Check for updates

NIST Series Technical Note
NIST TN 2311

Blueprint for Deploying 5G O-RAN Testbeds: A Guide to Using Diverse O-RAN Software Stacks

Peng Liu
Kyehwan Lee
Fernando J. Cintrón
Simeon Wuthier
Bhadresh Savaliya
Douglas Montgomery
Richard Rouil

This publication is available free of charge from:
https://doi.org/10.6028/NIST.TN.2311

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
U.S. DEPARTMENT OF COMMERCE

NIST Series Technical Note
NIST TN 2311

Blueprint for Deploying 5G O-RAN Testbeds: A Guide to Using Diverse O-RAN Software Stacks

Peng Liu
Associate, Wireless Networks Division
Communications Technology Laboratory
Prometheus Computing LLC, Bethesda, Maryland

Kyehwan Lee
Fernando J. Cintrón
Bhadresh Savaliya
Douglas Montgomery
Richard Rouil
Wireless Networks Division
Communications Technology Laboratory

Simeon Wuthier
Associate, Wireless Networks Division
Communications Technology Laboratory
Georgetown University, Washington, D.C.

This publication is available free of charge from:
https://doi.org/10.6028/NIST.TN.2311

October 2024

U.S. Department of Commerce
Gina M. Raimondo, Secretary

National Institute of Standards and Technology
Laurie E. Locascio, NIST Director and Under Secretary of Commerce for Standards and Technology

Certain equipment, instruments, software, or materials, commercial or non-commercial, are identified in this paper in order to specify the experimental procedure adequately. Such identification does not imply recommendation or endorsement of any product or service by NIST, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

NIST Technical Series Policies
Copyright, Use, and Licensing Statements
NIST Technical Series Publication Identifier Syntax

Publication History
Approved by the NIST Editorial Review Board on 2024-10-09

How to cite this NIST Technical Series Publication:
Liu P, Lee K, Cintrón FJ, Wuthier S, Savaliya B, Montgomery D, Rouil R (2024) Blueprint for Deploying 5G O-RAN Testbeds: A Guide to Using Diverse O-RAN Software Stacks. (National Institute of Standards and Technology, Gaithersburg, MD), NIST TN 2311. https://doi.org/10.6028/NIST.TN.2311

Author ORCID iDs
Peng Liu: 0000-0001-8237-5807
Kyehwan Lee: 0009-0002-0649-4235
Fernando J. Cintrón: 0000-0002-5602-1068
Simeon Wuthier: 0000-0003-4088-7518
Bhadresh Savaliya: 0009-0007-9407-4287
Douglas Montgomery: 0000-0002-5364-9474
Richard Rouil: 0000-0003-0387-0880

Contact Information
richard.rouil@nist.gov

Public Comment Period
Sep. 19, 2024 – Oct. 09, 2024

Submit Comments
richard.rouil@nist.gov
Mailing Address
100 Bureau Dr, Stop 6730
Gaithersburg, MD 20899

Abstract

This documentation serves as a blueprint for new researchers, offering a comprehensive guide on establishing an Open Radio Access Network (O-RAN) testbed from scratch. It details the O-RAN architecture and the supporting software stacks required for each component, and provides both aggregated and disaggregated deployment scenarios tested on our testbeds. The guide provides thorough installation instructions for each software stack we tested. In addition, a testbed example of a disaggregated scenario is used to demonstrate proper configurations and practical operations to test the connection and interoperation between the deployed O-RAN components. Moreover, this documentation introduces our innovative automation tool, designed to streamline the installation and configuration of some O-RAN components, ensuring a more efficient deployment process. This publication aims to equip researchers with the foundational knowledge and practical steps needed to initiate and manage their own O-RAN testbeds effectively.

Keywords

5G New Radio; FlexRIC; O-RAN SC Near-RT RIC; Open RAN; Open5GS; srsRAN 4G; srsRAN Project; Testbed; USRP.

Table of Contents

List of Tables

List of Figures

Acknowledgments

This work is partially funded by the Department of Homeland Security's Science and Technology Directorate (S&T).

Abbreviations

3GPP the Third Generation Partnership Project

AMF Access and Mobility Management Function

CLI Command Line Interface

CPU Central Processing Unit

DL Downlink

E2AP E2 Application Protocol

E2SM E2 Service Model

E2SM-KPM E2 Service Model - Key Performance Measurement

FDD Frequency Division Duplex

GTP-U GPRS Tunneling Protocol User Plane

HSS Home Subscriber Server

IP Internet Protocol

LTE Long Term Evolution

ML Machine Learning

NAT Network Address Translation

near-RT RIC near-real-time RAN intelligent controller

NF Network Function

NGAP Next Generation Application Protocol

non-RT RIC non-real-time RAN intelligent controller

NR New Radio

NRF Network Repository Function

O-CU O-RAN Central Unit

O-DU O-RAN Distributed Unit

O-RAN Open Radio Access Network

O-RU O-RAN Radio Unit

OAI OpenAirInterface

OS Operating System

OSC O-RAN Software Community

PCF Policy Control Function

PCRF Policy and Charging Rules Function

PLMN Public Land Mobile Network

PPS Pulse Per Second

RAN Radio Access Network

RF Radio Frequency

RIC RAN Intelligent Controller

RRM Radio Resource Management

RSRP Reference Signal Receive Power

RX Receiver

SA Standalone

SCTP Stream Control Transmission Protocol

SMO Service Management and Orchestration

TX Transmitter

UDR Unified Data Repository

UE User Equipment

UL Uplink

UPF User Plane Function

USIM Universal Subscriber Identity Module

USRP Universal Software Radio Peripheral

VM Virtual Machine

ZMQ Zero Message Queue

1. Introduction

Before the advent of Open Radio Access Network (O-RAN), Radio Access Networks (RANs) were typically dominated by proprietary, monolithic systems from individual vendors. This often resulted in high costs and limited flexibility. O-RAN addresses these challenges through open standards that enable interoperability between components from different vendors while adhering to the specifications defined by 3GPP.

In advancing O-RAN research and implementation, the development of testbeds is crucial. These testbeds provide controlled and replicable environments for validating and enhancing O-RAN technologies and network performance. They serve as platforms to test various deployment scenarios, assess interoperability between O-RAN components and software stacks from different vendors, and evaluate research outcomes for specific O-RAN tasks. Testbeds offer invaluable insights for large-scale deployments, ensuring that O-RAN solutions are robust, efficient, and ready for real-world applications. In this documentation, we provide a blueprint for new researchers, with a comprehensive guide on establishing an O-RAN testbed from scratch.

1.1. O-RAN Architecture

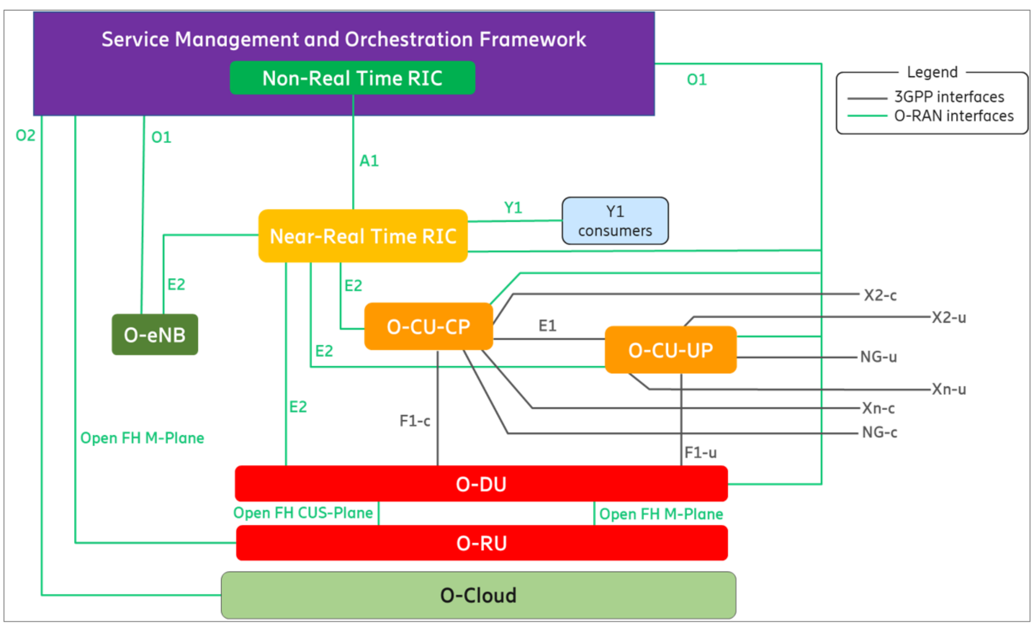


Fig. 1. O-RAN Architecture [1]

The O-RAN architecture was defined by O-RAN Alliance in [1], as shown in Fig. 1. Its key elements include

- Service Management and Orchestration (SMO), which is responsible for RAN domain management;

- non-real-time RAN intelligent controller (non-RT RIC), which runs in SMO and provides intelligent RAN optimization with service and policy management, Machine Learning (ML) model management, and enrichment information for near-real-time RAN intelligent controller (near-RT RIC) functions. The response interval is greater than 1 s. It provides the non-RT RIC applications, i.e., the rApps, to realize the functionality;
- near-RT RIC, which controls the E2 nodes (O-RAN Central Unit (O-CU), O-RAN Distributed Unit (O-DU)) with a response time between 10 ms and 1 s. The control is steered via the policies and enrichment data from non-RT RIC. Radio Resource Management (RRM) is a main function provided by near-RT RIC and is realized by means of E2 Service Models (E2SMs) on near-RT RIC Applications (xApps);
- O-CU, O-DU, and O-RAN Radio Unit (O-RU). The gNB functions defined by the Third Generation Partnership Project (3GPP) are disaggregated and distributed into these O-RAN components and the 3GPP defined interfaces are terminated as shown in Fig. 1. In addition, each of these O-RAN components can be managed to combine with one or more of the others.

In addition, O-Cloud provides a cloud computing platform that can host some of the O-RAN Network Functions (NFs) as mentioned above.

Some of the key connections between the above O-RAN components are realized by the interfaces defined as follows:

- A1 Interface, which exchanges information between non-RT and near-RT RICs to support the services of policy management, enrichment information, and ML model management;
- O1 Interface, which connects SMO with near-RT RIC and E2 nodes for NF management and orchestration;
- E2 Interface, which connects near-RT RIC with E2 nodes and provides near-RT services using E2 Application Protocol (E2AP) and E2SMs.

1.2. Software Stacks for O-RAN Components

The O-RAN participating organizations and researchers have been actively providing standard-compliant solutions to some of the O-RAN components. Some of the open-source options include:

non-RT RIC:

- non-RT RIC from O-RAN Software Community (OSC), which provides both non-RT RIC framework and rApps. It can be deployed in Kubernetes containers.

near-RT RIC:

- near-RT RIC from OSC, which provides both near-RT RIC framework and xApps. It can be deployed in Kubernetes clusters;
- FlexRIC, which provides both near-RT RIC framework and xApps. It can be deployed on a bare metal server/workstation or on a Virtual Machine (VM);
- near-RT RIC from SRS, which is built on the Release I of OSC near-RT RIC to support quick deployment and control over srsRAN gNB with xApps programmed in Python. It can be deployed in a docker container.

E2 Nodes and User Equipments (UEs):

- OpenAirInterface (OAI), which supports CU/DU split, E2 connection to near-RT RIC, and connection to OAI UEs. OAI gNB also supports Split 7.2 fronthaul interfaces to RUs, whereas some implementations also support Split 8. OAI gNB and UE can be deployed on a bare metal server/workstation, and users have found success in deployment in Kubernetes clusters;
- srsRAN, which includes srsRAN Project for 5G O-RAN gNB, and srsRAN 4G for 5G UE. In addition to gNB-UE connection, srsRAN gNB also supports CU/DU split, E2 connection to near-RT RIC, and Split 7.2 and Split 8 fronthaul interfaces to RUs. srsRAN supports deployments on Kubernetes, docker, and on a bare metal server/workstation, or VM.

In addition, 5G core network supports O-RAN architecture by providing NFs such as Access and Mobility Management Function (AMF), User Plane Function (UPF), etc. Some available options include Open5GS, Free5GC, Open5GCore, OAI 5G CN, etc.

This document aims to guide readers through the process of deploying a 5G O-RAN testbed, covering essential aspects such as hardware requirements, software installation of the tested software stacks for the O-RAN components, and deployment scenarios. We start with an introduction to the hardware specifications in our testbed deployments. Following this, we delve into the installation procedures for the software stacks associated with O-RAN components as well as 5G core network, where detailed configurations are discussed. Additionally, various deployment scenarios are documented, which includes both aggregated scenarios, where all or most of the O-RAN components are deployed in one server or workstation, and disaggregated scenarios, where each O-RAN component is deployed in an individual server/workstation and their communications are via a dedicated network. Among the various deployment scenarios, we present how we realize connections between UE and gNB via digital connection (Zero Message Queue (ZMQ)), direct cable connection, and Radio Frequency (RF) channel emulator. Furthermore, we facilitate the seamless deployment of the entire O-RAN testbed by introducing our automation tool. This tool enables automatic installation and configuration of various O-RAN components, as detailed in this documentation, significantly reducing the potential for human error and greatly accelerating deployment times. Finally, we will provide a detailed test example to demonstrate the practical application of a deployed testbed.

2. System Requirements

This section introduces the open-source software for the components of the deployed 5G O-RAN testbed, the deployment scenarios that have been tried, and the hardware and system prerequisites for the radios and servers/workstations.

2.1. Software

- 5G gNB: srsRAN Project,
- 5G UE: srsUE from srsRAN 4G.
- 5G Core: Open5GS,
- RAN Intelligent Controller (RIC): FlexRIC and OSC near-RT RIC

It is important to note that srsRAN 4G supports 5G Standalone (SA) in srsUE by modifying the srsUE configuration file.

2.2. Hardware and System Prerequisites

We use Ettus B210 and X310 for gNB, and Ettus B210 for srsUE. We are also in progress of validating X310 for srsUE, and exploring X410 for both gNB and srsUE. The Universal Software Radio Peripherals (USRPs) share the same Pulse Per Second (PPS) time source and 10 MHz frequency source from an octoclock.

The Central Processing Units (CPUs) in the servers and workstations include:

- Intel Xeon Gold 6246R with 16 cores @ 3.4 GHz,
- Intel Xeon Gold 6334 with 8 Cores @ 3.6 GHz,
- Intel Core i9-12900K with 16 Cores @ 2.4 GHz.

The OSC near-RT RIC can operate on Ubuntu 20.04 (preferred) or Ubuntu 22.04, whereas the other software are installed on Ubuntu 22.04. The servers/workstations on which the gNB and srsUE run have low-latency kernels, and the systems and BIOS are configured to achieve optimal performance [2]:

Low-latency kernel is installed by

```
$ sudo apt-get -y install linux-lowlatency
```

After rebooting the Operating System (OS), make sure `uname -r` indicates that low-latency kernel is successfully deployed:

```
$ uname -r

5.15.0-102-lowlatency
```

For power management:

- In `/etc/default/grub`, disable c-state by:

```
GRUB_CMDLINE_LINUX_DEFAULT="quiet processor.max_cstate=1
intel_idle.max_cstate=0 idle=poll"
```

  Followed by,

```
$ sudo update-grub2
```

- In `/etc/modprobe.d/blacklist.conf`, add the following line to the end of the file:

```
blacklist intel_powerclamp
```

  Then reboot the system.

- When rebooting, change the following items in BIOS:
  - Disable secure booting option,
  - Disable hyperthreading,
  - Enable virtualization,
  - Disable c-state power management functions, and
  - Enable real-time tuning and Intel Turbo boost.

  Although [2] indicates that p-state power management should also be disabled, we experience issues when running `srsran_performance` script provided by [3]. Hence it is excluded from this user manual.

- Set the scaling governor to `performance` by installing `cpufrequtils`

```
$ sudo apt-get install cpufrequtils
```

  and add the following line to `/etc/default/cpufrequtils`

```
GOVERNOR="performance"
```

  Next,

```
$ sudo systemctl disable ondemand.service

$ sudo /etc/init.d/cpufrequtils restart
```

- Verify power management configuration and CPU frequency using `i7z`

```
$ sudo apt install i7z

$ sudo i7z
```

All cores should have `C0 %` as 100 and `Halt(C1) %` as 0.

