/pcap/gnb1_gtpu.pcap
```

Figure 23 gNodeB pcap configuration

6.9 Running the test network

Running the test network involves starting multiple Docker Compose applications. An example to start deployment scenario 1 can be found in figure 24. Detailed step-by-step instructions on running the test network are available in the README.md file (appendix A).

```
# Start shared services
docker compose -f ./0-shared/docker-compose.yaml up -d

# Start the Open5GS Core
docker compose -f ./1-core/docker-compose.yaml up -d

# Start deployment scenario 1
docker compose -f ./3-srsran/1-single-ue/docker-compose.yaml up -d
```

Figure 24 Commands required to run the test network with deployment scenario 1

7 End-to-End connectivity

A central goal of this work was to have End-to-End 5G connectivity. This means that a UE can send and receive packets to/from the internet over a radio channel and a 5G core network. This chapter describes the flow of UE packets through the test network. Throughput measurements are done to show the capabilities of the test network.

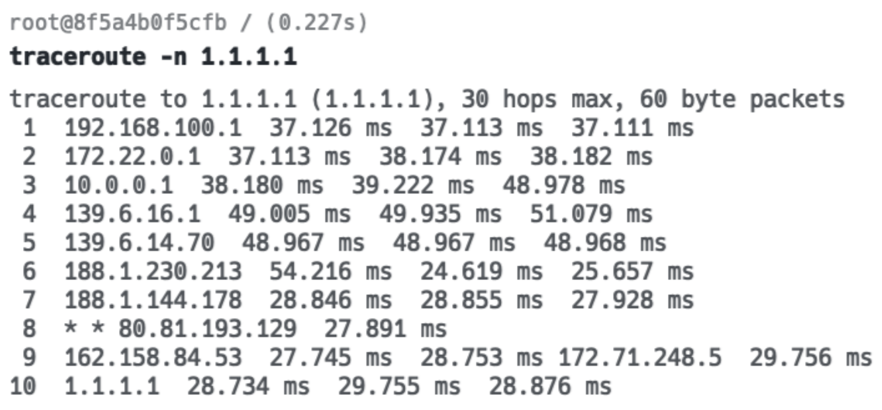
When the Open5GS UPF container starts, it creates an interface called *ogstun* with the 192.168.100.1 address used as a tunnel interface for GTP. An iptables rule is used in the UPF container for Network Address Translation:

iptables -t nat -A POSTROUTING -s 192.168.100.0/24 ! -o ogstun
-j MASQUERADE.

This rule ensures that outgoing packets from UEs with the private 192.168.100.0/24 source network address are translated to the host IP address to enable communication with the internet. After the simulated UE successfully connects to the gNodeB, a device called *tun_srsue* gets created inside the UE container. The device's IP address is dynamically assigned from the 192.168.100.0/24 network. Addresses are assigned by the Session Management Function (SMF) of the 5G core which keeps track of which addresses are already used by other UEs. When a UE sends packets over the simulated radio channel, they take the following path through the network: UE -> gNodeB -> AMF -> SMF -> UPF -> Internet.

In order to verify End-to-End connectivity, requests are sent from the simulated UE over the 5G network to the internet. For testing purposes, the CloudFlare DNS server 1.1.1.1 was used as a target for ICMP Echo requests. It was chosen because it is known to respond to ICMP Echo requests, has low latency, and is guaranteed to be outside the TH network.

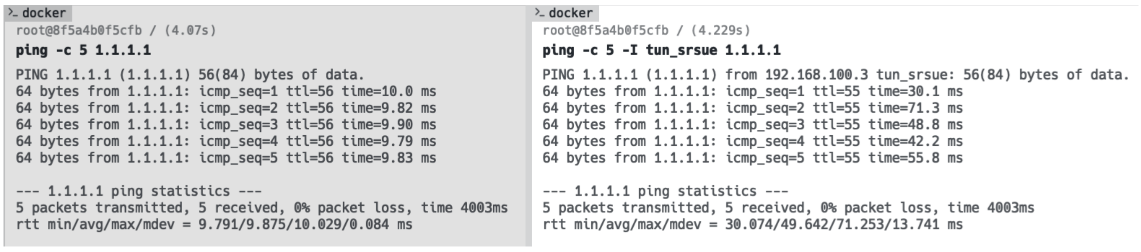
In order to perform requests to the internet, a shell is spawned within the UE container. By default, all outgoing requests will use the Docker network default gateway. The default route must be changed to use the *tun_srsue* device to ensure that the simulated radio channel is used. For each deployment scenario, a script exists to change the default routes of all UE containers accordingly. See the README.md file (appendix A) for details. Once the default route is updated, a traceroute reveals that the next hop uses the 192.168.100.1 address (see figure 25). This address belongs to the UPF *ogstun* interface.



```
root@8f5a4b0f5cfb / (0.227s)
traceroute -n 1.1.1.1
traceroute to 1.1.1.1 (1.1.1.1), 30 hops max, 60 byte packets
 1  192.168.100.1  37.126 ms  37.113 ms  37.111 ms
 2  172.22.0.1  37.113 ms  38.174 ms  38.182 ms
 3  10.0.0.1  38.180 ms  39.222 ms  48.978 ms
 4  139.6.16.1  49.005 ms  49.935 ms  51.079 ms
 5  139.6.14.70  48.967 ms  48.967 ms  48.968 ms
 6  188.1.230.213  54.216 ms  24.619 ms  25.657 ms
 7  188.1.144.178  28.846 ms  28.855 ms  27.928 ms
 8  * * 80.81.193.129  27.891 ms
 9  162.158.84.53  27.745 ms  28.753 ms 172.71.248.5  29.756 ms
10  1.1.1.1  28.734 ms  29.755 ms  28.876 ms
```

Figure 25 traceroute over simulated radio channel

The ping tool supports the -I flag, which allows to change the interface to which the ICMP Echo request is sent. That way, connectivity over the simulated radio channel can be verified without changing the default route. Figure 26 shows a side-by-side comparison of pings to the CloudFlare DNS server 1.1.1.1. The left side shows a ping over the Docker network. The right side shows a ping over the simulated radio channel. Both ping commands were run using five ICMP echo requests (one every second) and were launched simultaneously. It can be seen that connections over the simulated radio channel to the internet work without packet loss. However, a delay of around 40ms is introduced. This is expected, as radio simulation involves quite some overhead and is resource-intensive. As explained earlier, sending packets over the simulated radio channel involves extra hops. This can also be seen by the reduced TTL value for the simulated radio channel. The value is only reduced by one, even when multiple hops are involved, because of the GTP tunnel established between the gNodeB and the UPF.



