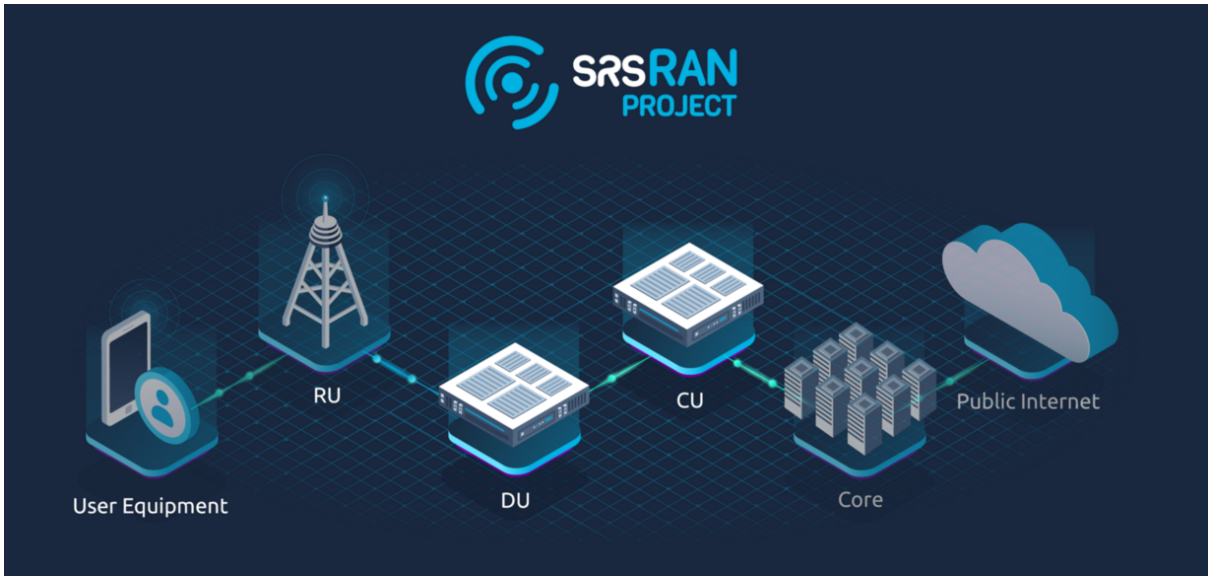
srsRAN Project Documentation

['Software Radio Systems']

Feb 21, 2024

GENERAL



The srsRAN Project is an open-source 5G CU/DU from SRS.
It is a complete RAN solution compliant with 3GPP and O-RAN Alliance specifications. The srsRAN Project includes the full L1/2/3 stack with minimal external dependencies. The software is portable across processor architectures and scalable from low-power embedded systems to cloudRAN, providing a powerful platform for mobile wireless research and development.

Get started with the srsRAN Project:

- *Install* srsRAN Project on your computer.
- Get up and *running*.
- More advanced users should read the Developers Guide.
- Get up to speed on 5G and OpenRAN with our Knowledge Base.

Useful resources:

- The srsRAN Project source code.
- Explore the srsRAN Project Discussions for news and user support.
- Discover the srsRAN Enterprise solution for Private 5G network deployments.

CHAPTER

ONE

GETTING STARTED

Install srsRAN Project on your computer:

- *Installation*

Run srsRAN Project:

- *Running*

Read the configuration reference:

- *Configuration reference*

Take a look at the source code:

- srsRAN Project on GitHub

Join the community for news and user support:

- srsRAN Project Discussions

Learn about the team behind the srsRAN Project:

- Software Radio Systems

CHAPTER

TWO

FEATURES AND ROADMAP

2.1 Current Features

- FDD/TDD supported, all FR1 bands
- 15/30 kHz subcarrier spacing
- All physical channels including PUCCH Format 1 and 2, excluding Sounding-RS
- Highly optimized LDPC and Polar encoder/decoder for ARM Neon and x86 AVX2/AVX512
- All RRC procedures excluding Handover
- All MAC procedures excluding power control
- Split 7.2 support using in-house OFH library
- Support for QAM-256
- 4x4 MIMO
- Slicing

2.2 Coming Soon

- Multi-cell support
- Handover
- CU/DU split deployment

CHAPTER

THREE

CONTRIBUTION GUIDELINES

Contributions to the srsRAN Project are always welcome. The easiest way to contribute is by issuing a pull request on the srsRAN Project repository. Take a look at our *code style guide* and please follow our *testing policy*.

We ask srsRAN Project contributors to agree to a Copyright License Agreement. This provides us with necessary permissions to use contributed code. For more information, see the FAQ below.

3.1 FAQ

3.1.1 1. What is a Copyright License Agreement (CLA) and why do I need one?

A Copyright License Agreement is a legal document in which you state you are entitled to contribute the code/documentation/translation to the project you're contributing to and are willing to have it used in distributions and derivative works. This means that should there be any kind of legal issue in the future as to the origins and ownership of any particular piece of code, then that project has the necessary forms on file from the contributor(s) saying they were permitted to make this contribution.

The CLA also ensures that once you have provided a contribution, you cannot try to withdraw permission for its use at a later date. People and companies can therefore use that software, confident that they will not be asked to stop using pieces of the code at a later date.

The agreements used by the srsRAN Project are standard documents provided by Project Harmony, a community-centered group focused on contributor agreements for free and open source software (FOSS). For more information, see www.harmonyagreements.org.

3.1.2 2. How do I complete and submit the CLA?

The srsRAN Project CLA for Individual contributions can be found `here`. The srsRAN Project CLA for Entity contributions can be found `here`. Download the appropriate CLA, then print, sign and scan the document before sending by email to licensing@srs.io.

3.1.3 3. How will my contributions to the srsRAN Project be used?

The srsRAN Project was created and is maintained by Software Radio Systems (SRS), a private limited company headquartered in Ireland. SRS provides the srsRAN Project under both the open-source AGPLv3 license and commercial licenses. SRS also sells proprietary software products which build upon the srsRAN Project codebase. In this way, we attempt to ensure the ongoing development, evolution and sustainability of the srsRAN Project.

Through the license agreements, we ask you to grant us permission to use your contributions within the srsRAN Project and to continue to provide the srsRAN Project under open-source and commercial licenses and within proprietary products. As we do not ask for copyright assignment, you retain complete ownership of your contributions and have the same rights to use or license those contributions which you would have had without entering into a license agreement.

3.1.4 4. If I do not own the copyright in the entire work of authorship?

If you do not own the copyright in the entire work of authorship, you can only contribute the work if the third-party works have been submitted separately by the upsream owner or under the upstream license.

If you have any questions about srsRAN Project licensing and contributions, please contact us at licensing@srs.io

CHAPTER

FOUR

REPORTING

4.1 Issues

For contributing code and reporting issues, use Github's pull request or issue tracking system.

4.2 Vulnerabilities

For issues or fixes that could potentially affect software security, please use the dedicated `security@srs.io` email for submissions.
Our target initial response time for any vulnerability report is less than 7 days.

4.3 Support

For support and help using srsRAN Project, check out the community driven discussion forum.

CHAPTER

FIVE

RELEASE NOTES

5.1 23.10

- Added downlink MIMO (up to 4 layers)
- Added Open Fronthaul (OFH) interface for split 7.2 ORAN radio units (optionally using DPDK)
- Added E2 interface including KPM and RAN control (RC) service model
- Added Timing Advance support
- Added Docker files
- Expose many more config parameters
- Other bug-fixes and improved stability and performance in all part

5.2 23.5

- Updated ASN.1 of RRC/NGAP/F1AP/E1AP to 3GPP Release 17.4
- Add UE capability transfer procedure
- Add support for srsUE
- CSI-RS in Downlink
- Add virtual RF driver to use with Amarisoft UE
- Expose further config parameters
- Other bug-fixes and improved stability and performance in all parts

5.3 23.3 (initial public release)

- FDD/TDD supported, all FR1 bands
- 15/30 KHz subcarrier spacing
- All physical channels including PUCCH Format 1 and 2, excluding Sounding-RS
- Highly optimized LDPC and Polar encoder/decoder for ARM Neon and x86 AVX2/AVX512
- All RRC procedures excluding Mobility, Paging and Reestablishment
- All MAC procedures excluding power control

- Split 8 support using Ettus/NI USRPs

CHAPTER

SIX

INSTALLATION GUIDE

The following steps need to be taken in order to download and build the srsRAN Project:

1. Install dependencies
2. Install RF driver
3. Clone the repository
4. Build the codebase

6.1 Build Tools and Dependencies

The srsRAN Project uses CMake and C++14. We recommend the following build tools:

- cmake
- gcc (v9.4.0 or later) **OR** Clang (v10.0.0 or later)

The srsRAN Project has the following necessary dependencies:

- libfftw
- libsctp
- yaml-cpp
- PolarSSL/mbedTLS
- googletest

You can install the required build tools and dependencies for various distributions as follows:

Ubuntu 22.04:

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

Fedora:

```
sudo yum install cmake make gcc gcc-c++ fftw-devel lksctp-tools-devel yaml-
↪cpp-devel mbedtls-devel gtest-devel
```

Arch Linux:

```
sudo pacman -S cmake make base-devel fftw mbedtls yaml-cpp lksctp-tools gtest
```

It is also recommended users install the following (although they are not required):

- Ccache: This will help to speed up re-compilation
- backward-cpp: This library helps to generate more informative backtraces in the stdout if an error occurs during runtime

---

6.2 RF-drivers

The srsRAN Project uses RF drivers to support different radio types.
Currently, only UHD is supported however additional drivers are under development:

- UHD

Note: We recommended the LTS version of UHD, i.e. either 3.15 or 4.0.

6.3 Clone and Build

First, clone the srsRAN Project repository:

```
git clone https://github.com/srsRAN/srsRAN_Project.git
```

Then build the code-base:

```
cd srsRAN_Project
mkdir build
cd build
cmake ../
make -j $(nproc)
make test -j $(nproc)
```

You can now run the gNB from `srsRAN_Project/build/apps/gnb/`. If you wish to install the srsRAN Project gNB, you can use the following command:

```
sudo make install
```

The *Running srsRAN Project* section of the documentation further discusses how to configure and run the gNB application.

6.4 Packages

srsRAN Project is available to download directly from packages for various linux distributions. Users looking for a simple installation who do not wish to edit the source code should use the package installation.

6.4.1 Ubuntu

Ubuntu users can download the srsRAN Project packages using the following commands:

```
sudo add-apt-repository ppa:softwareradiosystems/srsran-project
sudo apt-get update
sudo apt-get install srsran-project -y
```

The application can then be run with the:

```
sudo gnb -c <config file>
```

6.4.2 Arch Linux

Arch Linux users can download the srsRAN Project packages using an AUR helper, e.g. 'yay', using the following command:

```
yay -Sy srsran-project-git
```

This will build and install the latest version of srsRAN Project from git.

When installed from packages srsRAN Project example configs can be found in `/usr/share/srsran`. For info on these config files, see *here*

6.5 PHY testvectors

A number of PHY tests are based on MATLAB generated testvectors. By default, those tests are disabled. The following steps are required to enable them:

1. Download the PHY testvector set.
2. Copy the PHY testvectors to its location within srsRAN:

```
tar -xzf phy_testvectors.tar.gz -C /path_to_your_local_repository/srsgnb/
```

3. Enable the use of PHY testvectors by modifying the root CMakeLists.txt as shown below:

```
option(USE_PHY_TESTVECTORS   "Enable testvector PHY tests"              ON)
```

4. Rebuild srsRAN Project.

CHAPTER

SEVEN

RUNNING SRSRAN PROJECT

7.1 Baseline Requirements

To successfully run the srsRAN Project gNB you will need the following:

- A PC with a Linux based OS (Ubuntu 20.04 or later)
- A USRP device
- srsRAN Project (see the *Installation Guide*)
- A 3rd-party 5G core (we recommend Open5GS)
- A 3rd-party 5G UE

Recommended:

- External clock source

If you plan to connect the gNB to a COTS UE we recommend that you use an external clock source such as an Octoclock or GPSDO that is compatible with your RF-frontend, as the on-board clock within the USRP may not be accurate enough to enable a connection with the UE. This is discussed further in the relevant app note.

7.2 System Preparation

Before running the gNB application, we recommend tuning your system for best performance. We provide a script to configure known performance parameters:

- srsran_performance

The script does the following:

1. Sets the scaling governor to performance
2. Disables DRM KMS polling
3. Tunes network buffers (Ethernet based USRPs only)

Run the script as follows from the main project folder:

```
sudo ./scripts/srsran_performance
```

7.3 Running the gNB

If the gNB has been installed using `sudo make install` or installed from packages then you will be able to run the gNB from anywhere on your machine.
If you built the gNB from source and have not installed it, then you can run the gNB from: `/srsRAN_Project/build/apps/gnb`. In this folder you will find the gNB application binary.