As the connections between gNB, Open5GS and FlexRIC use the Stream Control Transmission Protocol (SCTP), it should be enabled on the corresponding servers by:

- installing `libsctp-dev`

  ```
  $ sudo apt install libsctp-dev
  ```

- using `lsmod | grep 'sctp'` to check if SCTP is enabled, and if nothing is returned, comment out the lines in `/etc/modprobe.d/sctp.conf`:

  ```
  #install sctp /bin/true
  ```

  and load the SCTP module by:

  ```
  $ sudo modprobe sctp
  ```

To make sure ZMQ or UHD can be used, each server/workstation that runs gNB or srsUE shall install the packages by:

```
$ sudo apt-get install libzmq3-dev libuhd-dev uhd-host
```

After that, download the UHD images by

```
$ sudo uhd_images_downloader
```

3. RAN Components

3.1. gNodeB: srsRAN Project Setup

3.1.1. Installation

The installation procedures below were tested on the srsRAN Project versions 23.5 and 23.10. We also had a successful test with version 23.10.1 with the updates until 12/22/2023 in GitHub repository. It is important to note that this latest updates required some accommodations on the FlexRIC and Open5GS versions/installations. For more information, please refer to Sections 4 and 5.

The supporting packages of the srsRAN Project gNB need to be installed [4]:

```
$ sudo apt-get install cmake make gcc g++ pkg-config libfftw3-dev
  libmbedtls-dev libsctp-dev libyaml-cpp-dev libgtest-dev
```

The srsRAN Project code can be downloaded and installed using the following commands [5]. It is important to note that to enable ZMQ, `-DENABLE_EXPORT=ON -DENABLE_ZEROMQ=ON` needs to be included when running `cmake`:

```
$ git clone https://github.com/srsran/srsRAN_Project.git

$ cd srsRAN_Project

$ mkdir build

$ cd build

$ cmake ../ -DENABLE_EXPORT=ON -DENABLE_ZEROMQ=ON
```

During `cmake`, it is important that ZMQ is found:

```
-- FINDING ZEROMQ.
-- Checking for module 'ZeroMQ'
--   No package 'ZeroMQ' found
-- Found libZEROMQ: /usr/local/include, /usr/local/lib/libzmq.so
```

If not, follow the instructions in Section 2.2 to enable SCTP.

Next,

```
$ make -j `nproc`

$ make test -j `nproc`

$ sudo make install
```

Make sure it passes all the tests.

3.1.2. Configuration

The gNB configuration file we used is based on the script in [6]. The corresponding changes are made accordingly as follows:

```
amf:
    addr: 5G core bind address
    bind_addr: gNB bind address
```

The addr is the Internet Protocol (IP) address that the 5G core binds to. In Open5GS, this address is configured at Next Generation Application Protocol (NGAP) address in AMF and GPRS Tunneling Protocol User Plane (GTP-U) address in UPF. The bind_addr is a local IP address that the gNB binds to. Users should make sure the interfaces with these IP addresses are accessible between each other, for example, they can be in the same subnet.

The RF front-end of the radio can be configured as follows:

When using Ettus B210, it shall be configured as

```
ru_sdr:
    device_driver: uhd
    device_args: type=b200
    clock: external
    sync: external
    srate: 11.52
    tx_gain: 75
    rx_gain: 35
```

When using Ettus X310, it shall be configured as

```
ru_sdr:
    device_driver: uhd
    device_args:type=x300,addr=x310_ip_addr,dboard_clock_rate=11.52e6,
time_source=external,clock_source=external
    clock: external
    sync: external
    srate: 11.52
    tx_gain: 30
    rx_gain: 5
```

where *x310_ip_addr* is the IP address configured for X310.

When ZMQ is used, the RF front-end shall be configured as

```
ru_sdr:
    device_driver: zmq
    device_args: tx_port=tcp://tx_ip:tx_port,rx_port=tcp://rx_ip:
rx_port,base_srate=11.52e6
    srate: 11.52
    tx_gain: 75
    rx_gain: 35
```

where *tx_ip* and *tx_port* are the IP of the interface and its port that gNB uses to transmit the digital samples, and *rx_ip* and *rx_port* are where gNB expects the digital samples are from.

The 5G cell configuration is as follows

```
cell_cfg:
    dl_arfcn: 368500
    band: 3
    channel_bandwidth_MHz: 10
    common_scs: 15
    plmn: "00101"
    tac: 7
    pdcch:
        dedicated:
            ss2_type: common
            dci_format_0_1_and_1_1: false
        common:
            ss0_index: 0
            coreset0_index: 6
    prach:
        prach_config_index: 1
```

It is important that `tac` and `plmn` should align with those in AMF of Open5GS.

The configuration for RIC connection is as follows:

```
e2:
    enable_du_e2: true
    addr: RIC bind address
    bind_addr: gNB bind address for RIC connection
    e2sm_kpm_enabled: true
```

where `addr` is the IP address where RIC binds to, and `bind_addr` is the local IP address where the gNB binds to for RIC connection. Users should also make sure the interfaces with these IP addresses are accessible between each other.

E2AP packet captures can be enabled using:

```
pcap:
    e2ap_enable: true
    e2ap_filename: /tmp/gnb_e2ap.pcap
```

3.2. 5G UE: srsRAN 4G

3.2.1. Installation

The srsUE is a part of srsRAN 4G. It is a 4G Long Term Evolution (LTE) UE with prototype 5G New Radio (NR) features [7]. It can be installed using packages or from source. As we experienced issues when installing with packages, in this section, we introduce how it is installed from source in our deployments.

The dependencies are installed with the following command

```
$ sudo apt-get install build-essential cmake libfftw3-dev libmbedtls-dev
  libboost-program-options-dev libconfig++-dev libsctp-dev
```

Next,

```
$ git clone https://github.com/srsRAN/srsRAN_4G.git

$ cd srsRAN_4G

$ mkdir build

$ cd build

$ cmake ../
```

During `cmake`, it is important that ZMQ is found:

```
-- FINDING ZEROMQ.
-- Checking for module 'ZeroMQ'
--   No package 'ZeroMQ' found
-- Found libZEROMQ: /usr/local/include, /usr/local/lib/libzmq.so
```

If not, follow the instructions in Section 2.2 to enable SCTP.

Finally,

```
$ make

$ make test

$ sudo make install

$ srsran_install_configs.sh user
```

Make sure it passes all the tests.

3.2.2. Configuration

The srsUE configuration file we used is based on the script in [8]. This script can be used with Ettus B210 when `tx_gain` and `rx_gain` are correctly configured. When ZMQ is used, the following changes need to be made:

```
device_name = zmq
device_args = tx_port=tcp://tx_ip:tx_port,rx_port=tcp://rx_ip:rx_port,
base_srate=11.52e6
```

where *tx_ip* and *tx_port* are the IP address of the interface and its port that srsUE uses to transmit the digital samples, and *rx_ip* and *rx_port* are where srsUE expects the digital samples are from.

3.3. E2 Simulator

The E2 simulator from the OSC's E2 simulator repository can be used to test the E2 interface on the installed OSC near-RT RIC. Users can easily implement E2 simulator using a docker container and build Dockerfile to generate the image.

3.3.1. Build and Installation

First check out `e2-interface` git source file from the repository, then build the docker image.

```
$ git clone https://gerrit.o-ran-sc.org/r/sim/e2-interface
$ apt-get install cmake g++ libsctp-dev
$ cd e2-interface/e2sim
$ vi Dockerfile_kpm ### modify last line to "CMD sleep 100000000"
$ mkdir build
$ cd build
$ cmake .. && make package && cmake .. -DDEV_PKG=1 && make package
$ cp *.deb ../e2sm_examples/kpm_e2sm/
$ cd ../
$ docker build -t oransim:0.0.999 . -f Dockerfile_kpm
```

The E2 simulator can be run in a docker container and execute with the commands below.

```
$ docker run -d -it --name oransim oransim:0.0.999
```

In the middle of building procedures, there was sleeping command into the Dockerfile, so E2 simulator should be run manually. The execution command is "`kpm_sim <IP address of SCTP> 36422`", however, it needs to know the IP address of the E2 termination point inside the near-RT RIC. This IP address can be found inside the Kubernetes service, `service-ricplt-e2term-sctp-alpha`.

```
$ Kubectl get svc -n ricplt | grep e2term-sctp

ricplt service-ricplt-e2term-sctp-alpha NodePort 10.96.147.226 sctp-
alpha:36422→32222 SCTP
```

3.3.2. Running E2 Simulator

The next table shows the execution result from running `kpm_sim`. It shows the connection was established successfully.

```
root@ /e2-interface/e2sim~$ docker exec -it oransim /bin/bash
root@44623223b91a:/playpen~$ kpm_sim 10.96.147.226 36422
[kpm_callbacks.cpp:63] Starting KPM simulator
[encode_kpm.cpp:49] short_name: ORAN-E2SM-KPM, func_desc: KPM Monitor,
e2sm_odi: OID123
[encode_kpm.cpp:72] Initialize event trigger style list structure
[encode_kpm.cpp:91] Initialize report style structure
%%about to register e2sm func desc for 0
%%about to register callback for subscription for func_id 0
Start E2 Agent (E2 Simulator
… </successfulOutcome>
</E2AP-PDU>
[E2AP] Unpacked E2AP-PDU: index = 2, procedureCode = 1
[e2ap_message_handler.cpp:80]
[E2AP] Received SETUP-RESPONSE-SUCCESS
```

3.3.3. E2 Connection Check from RIC Cluster

The connection status between the E2 simulator and the RIC cluster can be viewed by a simple curl command to one of the running pods in the RIC cluster, whose name is `service-ricplt-e2mgr-http` service point.

```
$ Kubectl get service -n ricplt | grep service-ricplt-e2mgr-http

ricplt service-ricplt-e2mgr-http ClusterIP 10.96.90.98 http:3800→0
```

Then, use `curl` command to verify the connection.

```
root@~$ curl -X GET http://10.96.90.98:3800/v1/nodeb/states
2>/dev/null|jq
[
   {
         "inventoryName": "gnb_734_373_16b8cef1",
         "globalNbId": {
           "plmnId": "373437",
           "nbId": "10110101110001100111011110001"
         },
         "connectionStatus": "CONNECTED"
   }
]
```

4. 5G Core

We utilize Open5GS to deliver 5G Core network functionalities. Open5GS can be installed with a package manager, or it can be built from sources [9]. In addition, srsRAN Project also provides a dockerized version to simplify the deployment [10]. In this section, we use these three installation methods to introduce the installation and configuration procedures we have tried in our 5G testbed.

When Open5GS is installed using a package manager, version v2.6.4 and v2.6.6 are compatible with srsRAN Project v23.5 and v23.10, whereas Open5GS v2.7.0 supports srsRAN Project v23.10.1 with additional required steps. When Open5GS v2.7.0 is installed from source, it has been tested to be compatible with srsRAN Project v23.10.1. The dockerized Open5GS provided by srsRAN Project has been tested to be working with all the three versions of srsRAN Project mentioned above.

4.1. Installation with Package Manager

Following the guidelines in [11], we tried installing Open5GS v2.6.4, v2.6.6, and v2.7.0 with package manager. V2.6.4 and v2.6.6 follow the identical steps as follows, whereas some updates in v2.7.0 require additional steps in order to function properly with the other O-RAN components and configurations, such as WebUI access and Network Repository Function (NRF) configuration. These steps will be addressed in this section.

4.1.1. MongoDB

MongoDB is used as database for NRF/Policy Control Function (PCF)/Unified Data Repository (UDR) and Policy and Charging Rules Function (PCRF)/Home Subscriber Server (HSS) [9]. It needs to be installed before Open5GS.

```
$ sudo apt update

$ sudo apt install gnupg

$ curl -fsSL https://pgp.mongodb.com/server-6.0.asc | sudo gpg -o
  /usr/share/keyrings/mongodb-server-6.0.gpg --dearmor

$ echo "deb [ arch=amd64,arm64 signed-by=/usr/share/keyrings/mongodb
  -server-6.0.gpg] https://repo.mongodb.org/apt/ubuntu jammy/mongodb-
  org/6.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-
  6.0.list

$ sudo apt update

$ sudo apt install -y mongodb-org

$ sudo systemctl enable --now mongod
```

4.1.2. Open5GS

```
$ sudo add-apt-repository ppa:open5gs/latest

$ sudo apt update

$ sudo apt install open5gs
```

4.1.3. WebUI

WebUI can be used to add and modify subscriber information to Open5GS using a web browser. Because of its interactive nature, it is more user friendly than the other subscriber editing methods, such as using a command line tool. To use WebUI, Ubuntu Desktop must be available on the server/workstation.

First, the dependency of WebUI, `Nodejs`, can be installed by:

```
$ sudo apt update

$ sudo apt install -y ca-certificates curl gnupg

$ sudo mkdir -p /etc/apt/keyrings

$ curl -fsSL https://deb.nodesource.com/gpgkey/nodesource-repo.gpg.key |
  sudo gpg --dearmor -o /etc/apt/keyrings/nodesource.gpg

$ NODE_MAJOR=20

$ echo "deb [arch=amd64,arm64 signed-by=/etc/apt/keyrings/
  nodesource.gpg] https://deb.nodesource.com/node_$NODE_MAJOR.x nodistro
  main" | sudo tee /etc/apt/sources.list.d/nodesource.list

$ sudo apt update

$ sudo apt install nodejs -y
```

Then, install WebUI by:

```
$ curl -fsSL https://open5gs.org/open5gs/assets/webui/install | sudo -E
  bash -
```

4.2. Configuration

4.2.1. AMF and UPF Configurations in 5G SA Mode

Once Open5GS is successfully installed, `systemctl` shows the running modules, and the status of each module should be active and running, as shown in Fig. 2.

Their configuration files can be found in `/etc/open5gs/` directory with extension `.yaml`.

```
open5gs-amfd.service                     loaded    active running   Open5GS AMF Daemon
open5gs-ausfd.service                    loaded    active running   Open5GS AUSF Daemon
open5gs-bsfd.service                     loaded    active running   Open5GS BSF Daemon
open5gs-hssd.service                     loaded    active running   Open5GS HSS Daemon
open5gs-mmed.service                     loaded    active running   Open5GS MME Daemon
open5gs-nrfd.service                     loaded    active running   Open5GS NRF Daemon
open5gs-nssfd.service                    loaded    active running   Open5GS NSSF Daemon
open5gs-pcfd.service                     loaded    active running   Open5GS PCF Daemon
open5gs-pcrfd.service                    loaded    active running   Open5GS PCRF Daemon
open5gs-scpd.service                     loaded    active running   Open5GS NRF Daemon
open5gs-sgwcd.service                    loaded    active running   Open5GS SGW-C Daemon
open5gs-sgwud.service                    loaded    active running   Open5GS SGW-U Daemon
open5gs-smfd.service                     loaded    active running   Open5GS SMF Daemon
open5gs-udmd.service                     loaded    active running   Open5GS UDM Daemon
open5gs-udrd.service                     loaded    active running   Open5GS UDR Daemon
open5gs-upfd.service                     loaded    active running   Open5GS UPF Daemon
open5gs-webui.service                    loaded    active running   Open5GS WebUI
```

Fig. 2. Open5GS running modules.

For 5G SA mode, to setup the 5G core, AMF and UPF bind addresses should align with the *5G core bind address* as specified in Section 3.1.2. For example, when Open5GS is deployed in the same server as the gNB, 127.0.0.5 can be used as the bind address for 5G core. Hence, the following configurations need to be made:

In `/etc/open5gs/amf.yaml`,

```
ngap:
    - addr: 127.0.0.5
guami:
    - plmn_id:
        mcc: 001
        mnc: 01
tai:
    - plmn_id:
        mcc: 001
        mnc: 01
      tac: 7
plmn_support:
    - plmn_id:
        mcc: 001
        mnc: 01
```

In `/etc/open5gs/upf.yaml`,

```
gtpu:
    - addr: 127.0.0.5
```

Next, restart AMF and UPF modules

```
$ sudo systemctl restart open5gs-amfd

$ sudo systemctl restart open5gs-upfd
```

Users should run `systemctl` and check if all the modules are in active and running status. If not, some configurations may need to be revisited.