```
docker
root@8f5a4b0f5cfb / (4.07s)
ping -c 5 1.1.1.1
PING 1.1.1.1 (1.1.1.1) 56(84) bytes of data.
64 bytes from 1.1.1.1: icmp_seq=1 ttl=56 time=10.0 ms
64 bytes from 1.1.1.1: icmp_seq=2 ttl=56 time=9.82 ms
64 bytes from 1.1.1.1: icmp_seq=3 ttl=56 time=9.90 ms
64 bytes from 1.1.1.1: icmp_seq=4 ttl=56 time=9.79 ms
64 bytes from 1.1.1.1: icmp_seq=5 ttl=56 time=9.83 ms

--- 1.1.1.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4003ms
rtt min/avg/max/mdev = 9.791/9.875/10.029/0.084 ms
```

```
docker
root@8f5a4b0f5cfb / (4.229s)
ping -c 5 -I tun_srsue 1.1.1.1
PING 1.1.1.1 (1.1.1.1) from 192.168.100.3 tun_srsue: 56(84) bytes of data.
64 bytes from 1.1.1.1: icmp_seq=1 ttl=55 time=30.1 ms
64 bytes from 1.1.1.1: icmp_seq=2 ttl=55 time=71.3 ms
64 bytes from 1.1.1.1: icmp_seq=3 ttl=55 time=48.8 ms
64 bytes from 1.1.1.1: icmp_seq=4 ttl=55 time=42.2 ms
64 bytes from 1.1.1.1: icmp_seq=5 ttl=55 time=55.8 ms