Run the gNB as follows, passing the YAML configuration file:

```
sudo gnb -c gnb_rf_b200_tdd_n78_10mhz.yml
```

Run the gNB with sudo to ensure threads are configured with the correct priority.
Run the gNB with an explicit configuration file. See the *configuration reference* for more details.
Example configuration files can be found in the `configs/` folder in the srsRAN Project codebase.

When running, the gNB should generate the following console output:

```
Available radio types: uhd.

---== srsRAN gNB (commit 77be7d339) ==---

[INFO] [UHD] linux; GNU C++ version 9.4.0; Boost_107100; UHD_4.2.0.HEAD-0-
↪g197cdc4f
Making USRP object with args 'type=b200'
Cell pci=1, bw=10 MHz, dl_arfcn=632628 (n78), dl_freq=3489.42 MHz, dl_ssb_
↪arfcn=632640, ul_freq=3489.42 MHz

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

Entering `t` will enable the console trace, see *here* for more details.

Configuration parameters can also be passed on the command line. To see the list of options, use:

```
./gnb --help
```

CHAPTER

EIGHT

CONSOLE REFERENCE

The gNB application runs in the console. When running, type `t` in the console to enable the metrics trace.

A sample output showing bi-directional traffic can be seen here:

```
          -------------DL-----------------|------------------UL-----------------
↪----
pci rnti  cqi  mcs  brate   ok  nok  (%) | pusch  mcs  brate   ok  nok  (%)  ␣
↪ bsr
  1 4601   15   28   139M  867    0   0% |  29.4   28   133M  779    0   0%  ␣
↪150k
  1 4601   15   28   138M  851   10   1% |  29.5   28   121M  704    6   0%  ␣
↪150k
  1 4601   15   28   138M  863    0   0% |  29.5   28   133M  778    0   0%  ␣
↪150k
  1 4601   15   28   139M  869    0   0% |  29.3   28   134M  781    0   0%  ␣
↪150k
  1 4601   15   28   130M  800   10   1% |  29.8   28   131M  745    2   0%  ␣
↪150k
  1 4601   15   28   138M  855    8   0% |  29.3   28   134M  772    4   0%  ␣
↪150k
  1 4601   15   28   138M  865    0   0% |  28.7   28   133M  778    0   0%  ␣
↪150k
  1 4601   15   28   137M  848   11   1% |  29.7   28   133M  767    2   0%  ␣
↪150k
  1 4601   15   28   135M  838    4   0% |  29.6   28   132M  749    4   0%  ␣
↪150k
  1 4601   15   28   139M  867    0   0% |  29.3   28   133M  779    0   0%  ␣
↪150k
  1 4601   15   28   139M  867    0   0% |  29.6   28   133M  778    0   0%  ␣
↪150k
```

Metrics are provided on a per-UE basis. The following metrics are provided:

pci
: Physical Cell Identifier

rnti
: Radio Network Temporary Identifier (UE identifier)

cqi
: Channel Quality Indicator reported by the UE (1-15)

mcs
: Modulation and coding scheme (0-28)

brate
: Bitrate (bits/sec)

ok
: Number of packets successfully sent

nok
: Number of packets dropped

(%)
: % of packets dropped

pusch
: PUSCH SINR (Signal-to-Interference-plus-Noise Ratio)

bsr
: Buffer Status Report - data waiting to be transmitted as reported by the UE (bytes)

CHAPTER

NINE

OUTPUTS

9.1 Logs

The srsRAN Project gNB application provides a highly configurable logging mechanism, with per-layer and per-component log levels.

Set the log file path and log levels in the gNB config file. See the *Configuration Reference* for more details.

The format used for all log messages is as follows:

Timestamp [Layer] [Level] [TTI] message

Where the fields are:

- Timestamp in *YYYY-MM-DDTHH:MM:SS.UUUUUU* format at which log message was generated
- Layer can be one of *MAC/RLC/PDCP/RRC/SDAP/NGAP/GTPU/RADIO/FAPI/F1U/DU/CU/LIB*. PHY layers are specified as downlink or uplink and with executor number e.g. *DL-PHY1*.
- Level can be one of *E/W/I/D* for error, warning, info and debug respectively.
- TTI is only shown for PHY or MAC messages and is in the format *SFN.sn* where SFN is System Frame Number and sn is slot number.

An example log file excerpt can be seen below:

```
2023-03-15T18:29:25.142200 [MAC     ] [I] [  276.14] UL PDU rnti=0x4601 ue=0␣
↪subPDUs: [lcid=1: len=96, SBSR: lcg=0 bs=0, SE_PHR: total_len=3, PAD:␣
↪len=424]
2023-03-15T18:29:25.142204 [RLC     ] [I] ue=0 SRB1 UL: RX PDU. pdu_len=96␣
↪dc=data p=1 si=full sn=0 so=0
2023-03-15T18:29:25.142226 [PDCP    ] [I] ue=0 SRB1 UL: RX PDU. type=data pdu_
↪len=94 sn=0 count=0
2023-03-15T18:29:25.142228 [PDCP    ] [I] ue=0 SRB1 UL: RX SDU. count=0
2023-03-15T18:29:25.142245 [RRC     ] [D] ue=0 SRB1 - Rx DCCH UL␣
↪rrcSetupComplete (88 B)
2023-03-15T18:29:25.142249 [RRC     ] [D] Content: [
  {
    "UL-DCCH-Message": {
      "message": {
        "c1": {
```

(continues on next page)

(continued from previous page)