4.2.2. NRF Configurations in Open5GS v2.7.0

It is important to note that the NRF configuration in Open5GS v2.7.0 does not update the Public Land Mobile Network (PLMN) information following the changes in AMF. When v2.7.0 is installed with package manager, PLMN needs to be manually updated in `/etc/open5gs/nrf.yaml`:

```
nrf:
    serving:
        - plmn_id:
            mcc: 001
            mnc: 01
```

4.2.3. Register Subscriber

We use the subscriber information as provided by srsRAN Project [5]:

```
opc = 63BFA50EE6523365FF14C1F45F88737D
k = 00112233445566778899aabbccddeeff
imsi = 001010123456780
```

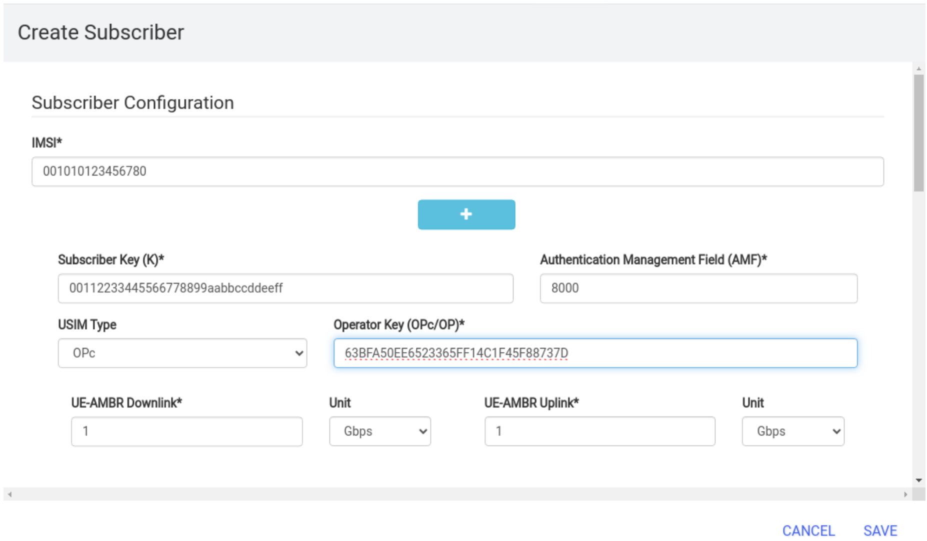
and the APN is:

```
apn = srsapn
apn_protocol = ipv4
```

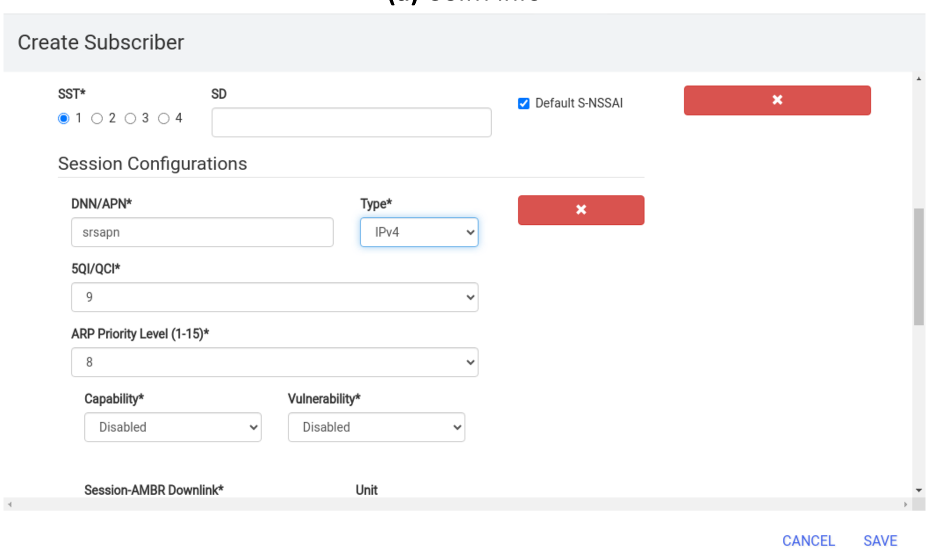
To register the subscriber, open a web browser and connect to `http://localhost:3000` (Note: In Open5GS v2.7.0, the port number is 9999.). Login with Username *admin* and Password *1423*. Once logged in, click on the **+** button at the bottom right corner of the browser to open the `Create Subscriber` window. As shown in Fig. 3, add the information above to the corresponding fields. All the other fields can be left unchanged.

4.2.4. Enable UE Access to Internet

To enable the UE access to the Internet, IP forwarding should be enabled and Network Address Translation (NAT) rules should be added:



(a) USIM info



(b) APN info

Fig. 3. Create Subscriber window in WebUI.

```
$ sudo sysctl -w net.ipv4.ip_forward=1

$ sudo sysctl -w net.ipv6.conf.all.forwarding=1

$ sudo iptables -t nat -A POSTROUTING -s 10.45.0.0/16 ! -o ogstun -j
  MASQUERADE

$ sudo ip6tables -t nat -A POSTROUTING -s 2001:db8:cafe::/48 ! -o ogstun
  -j MASQUERADE
```

It is important to note that these rules will be reset upon reboot unless they are saved using a command like `iptables-save`.

In addition, firewall should also be disabled:

```
$ sudo ufw disable

Firewall stopped and disabled on system startup

$ sudo ufw status

Status: inactive
```

4.3. Installation from Sources

In this section, we build Open5GS from sources following most of the instructions in [9]. The version we tried was v2.7.0 with updates until 01/03/2024, as the source code has been actively updated.

4.3.1. MongoDB

Please follow the instructions in Section 4.1.1 to install MongoDB.

4.3.2. TUN Device

When building from sources, the interface for TUN device needs to be added manually, and after each reboot, the IP addresses need to be configured:

```
$ sudo ip tuntap add name ogstun mode tun

$ sudo ip addr add 10.45.0.1/16 dev ogstun

$ sudo ip addr add 2001:db8:cafe::1/48 dev ogstun

$ sudo ip link set ogstun up
```

4.3.3. Open5GS

Install the dependencies:

```
$ sudo apt install python3-pip python3-setuptools python3-wheel ninja-
  build build-essential flex bison git cmake libsctp-dev libgnutls28-
  dev libgcrypt20-dev libssl-dev libidn11-dev libmongoc-dev libbson-
  dev libyaml-dev libnghttp2-dev libmicrohttpd-dev libcurl4-gnutls-dev
  libnghttp2-dev libtins-dev libtalloc-dev meson
```

Please note the version of `libgcrypt20-dev` is used.

Git clone Open5GS source code and install:

```
$ git clone https://github.com/open5gs/open5gs

$ cd open5gs

$ meson build --prefix=`pwd`/install

$ ninja -C build
```

Check if 5G core is compiled successfully:

```
$ ./build/tests/registration/registration
```

Run all tests:

```
$ cd build

$ meson test -v
```

If without issue, install Open5GS:

```
$ ninja install

$ cd ..
```

[9] introduced executing the NFs either individually or all together. When executed individually, the NF configuration files in `install/bin/` are used. `amf.yaml` and `upf.yaml` shall be modified as discussed in Section 4.2.1, and each NF shall be executed in a separate terminal by following ”Running Open5GS” section in [9].

When executing the NFs together, the configuration file in `build/configs/sample.yaml` is used. The `amf` and `upf` sections of this file shall be modified for 5G SA mode following the corresponding configurations in Section 4.2.1. After that, run the following command:

```
$ ./build/tests/app/5gc
```

4.3.4. WebUI

To install WebUI, nodejs can be installed following the instructions in Section 4.1.3. After that, in `open5gs` directory:

```
$ cd webui

$ npm ci
```

And WebUI can be executed by

```
$ npm run dev
```

4.3.5. Register Subscriber

Please follow the instructions in Section 4.2.3 to register subscriber.

4.3.6. Enable UE Access to Internet

Please follow the instructions in Section 4.2.4 to give UE access to the Internet.

4.4. Installation with Docker

Both Open5GS and srsRAN Project provide the options to build Open5GS into a docker container. As srsRAN Project provides a stable version to better fit the srsRAN gNB execution, we deployed the srsRAN option and it is thus introduced here.

The dockerized Open5GS provided by srsRAN Project uses the Open5GS v2.6.1. It has been configured with the default AMF IP address of 10.53.1.2, which is also the IP address of the docker container. It also uses the PLMN and UE information as introduced in Section 4.2.

To install Open5GS into the docker container, first install `docker-compose`:

```
$ sudo apt install docker-compose
```

You may need to add your user to docker group:

```
$ sudo gpasswd -a $USER docker

$ newgrp docker
```

After that, build the dockerized Open5GS by

```
$ cd docker

$ docker-compose up --build 5gc
```

4.4.1. Access Dockerized Open5GS from gNB

As dockerized Open5GS uses IP address of 10.53.1.2, the configuration file for gNB, as discussed in Section 3.1.2, needs to be updated by replacing the 5G core bind address in `amf` with 10.53.1.2. In addition, IP rules may be added on the gNB server.

4.4.2. Enable UE access to Internet

The default deployment of the docker container does not grant the UE access to the Internet. To give UE access to the Internet, log in to the docker container while it is running:

```
$ docker exec -t open5gs_5gc /bin/bash
```

Follow the steps in Section 4.2.4 to enable IP forwarding and edit iptables.

5. Near-RT RIC

5.1. FlexRIC Setup

As srsRAN Project is actively updated, it may not be compatible with all versions/branches of FlexRIC. A FlexRIC installation guideline is provided by srsRAN Project in [10], where the `e2ap-v2` branch of [12] is used, and a patch file is created to ensure E2 node is correctly connected to gNB. This installation was tested to be working with srsRAN Project v23.5 and v23.10, but failed when working with srsRAN Project v23.10.1. The `master` branch of the FlexRIC provides an installation that is compatible with srsRAN Project v23.10.1. In this section, we first introduce how to install the FlexRIC that is compatible with the current version of srsRAN Project (v23.10.1). In addition, the FlexRIC `e2ap-v2` branch installation is also provided.

5.1.1. FlexRIC Installation for srsRAN Project v23.10.1

[12] provides a guideline for FlexRIC installation. To make it compatible with srsRAN Project v23.10.1, the versions of E2AP and KPM need to be specified at `cmake` [13].

Before installing FlexRIC, the dependencies for `SWIG` and `FlexRIC` may need to be installed:

```
$ sudo apt install autotools-dev automake libpcre2-dev bison byacc

$ sudo apt install libsctp-dev python3.8 cmake-curses-gui libpcre2-dev
  python3-dev gcc-10 g++-10
```

If a newer version of Python is already installed, `python3.8` may not be needed.

Next, version v4.1 or greater of `SWIG` needs to be installed:

```
$ git clone https://github.com/swig/swig.git

$ cd swig

$ git checkout release-4.1

$ ./autogen.sh

$ ./configure --prefix=/usr/

$ make -j`nproc`

$ sudo make install
```

Clone FlexRIC:

```
$ git clone https://gitlab.eurecom.fr/mosaic5g/flexric.git

$ git checkout master

$ cd flexric

$ mkdir build
```

The IP address that FlexRIC binds to can either be changed in `flexric/flexric.conf` before installation, or it can be changed after installation in `/usr/local/etc/flexric/flexric.conf`. The default IP address is 127.0.0.1. It can be used when FlexRIC is deployed in the same server as the gNB. However, if FlexRIC is to be disaggregated from gNB, the IP address of the interface assigned for FlexRIC should be typed in here. For example, if the IP of the interface to be used by FlexRIC is 192.168.10.2, in `flexric/flexric.conf`, it should be assigned to `NEAR_RIC_IP` as follows:

```
[NEAR-RIC]
NEAR_RIC_IP = 192.168.10.2 # An IP example
```

To build FlexRIC, we will use `gcc-10` compiler, E2AP version 3 and KPM SM version 3 [13]:

```
$ cd build

$ CC=gcc-10 CXX=g++-10 cmake .. -DE2AP_VERSION=E2AP_V3 -
  DKPM_VERSION=KPM_V3_00

$ sudo make install

$ cd ..
```

If the IP address for nearRT-RIC was not changed before installation, it can be edited now in `/usr/local/etc/flexric/flexric.conf`:

```
[NEAR-RIC]
NEAR_RIC_IP = 192.168.10.2 # An IP example
```

5.1.2. FlexRIC Installation for srsRAN Project v23.5 and v23.10

Before downloading and installing FlexRIC, its dependencies need to be installed by:

```
$ sudo apt-get update

$ sudo apt-get install swig libsctp-dev cmake-curses-gui libpcre2-dev
  python3 python3-dev
```

and the patch file can be downloaded from https://docs.srsran.com/projects/project/en/latest/_downloads/d0bb1100d471824e1f5536ddd0765d0d/flexric.patch.

Next, download FlexRIC code and apply the patch file:

```
$ git clone https://gitlab.eurecom.fr/mosaic5g/flexric.git
$ cd flexric
$ git checkout e2ap-v2
$ git apply -v ./flexric.patch
```

As discussed in Section 5.1.1, the IP address for FlexRIC can either be changed before or after the installation. Refer to that section for additional information.

Next, as FlexRIC can only be built with `gcc-10`, `CC=gcc-10 CXX=g++-10` needs to be specified at `cmake`. FlexRIC can be built and installed by:

```
$ mkdir build
$ cd build
$ CC=gcc-10 CXX=g++-10 cmake ../
$ make
$ sudo make install
```

5.2. OSC Near-RT RIC Setup

5.2.1. Software Source and Dependency

This section introduces the 5G open-source implementation of RAN solution from OSC. First it needs to have the latest version of git software. Then, it will be able to download OSC's RIC implementation.

- Install the latest version of git

  ```
  $ sudo apt install git
  ```

- RIC source download with J-release

  ```
  $ git clone "https://gerrit.o-ran-sc.org/r/ric-plt/ric-dep" -b
    j-release
  ```

5.2.2. Kubernetes, Docker, Helm Chart Installation

First go to the directory, `ric-dep/bin`, to prepare Kubernetes and Docker software environments, run the installation file, `install_k8s_and_helm.sh` to install OSC's basic installation. This file installs Kubernetes control, administration, proxies, and more. Also the helm chart will be downloaded and installed as well as the docker container runtime.

```
~/ric-dep/bin~$ ./install_k8s_and_helm.sh
+ KUBEV=1.28.11
+ HELMV=3.14.4
+ DOCKERV=20.10.21
+ echo running ./install_k8s_and_helm.sh
running ./install_k8s_and_helm.sh
…
```

After running the installation file, check the status of the Kubernetes installation which should be similar to the next table. Ensure that the STATUS column displays "Running" status without errors for each component.

```
$ kubectl get po -A
NAMESPACE      NAME                              READY  STATUS    RESTARTS   AGE
kube-flannel   kube-flannel-ds-6bhtg             1/1    Running   0          59s
kube-system    coredns-76f75df574-ggzrk          1/1    Running   0          59s
kube-system    coredns-76f75df574-kvhb7          1/1    Running   0          59s
kube-system    etcd-ric                          1/1    Running   0          73s
kube-system    kube-apiserver-ric                1/1    Running   0          73s
kube-system    kube-controller-manager-ric       1/1    Running   0          76s
kube-system    kube-proxy-f277s                  1/1    Running   0          59s
kube-system    kube-scheduler-ric                1/1    Running   0          74s
```

5.2.3. Modify Service Platform Configuration File

Next step, it needs to modify a configuration file in `ric-dep/RECIPE_EXAMPLE/example_recipe_latest_stable.yaml`, or `example_recipe_oran_j_release`. Change the following section to the host node's real IP address.

```
extsvcplt:
riccp:"10.0.0.1" --> main internet interface ip
auxip:"10.0.0.1" --> main internet interface ip
```

5.2.4. Install Common Template to Helm

In the ric-dep/bin directory run the common template for helm chart and chartmuseum installation.

```
$ cd ric-dep/bin

$ ./install_common_templates_to_helm.sh
```

This command installs the following components:

- chartmuseum into helm with "helm servecm"
- ric-common template into helm by chartmuseum

The result should be similar to the following:

```
~/ric-dep/bin~$ ./install_common_templates_to_helm.sh
Installing servecm (Chart Manager) and common templates to helm3
Installed plugin: servecm
/root/.cache/helm/repository
   %  Total    %  Received  %  Xferd  Average  Speed   Time     Time     Time     Current
                                      Dload    Upload  Total    Spent    Left     Speed
 100  15.0M  100  15.0M     0      0  28.8M    0       --:--:-- --:--:-- --:--:-- 28.8M
linux-386/
linux-386/chartmuseum
linux-386/LICENSE
linux-386/README.md
servecm not yet running. sleeping for 2 seconds
nohup: appending output to 'nohup.out'
servcm up and running
/root/.cache/helm/repository
Successfully packaged chart and saved it to: /tmp/ric-common-3.3.2.tgz
Error: no repositories configured
"local" has been added to your repositories
checking that ric-common templates were added
NAME              CHART VERSION   APP VERSION   DESCRIPTION
local/ric-common  3.3.2                         Common templates for inclusion in other charts
```