--- 1.1.1.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4003ms
rtt min/avg/max/mdev = 30.074/49.642/71.253/13.741 ms
```

Figure 26 Side-by-side ping comparison

7.1 Throughput measurements

Using deployment scenario 1, the throughput over the simulated radio channel was measured using iperf3. The used iperf3 commands are in the README.md file (appendix A). An iperf3 server was launched on the host. Then, inside the UE container, the default route was updated to use the simulated radio channel. An iperf3 client was started in the container for 45 seconds using TCP and a bitrate of 100Mbit/s. An application layer throughput of around 52 Mbit/s was achieved. Figure 18 shows the monitoring results of this iperf3 session. Note that the bitrate displayed in the dashboard is higher than the value measured by iperf3. This is because the dashboard contains the link layer throughput, which includes protocol overhead from TCP, IPv4, and MAC-NR over UDP. The limiting factor was the CPU performance of the test server. Multiple cores are at 100% utilization during the measurement, as radio simulation is quite resource-intensive.

7.2 Handover

srsRAN release 24.04 introduced support for handover [40]. A handover is the process of a UE moving from one cell to another. The active connection of the UE gets seamlessly transferred from one cell to the other. With handover, it is possible to maintain continuous network connectivity while moving geographically. There are multiple types of handovers. *Intra gNB-DU* handover occurs when the source and target cell are provided by the same gNodeB DU. *Inter gNB-DU and Intra gNB-CU* handover occurs when the source and target cell belong to different gNodeB DUs, but both DUs are managed by the same CU. *Inter gNB-CU* handover describes the process of a UE moving from one cell to another cell, which is managed by a different CU.

Depending on the type of handover, different control messages are necessary to complete it. At the time of writing, srsRAN only supports the Intra gNB-DU handover type. One srsRAN gNodeB creates multiple cells that can vary by frequency or location (e.g., using tri-sector antennas). The gNodeB is then configured to instruct connected UEs to monitor signal quality metrics of neighboring cells in addition to its own cell. An A3 event is triggered when the signal quality of a neighboring cell surpasses that of the serving cell by a predefined threshold for a specified time. This event indicates that a neighboring cell has a better signal condition and may be more suitable for serving the UE. The UE sends the event to its current cell. Upon receiving the measurement report, the current cell evaluates the information and other factors (like cell load, UE speed, and network policies) to decide whether a handover is justified. If so, it initiates the handover process to the target cell.

The ability to initiate handovers would be an excellent addition to the test network, as handovers are a vital feature of mobile networks. The GNU Radio Companion was used to simulate that another cell has a better signal quality than the current cell, as explained in chapter 6.5.2. The path loss value of a UE is increased to simulate a UE moving further away from the cell tower. It turned out that the srsRAN_4G project used for the simulated UEs does not yet support A3 events and handover for 5G. Therefore, analyzing a 5G handover using the test network was impossible. This could be accomplished in the future using real radio hardware or after an update to the UE simulation software.

8 Extending the test network with RAN Intelligent Controllers

Based on the design explained in chapter 6, a fully functional 5G SA network with End-to-End connectivity is implemented. This network is extended with RICs that run xApps to monitor the network. One final goal of this work was to integrate srsRAN with an external O-RAN SC Near-RT RIC. However, during earlier stages of this work, FlexRIC was the only Near-RT RIC verified to work with srsRAN. For this reason, and because FlexRIC is more straightforward to deploy, this work initially focused on integrating srsRAN into FlexRIC. This is described in chapter 8.2. Once the FlexRIC integration was complete, the focus was on the O-RAN SC Near-RT RIC. Chapter 8.3 is about the integration of srsRAN with the O-RAN SC Near-RT RIC. Finally, chapter 8.4 compares both RICs and how they integrate into srsRAN.

8.1 srsRAN E2 service models

srsRAN implements the E2 interface to connect to a Near-RT RIC. The E2 interface is still under development, and its current version is limited in functionality. The srsRAN E2 interface only supports the *E2SM_RC* and *E2SM_KPM* service models. [41]

The *E2SM_RC* service model specifies capabilities for RAN control, including radio access control or radio bear control. srsRAN only supports control service style 2. As the radio channel in this work is emulated, most radio control operations do not apply. Therefore, this service model has not been evaluated further.

The *E2SM_KPM* service model enables the report of measurements from the disaggregated gNodeB. srsRAN supports all five report service styles: [42]

1. *E2 Node measurement* - Report KPIs for a specific node in the RAN.
2. *E2 Node Measurement for a single UE* - Report KPIs for a single UE.
3. *Condition-based, UE-level E2 Node Measurement* - Report KPIs for UEs that fulfill predefined conditions, such as signal degradation, handover failures, or congestion.
4. *Common Condition-based, UE-level Measurement* - Similar to style 3, except that the reporting conditions are common across multiple UEs, enabling synchronized reporting for a group of UEs experiencing similar conditions.
5. *E2 Node Measurement for multiple UEs* - Report aggregated KPIs for multiple UEs.

Condition-based reporting helps reduce unnecessary reporting traffic by only sending reports when significant events occur, optimizing network resource utilization.

The following six O-RAN defined metrics are exposed:

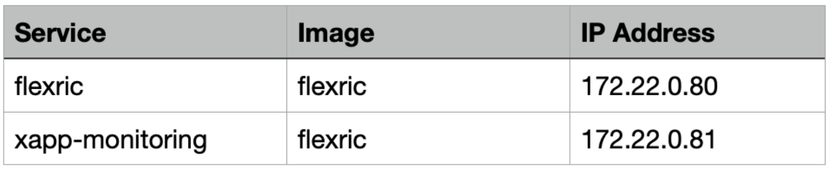
- *DRB.UEThpDl* - DL throughput
- *DRB.UEThpUl* - UL throughput
- *DRB.RlcPacketDropRateDl* - UL packet success rate
- *DRB.PacketSuccessRateUlgNBUu* - RLC DL packet drop rate
- *DRB.RlcSduTransmittedVolumeDL* - RLC DL transmitted Service Data Unit (SDU) volume
- *DRB.RlcSduTransmittedVolumeUL* - RLC UL transmitted SDU volume

The monitoring period is limited by srsRAN to 1s. srsRAN uses KPM version 3, which was published in March 2023. KPM version 4 was published in October 2023 but is not yet supported by srsRAN or the RICs evaluated in this work.

8.2 FlexRIC

FlexRIC is an O-RAN compatible Near-RT RIC. It was published by EURECOM in 2021 and is developed under the Mosaic5G community on GitLab [43]. Mosaic5G is part of the OpenAirInterface Software Alliance, a non-profit organization focusing on the development of 5G RAN and core network technologies [44].

The goal of FlexRIC is to be a “flexible and efficient software development kit (SDK) that enables to build specialized serviceoriented controllers” [45]. In contrast to the O-RAN SC reference implementation, FlexRIC does not require running all its components and xApps in isolated containers orchestrated by Kubernetes. The authors argue that enforcing the usage of Kubernetes "couples the RIC with a containerized technology, contrary to an open technologically agnostic design" [45] and introduces unneeded overhead.



Service	Image	IP Address
flexric	flexric	172.22.0.80
xapp-monitoring	flexric	172.22.0.81

Table 8 FlexRIC deployment

As shown in table 8, FlexRIC is deployed using a single Docker container. An additional container is deployed to run a monitoring xApp. Both containers share the same image, which contains the FlexRIC binary and the compiled xApp. FlexRIC is under active development and is developed in multiple git branches. For this deployment, the br-flexric branch is used, as SRS recommends. FlexRIC is built using the *-DKPM_ VERSION=KPM_ V3* flag to ensure KPM version 3 (the version that srsRAN uses) is used. By default, version 2 is used.

Deployment scenario 3 (see chapter 6.5.3) is used to deploy a gNodeB that connects to FlexRIC. Figure 29 shows how the gNodeB is configured to integrate FlexRIC. The E2 agent and KPM E2SM get enabled, and the address of FlexRIC is configured. During startup, the gNodeB initiates a connection to the RIC as explained in chapter 4.1.2.

Figure 27 shows how the xApp is configured to receive monitoring data. The name is set to KPM, which refers to the E2SM. The time is set to 1000 to get updated data every second. srsRAN only supports a period of 1 second. The format refers to the Report Service Style 1, which indicates that KPIs for a specific RAN node is requested. The ran_type configuration ensures that metrics from a gNodeB are returned. The xApp subscribes for two metrics: *DRB.UEThpDl* and *DRB.UEThpUl*.