```
          "rrcSetupComplete": {
            "rrc-TransactionIdentifier": 0,
            "criticalExtensions": {
              "rrcSetupComplete": {
                "selectedPLMN-Identity": 1,
                "registeredAMF": {
                  "amf-Identifier": "000000100000000001000000"
                },
                "guami-Type": "native",
                "dedicatedNAS-Message":
↪"7e01820be950137e004139000bf200f110020040e7000f272e04f070f0707100307e004139000bf200f11002(
↪"
              }
            }
          }
        }
      }
    }
  }
]
2023-03-15T18:29:25.142253 [RRC     ] [D] ue=0 "RRC Setup Procedure" finished␣
↪successfully
2023-03-15T18:29:25.142263 [NGAP    ] [I] ue=0 Sending InitialUeMessage (ran_
↪ue_id=0)
```

9.2 PCAPs

The srsRAN Project gNB can output PCAPs at the following layers:

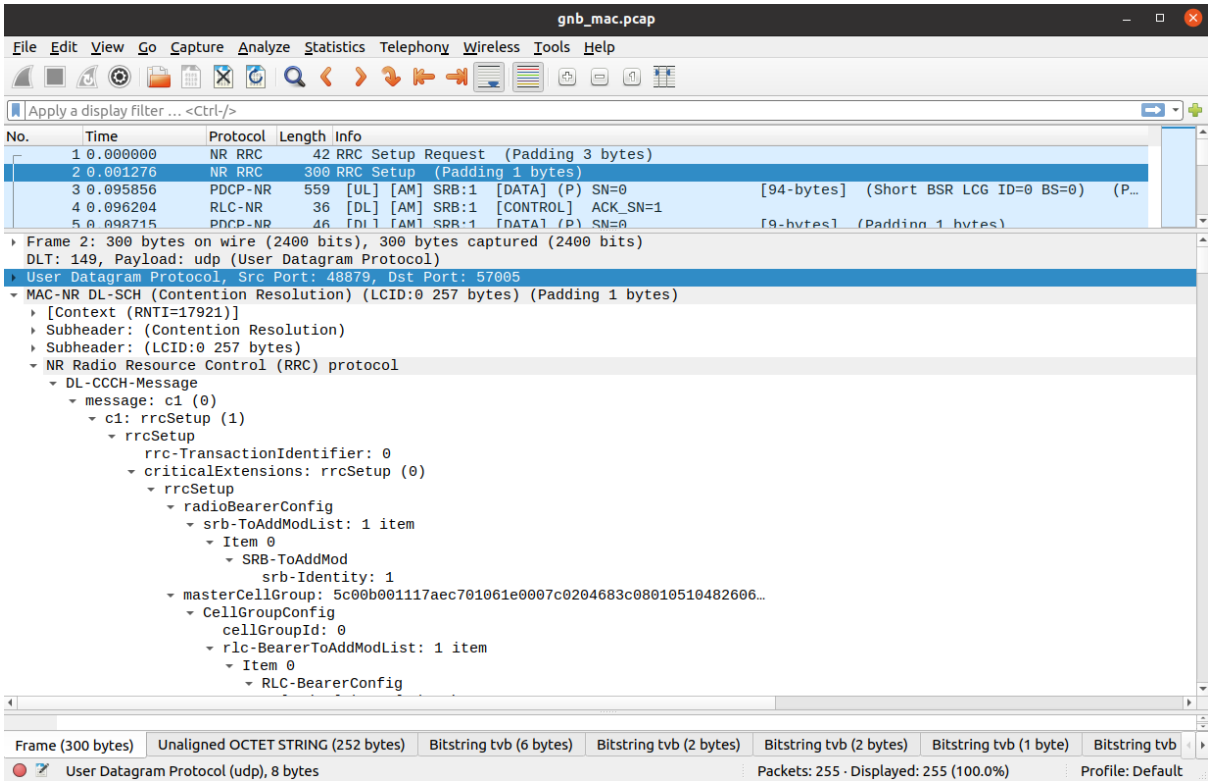
- MAC
- NGAP
- GTP-U
- E1AP
- F1AP
- E2AP

To output these PCAPs, they must first be enabled on a per-layer basis in the gNB configuration file. See the *Configuration Reference* for more details.

9.2.1 MAC

To analyze a MAC-layer PCAP using Wireshark, you will need to configure User DLT 149 for UDP and enable the mac_nr_udp protocol:

1. Go to Edit->Preferences->Protocols->DLT_USER->Edit and add an entry with DLT=149 and Payload protocol=udp.
2. Go to Analyze->Enabled Protocols->MAC-NR and enable mac_nr_udp
3. Go to Edit->Preferences->Protocols->MAC-NR: Enable both checkboxes “Attempt to…”; Set LCID->DRB mapping to “From configuration protocol”.

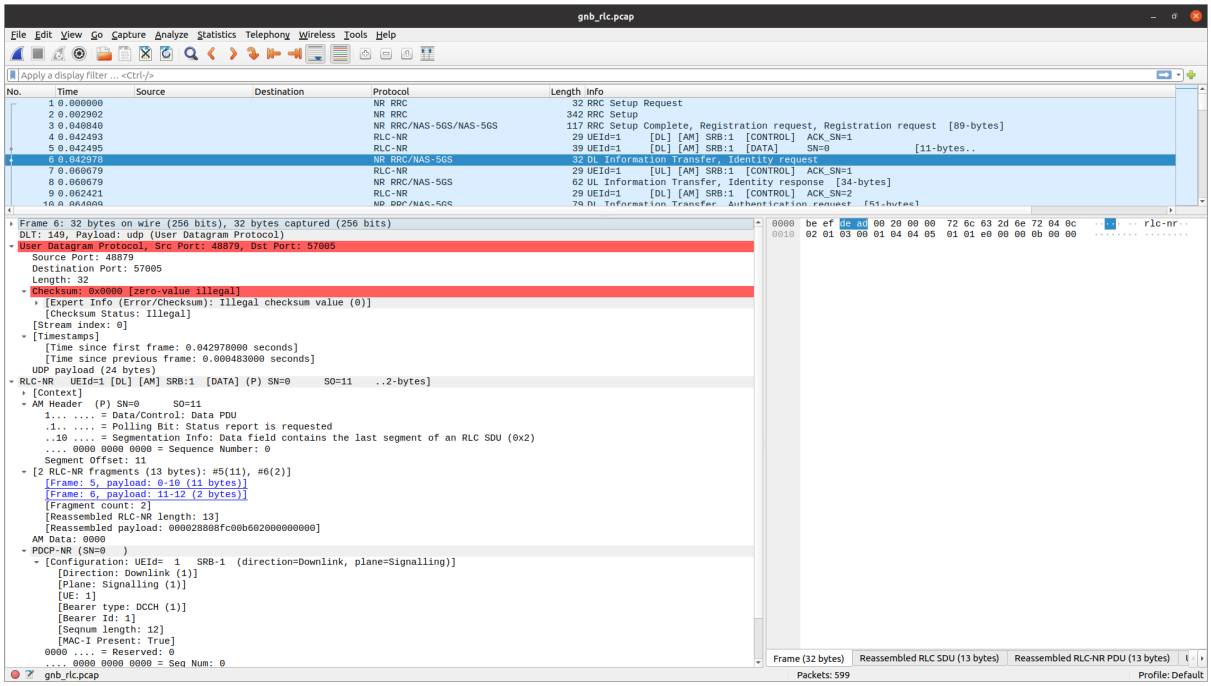


9.2.2 RLC

Note: To correctly view the RLC PCAPs you will need Wireshark v4.3.x or later.

To analyze a RLC-layer PCAP using Wireshark, you will need to configure User DLT 149 for UDP and enable the rlc_nr_udp protocol:

1. Go to Edit->Preferences->Protocols->DLT_USER->Edit and add an entry with DLT=149 and Payload protocol=udp.
2. Go to Analyze->Enabled Protocols->RLC-NR and enable rlc_nr_udp
3. Go to Edit->Preferences->Protocols->RLC-NR and configure according to your needs.

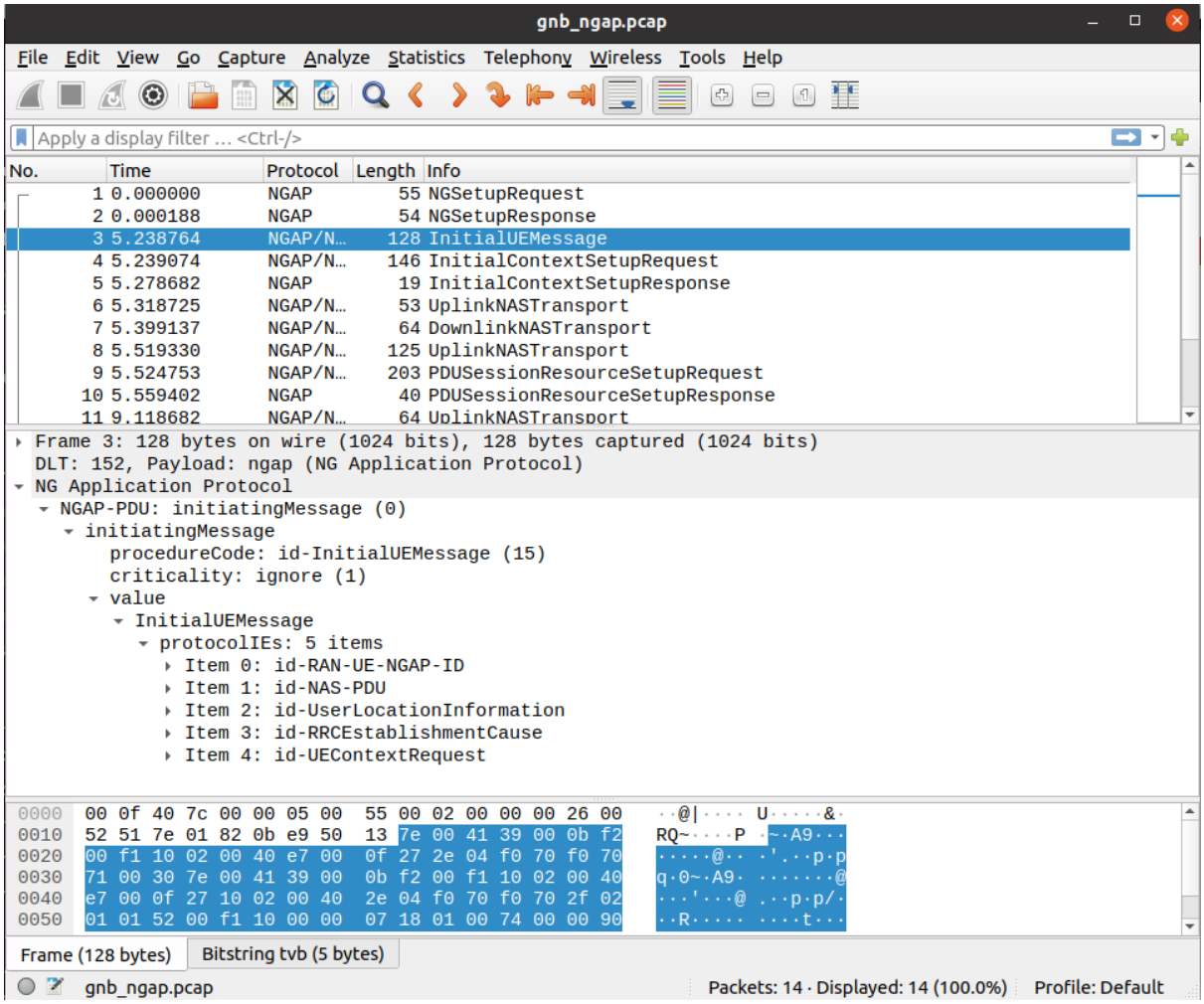


9.2.3 NGAP

To analyze an NGAP-layer PCAP using Wireshark, you will need to configure User DLT 152 for NGAP and enable detection and decoding 5G-EA0 ciphered messages:

1. Go to Edit->Preferences->Protocols->DLT_USER->Edit and add an entry with DLT=152 and Payload protocol=ngap.

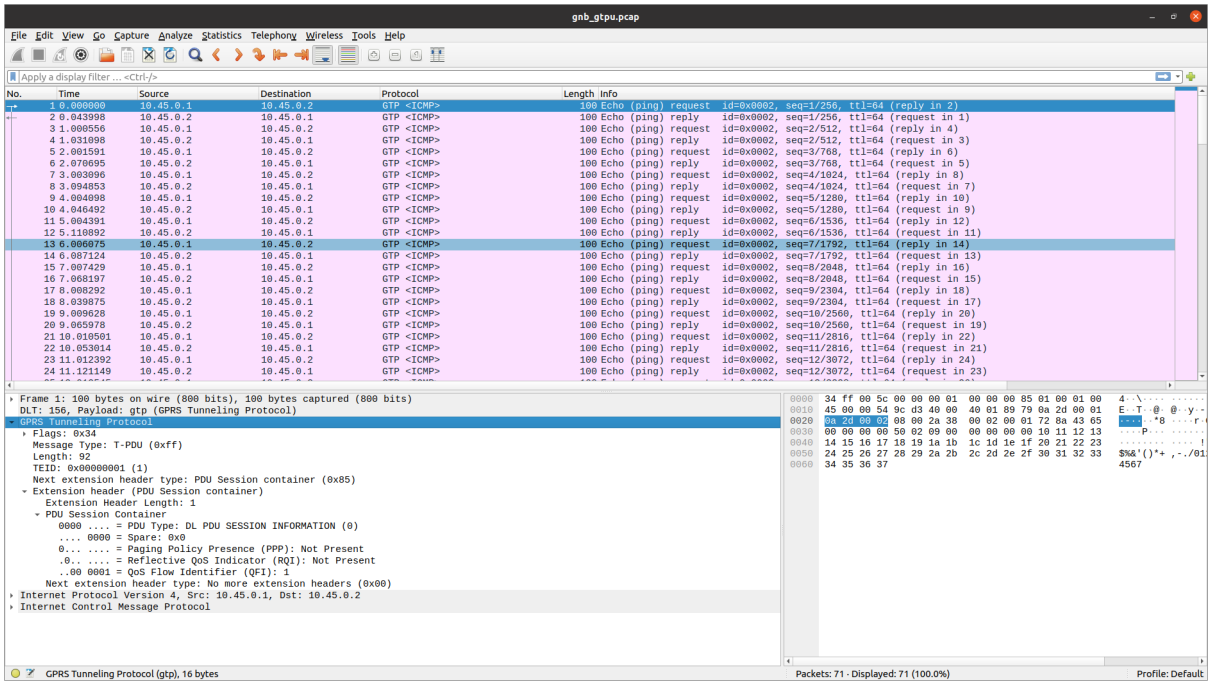
2. Go to Edit->Preferences->Protocols->NAS-5GS and enable “Try to detect and decode 5G-EA0 ciphered messages”.



9.2.4 GTP-U

To analyze a GTP-U PCAP using Wireshark, you will need to configure User DLT 156 for GTP:

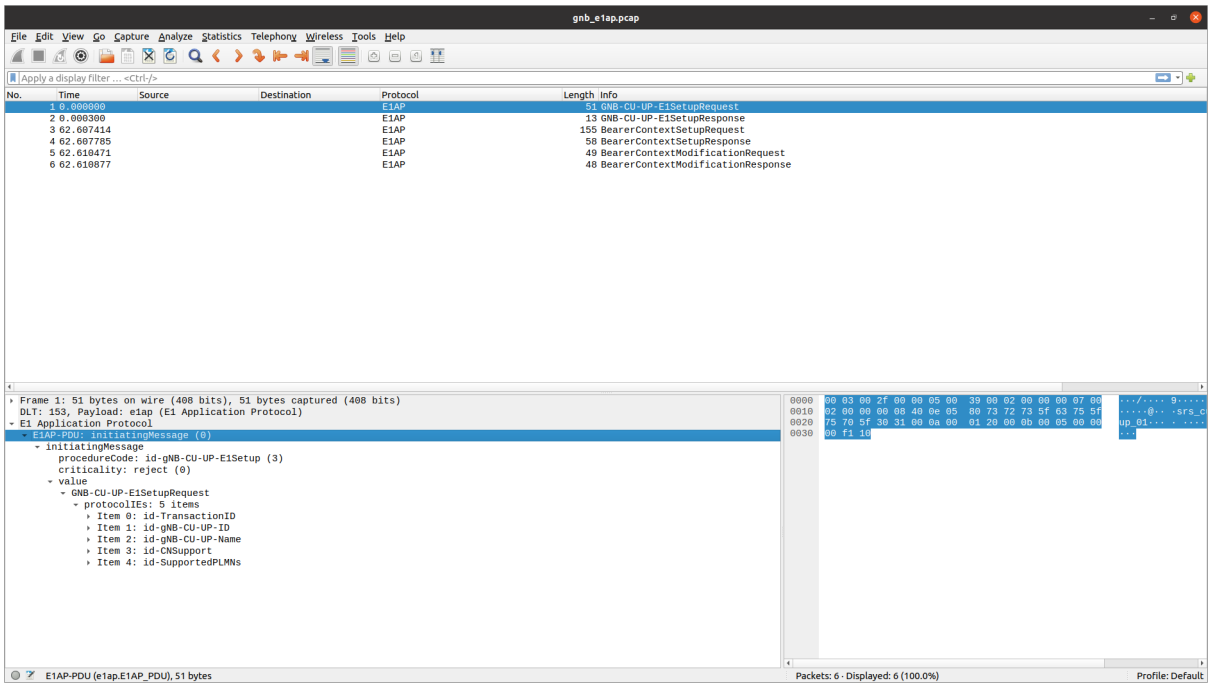
1. Go to Edit->Preferences->Protocols->DLT_USER->Edit and add an entry with DLT=156 and Payload Protocol=gtp.



9.2.5 E1AP

To analyze an E1AP PCAP using Wireshark, you will need to configure User DLT 153 for E1AP:

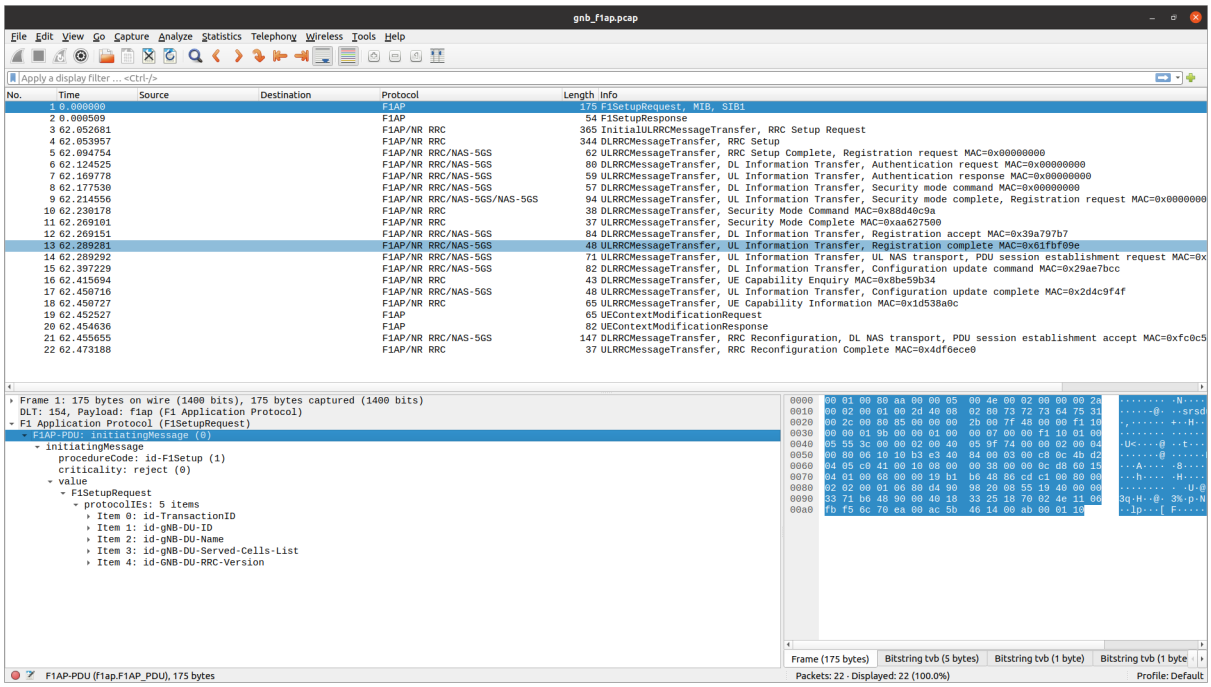
1. Go to Edit->Preferences->Protocols->DLT_USER->Edit and add an entry with DLT=153 and Payload Protocol=e1ap.



9.2.6 F1AP

To analyze an F1AP PCAP using Wireshark, you will need to configure User DLT 154 for F1AP:

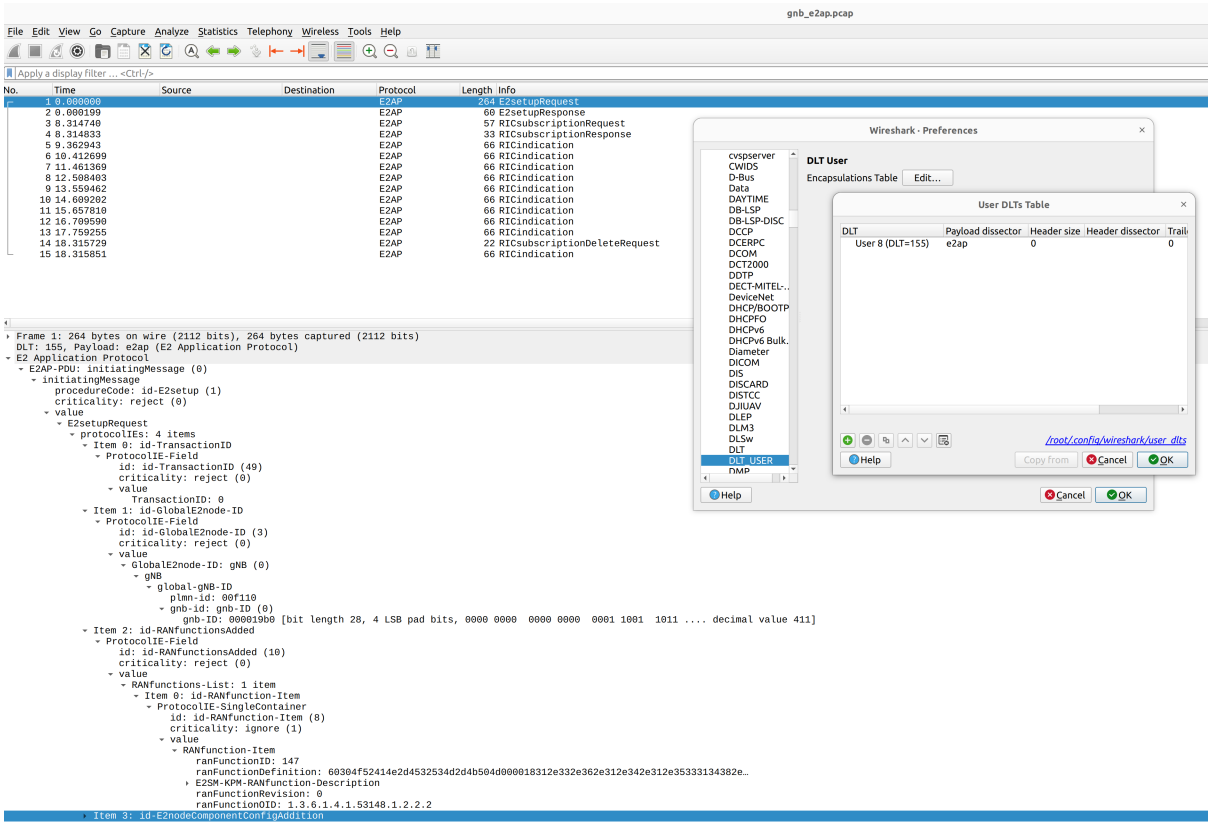
1. Go to Edit->Preferences->Protocols->DLT_USER->Edit and add an entry with DLT=154 and Payload Protocol=f1ap.



9.2.7 E2AP

To analyze an E2AP PCAP using Wireshark, you will need to configure User DLT 155 for E2AP:

1. Go to Edit->Preferences->Protocols->DLT_USER->Edit and add an entry with DLT=155 and Payload Protocol=e2ap.


gnb_e2ap.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocol Length Info
1 0.000000 E2AP 264 E2setupRequest
2 0.000199 E2AP 60 E2setupResponse
3 8.314740 E2AP 57 RICsubscriptionRequest
4 8.314833 E2AP 33 RICsubscriptionResponse
5 9.362943 E2AP 66 RICindication
6 10.412699 E2AP 66 RICindication
7 11.461369 E2AP 66 RICindication
8 12.508403 E2AP 66 RICindication
9 13.559462 E2AP 66 RICindication
10 14.609202 E2AP 66 RICindication
11 15.657810 E2AP 66 RICindication
12 16.709590 E2AP 66 RICindication
13 17.759255 E2AP 66 RICindication
14 18.315729 E2AP 22 RICsubscriptionDeleteRequest
15 18.315851 E2AP 66 RICindication
Frame 1: 264 bytes on wire (2112 bits), 264 bytes captured (2112 bits)
DLT: 155, Payload: e2ap (E2 Application Protocol)
E2 Application Protocol
E2AP-PDU: initiatingMessage (0)
initiatingMessage
procedureCode: id-E2setup (1)
criticality: reject (0)
value
E2setupRequest
protocolIEs: 4 items
Item 0: id-TransactionID
ProtocolIE-Field
id: id-TransactionID (49)
criticality: reject (0)
value
TransactionID: 0
Item 1: id-GlobalE2node-ID
ProtocolIE-Field
id: id-GlobalE2node-ID (3)
criticality: reject (0)
value
GlobalE2node-ID: gNB (0)
gNB
global-gNB-ID
plmn-id: 00f110
gnb-id: gnb-ID (0)
gnb-ID: 000019b0 [bit length 28, 4 LSB pad bits, 0000 0000 0000 0000 0001 1001 1011 .... decimal value 411]
Item 2: id-RANfunctionsAdded
ProtocolIE-Field
id: id-RANfunctionsAdded (10)
criticality: reject (0)
value
RANfunctions-List: 1 item
Item 0: id-RANfunction-Item
ProtocolIE-SingleContainer
id: id-RANfunction-Item (8)
criticality: ignore (1)
value
RANfunction-Item
ranFunctionID: 147
ranFunctionDefinition: 60304f52414e2d4532534d2d4b504d000018312e332e362e312e342e312e35333134382e…
E2SM-KPM-RANfunction-Description
ranFunctionRevision: 0
ranFunctionOID: 1.3.6.1.4.1.53148.1.2.2.2
Item 3: id-E2nodeComponentConfigAddition
Wireshark · Preferences
cvspserver
CWIDS
D-Bus
Data
DAYTIME
DB-LSP
DB-LSP-DISC
DCCP
DCERPC
DCOM
DCT2000
DDTP
DECT-MITEL-..
DeviceNet
DHCP/BOOTP
DHCPFO
DHCPv6
DHCPv6 Bulk..
Diameter
DICOM
DIS
DISCARD
DISTCC
DJIUAV
DLEP
DLM3
DLSw
DLT
DLT_USER
DMP
Help
DLT User
Encapsulations Table
Edit...
User DLTs Table
DLT
Payload dissector
Header size
Header dissector
Trail
User 8 (DLT=155)
e2ap
0
0
/root/config/wireshark/user_dlts
Help
Copy from
Cancel
OK
Cancel
OK

CHAPTER

TEN

CONFIGURATION REFERENCE

The srsRAN Project gNB application uses a YAML (.yml) configuration file.

The gNB comes with a number of example configuration files, these can be found in `srsRAN_Project/configs/` in the source code:

- B200 USRP @ 20 MHz in band 78 (TDD)
- N310 USRP @ 20 MHz in band 3 (FDD)

These configuration file examples provide a basic set-up to get users up and running, users can easily modify these to suit their use-case.

More example configuration files for various use cases can be found here. These include sample supplementary configuration files for:

- MIMO with a USRP
- Increasing QAM from 64 to 256
- Specific QOS configurations for voice, video, IMS, live-streaming and buffered video streaming
- Slicing support

These are intended to be used in conjunction with other configuration files, for example you might run the MIMO example as follows:

```
sudo ./gnb -c gnb_rf_n310_fdd_n3_20mhz.yml -c mimo.yml
```

Furthermore you may wish to use QAM256 in addition to the above. This can be done as follows:

```
sudo ./gnb -c gnb_rf_n310_fdd_n3_20mhz.yml -c mimo.yml -c qam256.yml
```

10.1 Format

All configuration parameters are presented here below in the following format:

parameter

- Optional/Required *TYPE* (*default*). *Description*. Format: *format description*. Supported: *supported values*.

10.2 Configuration Parameters

```
gnb_id: 411                             # Optional UINT (411). Sets the numerical␣
↪ID associated with the gNB.

gnb_id_bit_length:                      # Optional UNIT. Sets the bit length of␣
↪the gnb_id above. Format: integer between [22 - 32]

ran_node_name: srsgnb01                 # Optional TEXT (srsgnb01). Sets the text␣
↪ID associated with the gNB. Format: string without spaces.

cells:                                  # Optional TEXT. Sets the cell␣
↪configuration on a per cell basis, overwriting the default configuration␣
↪defined by
                                        # cell_cfg. Contains a list of cells,␣
↪each with parameters that overwrite the default config. This can only be
                                        # set via the configuration file. For␣
↪more information see the relevant example configuration file:
                                        # gnb_custom_cell_properties.yml. Each␣
↪list entry should begin with "-".

qos:                                    # Optional TEXT. Configures RLC and PDCP␣
↪radio bearers on a per 5QI basis. This can only be set via the␣
↪configuration file. Each list entry should begin with "-".

srbs:                                   # Optional TEXT. Configures signaling␣
↪radio bearers. Each list entry should begin with "-".

slicing:                                # Optional TEXT. Configure network␣
↪slicing options. This can only be set via the configuration file.
  -                                         # Configure Slice 1
    sst: 1                                      # Optional UINT (1). Sets the␣
↪Slice Service Type. Supported: [0 - 255].
    sd: 0                                       # Optional UINT (0). Sets the␣
↪Service Differentiator. Supported: [0-16777215].

amf:
  addr:                                 # Required TEXT. Sets the IP address␣
↪or hostname of the AMF. Format: IPV4 or IPV6 IP address.
  port: 38412                           # Optional UINT (38412). Sets the AMF␣
↪port. Format: integer between [20000 - 40000].
  bind_addr:                            # Required TEXT. Sets the local IP␣
↪address that the gNB binds to for receiving traffic from the AMF. Format:␣
↪IPV4 or IPV6 IP address.
  n2_bind_addr:                         # Optional TEXT. Sets local IP␣
↪address to bind for N2 interface. Format: IPV4 or IPV6 IP address.
  n3_bind_addr:                         # Optional TEXT. Sets local IP␣
↪address to bind for N3 interface. Format: IPV4 or IPV6 IP address.
  sctp_rto_initial:                     # Optional INT. Sets the initial␣
↪retransmission timeout when creating the SCTP connection.
```

(continues on next page)

(continued from previous page)

```
  sctp_rto_min:                         # Optional INT. Sets the minimum␣
↪retransmission timeout for the SCTP connection.
  sctp_rto_max:                         # Optional INT. Sets the maximum␣
↪retransmission timeout for the SCTP connection.
  sctp_initial_max_attempts:            # Optional INT. Sets the maximum␣
↪retransmission attempts for the initial SCTP connection.
  sctp_max_init_timeo:                  # Optional INT. Sets the maximum␣
↪retransmission timeout for the initial SCTP connection.
  udp_max_rx_msgs:                      # Optional INT. Sets the maximum␣
↪amount of messages RX in a single syscall.
  no_core: 0                            # Optional BOOLEAN (0). Setting to␣
↪true allows the gNB to run without a core. Supported: [0, 1].