5.2.5. Installing Near RT-RIC

In the `ric-dep/bin` directory, the next step is to install RIC components with helm chart. OSC's install script provides a shell script for installation of each version of helm chart.

```
~/ric-dep/bin~$ ./install -f ../RECIPE_EXAMPLE/example_recipe_latest_stable.yaml
namespace/ricplt created
namespace/ricinfra created
namespace/ricxapp created
Deploying RIC infra components [infrastructure dbaas appmgr rtmgr e2mgr e2term a1mediator subgmr
vespamgr o1mediator alarmmanager ]
Note that the following optional components are NOT being deployed: influxdb jaegeradapter. To
deploy them add them with -c to the default component list of the install command
configmap/ricplt-recipe created
Add cluster roles
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "local" chart repository
Update Complete. ⎈Happy Helming!⎈
Saving 7 charts
Downloading ric-common from repo http://127.0.0.1:8879/charts
Deleting outdated charts
NAME: r4-infrastructure
LAST DEPLOYED: Fri Aug 9 14:55:20 2024
NAMESPACE: ricplt
STATUS: deployed
REVISION: 1
TEST SUITE: None
…
```

After installation, the result can be checked with the ”`kubectl get pods -A`” command to look into the overall status. (-A : all name spaces)

```
~/ric-dep/bin~$ kubectl get pods -A
NAMESPACE      NAME                                                      READY   STATUS
kube-flannel   kube-flannel-ds-2xhvq                                     1/1     Running
kube-system    coredns-5dd5756b68-n5pxv                                  1/1     Running
kube-system    coredns-5dd5756b68-zs4km                                  1/1     Running
kube-system    etcd-ric                                                  1/1     Running
kube-system    kube-apiserver-ric                                        1/1     Running
kube-system    kube-controller-manager-ric                               1/1     Running
kube-system    kube-proxy-qsx2t                                          1/1     Running
kube-system    kube-scheduler-ric                                        1/1     Running
ricinfra       deployment-tiller-ricxapp-676dfd8664-vzhgh                1/1     Running
ricplt         deployment-ricplt-a1mediator-64fd4bf64-dv8bx              1/1     Running
ricplt         deployment-ricplt-alarmmanager-7d47d8f4d4-xwdph           1/1     Running
ricplt         deployment-ricplt-appmgr-79848f94c-fbgtf                  1/1     Running
ricplt         deployment-ricplt-e2mgr-856f655b4-vnqbj                   1/1     Running
ricplt         deployment-ricplt-e2term-alpha-d5fd5d9c6-454zr            1/1     Running
ricplt         deployment-ricplt-o1mediator-76c4646878-qff7k             1/1     Running
ricplt         deployment-ricplt-rtmgr-6556c5bc7b-kdh5z                  1/1     Running
ricplt         deployment-ricplt-submgr-66485ccc6c-p96d4                 1/1     Running
ricplt         deployment-ricplt-vespamgr-786666549b-6w5f8               1/1     Running
ricplt         r4-infrastructure-kong-5986fc7965-zrzss                   2/2     Running
ricplt         r4-infrastructure-prometheus-alertmanager-64f9876d6d-vf8v6   2/2  Running
ricplt         r4-infrastructure-prometheus-server-bcc8cc897-5t5tz       1/1     Running
ricplt         statefulset-ricplt-dbaas-server-0                         1/1     Running
```

5.2.6. RIC Application, xApps

5.2.6.1. Onboarding of xApp Using `dms_cli` tool

`dms_cli` offers a rich set of command line utilities to onboard one of example xApps, `hw-go`, to chartmuseum.

5.2.6.2. Chartmuseum

Using docker, chartmuseum can be easily run with the following command.

```
$ docker run --rm -u 0 -it -d -p 8090:8080 -e DEBUG=1 -e STORAGE=local
  -e STORAGE_LOCAL_ROOTDIR=/charts -v $(pwd)/charts:/charts
  chartmuseum/chartmuseum:latest
```

It will start downloading chartmuseum docker image from github and run the docker container afterwards.

The status of running docker container can be checked with the following "`docker ps`" command.

```
root@ric: /ric-dep/bin~$ docker ps
CONTAINER ID   IMAGE                              COMMAND            CREATED
STATUS         PORTS                                          NAMES
9499fdeea733   chartmuseum/chartmuseum:latest     "/chartmuseum"     4 minutes ago
Up 4 minutes   0.0.0.0:8090->8080/tcp, :::8090->8080/tcp     stupefied_lamport
```

Set up the environment variables for Command Line Interface (CLI) connection using the same port as used above.

```
#Set CHART_REPO_URL env variable
export CHART_REPO_URL=http://0.0.0.0:8090
```

5.2.6.3. Onboarder (dms_cli) Installation

First, check out onboader from OSC's git repository.

```
#Git clone appmgr

 $ git clone "https://gerrit.o-ran-sc.org/r/ric-plt/appmgr"
```

Next, configure and install.

```
#Change dir to xapp_onboarder

 $ cd appmgr/xapp_orchestrater/dev/xapp_onboarder


#If pip3 is not installed, install using the following command

 $ apt install python3-pip


#In case dms_cli binary is already installed, it can be uninstalled using
following command

 $ pip3 uninstall xapp_onboarder
```

```
#Install xapp_onboarder using following command

 $ pip3 install ./


#(followings commands are optional)

 $ chmod 755 /usr/local/bin/dms_cli

 $ chmod -R 755 /usr/local/lib/python3.8 (←it might be different version
   number depending on user's python version, maybe python3.6 etc)
```

Check the installation and running status with health check.

```
root: /ric-dep/bin/appmgr/xapp_orchestrater/dev/xapp_onboarder~$ dms_cli
health
True
```

If the result is '`true`', it means dms_cli process is ready to run. Otherwise, if you get failed or connection error such as "`Caused by: ConnectionError`", this is due to `CHART_REPO_URL` not set correctly or set to `Null` when the user left the shell and returned back again without this environment variable set. Setting this variable into bash resource file, bashrc or bash profile, would help with this issue.

5.2.6.4. `hw-go` xApp Build and Preparation

In order to test the xApp, first check out hw-go test file from the git repository.

```
$ git clone https://gerrit.o-ran-sc.org/r/ric-app/hw-go
```

Go to `hw-go` inside and build a docker image with the provided Dockerfile. In order to load a docker image from the local repository where chartmuseum is running, the docker images should be stored locally, then Kubernetes will retrieve this image from the repo of chartmuseum. The images will be used by Kubernetes to access and load into the clusters.

```
$ cd hw-go

$ docker build -t example.com:80/hw-go:1.2 .

$ export CHART_REPO_URL=http://0.0.0.0:8090
```

Edit `config-file.json` for dms_cli (xApp onboader) to get information in order for `dms_cli` to generate helm chart. Because we made the docker image in the name of "`example.com:80/hw-go:1.2`", we need the `config-json` file to match accordingly.

```
$ vi config/config-file.json
```

1. modify `tag = 1.2`, under `containers.image`
2. modify registry with `example.com:80` under `containers.image`
3. modify name with `hw-go`, under `containers.image`

```
$ docker save -o hw-go.tar example.com:80/hw-go:1.2

$ ctr -n=k8s.io image import hw-go.tar
```

5.2.6.5. Onboarding `hw-go` xApp and Install

In `hw-go` directory, two config files are needed to onboard the xApp. One is `config-file.json` and the other is `schema.json` file. The following table shows the xApp onboarding commands with dms_cli application and the output messages. The onboarding process is making helm chart `yaml` files, which are used for installing into the clusters.

```
$ dms_cli onboard ./config/config-file.json ./config/schema.json

httpGet:
  path: ' index .Values "readinessProbe" "httpGet" "path" | toJson '
  port: ' index .Values "readinessProbe" "httpGet" "port" | toJson '
initialDelaySeconds: ' index .Values "readinessProbe"
"initialDelaySeconds" | toJson '
periodSeconds: ' index .Values "readinessProbe" "periodSeconds" | toJson
'


$ dms_cli onboard ./config/config-file.json ./config/schema.json

httpGet:
  path: ' index .Values "livenessProbe" "httpGet" "path" | toJson '
  port: ' index .Values "livenessProbe" "httpGet" "port" | toJson '
initialDelaySeconds: ' index .Values "livenessProbe"
"initialDelaySeconds" | toJson '
periodSeconds: ' index .Values "livenessProbe" "periodSeconds" | toJson '

{
"status": "Created"
}
```

To deploy the hw-go image into the RIC cluster as a running pod, the install command will be used.

```
$ dms_cli install hw-go 1.0.0 ricxapp

status: OK
```

In case of uninstalling the previous installed xApp, hw-go pod in the ricxapp namesapce,

```
$ dms_cli uninstall hw-go ricxapp

status: OK
```

5.2.6.6. Checking xApp's Deployment Status

In the near RT RIC cluster, you will be able to check whether hw-go xApp is properly on-boarded or not by sending a curl query to “service-ricplt-appmgr” service.

First, in order to verify “service-ricplt-appmgr” service is running without problem, we need to check the RIC Kubernetes cluster.

```
$ kubectl get services -n ricplt | grep service-ricplt-appmgr

service-ricplt-appmgr-http   ClusterIP   10.109.23.238   <none>   8080/TCP
service-ricplt-appmgr-rmr    ClusterIP   10.111.218.86   <none>   4561/TCP,4560/TCP
```

There are two "`service-ricplt-appmgr`" related services. We need to pick up the name of "`service-ricplt-appmgr-http`" to check the xApp status.

```
$ curl http://service-ricplt-appmgr-http.ricplt:8080/ric/v1/xapps | jq .
Or use ip address in case of service name is not found
$ curl http://10.110.206.238:8080/ric/v1/xapps | jq .
  %   Total     %   Received  %   Xferd   Average   Speed    Time      Time      Time      Current
                                          Dload     Upload   Total     Spent     Left      Speed
 100  333     100   333       0       0   16650     0        --:--:--  --:--:--  --:--:--  16650
  [
   {
    "instances": [
     {
       "ip": "service-ricxapp-hw-go-rmr.ricxapp",
       "name": "hw-go",
       "policies": [
        1
       ],
       "port": 4560
       "rxMessages": [
        "RIC_SUB_RESP",
        "A1_POLICY_REQ",
        "RIC_HEALTH_CHECK_REQ"
       ],
       "status": "deployed",
       "txMessages": [
        "RIC_SUB_REQ",
        "A1_POLICY_RESP",
        "A1_POLICY_QUERY",
        "RIC_HEALTH_CHECK_RESP"
       ]
     }
    ],
    "name": "hw-go",
    "status": "deployed",
    "version": "1.0.0"
   }
  ]
```

In the RIC Kubernetes, `hw-go` xApp can be observed by getting pods status as shown in the following table using `ricxapp` namespace. (`-n ricxapp`)

```
$ kubectl get po -n ricxapp
NAME                            READY   STATUS    RESTARTS   AGE
ricxapp-hw-go-7c8945ccb6-tldk8  1/1     Running   0          9d
```

5.2.7. Interoperation with E2 Simulator

When the xApp is connected to the RIC cluster, xApp's initial configuration message, xApp register, and subscription request message will be transferred to the E2 simulator through the E2 functions in the cluster pods. Those message transactions are shown in the E2 simulator's logs described in the table below.

```
[e2sim.cpp:237] [SCTP] Received new data of size 47
in e2ap_handle_sctp_data()
decoding...
full buffer

length of data 47
result 0
index is 1
showing xer of data
```


```
<E2AP-PDU>
   <initiatingMessage>
    <procedureCode>8</procedureCode>
    <criticality><ignore/></criticality>
    <value>
      <RICsubscriptionRequest>
        <protocolIEs>
          <RICsubscriptionRequest-IEs>
            <id>29</id>
            <criticality><reject/></criticality>
            <value>
              <RICrequestID>
                <ricRequestorID>123</ricRequestorID>
                <ricInstanceID>4</ricInstanceID>
              </RICrequestID>
            </value>
          </RICsubscriptionRequest-IEs>
          <RICsubscriptionRequest-IEs>
            <id>5</id>
            <criticality><reject/></criticality>
            <value>
              <RANfunctionID>1</RANfunctionID>
            </value>
          </RICsubscriptionRequest-IEs>
          <RICsubscriptionRequest-IEs>
            <id>30</id>
            <criticality><reject/></criticality>
            <value>
              <RICsubscriptionDetails>
                <ricEventTriggerDefinition>01 02 03 04</ricEventTriggerDefinition>
                <ricAction-ToBeSetup-List>
                  <ProtocolIE-SingleContainer>
                    <id>19</id>
                    <criticality><ignore/></criticality>
                    <value>
                      <RICaction-ToBeSetup-Item>
                        <ricActionID>1</ricActionID>
                        <ricActionType><report/></ricActionType>
                        <ricActionDefinition>01 02 03 04</ricActionDefinition>
                        <ricSubsequentAction>
                          <ricSubsequentActionType><continue/></ricSubsequentActionType>
                          <ricTimeToWait><w20ms/></ricTimeToWait>
```

```
                </ricSubsequentAction>
               </RICaction-ToBeSetup-Item>
              </value>
             </ProtocolIE-SingleContainer>
            </ricAction-ToBeSetup-List>
           </RICsubscriptionDetails>
          </value>
        </RICsubscriptionRequest-IEs>
       </protocolIEs>
      </RICsubscriptionRequest>
    </value>
   </initiatingMessage>
</E2AP-PDU>
initiating message
[E2AP] Unpacked E2AP-PDU: index = 1, procedureCode = 8
...
```

6. Testbed Deployments

The software stacks for O-RAN components mentioned in Section 1.2 can be installed in several styles, such as installing from source or package manager on a bare metal machine, or installing in a docker container or kubernetes pods. Each installation style may require customized software or IP configurations on some or all the software to ensure that connections can be established between O-RAN components.

As each of the 5G testbed components can be deployed separately, they can either be deployed on the same server/workstation, or some/all components can be deployed on individual machines. In addition, the connection between gNB and UE can be

1. digital connection, where digital samples of the base band signals can be exchanged without being converted into RF signals,
2. RF connection using radio devices.

We use ZMQ messaging library for digital connection, and use USRPs for radio connection.

In this section, we introduce several deployment scenarios that were implemented and tested successfully on our testbeds. Each scenario is a combination of different software stacks, installation styles, and connection types. It is worth mentioning that of all the deployment scenarios in this section, we use the following RAN configurations:

- Frequency band: n3;
- Duplexing method: Frequency Division Duplex (FDD);
- Uplink center frequency: 1747.5 MHz;
- Downlink center frequency: 1842.5 MHz;
- Bandwidth: 10 MHz.

6.1. Aggregated Deployments

In the aggregated scenarios we have tried, the 5G Core, near-RT RIC, and gNB are installed on the same server, whereas the UE can be on the same server or on a different server. Both digital and RF connection have been tested.

In the following scenarios that uses FlexRIC as near-RT RIC, when FlexRIC is installed from source and Open5GS is installed from source or package manager, we can use the following local IP addresses for each component:

- Open5GS: 127.0.0.5,
- FlexRIC: 127.0.0.1,
- srsRAN gNB: 127.0.0.1.

User can install Open5GS from package manager following Section 4.1, or from sources following Section 4.3. The configurations in these sections use the above IP addresses as examples.

FlexRIC can be installed following Section 5.1.1 or 5.1.2, and the `127.0.0.1` local IP was default at `flexric/flexric.conf` so no IP configuration is needed for aggregated scenarios.

6.1.1. Installations on a Single Bare Metal Server, ZMQ Connection

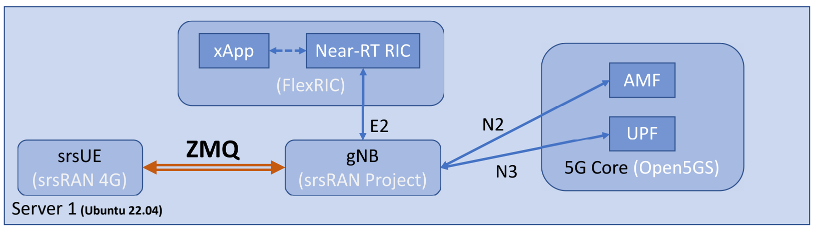


Fig. 4. Aggregated deployment: srsRAN gNB and UE, Open5GS, and FlexRIC. All components are installed on a bare metal server.

As illustrated in Fig. 4, the aggregated deployment on one server includes

- gNB: srsRAN Project,
- UE: srsRAN 4G,
- 5G core: Open5GS,
- near-RT RIC: FlexRIC,

and the connection between gNB and UE is via ZMQ. Because of its simplicity in configuration, a user can use this scenario as a test setup before more complicated function or network layout is introduced. The O-RAN components in this scenario can be installed following the Sections in Table 1.

Table 1. Installation Guide for Aggregated Scenario on One Server.