```
Sub_ORAN_SM_List = ({
  name = "KPM",
  time = 1000,
  format = 1,
  ran_type = "ngran_gNB",
  actions = (
    { name = "DRB.UEThpDl" },
    { name = "DRB.UEThpUl" }
  )
})
```

Figure 27 FlexRIC monitoring xApp configuration (shortened)
See /2-flexric/config/xapp-monitoring/xapp.conf in the git repository (appendix A) for the full configuration file.

It is required to start the xApp after the UE is successfully connected. Otherwise, FlexRIC can not decode the RICIndication message successfully and crashes. This is due to an issue with srsRAN, which sends an empty RICIndication message when no UE is connected [46]. Once the xApp runs, it prints the requested KPMs every second. As the xApp requested 2 KPMs (UE downlink and UE uplink), two lines will be printed per second. Values printed by the xApp are not labeled, but the order of the lines matches the order of the requested KPMs in the actions configuration of the xApp.

8.3 O-RAN SC Near-RT RIC

The O-RAN SC provides a reference implementation of a Near-RT RIC. The O-RAN SC is partnering with the O-RAN Alliance and the Linux Foundation to create an open RAN solution. This work uses the current I release, released in December 2023.

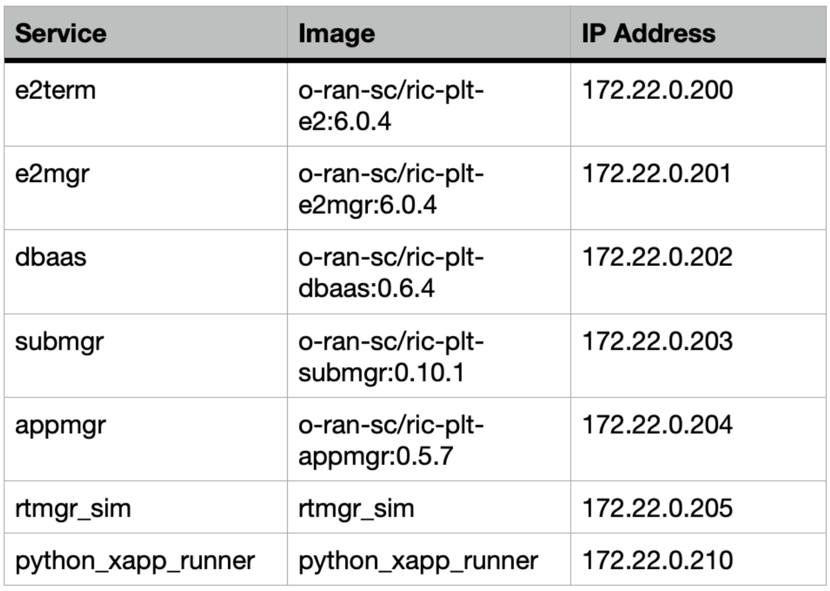
The O-RAN SC Near-RT RIC is meant to be deployed in a Kubernetes cluster. Kubernetes builds on container technology and provides an automated system for deploying, scaling, and managing containerized applications on a multi-node cloud infrastructure. It is the most widely used platform businesses worldwide use for container orchestration [47].

8.3.1 Docker Compose-based deployment

While integrating srsRAN into an external Kubernetes-based Near-RT RIC, unsolved problems were encountered. The default xApps provided by the O-RAN SC resulted in crashes or did not produce the expected output. Luckily, SRS was verifying the compatibility of srsRAN with the O-RAN SC Near-RT RIC at that time. Therefore, they provide a stripped-down version of the Near-RT RIC that does not require a Kubernetes cluster to run but instead uses plain Docker containers. The official O-RAN container images are used. Using Docker Compose instead of Kubernetes has the advantage that deployment is more straightforward, and the Near-RT RIC can run on the same host as the other network services. The Routing Manager (rtmgr) requires a Kubernetes environment and is therefore simulated using an implementation provided by SRS. Table 9 shows the Docker services, along with their image and IP address needed to deploy the O-RAN SC Near-RT RIC. SRS provides an xApp Python framework and a monitoring xApp that is also deployed.

Deployment scenario 4 (see chapter 6.5.4) is used to deploy a gNodeB that connects to the Near-RT RIC. Figure 29 shows how the gNodeB E2 interface is configured. The E2 agent and KPM E2SM get enabled, and the address of the Near-RT RIC is configured. During startup, the gNodeB initiates a connection to the RIC as explained in chapter 4.1.2.

The sample monitoring xApp provided by SRS has been slightly modified to integrate into the existing test network. Hardcoded IP addresses have been replaced to use environment variables instead. The source code, as well as deployment instructions and how to start the xApp, are available in the git repository (appendix A). Figure 28 shows a sample output that the xApp generates every second. Compared to the FlexRIC xApp, the output is enhanced using metadata.



Service	Image	IP Address
e2term	o-ran-sc/ric-plt-e2:6.0.4	172.22.0.200
e2mgr	o-ran-sc/ric-plt-e2mgr:6.0.4	172.22.0.201
dbaas	o-ran-sc/ric-plt-dbaas:0.6.4	172.22.0.202
submgr	o-ran-sc/ric-plt-submgr:0.10.1	172.22.0.203
appmgr	o-ran-sc/ric-plt-appmgr:0.5.7	172.22.0.204
rtmgr_sim	rtmgr_sim	172.22.0.205
python_xapp_runner	python_xapp_runner	172.22.0.210

Table 9 O-RAN SC Near-RT RIC deployment

```
RIC Indication Received from gnb_001_001_000004 for Subscription ID: 1
E2SM_KPM RIC Indication Content:
-ColletStartTime: 2024-05-29 09:33:25
-Measurements Data:
-granulPeriod: 100
--Metric: DRB.UEThpDl, Value: [53258]
--Metric: DRB.UEThpUl, Value: [965]
```

Figure 28 Python monitoring xApp output

8.3.2 Kubernetes based deployment

A separate host is used to deploy Kubernetes. Deploying Kubernetes on the same host used for the rest of the test network would interfere with the existing Docker Compose-based container stack. For a test network, a fully-fledged Kubernetes cluster is not feasible. Therefore, a more lightweight Kubernetes distribution called k3s is used. The details on deploying Kubernetes and the O-RAN Near-RT RIC are out of the scope of this work. A deployment script provided by PROCYDE is used to deploy Kubernetes and the Near-RT RIC.

Deployment scenario 5 introduced in chapter 6.5.5 is used to deploy a gNodeB that connects to the externally hosted Near-RT RIC. The environment variable EXTERNAL_E2TERM_IP is used to set the IP address of the external host.

While it was possible to establish an E2 connection between the gNodeB and the O-RAN SC Near-RT RIC, it was impossible to get a monitoring xApp running. The kpimon-go xApp provided by the O-RAN SC crashes when it tries to send the E2SubscriptionMessage. Also, it does not subscribe to newly connected E2 nodes.

All nodes need to be connected before the xApp starts. Using the python xApp from the Docker Compose-based deployment was impossible because it is tailored towards the Docker environment.