e2:
  enable_du_e2: 0                       # Optional BOOLEAN (0). Enables the␣
↪DU E2 agent. Supported: [0, 1].
  addr:                                 # Optional TEXT. Sets the RIC IP␣
↪address.
  port: 36421                           # Optional UINT (36421). Sets the RIC␣
↪Port. Supported: [20000 - 40000].
  bind_addr:                            # Optional TEXT. Sets the local IP␣
↪address to bind for RIC connection. Supported: IPv4 address.
  sctp_rto_initial:                     # Optional INT. Sets the initial␣
↪retransmission timeout when creating the SCTP connection.
  sctp_rto_min:                         # Optional INT. Sets the minimum␣
↪retransmission timeout for the SCTP connection.
  sctp_rto_max:                         # Optional INT. Sets the maximum␣
↪retransmission timeout for the SCTP connection.
  sctp_initial_max_attempts:            # Optional INT. Sets the maximum␣
↪retransmission attempts for the initial SCTP connection.
  sctp_max_init_timeo:                  # Optional INT. Sets the maximum␣
↪retransmission timeout for the initial SCTP connection.
  e2sm_kpm_enabled: 0                   # Optional BOOLEAN (0). Enabled the␣
↪E2SM KPM service module. Supported: [0, 1].
  e2sm_rc_enabled: 0                    # Optional BOOLEAN (0). Enabled the␣
↪E2SM RC service module. Supported: [0, 1].

cu_cp:
  max_nof_dus:                          # Optional UINT. Maximum number of DU␣
↪connections that the CU-CP may accept.
  max_nof_cu_ups:                       # Optional UINT. Maximum number of CU-
↪UP connections that the CU-CP may accept.
  inactivity_timer: 7200                # Optional INT (7200). Sets the UE/
↪PDU Session/DRB inactivity timer in seconds. Supported: [1 - 7200].
  ue_context_setup_timeout_s: 3         # Optional INT (3). Sets the timeout␣
↪for the reception of an InitialContextSetupRequest after an␣
↪InitialUeMessage was sent to the core, in seconds. If the value is reached,␣
↪the UE will be released.
```

(continues on next page)

(continued from previous page)

```
  mobility:
    trigger_handover_from_measurements: 0    # Optional BOOLEAN (0). Sets␣
↪whether or not to start HO if neighbor cells become stronger. Supported: [0,
↪ 1].
    cells:                                   # Optional TEXT. Sets the list␣
↪of cells known to the CU-CP, their configs (if not provided over F1) and␣
↪their respective neighbor cells.
      -                                        # Define Cell 1
        nr_cell_id:                            # Required UINT. The ID of␣
↪this serving cell.
        periodic_report_cfg_id:                # Optional UINT. The␣
↪periodical report configuration to use for this serving cell.
        ncells:                                # Required TEXT. List of␣
↪neighbor cells.
          nr_cell_id:                              # Required UINT. The ID␣
↪of this neighbor cell.
          report_configs:                          # Required TEXT. List␣
↪of report configurations to use for measurements of this neighbor cell.
        gnb_id:                                # Optional UINT. The ID of␣
↪this gNB.
        ssb_arfcn:                             # Optional UINT. The SSB␣
↪ARFCN of this serving cell. Must be present if not provided over F1.
        band:                                  # Optional UINT. The NR␣
↪band of this serving cell cell. Must be present if not provided over F1.
        ssb_scs:                               # Optional UINT. The SSB␣
↪subcarrier spacing of this serving cell in KHz. Must be present if not␣
↪provided over F1.
        ssb_period:                            # Optional UINT. The SSB␣
↪period of this serving cell in ms.  Must be present if not provided over F1.
↪
        ssb_offset:                            # Optional UINT. The SSB␣
↪offset of this serving cell. Must be present if not provided over F1.
        ssb_duration:                          # Optional UINT. The SSB␣
↪duration of this serving cell in subframes. Must be present if not provided␣
↪over F1.
    report_configs:                          # Optional TEXT. Sets the list␣
↪of report configurations to dynamically build a measurement configuration␣
↪sent to the UEs using the below values.
      -                                        # Define report configs for␣
↪cell 1
        report_cfg_id:                         # Required UINT. The ID of␣
↪this report configuration.
        report_type:                           # Required TEXT. The type␣
↪of the report. Supported: [event_triggered, periodical]. Note that␣
↪periodical reports are only supported for serving cells.
        report_interval_ms: 1024               # Optional UINT (1024). The␣
↪report interval in ms.
        a3_report_type:                        # Optional TEXT. A3 report␣
↪type. Supported: [rsrp, rsrq, sinr].
```

(continues on next page)

(continued from previous page)

```
        a3_offset_db:                           # Optional UINT. A3 offset␣
↪in dB used for measurement report trigger.
        a3_hysteresis_db:                       # Optional UINT. A3␣
↪hysteresis in dB used for measurement report trigger.
        a3_time_to_trigger_ms:                  # Optional UINT. Time in ms␣
↪during which A3 condition must be met before measurement report trigger.

  rrc:
    force_reestablishment_fallback: 0       # Optional BOOLEAN (0). Force␣
↪RRC re-establishment fallback to RRC setup. Supported: [0, 1].
    rrc_procedure_timeout_ms: 720           # Optional UINT (720). Sets the␣
↪timeout in ms used for RRC message exchange with UE. It needs to suit the␣
↪expected communication delay and account for potential retransmissions UE␣
↪processing delays, SR delays, etc. Supported: [0 - inf].

  security:
    integrity: not_needed                   # Optional TEXT (not_needed).␣
↪Sets the default integrity protection indication for DRBs.
    confidentiality: required               # Optional TEXT (required). Sets␣
↪the default integrity confidentiality indication for DRBs.
    nea_pref_list: nea0,nea2,nea1           # Optional TEXT (nea0,nea2,nea1).
↪ Ordered preference list for the selection of encryption algorithm (NEA).␣
↪Supported: [nea0,nea2,nea1,nea3].
    nia_pref_list: nia2,nia1                # Optional TEXT (nia2,nia1).␣
↪Ordered preference list for the selection of encryption algorithm (NIA).␣
↪Supported: [nia2,nia1,nia3].

cu_up:
  gtpu_queue_size: 2048                 # Optional INT (2048). Sets the GTP-U␣
↪queue size, in PDUs.
  gtpu_reordering_timer: 0              # Optional INT (0). Sets the GTP-U RX␣
↪reordering timer (in milliseconds).
  warn_on_drop: 0                       # Optional BOOLEAN (0). Enables the␣
↪option to log a warning for dropped packets in GTP-U and PDCP due to full␣
↪queues. Supported: [0, 1].

ntn:
  cell_specific_koffset: 0              # Optional INT (0). Sets the cell-
↪specific k-offset to be used for NTN. Supported: [0 - 1023].
  ta_common:                            # Optional UINT. Sets the TA common␣
↪offset.

  ephemeris_info_ecef:
    pos_x: 0                              # Optional INT (0). Sets the X␣
↪position of the satellite. Supported: [-67108864 - 67108863].
    pos_y: 0                              # Optional INT (0). Sets the Y␣
↪position of the satellite. Supported: [-67108864 - 67108863].
    pos_z: 0                              # Optional INT (0). Sets the Z␣
↪position of the satellite. Supported: [-67108864 - 67108863].
```

(continues on next page)

(continued from previous page)

```
    vel_x: 0                                # Optional INT (0). Sets the X␣
↪velocity of the satellite. Supported: [-32768 - 32767].
    vel_y: 0                                # Optional INT (0). Sets the Y␣
↪velocity of the satellite. Supported: [-32768 - 32767].
    vel_z: 0                                # Optional INT (0). Sets the Z␣
↪velocity of the satellite. Supported: [-32768 - 32767].
  ephemeris_orbital:
    semi_major_axis: 0                      # Optional FLOAT (0). Sets the semi-
↪major axis of the satellite. Supported: [0 - 10000000000].
    eccentricity: 0                         # Optional FLOAT (0). Sets the␣
↪eccentricity of the satellite.
    periapsis: 0                            # Optional FLOAT (0). Sets the␣
↪periapsis of the satellite.
    longitude: 0                            # Optional FLOAT (0). Sets the␣
↪longitude of the satellites angle of ascending node.
    inclination: 0                          # Optional FLOAT (0). Sets the␣
↪inclination of the satellite.
    mean_anomaly: 0                         # Optional FLOAT (0). Sets the mean␣
↪anomaly of the satellite.

ru_ofh:
# Many of the following values are optional as they have default values. In␣
↪practice, all of the following parameters should be defined by the user, as␣
↪they will need
# to be configured specifically for the RU being used. Failing to configure␣
↪this parameters correctly may result in the RU failing to connect correctly␣
↪to the DU.
  gps_alpha: 0                          # Optional FLOAT (0). Sets the GPS␣
↪alpha. Supported: [0 - 1.2288e+07].
  gps_beta: 0                           # Optional INT (0). Sets the GPS beta.
↪ Supported: [-32768 - +32767].
  ru_bandwidth_MHz: 0                   # Required UINT (0). Sets the channel␣
↪bandwidth in MHz. Supported: [5,10,15,20,25,30,40,50,60,70,80,90,100].
  t1a_max_cp_dl: 500                    # Optional INT (500). Sets T1a␣
↪maximum value for downlink control-plane. Supported: [0 - 1960].
  t1a_min_cp_dl: 258                    # Optional INT (258). Sets T1a␣
↪minimum value for downlink control-plane. Supported: [0 - 1960].
  t1a_max_cp_ul: 500                    # Optional INT (500). Sets T1a␣
↪maximum value for uplink control-plane. Supported: [0 - 1960].
  t1a_min_cp_ul: 258                    # Optional INT (258). Sets T1a␣
↪minimum value for uplink control-plane. Supported: [0 - 1960].
  t1a_max_up: 300                       # Optional INT (300).Sets T1a maximum␣
↪value for uer-plane. Supported: [0 - 1960].
  t1a_min_up: 85                        # Optional INT (85). Sets T1a minimum␣
↪value for user-plane. Supported: [0 - 1960].
  ta4_max: 300                          # Optional UINT (300). Sets the Ta4␣
↪maximum value for User-Plane. Supported: [0 - 1960].
  ta4_min: 85                           # Optional UINT (85). Sets the Ta4␣
↪maximum value for User-Plane. Supported: [0 - 1960].
```

(continues on next page)

(continued from previous page)

```
  is_prach_cp_enabled: 0                # Optional BOOLEAN (0). Sets PRACH␣
↪control-plane enabled flag. Supported: [0, 1].
  is_dl_broadcast_enabled: 0            # Optional BOOLEAN (0). Sets downlink␣
↪broadcast enabled flag. Supported: [0, 1].
  ignore_ecpri_seq_id: 0                # Optional BOOLEAN (0). Sets whether␣
↪or not to ignore eCPRI sequence ID field value. Supported [0, 1].
  ignore_ecpri_payload_size: 0          # Optional BOOLEAN (0). Sets whether␣
↪or not to ignore eCPRI payload size field value. Supported [0, 1].
  warn_unreceived_ru_frames: 1          # Optional BOOLEAN (1). Sets whether␣
↪or not to send a warning when there are unreceived Radio Unit frames.␣
↪Supported [0, 1].
  compr_method_ul: bfp                  # Optional TEXT (bfp). Sets the␣
↪uplink compression method. Supported: [none, bfp, bfp selective, block␣
↪scaling, mu law, modulation, modulation selective].
  compr_bitwidth_ul: 9                  # Optional UINT (9). Sets the uplink␣
↪compression bit width. Supported: [1 - 16].
  compr_method_dl: bfp                  # Optional TEXT (bfp). Sets the␣
↪downlink compression method. Supported: [none, bfp, bfp selective, block␣
↪scaling, mu law, modulation, modulation selective].
  compr_bitwidth_dl: 9                  # Optional UINT (9). Sets the␣
↪downlink compression bit width. Supported: [1 - 16].
  compr_method_prach: none              # Optional TEXT (none). Sets the␣
↪PRACH compression method. Supported: [none, bfp, bfp selective, block␣
↪scaling, mu law, modulation, modulation selective].
  compr_bitwidth_prach: 16              # Optional UINT (16). Sets the PRACH␣
↪compression bit width. Supported [1 - 16].
  enable_ul_static_compr_hdr: 1         # Optional BOOLEAN (1). Uplink static␣
↪compression header enabled flag. Supported: [0 . 1].
  enable_dl_static_compr_hdr: 1         # Optional BOOLEAN (1). Downlink␣
↪static compression header enabled flag. Supported: [0 . 1].
  iq_scaling: 0.35                      # Optional FLOAT (0.35). Sets the IQ␣
↪scaling factor. Supported: [0 - 1].
  cells:                                # Optional TEXT. Sets the hardware␣
↪specific cell configuration on a per cell basis.
    -                                       # Cell 1
      network_interface: enp1s0f0           # Optional TEXT (enp1s0f0).␣
↪Sets the ethernet network interface name for the RU or PCIe identifier when␣
↪using DPDK. Format: a string, e.g. [interface_name].
      ru_mac_address: 70:b3:d5:e1:5b:06     # Optional TEXT␣
↪(70:b3:d5:e1:5b:06). Sets the RU MAC address. Format: a string, e.g.␣
↪[AA:BB:CC:DD:11:22:33].
      du_mac_address: 00:11:22:33:00:77     # Optional TEXT␣
↪(00:11:22:33:00:77). Sets the DU MAC address. Format: a string, e.g.␣
↪[AA:BB:CC:DD:11:22:33].
      vlan_tag: 1                           # Optional UINT (1). Sets the V-
↪LAN tag control information field. Supported: [1 - 4094].
      ru_prach_port_id: [4]                 # Optional UINT (4). Sets the␣
↪RU PRACH eAxC port ID. Supported: [0 - 65535].
      ru_dl_port_id: [0, 1]                 # Optional UINT (0, 1). Sets␣
```

(continues on next page)

(continued from previous page)