Software	Installation Guide	Server
srsRAN gNB	Section 3.1	Server 1
srsUE	Section 3.2	Server 1
FlexRIC	Section 5.1.1 or 5.1.2	Server 1
Open5GS	Section 4.1 or 4.3	Server 1

srsRAN gNB and UE can be installed following Sections 3.1 and 3.2, respectively. In srsRAN gNB's configuration file, Open5GS can be bound to gNB by

```
amf:
    addr: 127.0.0.5 # 5G core bind address
    bind_addr: 127.0.0.1 # gNB bind address
```

and FlexRIC can be bound to gNB by

```
e2:
    enable_du_e2: true
    addr: 127.0.0.1 # RIC bind address
    bind_addr: 127.0.0.1 # gNB bind address for RIC connection
    e2sm_kpm_enabled: true
```

To establish connection via ZMQ, we use `127.0.0.1:2000` as the interface between gNB Transmitter (TX) and UE Receiver (RX), and `127.0.0.1:2001` between UE TX and gNB RX. In gNB configuration file:

```
ru_sdr:
    device_driver: zmq
    device_args: tx_port=tcp://127.0.0.1:2000,rx_port=tcp://127.0.0.1:
2001,base_srate=11.52e6
    srate: 11.52
    tx_gain: 75
    rx_gain: 35
```

and in UE configuration file:

```
device_name = zmq
device_args = tx_port=tcp://127.0.0.1:2001,rx_port=tcp://127.0.0.1:2000,
base_srate=11.52e6
```

6.1.2. Installations with gNB and UE on different bare metal servers, ZMQ or USRP Connections

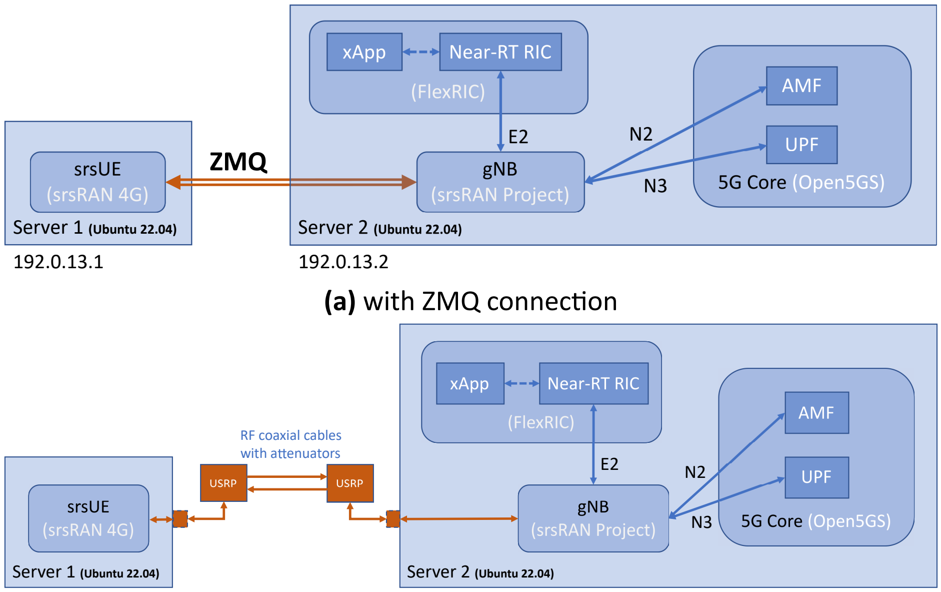
As illustrated in Fig. 5, the aggregated deployment has

- UE: srsRAN 4G

on Server 1, and

- gNB: srsRAN Project,
- 5G core: Open5GS,
- near-RT RIC: FlexRIC

on Server 2. The O-RAN components in this scenario can be installed following the Sections listed in Table 2.



(a) with ZMQ connection

(b) with USRP connection

Fig. 5. Aggregated deployment: srsRAN gNB and UE, Open5GS, and FlexRIC. All components are installed on bare metal servers.

Table 2. Installation Guide for Aggregated Scenario on Two Servers.

Software	Installation Guide	Server
srsRAN gNB	Section 3.1	Server 2
srsUE	Section 3.2	Server 1
FlexRIC	Section 5.1.1 or 5.1.2	Server 2
Open5GS	Section 4.1 or 4.3	Server 2

The Open5GS and FlexRIC can connect to the gNB as described in Section 6.1.1.

6.1.2.1. ZMQ Connection

In the scenario shown in Fig. 5a, ZMQ connection can be used to validate installations and connections between O-RAN components, before radio units are added, where RF configurations can lead to more complicated system calibration. ZMQ connection is established across a dedicated subnet of `192.0.13.x`. A simple `ping` function can be used to verify the connectivity between Server 1 and Server 2.

To establish ZMQ connection, the following changes need to be made on gNB and UE configuration files:

1. srsRAN gNB on Server 2 sends digital signal samples via `192.0.13.2:2000` and listens for samples from `192.0.13.1:2001`. The configuration file `ru_sdr` section can be configured as

```
ru_sdr:
    device_driver: zmq
    device_args: tx_port=tcp://192.0.13.2:2000,rx_port=tcp://
192.0.13.1:2001,base_srate=11.52e6
    srate: 11.52
    tx_gain: 75
    rx_gain: 35
```

2. srsRAN UE on Server 1 sends digital signal samples via `192.0.13.1:2001` and listens for samples from `192.0.13.2:2000`. The following lines in UE configuration file needs to be changed:

```
device_name = zmq
device_args = tx_port=tcp://192.0.13.1:2001,rx_port=tcp://
192.0.13.2:2000,base_srate=11.52e6
```

6.1.2.2. USRP Connection

Fig. 5b provides the first deployment we tested with Ettus B210 USRPs. It is important to note that 50 dB attenuation is added between the TX and RX ports.

The USRP connection can be established as follows:

1. srsRAN gNB configuration file on Server 2:

```
ru_sdr:
    device_driver: uhd
    device_args: type=b200
    clock: external
    sync: external
    srate: 11.52
    tx_gain: 75
    rx_gain: 35
```

2. srsRAN UE configuration file on Server 1:

```
[rf]
freq_offset = 0
tx_gain = 80
rx_gain = 35
srate = 11.52e6
nof_antennas = 1

device_name = uhd
device_args = clock=external
time_adv_nsamples = 300
```

6.1.3. Installations on a Single Bare Metal Server, Multiple UEs and One gNB, ZMQ Connection

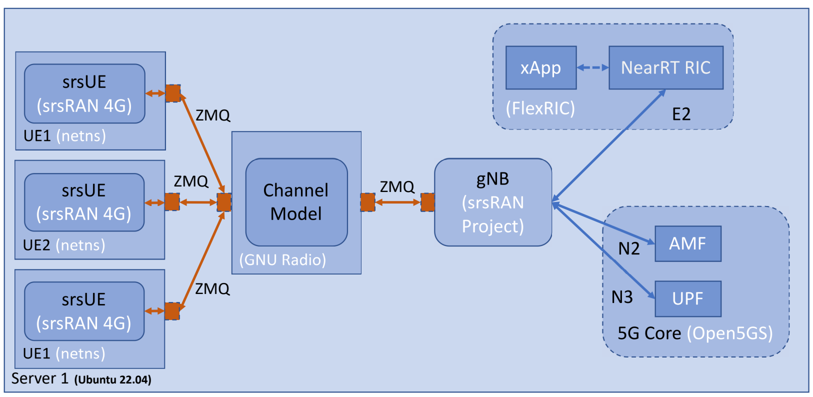


Fig. 6. Aggregated deployment with multiple UEs: srsRAN gNB and UEs, ZMQ connection, Open5GS, and FlexRIC. All components are installed on bare metal servers.

The O-RAN components can be installed following Table 1. Following the srsRAN instruction in [5], we connect more than one srsRAN UEs to the gNB with ZMQ (Fig. 6). The channel model is created using GNURadio, as shown in Fig. 7, where the TX samples from the UEs are added to form one data stream and is then sent to gNB RX, and the TX samples from gNB are duplicated and sent to each UE.

To have multiple UE instances running on the same machine, each UE shall be isolated within its own network namespace, with its unique Universal Subscriber Identity Module (USIM). Their USIM shall also be registered in Open5GS, following the steps in Section 4.1.3. Table 3 provides the USIMs and network namespaces of the UEs that are used in our deployment, and the changes we made to UE1 configuration file is as follows:

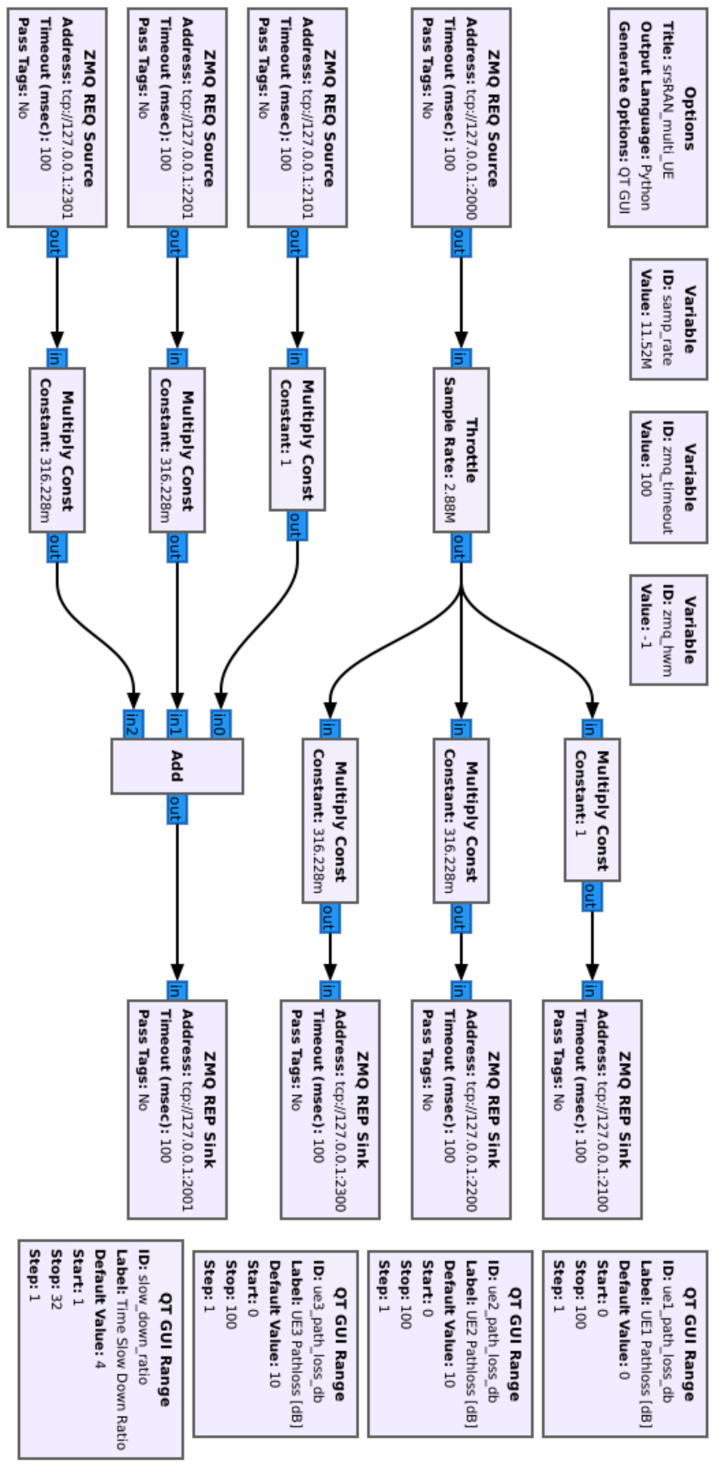


Fig. 7. GNURadio channel model for the aggregated multi-UE scenario with ZMQ.

Table 3. UE Configuration Information.

UE	UE1	UE2	UE3
OPc	63bfa50ee6523365ff14c1f45f88737d		
K	00112233445566778899aabbccdde**eff**	...**f00**	...**f01**
IMSI	0010101234567**80**	...**90**	...**91**
IMEI	35349006987331**9**	...**8**	...**2**
netns	ue1	ue2	ue3
TX Port	2101	2201	2301
RX Port	2100	2200	2300

```
[rf]
......
device_name = zmq
device_args = tx_port=tcp://127.0.0.1:2101,rx_port=tcp://127.0.0.1:2100,
base_srate=11.52e6
......
[pcap]
enable = none
mac_filename = /tmp/ue1_mac.pcap
mac_nr_filename = /tmp/ue1_mac_nr.pcap
nas_filename = /tmp/ue1_nas.pcap

[log]
......
filename = /tmp/ue1.log
......
[usim]
mode = soft
algo = milenage
opc = 63BFA50EE6523365FF14C1F45F88737D
k = 00112233445566778899aabbccddeeff
imsi = 001010123456780
imei = 353490069873319
......
[gw]
netns = ue1
ip_devname = tun_srsue
ip_netmask = 255.255.255.0
......
```

The configuration files for the other UEs shall change accordingly based on Table 3. Changes are depicted as underlined text.

6.1.4. Deployment with Kubernetes OSC Near-RT RIC, Containerized Open5GS, E2 Simulator, and ZMQ Connection between gNB and UE

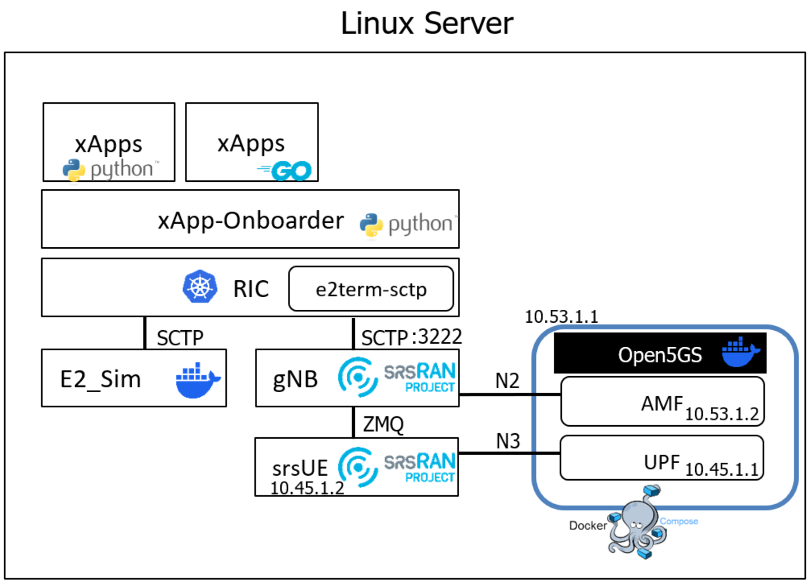


Fig. 8. Deployment with aggregated srsRAN gNB and UE, E2 Simulator, Kubernetes OSC Near-RT RIC, and Dockerized Open5GS 5G Core

For testing 5G core and 5G RAN components within a single server, it is very efficient to modularize each part as a docker container or cluster pod as it can reduce the compatibility issues that might happen in the middle of compilation and installation.

As shown in Fig. 8, We are using Kubernetes cluster for near-RT RIC, xApps as well as xApp-Onboarder, and docker or docker compose are used for 5G RAN, 5G core components. The O-RAN components in this scenario can be installed following the Sections in Table 4.

Table 4. Installation Guide for Aggregated Scenario on a Single Server with OSC Near-RT RIC.

Software	Installation Guide	Server
srsRAN gNB	Section 3.1	Server 1
srsUE	Section 3.2	Server 1
E2 Simulator	Section 3.3	Server 1
OSC Near-RT RIC	Section 5.2	Server 1
Open5GS	Section 4.4	Server 1

For 5G core solution in our testbed, the `docker compose` is used to deploy Open5GS software without having to install from the source code. Additionally, gNB and UE might also be deployed by being instantiated as docker containers. gNB and srsrUE compilation from the source code normally require somewhat intricate and strict CPU requirements and specific environments in the middle of compiling procedures. This often leads to failures in compiling procedures. To easily duplicate the installation procedures without any issues and difficulties for the same testbed environment and configuration, it is recommended to make docker container instances of the gNB and the srsUE.

The northbound interfaces of the gNB and E2 Simulator are connected to `e2term-sctp` pod in RIC cluster with SCTP protocol by the port number 32222. The southbound interface of the gNB is connected to srsUE southbound with ZMQ driver that is able to send a traffic without using RF hardware equipment.

6.2. Disaggregated Deployments

In this section, we provide several scenarios where each O-RAN component is deployed on a separate server/workstation. We first introduce the single-UE scenarios, including ZMQ and USRP connections. Next, we discuss how we use a channel emulator to provide RF connections between a gNB and multiple UEs. Finally, we discuss how we make gNB communicate with an open5GS in a docker container.