```
e2:
  enable_du_e2: true
  e2sm_kpm_enabled: true
  addr: RIC_IP
  bind_addr: GNB_IP
  port: 36421
```

Figure 29 gNodeB E2 configuration for RIC integration

8.4 FlexRIC vs. O-RAN SC Near-RT RIC

It was possible to integrate both RICs into srsRAN in terms of establishing an E2 connection from the gNodeB to the RIC. Both RICs provide open-source xApps for basic monitoring. However, for both RICs, the deployment of xApps could have been more straightforward and required manual patches to components. Components need to be started in the correct order, and fail-safes like reconnecting on connection loss or detecting newly connected RAN nodes are not implemented or do not work reliably. During testing, crashes were experienced on multiple components involved. Running a monitoring xApp successfully on the Kubernetes-based O-RAN SC Near-RT RIC was impossible.

Table 10 shows a comparison in CPU and memory requirements between a minimal FlexRIC and O-RAN SC Near-RT RIC deployment. The O-RAN RIC requires more resources than FlexRIC. This is because the O-RAN RIC is split into multiple functions, each running inside its own container. Despite that, the resource requirements of the O-RAN deployment are manageable and should not be problematic. Performance and resource requirements in the real world using hundreds of connected E2 nodes could not be evaluated in this work. The authors of FlexRIC see the disaggregation of functions into multiple containers as overhead [45], but the usage of containerization and Kubernetes has some essential benefits. Components can be individually managed and scaled based on resource requirements. Isolating components in containers has security benefits, especially when it comes to running external xApps. Security policies can be applied individually per component. To summarize, FlexRIC is more straightforward to deploy and requires fewer resources, but the O-RAN Near-RT RIC profits from a cloud-native deployment, especially regarding scalability. The FlexRIC milestone contains an entry to support cloud-native deployment using Helm on Kubernetes, which is still in progress to the author's knowledge [48].

```
NAME                  CPU %      MEM USAGE  /  LIMIT
xapp-monitoring       0.01%      1.516MiB  /  15.61GiB
flexric               0.00%      2.195MiB  /  15.61GiB