```
↪the RU downlink eAxC port ID. Format: vector containing all DL eXaC ports,␣
↪e.g. [0, ...\ , N].
      ru_ul_port_id: [0]                    # Optional UINT (0). Sets the␣
↪RU uplink eAxC port ID. Supported: [0 - 65535].

ru_sdr:
  srate: 61.44                            # Required FLOAT (61.44). Sets the␣
↪sampling rate of the RF-frontend in MHz.
  device_driver: uhd                      # Required TEXT (uhd). RF device␣
↪driver name. Supported: [uhd, zmq].
  device_args:                            # Optional TEXT. An argument that␣
↪gets passed to the selected RF driver.
  tx_gain: 50                             # Required FLOAT (50). Sets the␣
↪transmit gain in dB. Supported: [0 - max value supported by radio].
  rx_gain: 60                             # Required FLOAT (60). Sets the␣
↪receive gain in dB. Supported: [0 - max value supported by radio].
  freq_offset: 0                          # Optional FLOAT (0). Sets the␣
↪frequency offset in Hertz.
  clock_ppm: 0                            # Optional FLOAT (0). Sets the clock␣
↪calibration in Parts Per Million (PPM).
  lo_offset: 0                            # Optional FLOAT (0). Shifts the␣
↪local oscillator frequency in MHz away from the center frequency to move LO␣
↪leakage out of the channel.
  clock: default                          # Optional TEXT (default). Specify␣
↪the RF device source for timestamping. Supported: [default, internal,␣
↪external, gpsdo].
  sync: default                           # Optional TEXT (default). Specify␣
↪the RF device oscillator reference synchronization source. Supported:␣
↪[default, internal, external, gpsdo].
  otw_format: default                     # Optional TEXT (default). Specific␣
↪the over-the-wire format. Supported: [default, sc8, sc12, sc16].
  time_alignment_calibration: auto        # Optional TEXT (auto). Compensates␣
↪for any reception and transmission time misalignment inherent to the RF␣
↪device. Positive values reduce the RF transmission delay with respect to␣
↪the RF reception. Negative values have the opposite effect.

  amplitude control:
    tx_gain_backoff: 12.0                 # Optional FLOAT (12.0). Sets␣
↪baseband gain back-off in dB. This accounts for the signal PAPR and is␣
↪applied regardless of clipping settings. Format: positive float.
    enable_clipping: false                # Optional BOOL (false). Sets␣
↪clipping of the baseband samples on or off. If enabled, samples that exceed␣
↪the power ceiling are clipped.
    ceiling: -0.1                         # Optional FLOAT (-0.1). Sets␣
↪the power ceiling in dB, relative to the full scale amplitude of the radio.␣
↪Format: negative float or 0.

  expert_cfg:
    low_phy_dl_throttling: 0              # Optional FLOAT (0). Throttles␣
```

(continues on next page)

(continued from previous page)

```
↪the lower PHY DL baseband generation. Setting to 0 disables throttling.␣
↪Supported: any value in the range [0 - 1].
    discontinuous_tx: 0                    # Optional Boolean (0). Enables␣
↪discontinuous transmission mode for the radio front-ends supporting it.␣
↪Supported: [0, 1].
    power_ramping_time_us: 0               # Optional FLOAT (0). Specifies␣
↪the power ramping time in microseconds, it proactively initiates the␣
↪transmission and mitigates transient effects. It is recommended to␣
↪configure this parameter carefully, taking into account the characteristics␣
↪of the transmit chain in order to achieve optimal performance

ru_dummy:
  dl_processing_delay: 1               # Optional UINT (1). Sets the␣
↪downlink processing delay in number of slots.

cell_cfg:
  pci: 1                               # Required UINT (1). Sets the␣
↪Physical Cell ID. Supported: [0-1007].
  dl_arfcn: 536020                     # Required UINT (536020). Sets the␣
↪Downlink ARFCN.
  band: auto                           # Optional TEXT (auto). Sets the NR␣
↪band being used for the cell. If not specified, will be set automatically␣
↪based on ARFCN. Supported: all release 17 bands.
  common_scs: 15                       # Required UINT (15). Sets the␣
↪subcarrier spacing in KHz to be used by the cell. Supported: [15, 30].
  channel_bandwidth_MHz: 20            # Required UINT (20). Sets the␣
↪channel Bandwidth in MHz, the number of PRBs will be derived from this.␣
↪Supported: [5, 10, 15, 20, 25, 30, 40, 50, 60, 70, 80, 90, 100].
  nof_antennas_dl: 1                   # Optional UINT (1). Sets the number␣
↪of antennas for downlink transmission. Supported: [1, 2, 4].
  nof_antennas_ul: 1                   # Optional UINT (1). Sets the number␣
↪of antennas for uplink transmission. Supported: [1, 2, 4].
  plmn: 00101                          # Required TEXT (00101). Sets the␣
↪Public Land Mobile Network code. Format: 7-digit PLMN code containing MCC &␣
↪MNC.
  tac: 7                               # Required UINT (7). Sets the␣
↪Tracking Area Code.
  q_rx_lev_min: -70                    # Optional INT (-70). Sets the␣
↪required minimum received RSRP level for cell selection/re-selection, in␣
↪dBm. Supported: [-70 - -22].
  q_qual_min: -20                      # Optional INT (-20). Sets the␣
↪required minimum received RSRQ level for cell selection/re-selection, in dB.
↪ Supported: [-43 - -12].
  pcg_p_nr_fr1: 10                     # Optional INT (10). Sets the maximum␣
↪total TX power to be used by the UE in this NR cell group across in FR1.␣
↪Supported: [-30 - +23].

  mac_cell_group:
    bsr_cfg:                           # Buffer status report␣
```

(continues on next page)

(continued from previous page)

```
↪configuration parameters
      periodic_bsr_timer: 10                    # Optional UINT (10). Sets␣
↪the periodic Buffer Status Report Timer value in nof. subframes. Value 0␣
↪equates to infinity. Supported: [1, 5, 10, 16, 20, 32, 40, 64, 80, 128, 160,
↪ 320, 640, 1280, 2560, 0].
      retx_bsr_timer: 80                        # Optional UINT (80). sets␣
↪the retransmission Buffer Status Report Timer value in nof. subframes.␣
↪Supported: [10, 20, 40, 80, 160, 320, 640, 1280, 2560, 5120, 10240].
      lc_sr_delay_timer:                        # Optional TEXT. Sets the␣
↪logical Channel SR delay timer in nof. subframes. Supported: [10, 20, 40,␣
↪80, 160, 320, 640, 1280, 2560, 5120, 10240].
    phr_cfg:                                    # Power Headroom report␣
↪configuration parameters
      phr_prohibit_timer: 10                    # Optional UINT (10). Sets␣
↪the PHR prohibit timer in nof. subframes. Supported: [0, 10, 20, 50, 100,␣
↪200, 500, 1000].
    sr_cfg:                                     # Scheduling Request␣
↪configuration parameters
      sr_trans_max: 64                          # Optional UINT (64). Sets␣
↪the maximum number of SR transmissions. Supported: [4, 8, 16, 32, 64].
      sr_prohibit_timer:                        # Optional TEXT. Sets the␣
↪timer for SR transmission on PUCCH in ms. Supported: [1, 2, 4, 8, 16, 32,␣
↪64].

  ssb:
    ssb_period: 10                              # Optional UINT (10). Sets the␣
↪period of SSB scheduling in milliseconds. Supported: [5, 10, 20].
    ssb_block_power_dbm: -16                    # Optional INT (-16). Sets the SS␣
↪PBCH block power in dBm. Supported: [-60 - +50].
    pss_to_sss_epre_db: 0                       # Optional UINT (0). Sets the␣
↪Synchronization Signal Block Primary Synchronization Signal to Secondary␣
↪Synchronization Signal Energy Per Resource Element ratio in dB. Supported:␣
↪[0, 3].

  sib:
    si_window_length: 160                       # Optional UINT (160). Sets the␣
↪length of the SI scheduling window, in slots. It must be always shorter or␣
↪equal to the period of the SI message. Supported: [5,10,20,40,80,160,320,
↪640,1280].
    si_sched_info:                              # Optional TEXT. Configures the␣
↪scheduling for each of the SI-messages broadcast by the gNB.
    t300: 1000                                  # Optional UINT (1000). Sets the␣
↪RRC Connection Establishment timer in ms. The timer starts upon␣
↪transmission of RRCSetupRequest. Supported: [100,200,300,400,600,1000,1500,
↪2000].
    t301: 1000                                  # Optional UINT (1000). Sets the␣
↪RRC Connection Re-establishment timer in ms. The timer starts upon␣
↪transmission of RRCReestablishmentRequest. Supported: [100,200,300,400,600,
↪1000,1500,2000].
```

(continues on next page)

(continued from previous page)

```
    t310: 1000                                  # Optional UINT (1000). Sets the␣
↪Out-of-sync timer in ms. The timer starts upon detecting physical layer␣
↪problems for the SpCell i.e. upon receiving N310 consecutive out-of-sync␣
↪indications from lower layers. Supported: [0,50,100,200,500,1000,2000].
    n310: 1                                     # Optional UINT (1). Sets the Out-
↪of-sync counter. The counter is increased upon reception of "out-of-sync"␣
↪from lower layer while the timer T310 is stopped. Starts the timer T310,␣
↪when configured value is reached. Supported: [1,2,3,4,6,8,10,20].
    t311: 3000                                  # Optional UINT (3000). Sets the␣
↪RRC Connection Re-establishment procedure timer in ms. The timer starts␣
↪upon initiating the RRC connection re-establishment procedure. Supported:␣
↪[1000,3000,5000,10000,15000,20000,30000].
    n311: 1                                     # Optional UINT (1). Sets the In-
↪sync counter. The counter is increased upon reception of the "in-sync" from␣
↪lower layer while the timer T310 is running. Stops the timer T310, when␣
↪configured value is reached.Supported: [1,2,3,4,5,6,8,10].
    t319: 1000                                  # Optional UINT (1000). Sets the␣
↪RRC Connection Resume timer in ms. The timer starts upon transmission of␣
↪RRCResumeRequest or RRCResumeRequest1. Supported: [100,200,300,400,600,1000,
↪1500,2000].

  ul_common:
    p_max:                                      # Optional TEXT. Sets maximum␣
↪transmit power allowed in this serving cell. Supported: [-30 - +23].
    max_pucchs_per_slot: 31                     # Optional INT (31). Sets the␣
↪maximum number of PUCCH grants that can be allocated per slot. Supported:␣
↪[1 - 64].
    max_ul_grants_per_slot: 32                  # Optional INT (32). Sets the␣
↪maximum number of UL grants that can be allocated per slot. Supported: [1 -␣
↪80].

  pdcch:
    common:
      coreset0_index:                           # Optional INT. Sets the␣
↪CORESET 0 index. Supported: [0 - 15].
      ss1_n_candidates: {0, 0, 1, 0, 0}         # Optional UINT ({0, 0, 1, 0,␣
↪0}). Sets the number of PDCCH candidates per aggregation level for␣
↪SearchSpace#1. Supported: any 5 value array containing the following UINT␣
↪values [0, 1, 2, 3, 4, 5, 6, 7, 8].
      ss0_index: 0                              # Optional UINT (0). Sets the␣
↪SearchSpace#0 index. Supported: [0 - 15].
      max_coreset0_duration:                    # Optional INT. Sets the␣
↪maximum CORESET#0 duration in OFDM symbols to consider when deriving CORESET
↪#0 index. Supported: [1 - 2].
    dedicated:
      coreset1_rb_start: 1                      # Optional INT (0). Sets the␣
↪starting common resource block (CRB) number for CORESET 1, relative to CRB0.
↪ Supported: [0 - 275].
      coreset1_l_crb:                           # Optional INT (Across entire␣
```

(continues on next page)

(continued from previous page)