Fig. 9 and Fig. 10 illustrate the system setups of the testbeds. The servers for srsRAN gNB, FlexRIC, and Open5GS are physically connected via Ethernet cables and a switch on the 192.0.13.x subnet. Specifically, their IP addresses are as follows

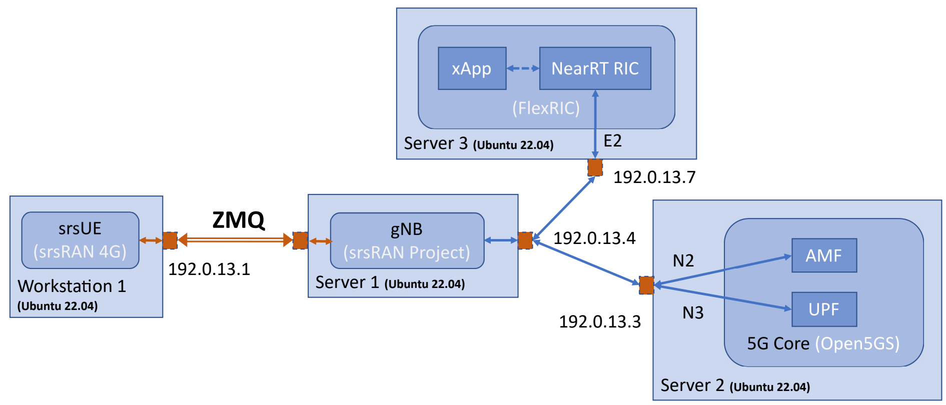
- Open5GS Server: 192.0.13.3
- srsRAN gNB Server: 192.0.13.4
- FlexRIC Server: 192.0.13.7

6.2.1. Installations on Bare Metal Servers, with ZMQ or USRP Connection

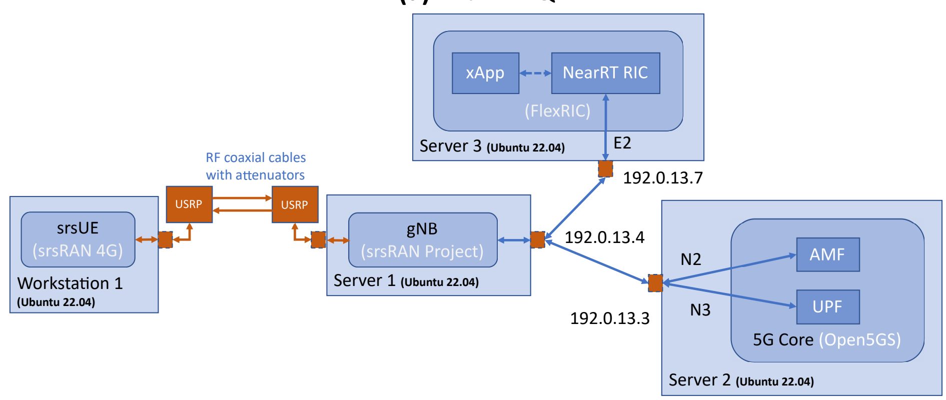
These scenarios have srsRAN gNB and UE, FlexRIC, and Open5GS installed on the bare metal servers/workstations, where we successfully installed and configured Open5GS with either package manager or from source. The O-RAN components in this scenario can be installed following the Sections in Table 5.

Table 5. Installation Guide for Disaggregated Scenario.

Software	Installation Guide	Server
srsRAN gNB	Section 3.1	Server 1
srsUE	Section 3.2	Workstation 1
FlexRIC	Section 5.1.1 or 5.1.2	Server 3
Open5GS	Section 4.1 or 4.3	Server 2



(a) with ZMQ



(b) with USRP

Fig. 9. Disaggregated deployments

To configure Open5GS, after it is installed, follow instructions in Section 4.2.1 and replace `127.0.0.5` with `192.0.13.3` in `amf.yaml: ngap: addr` and `upf.yaml: gtpu: addr`.

Follow Section 5.1.1 to change FlexRIC's IP address to `192.0.13.7`: it can either be added to `flexric.conf` before installation, or replace in `/usr/local/etc/flexric/flexric.conf` after it is installed.

Finally, srsRAN gNB can be bound to `192.0.13.4` and communicate with Open5GS and FlexRIC by

```
amf:
    addr: 192.0.13.3 # 5G core bind address
    bind_addr: 192.0.13.4 # gNB bind address
......
e2:
    enable_du_e2: true
    addr: 192.0.13.7 # RIC bind address
    bind_addr: 192.0.13.4 # gNB bind address for RIC connection
    e2sm_kpm_enabled: true
```

6.2.1.1. ZMQ Connection

To establish ZMQ connection, the following changes need to be made on gNB and UE configuration files:

1. srsRAN gNB on Server 1 sends digital signal samples via `192.0.13.4:2000` and listens for samples from `192.0.13.1:2001`. The configuration file `ru_sdr` section can be configured as:

```
ru_sdr:
    device_driver: zmq
    device_args: tx_port=tcp://192.0.13.4:2000,rx_port=tcp://
192.0.13.1:2001,base_srate=11.52e6
    srate: 11.52
    tx_gain: 75
    rx_gain: 35
```

2. srsRAN UE on Workstation 1 sends digital signal samples via `192.0.13.1:2001` and listens for samples from `192.0.13.4:2000`. The following lines in UE configuration file needs to be changed:

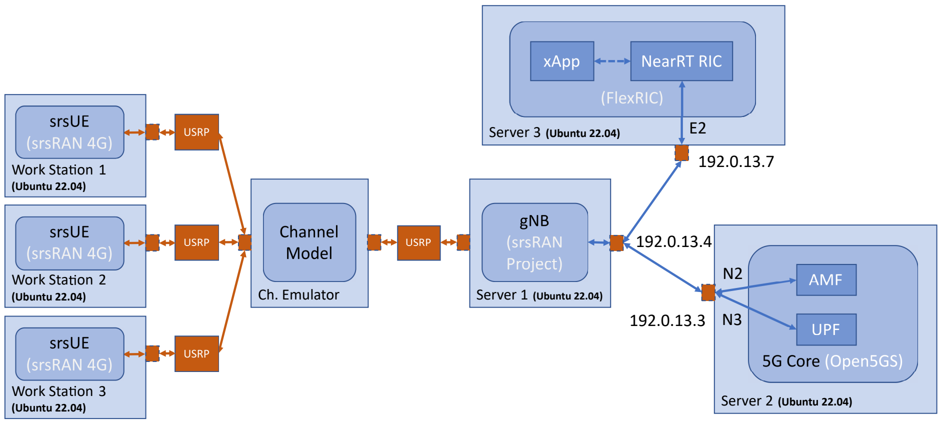
```
device_name = zmq
device_args = tx_port=tcp://192.0.13.1:2001,rx_port=tcp://
192.0.13.4:2000,base_srate=11.52e6
```

6.2.1.2. USRP Connection

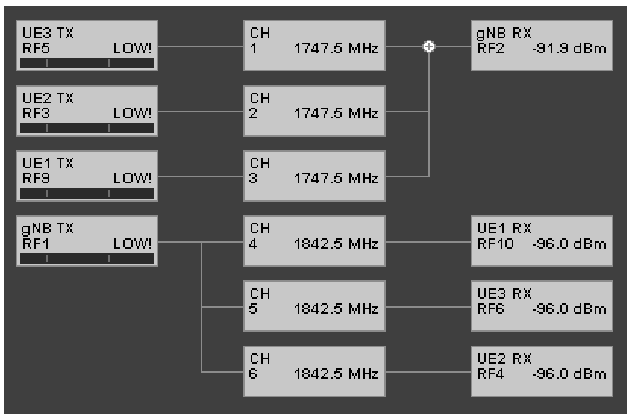
Fig. 9b provides the first disaggregated deployment we tested with Ettus B210 USRPs. The attenuation between TX and RX ports is 50 dB.

The USRP connection can be established following the same configuration changes in Section 6.1.2.2.

6.2.2. Installations on Bare Metal Servers, USRP Connection, Multiple UEs via Channel Emulator



(a) Multi-UE deployment scenario



(b) Channel emulator model for 3-UE deployment.

Fig. 10. Disaggregated deployment with multiple UEs and RF connection.

In addition to the one-UE scenarios, we were able to realize multi-UE connections to a single gNB (Fig. 10a). Each srsRAN UE is installed on a separate workstation, as listed in Table 6. The gNB and UE establish connections using Ettus B210 USRPs and a Propsim F32 channel emulator. The channel model (Fig. 10b) can be programmed to support designated uplink and downlink frequencies, and each channel's bandwidth can be configured up to 40 MHz. Additionally, each uplink or downlink channel provides 43 dB total attenuation to the input RF signal.

Table 6. Installation Guide for Disaggregated Scenario with Multiple UEs.

Software	Installation Guide	Server
srsRAN gNB	Section 3.1	Server 1
srsUE1	Section 3.2	Workstation 1
srsUE2	Section 3.2	Workstation 2
srsUE2	Section 3.2	Workstation 3
FlexRIC	Section 5.1.1 or 5.1.2	Server 3
Open5GS	Section 4.1 or 4.3	Server 2

To realize this deployment, we use the same USIMs for the UEs as listed in Fig. 3. These USIMs will need to be registered in Open5GS following Section 4.1.3 instructions. We provide the necessary changes to UE1 configuration file as an example here:

```
[rf]
freq_offset = 0
tx_gain = 75
rx_gain = 35
srate = 11.52e6
nof_antennas = 1

device_name = uhd
device_args = clock=external
time_adv_nsamples = 398
......
[usim]
mode = soft
algo = milenage
```

```
opc = 63BFA50EE6523365FF14C1F45F88737D
k = 00112233445566778899aabbccddeeff
imsi = 001010123456780
imei = 353490069873319
......
```

and the configuration files for UE2 and UE3 shall change accordingly.

6.2.3. Deployment with Containerized Open5GS

Besides installing Open5GS on a bare metal server, a dockerized Open5GS is provided in srsRAN Project to simplify the installation. The O-RAN components as well as the dockerized Open5GS in this scenario can be installed following the Sections in Table 7.

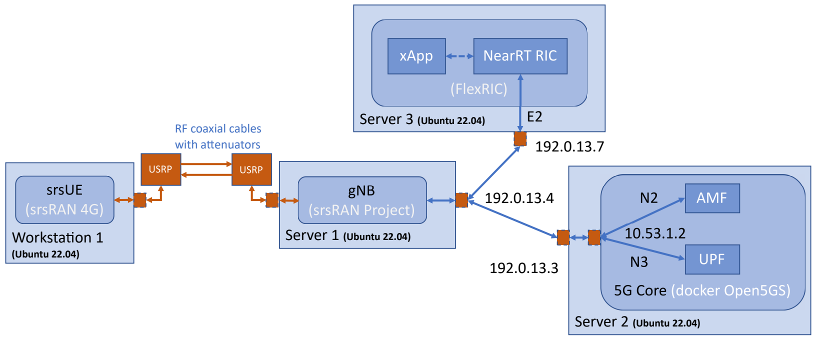


Fig. 11. Deployment with dockerized Open5GS: single-UE, USRP connection

Table 7. Installation Guide for Disaggregated Scenario with Dockerized Open5GS.

Software	Installation Guide	Server
srsRAN gNB	Section 3.1	Server 1
srsUE	Section 3.2	Workstation 1
FlexRIC	Section 5.1.1 or 5.1.2	Server 3
Open5GS	Section 4.4	Server 2

As the dockerized Open5GS uses the default IP address of `10.53.1.2`, we focus on how to configure gNB and its server's IP rules so that connection can be established between srsRAN gNB and Open5GS.

As shown in Fig. 11, the deployment has the same system setup as the disaggregated deployment in Fig. 9b, except that the Open5GS uses the same server but is encapsulated in a docker container and is using `10.53.1.2` as its IP address.

Server 2 should have access to `10.53.1.2` upon successful installation of the dockerized Open5GS. A user can verify the connectivity using "`ping 10.53.1.2`" on Server 2 terminal.

srsRAN gNB binds to Open5GS by configuring `amf: addr` in gNB configuration file:

```
amf:
    addr: 10.53.1.2 # 5G core bind address
    bind_addr: 192.0.13.4 # gNB bind address
```

In addition, gNB server (Server 1) can gain access to 10.53.1.2 in Server 2 by

```
$ sudo ip route add 10.53.1.0/24 via 192.0.13.3
```

The other configurations shall align with those for Fig. 9b scenario.

7. Automation Tool

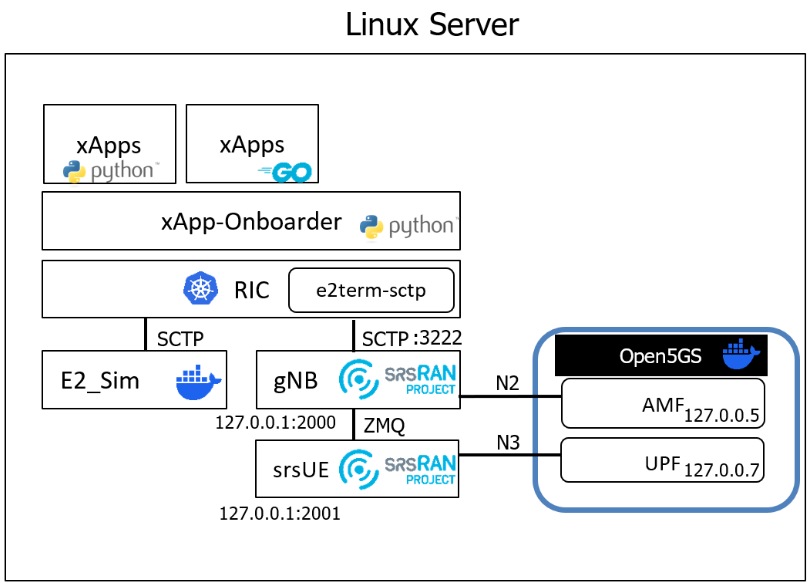


Fig. 12. Deployment Scenario from Automation Tool

The automation tool, O-RAN-Testbed-Automation, is based on the deployment scenario introduced in Section 6.1.4 and installs Open5GS from source. It streamlines the installation of components listed in Table 8 and automatically configures IP addresses, as illustrated in Fig. 12. The repository is available in [14].

Table 8. Installation Guide Followed by Automation Tool.

Software	Installation Guide
srsRAN gNB	Section 3.1
srsUE	Section 3.2
E2 Simulator	Section 3.3
OSC Near-RT RIC	Section 5.2
Open5GS	Section 4.3

The testbed deployment is packaged as an executable set of Bash shell scripts designed to run on a fresh Ubuntu-based Linux operating system. The `full_install.sh` script in each component directory installs dependencies, clones the repository, builds the source code, and deploys the respective application on the host machine. After installation, the `generate_configurations.sh` script modifies the default configuration files (.yaml or .conf) to set IP addresses, ports, and other settings necessary for proper application operation. The `run.sh` script starts the application, `stop.sh` halts it, and `is_running.sh` outputs the application's running status to `stdout`.

The automation example of the deployment scenario in Fig. 12 provides ZMQ connection between the gNB and the UE. Users can also enable additional functionalities, such as connecting to USRPs, by modifying the configuration files in the `configs` directory.

Additionally, [14] offers a comprehensive installation guide for deploying a fully disaggregated testbed.