NAME                  CPU %      MEM USAGE  /  LIMIT
ric_dbaas             0.13%      2.445MiB  /  15.61GiB
ric_appmgr            0.00%      8.262MiB  /  15.61GiB
python_xapp_runner    0.00%      1.668MiB  /  15.61GiB
ric_e2term            0.03%      2.23MiB  /  15.61GiB
ric_submgr            0.04%      10.87MiB  /  15.61GiB
ric_e2mgr             0.02%      7.188MiB  /  15.61GiB
ric_rtmgr_sim         0.00%      2.082MiB  /  15.61GiB
```

Table 10 Resource usage comparison between FlexRIC (top) and O-RAN-SC RIC (bottom)

Functionality-wise, both RICs support the E2 service models that srsRAN exposes. Additional RIC functionality is not evaluated in this work. FlexRIC and the O-RAN SC Near-RT RIC are actively developed. During this work, some problems experienced have already been resolved by new releases. Due to missing production-grade deployment capabilities and scalability, FlexRIC is more suitable for research and experimentation. Due to its roots as a research project, it does not follow the O-RAN spec as closely as the O-RAN SC implementation. The O-RAN SC Near-RT RIC is more suitable for production deployments.

9 Outlook

In this work, a 5G network was implemented. This includes a 5G Standalone core network, a Radio Access Network, as well as RAN Intelligent Controllers. The RAN is implemented using srsRAN, which provides a gNodeB implementation. The User Equipment is simulated using the srsUE software. A radio channel is emulated using zeroMQ.

It was possible to fulfill all the goals set for this work. A 5G network with End-to-End connectivity was implemented using open-source software. The deployment is straightforward and portable due to the use of Docker Compose. Packet captures, Logs and Monitoring dashboards provide visibility into the test network. Different deployment scenarios connect one or multiple UEs and integrate various RICs. The test network is extendable by adding new deployment scenarios with different configurations. New upstream software updates did not break existing test network deployments, and it was possible to integrate them manually. A README was created that contains step-by-step instructions on how to deploy the test network.

The test network provides a solid foundation for future researchers in open RAN. A possible extension to the test network would be to use real radio hardware. That way, the exposed metrics would be much more valuable. Also, handover and MIMO could be tested. Once the Y1 interface is implemented by the O-RAN SC, adding Y1 consumers to the test network would be interesting. A SMO Framework and Non-RT RIC could be added to the test network. With the rise of artificial intelligence, RAN automation based on AI incorporated into rApps and xApps is becoming an interesting research topic.

The open RAN is continuously evolving. During this work, SRS, the O-RAN SC, and other contributors continuously updated their open source O-RAN software, making them more production-ready, step by step. Ericsson, a major player in the mobile network industry, joined the OpenAirInterface Software Alliance this year, and major telecom providers are already evaluating O-RAN solutions. SRS plays a vital role in the adaptation of O-RAN, by providing open-source O-RAN-compatible RAN solutions. Other vendors, like the OpenAirInterface Software Alliance, are developing similar solutions. This competition will likely accelerate innovation. Looking forward, O-RAN will significantly transform the mobile industry by fostering a more competitive, cost-efficient, and innovative ecosystem.

References

[1] "O-RAN ALLIANCE e.V," [Online]. Available: `https://www.o-ran.org/` (visited on 05/31/2024).

[2] "Report ITU-R M.2410-0; Minimum requirements related to technical performance for IMT-2020 radio interface(s)," ITU, Ed., Nov. 2017. [Online]. Available: `https://www.itu.int/dms_pub/itu-r/opb/rep/R-REP-M.2410-2017-PDF-E.pdf` (visited on 03/29/2024).

[3] Prof. Dr. Dettmar, "Standards and history of mobile and wireless radio," Aug. 15, 2022.

[4] Puneet Jain, "Rel-18 Status and Rel-19 Progress in TSG SA," 3GPP, Ed., Nov. 2023. [Online]. Available: `https://www.3gpp.org/technologies/rel-18` (visited on 03/29/2024).

[5] E. Dahlman, S. Parkvall, and J. Sköld, "Radio-Interface Architecture," in *5G NR: The Next Generation Wireless Access Technology*, Elsevier, 2018, pp. 73–102, ISBN: 978-0-12-814323-0. DOI: `10.1016/B978-0-12-814323-0.00006-5`. [Online]. Available: `https://linkinghub.elsevier.com/retrieve/pii/B9780128143230000065` (visited on 05/07/2024).

[6] Alexander Geckeler. "„5G Plus" startet im gesamten 5G Netz von o2: o2 Telefónica aktiviert neueste Netztechnologie für die Digitalisierung Deutschlands," [Online]. Available: `https://www.telefonica.de/news/corporate/2023/10/5g-plus-startet-im-gesamten-5g-netz-von-o2-o2-telefonica-aktiviert-neueste-netztechnologie-fuer-die-digitalisierung-deutschlands.html` (visited on 05/23/2024).

[7] Tobias Krzossa. "5G-Netzausbau: Vodafone schaltet 5G-Standalone großflächig frei," [Online]. Available: `https://newsroom.vodafone.de/netz/vodafone-startet-umruestung-zum-modernsten-5g-netz-europas` (visited on 05/23/2024).

[8] Volker Briegleb, "Telekom öffnet 5G Standalone 2024 für Privatkunden," *heise online*, Dec. 12, 2023. [Online]. Available: `https://www.heise.de/news/Telekom-5G-Standalone-fuer-Privatkunden-im-zweiten-Halbjahr-2024-9572780.html` (visited on 05/23/2024).

[9] 3GPP, *General Packet Radio System (GPRS) Tunnelling Protocol User Plane (GTPv1-U)*, version 17.4.0, Sep. 2022. [Online]. Available: `https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=1699` (visited on 05/23/2024).

[10] M. Polese, L. Bonati, S. D'Oro, S. Basagni, and T. Melodia. "Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges." arXiv: 2202.01032 [cs, eess], [Online]. Available: http://arxiv.org/abs/2202.01032 (visited on 01/18/2024), preprint.

[11] O-RAN Work Group 1, *O-RAN Architecture Description*, version 11.00, Feb. 2024. [Online]. Available: https://orandownloadsweb.azurewebsites.net/download?id=564 (visited on 02/29/2024).

[12] 3GPP, *NG-RAN; Architecture description*, version 17.7.0, Dec. 2023. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3219 (visited on 03/07/2024).

[13] 3GPP, *System architecture for the 5G System (5GS)*, version 17.11.0, Dec. 19, 2023. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3144 (visited on 03/08/2024).

[14] O-RAN Work Group 3, *Y1 interface: General Aspects and Principles*, version 01.00, Nov. 18, 2023. [Online]. Available: https://orandownloadsweb.azurewebsites.net/download?id=585 (visited on 02/29/2024).

[15] O-RAN Work Group 2, *Non-RT RIC: Architecture*, version 05.00, Nov. 16, 2023. [Online]. Available: https://orandownloadsweb.azurewebsites.net/download?id=573 (visited on 03/14/2024).