```
↪BW of cell). Sets the length of CORESET 1 in number of CRBs. Supported: [0 -
↪ 275].
      coreset1_duration:                        # Optional INT (Max(2, Nof.␣
↪CORESET#0 symbols)). Sets the duration of CORESET 1 in number of OFDM␣
↪symbols. Supported: [1 - 2].
      ss2_n_candidates: {0, 0, 0, 0, 0}         # Optional UINT ({0, 0, 0, 0,␣
↪0}). Sets the number of PDCCH candidates per aggregation level for␣
↪SearchSpace#2. Supported: any 5 value array containing the following UINT␣
↪values [0, 1, 2, 3, 4, 5, 6, 7, 8].
      dci_format_0_1_and_1_1: 1                 # Optional BOOLEAN (1). Sets␣
↪whether to use non-fallback or fallback DCI format in UE SearchSpace#2.␣
↪Supported: [0, 1].
      ss2_type: ue_dedicated                    # Optional TEXT (ue_
↪dedicated). Sets the SearchSpace type for UE dedicated SearchSpace#2.␣
↪Supported: [common, ue_dedicated].

  pdsch:
    min_ue_mcs: 0                               # Optional UINT (0). Sets a␣
↪minimum PDSCH MCS value to be used for all UEs. Supported: [0 - 28].
    max_ue_mcs: 28                              # Optional UINT (28). Sets a␣
↪maximum PDSCH MCS value to be used for all UEs. Supported: [0 - 28].
    fixed_rar_mcs: 0                            # Optional UINT (0). Sets a fixed␣
↪RAR MCS value for all UEs. Supported: [0 - 28].
    fixed_sib1_mcs: 5                           # Optional UINT (5). Sets a fixed␣
↪SIB1 MCS for all UEs. Supported: [0 - 28].
    nof_harqs: 16                               # Optional UNIT (16). Sets the␣
↪number of Downlink HARQ processes. Supported [2, 4, 6, 8, 10, 12, 16].
    max_nof_harq_retxs: 4                       # Optional UINT (4). Sets the␣
↪maximum number times a DL HARQ can be retransmitted before it is discarded.␣
↪Supported: [0 - 4].
    max_consecutive_kos: 100                    # Optional UINT (100). Sets the␣
↪maximum number of consecutive HARQ-ACK KOs before an RLF is reported.␣
↪Supported: [0 - inf].
    rv_sequence: [0,2,3,1]                      # Optional UINT (0,2,3,1). Sets␣
↪the redundancy version sequence to use for PDSCH. Supported: any␣
↪combination of [0, 1, 2, 3].
    mcs_table: qam64                            # Optional TEXT (qam64). Sets the␣
↪MCS table to use for PDSCH. Supported: [qam64, qam256].
    min_rb_size: 1                              # Optional UINT (1). Sets the␣
↪minimum RB size for the UE PDSCH resource allocation. Supported: [1 - 275].
    max_rb_size: 275                            # Optional UINT (275). Sets the␣
↪maximum RB size for the UE PDSCH resource allocation. Supported: [1 - 275].
    max_pdschs_per_slot: 35                     # Optional UINT (35). Sets the␣
↪maximum number of PDSCH grants per slot, including SIB, RAR, Paging and UE␣
↪data grants. Supported: [1 - 35].
    max_alloc_attempts: 35                      # Optional UINT (35). Sets the␣
↪maximum number of DL or UL PDCCH grant allocation attempts per slot before␣
↪scheduler skips the slot. Supported: [1 - 35].
    olla_cqi_inc_step: 0.001                    # Optional FLOAT (0.001). Sets␣
```

(continues on next page)

(continued from previous page)

```
↪the outer-loop link adaptation (OLLA) increment value. The value 0 means␣
↪that OLLA is disabled. Supported: [0 - 1].
    olla_target_bler: 0.01                  # Optional FLOAT (0.01). Sets the␣
↪target DL BLER set in Outer-loop link adaptation (OLLA) algorithm.␣
↪Supported: [0 - 0.5].
    olla_max_cqi_offset: 4                  # Optional FLOAT (4). Sets the␣
↪maximum offset that the Outer-loop link adaptation (OLLA) can apply to CQI.␣
↪Supported: positive float.
    dc_offset:                              # Optional TEXT. Sets the direct␣
↪Current (DC) Offset in number of subcarriers, using the common SCS as␣
↪reference for carrier spacing, and the center of the gNB DL carrier as DC␣
↪offset value 0. The user can additionally
                                            # set "outside" to define that␣
↪the DC offset falls outside the DL carrier or "undetermined" in the case␣
↪the DC offset is unknown. Supported: [-1650 - 1649] OR [outside,
↪undetermined,center].
    harq_la_cqi_drop_threshold:             # Optional UINT. Sets the link␣
↪Adaptation (LA) threshold for drop in CQI of the first HARQ transmission␣
↪above which HARQ retransmissions are cancelled. Set this value to 0 to␣
↪disable this feature. Supported: [0 - 15].
    harq_la_ri_drop_threshold:              # Optional UINT. Sets the link␣
↪Adaptation (LA) threshold for drop in nof. layers of the first HARQ␣
↪transmission above which HARQ retransmission is cancelled. Set this value␣
↪to 0 to disable this feature. Supported: [0 - 4].
    dmrs_additional_position:               # Optional UINT. Sets the PDSCH␣
↪DMRS additional position. Supported: [0 - 3].

  pusch:
    min_ue_mcs: 0                           # Optional UINT (0). Sets a␣
↪minimum PUSCH MCS value to be used for all UEs. Supported: [0 - 28].
    max_ue_mcs: 28                          # Optional UINT (28). Sets a␣
↪maximum PUSCH MCS value to be used for all UEs. Supported: [0 - 28].
    max_consecutive_kos: 100                # Optional UINT (100). Sets the␣
↪maximum number of consecutive CRC KOs before an RLF is reported. Supported:␣
↪[0 - inf].
    rv_sequence: [0]                        # Optional UINT (0). Sets the␣
↪redundancy version sequence to use for PUSCH. Supported: any combination of␣
↪[0, 1, 2, 3].
    mcs_table: qam64                        # Optional TEXT (qam64). Sets the␣
↪MCS table to use for PUSCH. Supported: [qam64, qam256].
    msg3_delta_preamble: 6                  # Optional INT (6). Sets the MSG3␣
↪DeltaPreamble power offset between MS3 and RACH preamble transmission.␣
↪Supported: [-1 - 6].
    p0_nominal_with_grant: -76              # Optional INT (-76). Sets the P0␣
↪value for PUSCH grant (except MSG3), in dBm. Supported: multiples of 2␣
↪within the range [-202, 24].
    msg3_delta_power: 8                     # Optional INT (8). Sets the␣
↪target power level at the network receiver side, in dBm. Supported:␣
↪multiples of 2 within the range [-6, 8].
```

(continues on next page)

(continued from previous page)

```
    max_puschs_per_slot: 16                 # Optional UINT (16). Sets the␣
↪maximum number of PUSCH grants per slot. Supprted: [1 - 16].
    b_offset_ack_idx_1: 9                   # Optional UINT (9). Sets the␣
↪betaOffsetACK-Index1 part of UCI-OnPUSCH. Supported: [0 - 31].
    b_offset_ack_idx_2: 9                   # Optional UINT (9). Sets the␣
↪betaOffsetACK-Index2 part of UCI-OnPUSCH. Supported: [0 - 31].
    b_offset_ack_idx_3: 9                   # Optional UINT (9). Sets the␣
↪betaOffsetACK-Index3 part of UCI-OnPUSCH. Supported: [0 - 31].
    beta_offset_csi_p1_idx_1: 9             # Optional UINT (9). Sets the b_
↪offset_csi_p1_idx_1 part of UCI-OnPUSCH. Supported: [0 - 31].
    beta_offset_csi_p1_idx_2: 9             # Optional UINT (9). Sets the b_
↪offset_csi_p1_idx_2 part of UCI-OnPUSCH. Supported: [0 - 31].
    beta_offset_csi_p2_idx_1: 9             # Optional UINT (9). Sets the b_
↪offset_csi_p2_idx_1 part of UCI-OnPUSCH. Supported: [0 - 31].
    beta_offset_csi_p2_idx_2: 9             # Optional UINT (9). Sets the b_
↪offset_csi_p2_idx_2 part of UCI-OnPUSCH. Supported: [0 - 31].
    min_k2: 4                               # Optional UINT (4). Sets the␣
↪minimum value of K2 (difference in slots between PDCCH and PUSCH).␣
↪Supported: [1 - 4].
    dc_offset:                              # Optional TEXT. Sets the direct␣
↪Current (DC) Offset in number of subcarriers, using the common SCS as␣
↪reference for carrier spacing, and the center of the gNB UL carrier as DC␣
↪offset value 0. The user can additionally
                                            # set "outside" to define that␣
↪the DC offset falls outside the DL carrier or "undetermined" in the case␣
↪the DC offset is unknown. Supported: [-1650 - 1649] OR [outside,
↪undetermined,center].
    olla_snr_inc_step: 0.001                # Optional FLOAT (0.001). Sets␣
↪the outer-loop link adaptation (OLLA) increment value. The value 0 means␣
↪that OLLA is disabled. Supported: [0 - 1].
    olla_target_bler: 0.01                  # Optional FLOAT (0.01). Sets the␣
↪target UL BLER set in Outer-loop link adaptation (OLLA) algorithm.␣
↪sUPPORTED: [0 - 0.5].
    olla_max_snr_offset: 5                  # Optional FLOAT (5). Sets the␣
↪maximum offset that the Outer-loop link adaptation (OLLA) can apply to␣
↪estimate the UL SNR. Supported: positive float.
    dmrs_additional_position:               # Optional UINT. Sets the PUSCH␣
↪DMRS additional position. Supported: [0 - 3].

  pucch:
    p0_nominal: -90                         # Optional UINT (-90). Sets the␣
↪power control parameter P0 for PUCCH transmissions in dBm. Supported:␣
↪multiples of 2 and within the [-202, 24] interval.
    sr_period_ms: 40                        # Optional UINT (40). Sets the SR␣
↪period in milliseconds. Supported: [1,2,4,8,10,16,20,40,80,160,320].
    f1_nof_ue_res_harq: 3                   # Optional UINT (3). Sets the␣
↪number of PUCCH F1 resources available per UE for HARQ. Supported: [1 - 8].
    f1_nof_cell_res_sr: 2                   # Optional UINT (2). Sets the␣
↪number of PUCCH F1 resources available per cell for SR. Supported: [1 - 10].
```

(continues on next page)

(continued from previous page)

```
    f1_nof_symbols: 14                      # Optional UINT (14). Sets the␣
↪number of symbols for PUCCH F1 resources. Supported: [4 - 14].
    f1_enable_occ: 0                        # Optional BOOLEAN (0). Enables␣
↪OCC for PUCCH F1. Supported: [0, 1].
    f1_nof_cyclic_shifts: 1                 # Optional UINT (1). Sets the␣
↪number of possible cyclic shifts available for PUCCH F1 resources.␣
↪Supported: [1,2,3,4,6,12].
    f1_intraslot_freq_hop: 0                # Optional BOOLEAN (0). Enables␣
↪intra-slot frequency hopping for PUCCH F1. Supported: [0, 1].
    nof_cell_harq_pucch_res_sets: 1         # Optional UINT (1). Sets the␣
↪number of separate PUCCH resource sets for HARQ-ACK that are available in␣
↪the cell. The higher the number of sets, the lower the chances UEs have to␣
↪share the same PUCCH resources. Supported: [1 - 10].
    f2_nof_ue_res_harq: 6                   # Optional UINT (6). Sets the␣
↪number of PUCCH F2 resources available per UE for HARQ. Supported: [1 - 8].
    f2_nof_cell_res_csi: 1                  # Optional UINT (1). Sets the␣
↪number of PUCCH F2 resources available per cell for CSI. Supported: [1 -␣
↪10].
    f2_nof_symbols: 2                       # Optional UINT (2). Sets the␣
↪number of symbols for PUCCH F2 resources. Supported: [1 - 2].
    f2_max_nof_rbs: 1                       # Optional UINT (1). Sets the max␣
↪number of RBs for PUCCH F2 resources. Supported: [1 - 16].
    f2_max_payload:                         # Optional INT. Sets the max␣
↪number of payload bits for PUCCH F2 resources. Supported [1 - 11].
    f2_max_code_rate: dot35                 # Optional TEXT (dot35). Sets the␣
↪PUCCH F2 max code rate. Supported: [dot08, dot15, dot25, dot35, dot45,␣
↪dot60, dot80].
    f2_intraslot_freq_hop: 0                # Optional BOOLEAN (0). Enables␣
↪intra-slot frequency hopping for PUCCH F2. Supported: [0, 1].
    min_k1: 4                               # Optional UINT (4). Sets the␣
↪minimum value of K1 (difference in slots between PDSCH and HARQ-ACK). Lower␣
↪k1 values will reduce latency, but place a stricter requirement on the UE␣
↪decode latency. Supported: [1 - 4].
    max_consecutive_kos: 100                # Optional UINT (100). Sets the␣
↪maximum number of consecutive undecoded PUCCH F2 for CSI before an Radio␣
↪Link Failure is reported.