8. Test Setup

8.1. Testbed

The test was performed using the deployment scenario in Fig. 9b, where Ettus B210 USRPs and RF cabled connections are used and 50 dB attenuation is added between each TX and RX pair. The servers are on the 192.0.13.x subnet:

- gNB Server: 192.0.13.4
- Open5GS Server: 192.0.13.3
- FlexRIC Server: 192.0.13.7

8.2. Scripts and Configurations

The script for each testbed component is configured as follows:

gNB.yaml:

```
amf:
    addr: 192.0.13.3
    bind_addr: 192.0.13.4
ru_sdr:
    device_driver: uhd
    device_args: type=b200
    clock: external
    sync: external
    srate: 11.52 tx_gain: 75
    rx_gain: 35
cell_cfg:
    dl_arfcn: 368500
    band: 3
    channel_bandwidth_MHz: 10
    common_scs: 15
    plmn: "00101"
    tac: 7
    pdcch:
        dedicated:
            ss2_type: common
            dci_format_0_1_and_1_1: false
        common:
            ss0_index: 0
            coreset0_index: 6
```

```
    prach:
        prach_config_index: 1
log:
    filename: /tmp/gnb.log
    all_level: info
    hex_max_size: 0
pcap:
    mac_enable: false
    mac_filename: /tmp/gnb_mac.pcap
    ngap_enable: false
    ngap_filename: /tmp/gnb_ngap.pcap
    e2ap_enable: true
    e2ap_filename: /tmp/gnb_e2ap.pcap
e2:
    enable_du_e2: true
    addr: 192.0.13.7
    bind_addr: 192.0.13.4
    e2sm_kpm_enabled: true
```

Note: The `gnb.yaml` above can be used when Open5GS is installed on the bare metal Server 3, either with package manager or from source. When Open5GS is installed in a docker container, 10.53.1.2 is the IP address that Open5GS binds to, as discussed in Section 4.4.1. Thus `amf->addr` in the above configuration file shall be changed to 10.53.1.2. The user should also `ping 10.53.1.2` from Server 2 to check if the dockerized Open5GS is accessible. If not, IP rules shall be added to Server 2, which, in this deployment scenario, shall be:

```
$ sudo ip route add 10.53.1.0/24 via 192.0.13.3
```

ue.conf:

```
[rf]
freq_offset = 0
tx_gain = 80
rx_gain = 35
srate = 11.52e6
nof_antennas = 1

device_name = uhd
device_args = clock=external
time_adv_nsamples = 300

[rat.eutra]
dl_earfcn = 2850
nof_carriers = 0
```

```
[rat.nr]
bands = 3
nof_carriers = 1
max_nof_prb = 52
nof_prb = 52

[pcap]
enable = none
mac_filename = /tmp/ue_mac.pcap
mac_nr_filename = /tmp/ue_mac_nr.pcap
nas_filename = /tmp/ue_nas.pcap

[log]
all_level = info
phy_lib_level = none
all_hex_limit = 32
filename = /tmp/ue.log
file_max_size = -1
[usim]
mode = soft
algo = milenage
opc = 63BFA50EE6523365FF14C1F45F88737D
k = 00112233445566778899aabbccddeeff
imsi = 001010123456780
imei = 353490069873319

[rrc]
release = 15
ue_category = 4

[nas]
apn = srsapn
apn_protocol = ipv4

[gui]
enable = false
```

Open5GS:

If Open5GS is installed with Package Manager:

In `/etc/open5gs/amf.yaml`,

```
ngap:
    - addr: 192.0.13.3
guami:
    - plmn_id:
        mcc: 001
        mnc: 01
tai:
    - plmn_id:
        mcc: 001
        mnc: 01
      tac: 7
plmn_support:
    - plmn_id:
        mcc: 001
        mnc: 01
```

In `/etc/open5gs/upf.yaml`,

```
gtpu:
    - addr: 192.0.13.3
```

In `/etc/open5gs/nrf.yaml`, check if `mcc` is 001 and `mnc` is 01. If not, make the changes. If `mcc` and `mnc` are not defined, it is not necessary to add the configuration.

```
nrf:
    serving:
        - plmn_id:
            mcc: 001
            mnc: 01
```

If changes are made in NRF configuration file, restart NRF module by

```
$ sudo systemctl restart open5gs-nrfd
```

Next, restart AMF and UPF modules

```
$ sudo systemctl restart open5gs-amfd

$ sudo systemctl restart open5gs-upfd
```

Run `systemctl` and check if all the modules are in active and running status. Complete the rest of the configuration as discussed in Section 4.

If Open5GS is installed from source:

All the NFs can be started simultaneously using the configurations in `open5gs/build/configs/sample.yaml`. To do this, user should modify the amf, upf, and nrf sections in this file, using the configuration information above.

If Open5GS is installed with Docker provided by srsRAN Project:

Follow the instructions in Section 4.4 to complete the installation.

FlexRIC:

After FlexRIC is installed, check `/usr/local/etc/flexric/flexric.conf` and make sure 192.0.13.7 is assigned to `NEAR_RIC_IP`:

```
[NEAR-RIC]
NEAR_RIC_IP = 192.0.13.7
```

Next, follow the instructions in Section 5 and finish the installation.

8.3. Running Testbed

Open5GS

If Open5GS is installed with Package Manager:

All the NFs are in running state automatically. Nothing needs to be done.

If Open5GS is installed from source:

Run the following commands:

```
$ cd open5gs

$ ./build/tests/app/5gc
```

If Open5GS is installed with Docker provided by srsRAN Project:

```
$ cd srsRAN_Project/docker/

$ docker-compose up 5gc
```

FlexRIC

To run the testbed, users should first start NearRT-RIC in FlexRIC on Server 4 by

```
$ ./flexric/build/examples/ric/nearRT-RIC
```

The following logs should be displayed:

```
Setting the config -c file to /usr/local/etc/flexric/flexric.conf
Setting path -p for the shared libraries to /usr/local/lib/flexric/
[NEAR-RIC]: nearRT-RIC IP Address = 192.0.13.7, PORT = 36421
[NEAR-RIC]: Initializing
[NEAR-RIC]: Loading SM ID = 144 with def = PDCP_STATS_V0
[NEAR-RIC]: Loading SM ID = 147 with def = ORAN-E2SM-KPM
[NEAR-RIC]: Loading SM ID = 142 with def = MAC_STATS_V0
```

```
[NEAR-RIC]: Loading SM ID = 145 with def = SLICE_STATS_V0
[NEAR-RIC]: Loading SM ID = 146 with def = TC_STATS_V0
[NEAR-RIC]: Loading SM ID = 143 with def = RLC_STATS_V0
[NEAR-RIC]: Loading SM ID = 148 with def = GTP_STATS_V0
[iApp]: Initializing ...
[iApp]: nearRT-RIC IP Address = 192.0.13.7, PORT = 36422
fd created with 6
```

gNB

Next, log in to Server 2, in the directory that contains `gnb.yaml`:

```
$ sudo gnb -c gnb.yaml
```

The following log will be displayed if gNB is launched successfully:

```
Lower PHY in quad executor mode.

--== srsRAN gNB (commit 5e6f50a20) ==--

Connecting to AMF on 192.0.13.3:38412
Available radio types: uhd and zmq.
[INFO] [UHD] linux; GNU C++ version 7.5.0; Boost_106501; UHD_4.4.0.0-
0ubuntu1~bionic1
[INFO] [LOGGING] Fastpath logging disabled at runtime.
Making USRP object with args 'type=b200'
[INFO] [B200] Detected Device: B210
[INFO] [B200] Operating over USB 3.
[INFO] [B200] Initialize CODEC control...
[INFO] [B200] Initialize Radio control...
[INFO] [B200] Performing register loopback test...
[INFO] [B200] Register loopback test passed
[INFO] [B200] Performing register loopback test...
[INFO] [B200] Register loopback test passed
[INFO] [B200] Setting master clock rate selection to 'automatic'.
[INFO] [B200] Asking for clock rate 16.000000 MHz...
[INFO] [B200] Actually got clock rate 16.000000 MHz.
[INFO] [MULTI_USRP] Setting master clock rate selection to 'manual'.
[INFO] [B200] Asking for clock rate 11.520000 MHz...
[INFO] [B200] Actually got clock rate 11.520000 MHz.
Connecting to NearRT-RIC on 192.0.13.7:36421
Cell pci=1, bw=10 MHz, dl_arfcn=368500 (n3), dl_freq=1842.5 MHz,
dl_ssb_arfcn=368410, ul_freq=1747.5 MHz

==== gNodeB started ===
Type <t> to view trace
```

If connected to NearRT-RIC and Open5GS successfully, NearRT-RIC logs will display

```
Setting the config -c file to /usr/local/etc/flexric/flexric.conf
Setting path -p for the shared libraries to /usr/local/lib/flexric/
[NEAR-RIC]: nearRT-RIC IP Address = 192.0.13.7, PORT = 36421
[NEAR-RIC]: Initializing
[NEAR-RIC]: Loading SM ID = 144 with def = PDCP_STATS_V0
[NEAR-RIC]: Loading SM ID = 147 with def = ORAN-E2SM-KPM
[NEAR-RIC]: Loading SM ID = 142 with def = MAC_STATS_V0
[NEAR-RIC]: Loading SM ID = 145 with def = SLICE_STATS_V0
[NEAR-RIC]: Loading SM ID = 146 with def = TC_STATS_V0
[NEAR-RIC]: Loading SM ID = 143 with def = RLC_STATS_V0
[NEAR-RIC]: Loading SM ID = 148 with def = GTP_STATS_V0
[iApp]: Initializing ...
[iApp]: nearRT-RIC IP Address = 192.0.13.7, PORT = 36422
fd created with 6

Received message with id = 411, port = 30397
[E2AP] Received SETUP-REQUEST from PLMN 1. 1 Node ID 411 RAN type
ngran_gNB
[NEAR-RIC]: Accepting RAN function ID 147 with def = `0ORAN-E2SM-KPM
[NEAR-RIC]: Accepting interfaceType 0
```

and on Server 3, the following logs will be added to the end of AMF logs in `/var/log/open5gs/amf.log`:

```
10/30 11:33:58.139: [amf] INFO: gNB-N2 accepted[192.0.13.4]:46777 in
ng-path module (../src/amf/ngap-sctp.c:113)
10/30 11:33:58.139: [amf] INFO: gNB-N2 accepted[192.0.13.4] in master_sm
module (../src/amf/amf-sm.c:741)
10/30 11:33:58.139: [amf] INFO: [Added] Number of gNBs is now 1
(../src/amf/context.c:1185)
10/30 11:33:58.139: [amf] INFO: gNB-N2[192.0.13.4] max_num_of_ostreams :
30 (../src/amf/amf-sm.c:780)
```

srsUE

To start srsUE, log in to Server 1, in the directory that contains `ue.conf`:

```
$ sudo srsue ue.conf
```

If srsUE starts and connection between gNB and srsUE is established, the following logs will be displayed with `RRC NR reconfiguration successful`, and an IP address with 10.45.x.x is assigned to srsUE.

```
Active RF plugins: libsrsran_rf_uhd.so libsrsran_rf_zmq.so
Inactive RF plugins:
Reading configuration file ue_usrp_disaggregated_test_from_sdrd4.conf...

Built in Release mode using commit fa56836b1 on branch master.

Opening 1 channels in RF device=uhd with args=clock=external
Supported RF device list: UHD zmq file
[INFO] [UHD] linux; GNU C++ version 11.2.0; Boost_107400; UHD_4.1.0.5-3
[INFO] [LOGGING] Fastpath logging disabled at runtime.
[INFO] [B200] Loading firmware image: /usr/share/uhd/images/usrp
_b200_fw.hex...
Opening USRP channels=1, args: type=b200,master_clock_rate=23.04e6
[INFO] [UHD RF] RF UHD Generic instance constructed
[INFO] [B200] Detected Device: B210
[INFO] [B200] Loading FPGA image: /usr/share/uhd/images/usrp_b210
_fpga.bin...
[INFO] [B200] Operating over USB 3.
[INFO] [B200] Detecting internal GPSDO....
[INFO] [GPS] No GPSDO found
[INFO] [B200] Initialize CODEC control...
[INFO] [B200] Initialize Radio control...
[INFO] [B200] Performing register loopback test...
[INFO] [B200] Register loopback test passed
[INFO] [B200] Performing register loopback test...
[INFO] [B200] Register loopback test passed
[INFO] [B200] Asking for clock rate 23.040000 MHz...
[INFO] [B200] Actually got clock rate 23.040000 MHz.
Setting manual TX/RX offset to 300 samples
Waiting PHY to initialize ... done!
Attaching UE...
Random Access Transmission: prach_occasion=0, preamble_index=0, ra-
rnti=0x39, tti=2574
Random Access Complete. c-rnti=0x4601, ta=0
RRC Connected
PDU Session Establishment successful. IP 10.45.0.33
RRC NR reconfiguration successful.
```

and in the AMF logs, the following recordings will be added to the end of `/var/log/open5gs/amf.log`:

```
10/30 13:11:03.707: [amf] INFO: InitialUEMessage (../src/amf/ngap-
handler.c:401)
10/30 13:11:03.707: [amf] INFO: [Added] Number of gNB-UEs is now 1
(../src/amf/context.c:2523)
10/30 13:11:03.707: [amf] INFO: RAN_UE_NGAP_ID[0] AMF_UE_NGAP_ID[4]
TAC[7] CellID[0x19b0] (../src/amf/ngap-handler.c:562)
10/30 13:11:03.707: [amf] INFO: [suci-0-001-01-0000-0-0-0123456780] known
UE by SUCI (../src/amf/context.c:1787)
10/30 13:11:03.707: [gmm] INFO: Registration request (../src/amf/gmm-
sm.c:1061)
10/30 13:11:03.707: [gmm] INFO: [suci-0-001-01-0000-0-0-0123456780] SUCI
(../src/amf/gmm-handler.c:157)
10/30 13:11:03.709: [amf] INFO: [imsi-001010123456780:1] Release SM
context [204] (../src/amf/amf-sm.c:491)
10/30 13:11:03.709: [amf] INFO: [imsi-001010123456780:1] Release SM
Context [state:31] (../src/amf/nsmf-handler.c:1027)
10/30 13:11:03.709: [amf] INFO: [Removed] Number of AMF-Sessions is now 0
(../src/amf/context.c:2551)
10/30 13:11:03.940: [gmm] INFO: [imsi-001010123456780] Registration
complete (../src/amf/gmm-sm.c:1993)
10/30 13:11:03.940: [amf] INFO: [imsi-001010123456780] Configuration
update command (../src/amf/nas-path.c:612)
10/30 13:11:03.940: [gmm] INFO: UTC [2023-10-30T17:11:03]
Timezone[0]/DST[0] (../src/amf/gmm-build.c:558)
10/30 13:11:03.940: [gmm] INFO: LOCAL [2023-10-30T13:11:03] Timezone[-
14400]/DST[1] (../src/amf/gmm-build.c:563)
10/30 13:11:03.940: [amf] INFO: [Added] Number of AMF-Sessions is now 1
(../src/amf/context.c:2544)
10/30 13:11:03.940: [gmm] INFO: UE SUPI[imsi-001010123456780] DNN[srsapn]
S_NSSAI[SST:1 SD:0xffffff] (../src/amf/gmm-handler.c:1247)
10/30 13:11:03.965: [gmm] INFO: [imsi-001010123456780] No GUTI allocated
(../src/amf/gmm-sm.c:1323)
10/30 13:11:04.093: [amf] INFO: [imsi-001010123456780:1:11][0:0:NULL]
/nsmf-pdusession/v1/sm-contexts/smContextRef/modify (../src/amf/nsmf-
handler.c:837)
```

8.4. Tests

Once connection between srsUE and gNB is established, tests can be performed on the testbed.

8.4.1. Ping

In this test setup, the IP addresses are

- 5G Core: 10.45.0.1,
- UE: 10.45.0.33 (Note UE's IP address changes every time srsUE restarts).