[16] 3GPP, *Study on Central Unit (CU) - Distributed Unit (DU) lower layer split for NR*, version 15.0.0, Dec. 2017. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3364 (visited on 03/28/2024).

[17] Roger Niklasson, Rajavarma Bhyrraju, Kedar Thakar, David Espadas, Gustavo Hylander, and Justin Paul, "An intelligent platform: The use of O-RAN's SMO as the enabler for openness and innovation in the RAN domain," Ericsson, Nov. 2021. [Online]. Available: https://www.ericsson.com/en/reports-and-papers/white-papers/smo-enabling-intelligent-ran-operations (visited on 05/06/2024).

[18] 3GPP, *F1 application protocol (F1AP)*, version 17.8.0, Mar. 2024. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3260.

[19] 3GPP, *E1 Application Protocol (E1AP)*, version 17.8.0, Mar. 2024. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3957 (visited on 05/08/2024).

[20] 3GPP, *E1 general aspects and principles*, version 17.8.0, Dec. 2023. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3954 (visited on 05/08/2024).

[21] 3GPP, *NG Application Protocol (NGAP)*, version 17.8.0, Mar. 2024. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3223 (visited on 05/23/2024).

[22] O-RAN Work Group 3, *Near-RT RIC Architecture*, version 05.00, Jul. 29, 2023. [Online]. Available: https://orandownloadsweb.azurewebsites.net/download?id=518 (visited on 02/29/2024).

[23] O-RAN Work Group 3, *E2 General Aspects and Principles (E2GAP*, version 05.00, Nov. 21, 2023. [Online]. Available: https://orandownloadsweb.azurewebsites.net/download?id=581 (visited on 05/22/2024).

[24] O-RAN Work Group 2, *A1 interface: General Aspects and Principles*, version 03.02, Nov. 30, 2023. [Online]. Available: https://orandownloadsweb.azurewebsites.net/download?id=567 (visited on 05/08/2024).

[25] O-RAN Work Group 10, *Operations and Maintenance Interface*, version 12.00, Oct. 30, 2023. [Online]. Available: https://orandownloadsweb.azurewebsites.net/download?id=615 (visited on 05/08/2024).

[26] O-RAN Work Group 6, *O2 Interface General Aspects and Principles*, version 06.00, Nov. 28, 2023. [Online]. Available: https://orandownloadsweb.azurewebsites.net/download?id=601 (visited on 05/22/2024).

[27] "O-RAN SC Nexus Repository," [Online]. Available: https://nexus3.o-ran-sc.org/#browse/browse:docker.public:v2%2Fo-ran-sc (visited on 05/22/2024).

[28] "Docker Hub Container Image Library," [Online]. Available: https://hub.docker.com/ (visited on 05/22/2024).

[29] L. Mamushiane, A. Lysko, H. Kobo, and J. Mwangama, "Deploying a Stable 5G SA Testbed Using srsRAN and Open5GS: UE Integration and Troubleshooting Towards Network Slicing," in *2023 International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (icABCD)*, Aug. 2023. DOI: 10.1109/icABCD59051.2023.10220512. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10220512 (visited on 01/18/2024).

[30] *Open5gs/open5gs*, Open5GS, [Online]. Available: https://github.com/open5gs/open5gs (visited on 03/01/2024).

[31] "Open5GS - Open Collective," [Online]. Available: https://opencollective.com/open5gs (visited on 04/30/2024).

[32] *Srsran/srsRAN_Project*, srsRAN, [Online]. Available: https://github.com/srsran/srsRAN_Project (visited on 03/01/2024).

[33] *Srsran/srsRAN_4G*, srsRAN, [Online]. Available: https://github.com/srsran/srsRAN_4G (visited on 03/01/2024).

[34] "Features and Roadmap — srsRAN Project documentation," [Online]. Available: https://docs.srsran.com/projects/project/en/latest/general/source/2_features_and_roadmap.html#coming-soon (visited on 05/22/2024).

[35] "ZeroMQ Documentation," ZeroMQ, [Online]. Available: https://zeromq.org/get-started/ (visited on 05/22/2024).

[36] "ZeroMQ Documentation Chapter 2 - Sockets and Patterns," [Online]. Available: https://zguide.zeromq.org/docs/chapter2/ (visited on 04/30/2024).

[37] "srsRAN gNB with srsUE — srsRAN Project documentation," [Online]. Available: https://docs.srsran.com/projects/project/en/latest/tutorials/source/srsUE/source/index.html#gnu-radio-companion (visited on 05/22/2024).

[38] "Problems when trying to run FlexRic and xApp with a COTS UE · srsran/srsRAN_Project · Discussion #567," [Online]. Available: https://github.com/srsran/srsRAN_Project/discussions/567 (visited on 05/02/2024).

[39] "Outputs — srsRAN Project documentation," [Online]. Available: https://docs.srsran.com/projects/project/en/latest/user_manuals/source/outputs.html#pcaps (visited on 05/22/2024).

[40] "Release srsRAN Project 24.04 · srsran/srsRAN_Project," GitHub, [Online]. Available: https://github.com/srsran/srsRAN_Project/releases/tag/release_24_04 (visited on 05/06/2024).

[41] "O-RAN NearRT-RIC and xApp — srsRAN Project documentation," [Online]. Available: https://docs.srsran.com/projects/project/en/latest/tutorials/source/near-rt-ric/source/index.html (visited on 05/02/2024).

[42] O-RAN Work Group 3, *E2 Service Model (E2SM) KPM*, version 04.00, Jul. 29, 2023. [Online]. Available: https://orandownloadsweb.azurewebsites.net/download?id=514 (visited on 05/22/2024).

[43] "mosaic5g / Flexric · GitLab," GitLab, [Online]. Available: https://gitlab.eurecom.fr/mosaic5g/flexric (visited on 04/30/2024).

[44] "Mosaic5G," [Online]. Available: https://mosaic5g.io/ (visited on 05/02/2024).

[45] R. Schmidt, M. Irazabal, and N. Nikaein, "FlexRIC: An SDK for next-generation SD-RANs," in *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '21, New York, NY, USA: Association for Computing Machinery, Dec. 3, 2021, pp. 411–425, ISBN: 978-1-4503-9098-9. DOI: `10.1145/3485983.3494870`. [Online]. Available: `https://doi.org/10.1145/3485983.3494870` (visited on 05/02/2024).

[46] "Which E2SM KPM version does srsRAN use? · srsran/srsRAN_Project · Discussion #596," GitHub, [Online]. Available: `https://github.com/srsran/srsRAN_Project/discussions/596` (visited on 05/02/2024).

[47] "Production-Grade Container Orchestration," [Online]. Available: `https://kubernetes.io/` (visited on 05/22/2024).

[48] "Enable cloud-native deployment · Meilensteine · mosaic5g / Flexric · GitLab," GitLab, [Online]. Available: `https://gitlab.eurecom.fr/mosaic5g/flexric/-/milestones/2` (visited on 05/02/2024).

Disclaimer on the Use of Artificial Intelligence

This thesis has benefited from using Artificial Intelligence (AI) technologies. The TH Cologne GPT-Lab, introduced at the beginning of May 2024, was instrumental during the late stages of research in aggregating supplemental information. The application of AI was limited to supporting aspects such as enhancing the literature review and facilitating the understanding of complex technical standards. The Grammarly AI was used to check for spelling and grammar mistakes and to perform plagiarism checks.

Appendices

A Git repository

Contains the source code, configuration files, and documentation for the test network. Available at the DN.Lab GitLab server under 5G-Projects / srsRAN.

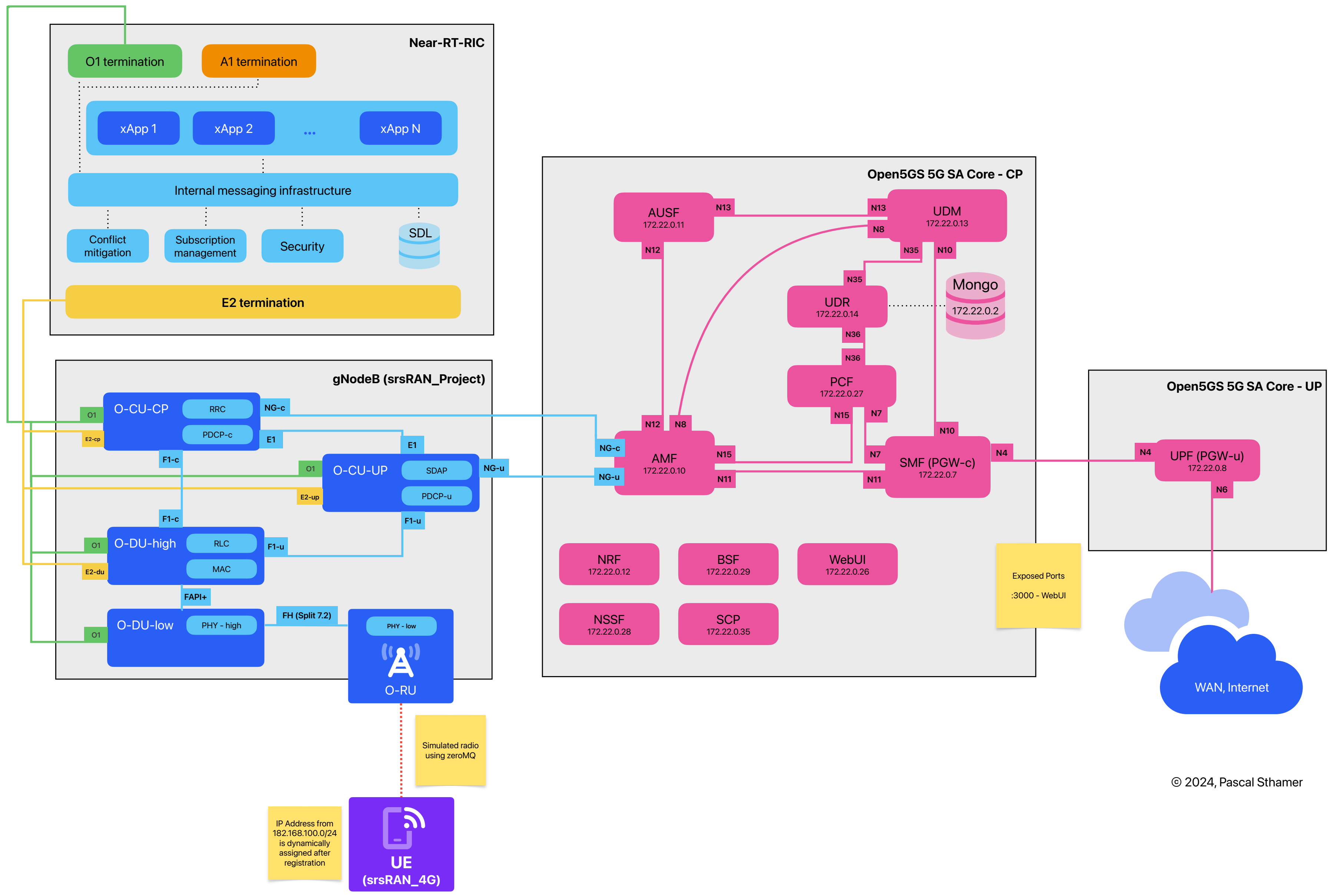
B FlexRIC Dockerfile

```
FROM ubuntu:jammy

ENV DEBIAN_FRONTEND=noninteractive

# Install updates and dependencies, make gcc-10 the default (flexric
   doesn't compile with gcc-11, which is the Ubuntu 22 default)
RUN apt-get update && \
    apt-get -y install build-essential cmake make gcc-10 git swig
   libsctp-dev python3 cmake-curses-gui python3-dev pkg-config
   libconfig-dev libconfig++-dev net-tools iputils-ping iproute2 && \
    update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-10 10

# Cloning takes a while - make it a single docker layer for caching
   purposes if the build command changes
RUN git clone --branch br-flexric --single-branch --depth 1
   https://gitlab.eurecom.fr/mosaic5g/flexric.git

RUN cd flexric && mkdir build && cd build && \
    cmake -DKPM_VERSION=KPM_V3 -DXAPP_DB=NONE_XAPP ../ && make
   -j`nproc` && make install

CMD /usr/local/bin/flexric/ric/nearRT-RIC
```

C docker-compose.yaml file for shared monitoring services

```yaml
version: "3.9"
name: shared
services:
  influxdb:
    image: influxdb:2.7
    container_name: influxdb
    env_file:
      - .env
    volumes:
      - influxdbdata:/var/lib/influxdb2
      - /etc/timezone:/etc/timezone:ro
      - /etc/localtime:/etc/localtime:ro
    ports:
      - "8086:8086/tcp"
    networks:
      default:
        ipv4_address: ${INFLUXDB_IP}

  prometheus:
    image: ubuntu/prometheus:2-22.04_stable
    container_name: prometheus
    env_file:
      - .env
    volumes:
      - ./config/prometheus.yml:/etc/prometheus/prometheus.yml:ro
      - /etc/timezone:/etc/timezone:ro
      - /etc/localtime:/etc/localtime:ro
    ports:
      - "9090:9090/tcp"
    networks:
      default:
        ipv4_address: ${PROMETHEUS_IP}

  grafana:
    image: grafana/grafana-oss:10.3.1
    container_name: grafana
    env_file:
      - .env
    volumes:
      - grafanadata:/var/lib/grafana
      - ./config/grafana/provisioning:/etc/grafana/provisioning
      - ./config/grafana/dashboards:/etc/dashboards
      - /etc/timezone:/etc/timezone:ro
      - /etc/localtime:/etc/localtime:ro
    ports:
      - "8080:3000/tcp"
    networks:
      default:
        ipv4_address: ${GRAFANA_IP}

networks:
  default:
    ipam:
      config:
        - subnet: ${TEST_NETWORK_SUBNET}
    name: ${TEST_NETWORK_NAME}

volumes:
  influxdbdata: {}
  grafanadata: {}
```

D Test network architecture diagramm


Near-RT-RIC
O1 termination
A1 termination
xApp 1
xApp 2
...
xApp N
Internal messaging infrastructure
Conflict mitigation
Subscription management
Security
SDL
E2 termination
gNodeB (srsRAN_Project)
O-CU-CP
RRC
PDCP-c
NG-c
E1
O1
E2-cp
F1-c
O-CU-UP
SDAP
PDCP-u
NG-u
E2-up
F1-u
O-DU-high
RLC
MAC
E2-du
FAPI+
O-DU-low
PHY - high
FH (Split 7.2)
PHY - low
O-RU
Simulated radio using zeroMQ
IP Address from 182.168.100.0/24 is dynamically assigned after registration
UE (srsRAN_4G)
Open5GS 5G SA Core - CP
AUSF 172.22.0.11
UDM 172.22.0.13
N13
N8
N12
N35
N10
UDR 172.22.0.14
Mongo 172.22.0.2
N36
PCF 172.22.0.27
N15
N7
AMF 172.22.0.10
N11
SMF (PGW-c) 172.22.0.7
N4
NRF 172.22.0.12
BSF 172.22.0.29
WebUI 172.22.0.26
NSSF 172.22.0.28
SCP 172.22.0.35
Exposed Ports
:3000 - WebUI
Open5GS 5G SA Core - UP
UPF (PGW-u) 172.22.0.8
N6
WAN, Internet
© 2024, Pascal Sthamer