As IP route is created on 5G Core's and UE's servers once connection is established, the ping packets between these two nodes go through the assigned tunnel interface:

On Open5GS server:

```
$ ip route

10.45.0.0/16 dev ogstun proto kernel scope link src 10.45.0.1
```

On srsUE server:

```
$ ip route

10.45.0.0/24 dev tun_srsue proto kernel scope link src 10.45.0.33
```

To ping UE from 5G Core, on the Open5GS server,

```
$ ping 10.45.0.33 -c 10

PING 10.45.0.33 (10.45.0.33) 56(84) bytes of data.
64 bytes from 10.45.0.33: icmp_seq=1 ttl=64 time=22.4 ms
64 bytes from 10.45.0.33: icmp_seq=2 ttl=64 time=21.8 ms
64 bytes from 10.45.0.33: icmp_seq=3 ttl=64 time=40.8 ms
64 bytes from 10.45.0.33: icmp_seq=4 ttl=64 time=39.8 ms
64 bytes from 10.45.0.33: icmp_seq=5 ttl=64 time=38.8 ms
64 bytes from 10.45.0.33: icmp_seq=6 ttl=64 time=36.7 ms
64 bytes from 10.45.0.33: icmp_seq=7 ttl=64 time=36.8 ms
64 bytes from 10.45.0.33: icmp_seq=8 ttl=64 time=35.7 ms
64 bytes from 10.45.0.33: icmp_seq=9 ttl=64 time=34.7 ms
64 bytes from 10.45.0.33: icmp_seq=10 ttl=64 time=32.7 ms

--- 10.45.0.33 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9009ms
rtt min/avg/max/mdev = 21.806/34.017/40.800/6.366 ms
```

To ping 5G Core from UE, on srsUE server,

```
$ ping 10.45.0.1 -c 10

PING 10.45.0.1 (10.45.0.1) 56(84) bytes of data.
64 bytes from 10.45.0.1: icmp_seq=1 ttl=64 time=42.5 ms
64 bytes from 10.45.0.1: icmp_seq=2 ttl=64 time=41.4 ms
64 bytes from 10.45.0.1: icmp_seq=3 ttl=64 time=39.3 ms
64 bytes from 10.45.0.1: icmp_seq=4 ttl=64 time=39.4 ms
64 bytes from 10.45.0.1: icmp_seq=5 ttl=64 time=38.4 ms
64 bytes from 10.45.0.1: icmp_seq=6 ttl=64 time=37.3 ms
```

```
64 bytes from 10.45.0.1: icmp_seq=7 ttl=64 time=35.3 ms
64 bytes from 10.45.0.1: icmp_seq=8 ttl=64 time=35.3 ms
64 bytes from 10.45.0.1: icmp_seq=9 ttl=64 time=34.3 ms
64 bytes from 10.45.0.1: icmp_seq=10 ttl=64 time=33.3 ms

--- 10.45.0.1 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9009ms
rtt min/avg/max/mdev = 33.338/37.642/42.454/2.907 ms
```

User can also ping a website's hostname (for example, Google) by

```
$ ping www.google.com -I tun_srsue -c 10

PING www.google.com (142.250.72.4) from 10.45.0.14 tun_srsue: 56(84)
bytes of data.
64 bytes from den08s06-in-f4.1e100.net (142.250.72.4): icmp_seq=1 ttl=104
time=66.0 ms
64 bytes from den08s06-in-f4.1e100.net (142.250.72.4): icmp_seq=2 ttl=104
time=63.3 ms
64 bytes from den08s06-in-f4.1e100.net (142.250.72.4): icmp_seq=3 ttl=104
time=64.4 ms
64 bytes from den08s06-in-f4.1e100.net (142.250.72.4): icmp_seq=4 ttl=104
time=60.4 ms
64 bytes from den08s06-in-f4.1e100.net (142.250.72.4): icmp_seq=5 ttl=104
time=61.3 ms
64 bytes from den08s06-in-f4.1e100.net (142.250.72.4): icmp_seq=6 ttl=104
time=80.3 ms
64 bytes from den08s06-in-f4.1e100.net (142.250.72.4): icmp_seq=7 ttl=104
time=79.3 ms
64 bytes from den08s06-in-f4.1e100.net (142.250.72.4): icmp_seq=8 ttl=104
time=78.3 ms
64 bytes from den08s06-in-f4.1e100.net (142.250.72.4): icmp_seq=9 ttl=104
time=77.3 ms
64 bytes from den08s06-in-f4.1e100.net (142.250.72.4): icmp_seq=10
ttl=104 time=76.5 ms

--- www.google.com ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9009ms
rtt min/avg/max/mdev = 60.439/70.725/80.340/7.817 ms
```

It is important to note that to force the ping packets go through the tunnel interface for srsUE (`tun_srsue`), `-I tun_srsue` has to be added to the command.

If ping cannot reach a host name, refer to Section 4.2.4 for troubleshooting.

8.4.2. Iperf3

Iperf3 can be used to test the Uplink (UL) and Downlink (DL) throughput. To set up the UL test, on Open5GS server

```
$ iperf3 -s -i 1
```

and on srsUE server

```
$ iperf3 -c 10.45.0.1 -b 15M -i 1 -t 60
```

where the bandwidth `-b` and time duration `-t` in seconds can be specified as needed.

Uplink Test

An iperf3 test result for UL is shown here:

On Open5GS Server:

```
$ iperf3 -s -i 1

-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 10.45.0.33, port 44714
[ 5] local 10.45.0.1 port 5201 connected to 10.45.0.33 port 44728
[ ID]  Interval          Transfer      Bitrate
[ 5]   0.00-1.00    sec  1.51  MBytes  12.7  Mbits/sec
[ 5]   1.00-2.00    sec  1.70  MBytes  14.3  Mbits/sec
[ 5]   2.00-3.00    sec  1.69  MBytes  14.2  Mbits/sec
[ 5]   3.00-4.00    sec  1.40  MBytes  11.7  Mbits/sec
[ 5]   4.00-5.00    sec  2.02  MBytes  17.0  Mbits/sec
[ 5]   5.00-6.00    sec  1.74  MBytes  14.6  Mbits/sec
[ 5]   6.00-7.00    sec  1.71  MBytes  14.4  Mbits/sec
[ 5]   7.00-8.00    sec  1.71  MBytes  14.4  Mbits/sec
[ 5]   8.00-9.00    sec  1.73  MBytes  14.5  Mbits/sec
[ 5]   9.00-10.00   sec  1.74  MBytes  14.6  Mbits/sec
[ 5]   10.00-10.17  sec  295   KBytes  4.5   Mbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID]  Interval         Transfer      Bitrate
[ 5]   0.00-10.17  sec  17.2  MBytes  14.2  Mbits/sec
```

On srsUE Server:

```
$ iperf3 -c 10.45.0.1 -b 15M -i 1 -t 10

Connecting to host 10.45.0.1, port 5201
[ 5] local 10.45.0.33 port 44728 connected to 10.45.0.1 port 5201
[ ID]  Interval         Transfer      Bitrate          Retr  Cwnd
[ 5]   0.00-1.00  sec  1.75  MBytes  14.7  Mbits/sec  0     129  KBytes
```

```
[ 5]  1.00-2.00   sec  1.88  MBytes  15.7  Mbits/sec  0   216  KBytes
[ 5]  2.00-3.00   sec  1.75  MBytes  14.7  Mbits/sec  0   303  KBytes
[ 5]  3.00-4.00   sec  1.62  MBytes  13.6  Mbits/sec  12  261  KBytes
[ 5]  4.00-5.00   sec  2.00  MBytes  16.8  Mbits/sec  0   296  KBytes
[ 5]  5.00-6.00   sec  1.75  MBytes  14.7  Mbits/sec  0   333  KBytes
[ 5]  6.00-7.00   sec  1.75  MBytes  14.7  Mbits/sec  0   355  KBytes
[ 5]  7.00-8.00   sec  1.75  MBytes  14.7  Mbits/sec  4   263  KBytes
[ 5]  8.00-9.00   sec  1.75  MBytes  14.7  Mbits/sec  0   286  KBytes
[ 5]  9.00-10.00  sec  1.75  MBytes  14.7  Mbits/sec  0   298  KBytes
- - - - - - - - - - - - - - - - - - - - - - - - - -
[ ID]  Interval         Transfer      Bitrate          Retr
[ 5]   0.00-10.00  sec  17.8  MBytes  14.9  Mbits/sec  16    sender
[ 5]   0.00-10.17  sec  17.2  MBytes  14.2  Mbits/sec        receiver

iperf Done.
```

To test the DL throughput, on srsUE server

```
$ iperf3 -s -i 1
```

and on Open5GS server

```
$ iperf3 -c 10.45.0.33 -b 15M -i 1 -t 60
```

Another approach to reversing the direction of data transmission is to simply add `-R` at the end of the command at the client side.

Downlink Test

An iperf3 test result for DL is shown here:

On srsUE Server:

```
$ iperf3 -s -i 1

-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 10.45.0.1, port 41844
[ 5] local 10.45.0.33 port 5201 connected to 10.45.0.1 port 41850
[ ID]  Interval        Transfer      Bitrate
[ 5]   0.00-1.00  sec  1.70  MBytes  14.3  Mbits/sec
[ 5]   1.00-2.00  sec  1.84  MBytes  15.4  Mbits/sec
[ 5]   2.00-3.00  sec  1.65  MBytes  13.9  Mbits/sec
[ 5]   3.00-4.00  sec  1.90  MBytes  15.9  Mbits/sec
[ 5]   4.00-5.00  sec  1.79  MBytes  15.0  Mbits/sec
[ 5]   5.00-6.00  sec  1.81  MBytes  15.2  Mbits/sec
[ 5]   6.00-7.00  sec  1.77  MBytes  14.8  Mbits/sec
[ 5]   7.00-8.00  sec  1.80  MBytes  15.1  Mbits/sec
```

```
[ 5]  8.00-9.00    sec  1.79  MBytes  15.0  Mbits/sec
[ 5]  9.00-10.00   sec  1.67  MBytes  14.0  Mbits/sec
[ 5]  10.00-10.07  sec  256   KBytes  30.1  Mbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID]  Interval         Transfer      Bitrate
[ 5]   0.00-10.07  sec  17.9  MBytes  15.0  Mbits/sec
```

On Open5GS Server:

```
$ iperf3 -c 10.45.0.33 -b 15M -i 1 -t 10

Connecting to host 10.45.0.33, port 5201
[ 5] local 10.45.0.1 port 41850 connected to 10.45.0.33 port 5201
[ ID]  Interval         Transfer      Bitrate          Retr  Cwnd
[ 5]   0.00-1.00   sec  1.82  MBytes  15.3  Mbits/sec  0     151  KBytes
[ 5]   1.00-2.00   sec  1.88  MBytes  15.7  Mbits/sec  0     154  KBytes
[ 5]   2.00-3.00   sec  1.75  MBytes  14.7  Mbits/sec  0     170  KBytes
[ 5]   3.00-4.00   sec  1.62  MBytes  14.7  Mbits/sec  12    216  KBytes
[ 5]   4.00-5.00   sec  1.75  MBytes  14.7  Mbits/sec  0     244  KBytes
[ 5]   5.00-6.00   sec  1.88  MBytes  15.7  Mbits/sec  0     259  KBytes
[ 5]   6.00-7.00   sec  1.75  MBytes  14.7  Mbits/sec  0     259  KBytes
[ 5]   7.00-8.00   sec  1.75  MBytes  14.7  Mbits/sec  4     280  KBytes
[ 5]   8.00-9.00   sec  1.88  MBytes  15.7  Mbits/sec  0     299  KBytes
[ 5]   9.00-10.00  sec  1.75  MBytes  14.7  Mbits/sec  0     299  KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID]  Interval         Transfer      Bitrate          Retr
[ 5]   0.00-10.00  sec  17.9  MBytes  15.1  Mbits/sec  16    sender
[ 5]   0.00-10.07  sec  17.9  MBytes  15.0  Mbits/sec        receiver

iperf Done.
```

8.4.3. xApp

[10] provides an example of xApp from FlexRIC, *xapp_kpm_moni*, which connects to NearRT-RIC and uses O-RAN E2 Service Model - Key Performance Measurement (E2SM-KPM) to monitor the Reference Signal Receive Power (RSRP).

To start E2SM-KPM xApp, the following app shall start on the FlexRIC server while the testbed is running

```
$ ./flexric/build/examples/xApp/c/monitor/xapp_kpm_moni
```

When the connection to NearRT-RIC is established, similar logs as follows will be added to the NearRT-RIC terminal:

```
[iApp]: E42 SETUP-REQUEST received
[iApp]: E42 SETUP-RESPONSE sent
[iApp]: SUBSCRIPTION-REQUEST xapp_ric_id->ric_id.ran_func_id 147
[E2AP] SUBSCRIPTION REQUEST generated
[NEAR-RIC]: nb_id 411 port = 26268
```

Some of the logs displayed on the terminal of E2SM-KPM xApp is shown here:

```
......
 [xApp]: E42 SETUP-REQUEST sent
adding event fd = 8 ev-> 4
[xApp]: E42 SETUP-RESPONSE received
[xApp]: xApp ID = 8
Registered E2 Nodes = 1
Pending event size before remove = 1
Connected E2 nodes = 1
Registered node 0 ran func id = 147
 Generated of req_id = 1
[xApp]: RIC SUBSCRIPTION REQUEST sent
adding event fd = 8 ev-> 5
[xApp]: SUBSCRIPTION RESPONSE received
Pending event size before remove = 1
[xApp]: Successfully SUBSCRIBED to ran function = 147
Received RIC Indication:
---Metric: RSRP: Value: 32 Received RIC Indication:
---Metric: RSRP: Value: 34
Received RIC Indication:
---Metric: RSRP: Value: 33
Received RIC Indication:
---Metric: RSRP: Value: 32
Received RIC Indication:
---Metric: RSRP: Value: 32
Received RIC Indication:
---Metric: RSRP: Value: 34
......
```

9. Conclusion and Future Work

This documentation presents a foundational blueprint for new researchers embarking on the journey of setting up an O-RAN testbed from the ground up. We have provided an overview of the O-RAN architecture and its associated software stacks, alongside an introduction to the various deployment scenarios that have been tested on our testbeds, including both aggregated and disaggregated setups. We introduced the automation tool that significantly enhances the deployment process of the O-RAN testbed by streamlining installations and configurations, ultimately improving efficiency and reducing the potential for errors. The installation instructions for each software stack are outlined to facilitate seamless deployment, and a testbed example of a disaggregated deployment scenario demonstrates the configuration and operation of 5G O-RAN testbed towards successful application and interoperation.

This guide is designed to be a comprehensive starting point for researchers, offering essential insights and practical guidance towards successful testbed implementation and operation. By following these instructions, researchers can effectively replicate and build upon our testbed framework, paving the way for further exploration and development in O-RAN control, application, and use cases.

In the future updates, we will expand this blueprint to include additional deployment scenarios and software integrations, further enhancing its utility for researchers. Specifically, we will provide instructions for incorporating the OSC near-RT RIC provided by srs, including its RAN Control functions and xApp modules, as well as the OSC non-RT RIC. We will also incorporate the instructions for OAI gNB and UE when higher-end USRPs of X410 are used. These additions will offer a broader perspective on O-RAN testbed deployment and extend the availability and interoperability of the O-RAN software stacks.

In addition, we plan to extend the deployment and test automation to include other software stack options. This enhancement will further improve deployment efficiency, reduce manual operation errors, and enable extensive testing on our O-RAN testbeds. Moreover, it will offer repeatable and rapid deployment solutions with diverse software stack choices for new researchers.

References

[1] O-RANWG1OAD (2024) O-RAN Work Group 1 (Use Cases and Overall Architecture): O-RAN Architecture Description (O-RAN Alliance), Standard.

[2] Open Networking Foundation (2023) ”Hardware Installation - Prerequisites”. Accessed: 2023-10-24. Available at https://docs.sd-ran.org/master/sdran-in-a-box/docs/HW_Installation_prereq.html.

[3] srsRAN (2023) ”Running srsRAN Project”. Accessed: 2023-10-24. Available at https://docs.srsran.com/projects/project/en/latest/user_manuals/source/running.html.

[4] srsRAN (2023) ”srsRAN Project - Installation Guide”. Accessed: 2023-10-24. Available at https://docs.srsran.com/projects/project/en/latest/user_manuals/source/installation.html.

[5] srsRAN (2023) ”srsRAN Project - srsRAN gNB with srsUE”. Accessed: 2023-10-24. Available at https://docs.srsran.com/projects/project/en/latest/tutorials/source/srsUE/source/index.html.

[6] srsRAN (2023) ”gnb_rf_b210_fdd_srsue.yml”. Accessed: 2023-10-24. Available at https://github.com/srsran/srsRAN_Project/blob/main/configs/gnb_rf_b210_fdd_srsUE.yml.

[7] srsRAN (2023) ”srsRAN 4G Features”. Accessed: 2023-10-26. Available at https://docs.srsran.com/projects/4g/en/latest/feature_list.html.

[8] srsRAN (2023) ”ue_rf.conf”. Accessed: 2023-10-26. Available at https://docs.srsran.com/projects/project/en/latest/_downloads/900a04eeabbe80c1bb9f3e571afaa804/ue_rf.conf.

[9] Open5GS (2023) ”Building Open5GS from Sources”. Accessed: 2023-10-25. Available at https://open5gs.org/open5gs/docs/guide/02-building-open5gs-from-sources/.

[10] srsRAN (2023) ”O-RAN NearRT-RIC and xApp”. Accessed: 2023-10-25. Available at https://docs.srsran.com/projects/project/en/latest/tutorials/source/flexric/source/index.html.

[11] Open5GS (2023) ”Quickstart”. Accessed: 2023-10-25. Available at https://open5gs.org/open5gs/docs/guide/01-quickstart/#:~:text=restart%20open5gs%2Dsgwud-,Setup%20a%205G%20Core,-You%20will%20need.

[12] EURECOM (2024) ”Flexric”. Accessed: 2024-01-09. Available at https://gitlab.eurecom.fr/mosaic5g/flexric.

[13] srsRAN (2023) ”Unknown RAN Function ID”. Accessed: 2023-12-21. Available at https://github.com/srsran/srsRAN_Project/discussions/368#discussioncomment-7909775.

[14] Simeon Wuthier (2024) ”O-RAN-Testbed-Automation”. Accessed: 2024-10-04. Available at https://github.com/usnistgov/O-RAN-Testbed-Automation.