  prach:
    prach_config_index:                     # Optional UINT. Sets the PRACH␣
↪configuration index. If not set, the value is derived, so that the PRACH␣
↪fits in an UL slot. Supported: [0 - 255].
    prach_root_sequence_index: 1            # Optional UINT (1). Sets the␣
↪PRACH Roost Sequence Index (RSI), which determines the Zadoff-Chu (ZC)␣
↪sequence used. Supported: [0 - 837]. If the PRACH configuration index is␣
↪larger than 86, you cannot set a PRACH RSI of more than 137.
    zero_correlation_zone: 0                # Optional UINT (0). Sets the␣
↪Zero Correlation Zone, which determines the size of the cyclic shift and␣
↪the number of preamble sequences which can be generated from each Root␣
↪Sequence Index. Supported: [0 - 15].
```

(continues on next page)

(continued from previous page)

```
    fixed_msg3_mcs: 0                       # Optional UINT (0). Sets a fixed␣
↪Msg3 MCS. Supported: [0 - 28].
    max_msg3_harq_retx: 4                   # Optional UINT (4). Sets the␣
↪maximum number of Msg3 HARQ retransmissions. Supported: [0 - 4].
    total_nof_ra_preambles:                 # Optional TEXT. Sets the number␣
↪of different PRACH preambles. Supported: [1 - 64].
    prach_frequency_start:                  # Optional INT. Set Offset of␣
↪lowest PRACH transmission occasion in frequency domain respective to PRB 0,␣
↪in PRBs. Supported: [0 - (MAX_NOF_PRB - 1)].
    preamble_rx_target_pw: -100             # Optional INT (-100). Sets the␣
↪Target power level at the network receiver side, in dBm. Supported:␣
↪multiples of 2 within range [-202, -60].
    preamble_trans_max: 7                   # Optional UINT (7). Sets the max␣
↪number of RA preamble transmissions performed before declaring a failure.␣
↪Supported: [3, 4, 5, 6, 7, 8, 10, 20, 50, 100, 200].
    power_ramping_step_db: 4                # Optional UINT (4). Sets the␣
↪power ramping steps for PRACH. Supported: [0, 2, 4 , 6].
    ports:                                  # Optional UINT. Sets the list of␣
↪atenna ports. Expected value is a UINT string of antenna port IDs.
    nof_ssb_per_ro: 1                       # Optional FLOAT (1). Sets the␣
↪number of SSBs per RACH occasion. Supported: [1/8, 1/4, 1/2, 1, 2, 4, 8,␣
↪16].
    nof_cb_preambles_per_ssb: 4             # Optional UINT (4). Sets the␣
↪number of contention based preambles per SSB. Supported: [1 - 64].

  tdd_ul_dl_cfg:
    dl_ul_tx_period: 10                     # Optional INT (10). Sets the TDD␣
↪pattern periodicity in slots. The combination of this value and the chosen␣
↪numerology must lead to a TDD periodicity of 0.5, 0.625, 1, 1.25, 2, 2.5, 3,
↪ 4, 5 or 10 milliseconds. Supported: [2 - 80].
    nof_dl_slots: 6                         # Optional INT (6). Number of␣
↪consecutive full Downlink slots. Supported: [0-80].
    nof_dl_symbols: 8                       # Optional INT (8). Number of␣
↪Downlink symbols at the beginning of the slot following full Downlink slots.
↪ Supported: [0-13].
    nof_ul_slots: 3                         # Optional INT (3). Number of␣
↪consecutive full Uplink slots. Supported: [0 - 80].
    nof_ul_symbols: 0                       # Optional INT (0). Number of␣
↪Uplink symbols at the end of the slot preceding the first full Uplink slot.␣
↪Supported: [0-13].
    pattern2:
      dl_ul_tx_period: 10                       # Optional INT (10). Sets the␣
↪TDD pattern periodicity in slots. The combination of this value and the␣
↪chosen numerology must lead to a TDD periodicity of 0.5, 0.625, 1, 1.25, 2,␣
↪2.5, 3, 4, 5 or 10 milliseconds. Supported: [2 - 80].
      nof_dl_slots: 6                       # Optional INT (6). Number of␣
↪consecutive full Downlink slots. Supported: [0-80].
      nof_dl_symbols: 8                     # Optional INT (8). Number of␣
↪Downlink symbols at the beginning of the slot following full Downlink slots.
```

(continues on next page)

(continued from previous page)

```
↪ Supported: [0-13]
      nof_ul_slots: 3                          # Optional INT (3). Number of␣
↪consecutive full Uplink slots. Supported: [0 - 80].
      nof_ul_symbols: 0                        # Optional INT (0). Number of␣
↪Uplink symbols at the end of the slot preceding the first full Uplink slot.␣
↪Supported: [0-13].

  paging:
    pg_search_space_id: 1                    # Optional UINT (1). Sets the␣
↪SearchSpace to use for Paging. Supported: [0, 1].
    default_pg_cycle_in_rf: 128              # Optional UINT (128). Sets the␣
↪default Paging cycle in nof. Radio Frames. Supported: [32,64,128,256].
    nof_pf_per_paging_cycle: oneT            # Optional TEXT (oneT). Sets the␣
↪number of paging frames per DRX cycle. Supported: [oneT, halfT, quarterT,␣
↪oneEighthT, oneSixteethT].
    pf_offset: 0                             # Optional UINT (0). Sets the␣
↪paging frame offset. Supported: [0 - (nof_pf_per_paging_cycle - 1)].
    nof_po_per_pf: 1                         # Optional UINT (1). Sets the␣
↪number of paging occasions per paging frame. Supported: [1, 2, 4].

  csi:
    csi_rs_enabled: 1                        # Optional BOOLEAN (1). Enables␣
↪CSI-RS resources and CSI reporting. Supported: [0, 1].
    csi_rs_period: 20                        # Optional UINT (20). Sets the␣
↪CSI-RS period in milliseconds. Supported: [10, 20, 40, 80].
    meas_csi_rs_slot_offset: 2               # Optional UINT (2). Sets the␣
↪slot offset of first CSI-RS resource used for measurement.
    tracking_csi_rs_slot_offset: 12          # Optional UINT (12). Sets the␣
↪slot offset of the first CSI-RS slot used for tracking.
    zp_csi_rs_slot_offset:                   # Optional TEXT. Sets the slot␣
↪offset of the ZP CSI-RS resources.
    pwr_ctrl_offset: 0                       # Optional INT (0). Sets the␣
↪power offset of PDSCH RE to NZP CSI-RS RE in dB. Supported: [-8 - 15].

expert_phy:
  max_proc_delay: 5                          # Optional INT (5). Sets the␣
↪maximum allowed DL processing delay in slots. Supported: [1 - 30].
  pusch_dec_max_iterations: 6                # Optional UINT (6). Sets the␣
↪number of PUSCH LDPC decoder iterations. Format: Positive integer greater␣
↪than 0.
  pusch_dec_enable_early_stop: 1             # Optional BOOL (1). Enables the␣
↪PUSCH decoder early stopping mechanism.
  pusch_sinr_calc_method: post_equalization # Optional TEXT (evm). Sets the␣
↪PUSCH SINR calculation method. Supported: [channel_estimator, post_
↪equalization, evm].
  max_request_headroom_slots: 0              # Optional UINT (0). Sets the␣
↪maximum request headroom size in slots. Supported: [0 - 3].

buffer_pool:
```

(continues on next page)

(continued from previous page)

```
  nof_segments: 1048576                # Optional UINT (1048576). Sets the␣
↪number of segments allocated by the buffer pool.
  segment_size: 2048                   # Optional UINT (2048). Sets the size␣
↪of each buffer pool segment in bytes.

test_mode:
  test_ue:
    rnti: 0                            # Optional ENUM (0). Sets the C-
↪RNTI of the UE. Supported: [0 - 65519].
    nof_ues: 1                         # Optonal UINT (1). Sets the number␣
↪of test UE(s) created. Supported [0 - 1024].
    auto_ack_indication_delay:         # Optional TEXT. Sets the delay␣
↪before the UL and DL HARQs are automatically ACKed. This feature should␣
↪only be used if the UL PHY is not operational.
    pdsch_active: 1                    # Optional BOOLEAN (1). Enables the␣
↪PDSCH of the UE.
    pusch_active: 1                    # Optional BOOLEAN (1). Enables the␣
↪PUSCH of the UE.
    cqi: 15                            # Optional UINT (15). Sets the␣
↪Channel Quality Information to be forwarded to the test UE. Supported: [1 -␣
↪15].
    pmi: 0                             # Optional UINT (0). Sets the␣
↪Precoder Matrix Indicator to be forwarded to test UE. Supported: [0 - 3].
    ri: 1                              # Optional UINT (1). Sets the Rank␣
↪Indicator to be forwarded to the test UE. Supported: [1 - 4].
    i_1_1: 0                           # Optional INT (0). Sets the␣
↪Precoder Matrix codebook index "i_1_1" to be forwarded to test UE, in the␣
↪case of more than 2 antennas. Supported: [0 - 7].
    i_1_3: 0                           # Optional INT (0). Sets the␣
↪Precoder Matrix codebook index "i_1_3" to be forwarded to test UE, in the␣
↪case of more than 2 antennas. Supported: [0 - 1].
    i_2: 0                             # Optional INT (0). Sets the␣
↪Precoder Matrix codebook index "i_2" to be forwarded to test UE, in the␣
↪case of more than 2 antennas. Supported: [0 - 3].

expert_execution:
  cell_affinities:                     # Optional TEXT. Sets the cell CPU␣
↪affinities configuration on a per cell basis. Entry order is the same as␣
↪the order in the defined cell list.
    -
      l1_dl_cpus:                        # Optional TEXT. Sets the CPU␣
↪core(s) assigned to L1 downlink tasks. Supported: [1, 2, 3 , ..., N]. Where␣
↪N is the number of total cores available.
      l1_ul_cpus:                        # Optional TEXT. Sets the CPU␣
↪core(s) assigned to L1 uplink tasks. Supported: [1, 2, 3 , ..., N].
      l1_dl_pinning:                     # Optional TEXT. Sets the policy␣
↪used for assigning CPU cores to L1 downlink tasks.
      l1_ul_pinning:                     # Optional TEXT. Sets the policy␣
↪used for assigning CPU cores to L1 uplink tasks.
```

(continues on next page)

(continued from previous page)

```
    l2_cell_cpus:                           # Optional TEXT. Sets the CPU␣
↪core(s) assigned to L2 cells tasks. Supported: [1, 2, 3 , ..., N].
    l2_cell_pinning:                        # Optional TEXT. Sets the policy␣
↪used for assigning CPU cores to L2 cell tasks.
    ru_cpus:                                # Optional TEXT. Sets the CPU␣
↪core(s) used for the Radio Unit tasks. Supported: [1, 2, 3 , ..., N].
    ru_pinning:                             # Optional TEXT. Sets the policy␣
↪used for assigning CPU cores to Radio Unity tasks.
  affinities:
    low_priority_cpus:                      # Optional TEXT. Sets the CPU␣
↪core(s) assigned to low priority tasks. Supported: [1, 2, 3 , ..., N].
    low_priority_pinning:                   # Optional TEXT. Sets the policy␣
↪used for assigning CPU cores to low priority tasks.
    ioslated_cpus:                          # Optional TEXT. Sets the CPU␣
↪core(s) isolated for the gNB application. Supported: [1, 2, 3 , ..., N].
  threads:
    non_rt:
      nof_non_rt_threads: 4                 # Optional UINT (4). Sets the␣
↪number of non real time threads for processing of CP and UP data in upper␣
↪layers.
    upper_phy:
      pdsch_processor_type: auto            # Optional TEXT (auto). Sets␣
↪the PDSCH processor type. Supported: [auto, generic, concurrent, lite].
    nof_pusch_decoder_threads: 1            # Optional UINT (1). Sets the␣
↪number of threads used to encode PUSCH.
    nof_ul_threads: 1                       # Optional UINT (1). Sets the␣
↪number of upprt PHY threads to proccess uplink.
    nof_dl_threads: 2                       # Optional UINT (1). Sets the␣
↪number of upprt PHY threads to proccess downlink.
    lower_phy:
      execution_profile:                    # Optional TEXT. Sets the lower␣
↪physical layer executor profile. Supported: [single, dual, quad].
    ofh:
      enable_dl_parallelization: 1          # Optional BOOLEAN. Sets the␣
↪Open Fronthaul downlink parallelization flag. Supported: [0, 1].

hal:
  eal_args:                             # Optional TEXT. EAL configuration␣
↪parameters used to initialize DPDK.

log:
# All gNB layers and components can be configured independently to output at␣
↪various levels of detail. Logs can be configured to the following levels␣
↪(from lowest to highest levels of detail):
# none, error, warning, info, debug
  filename:                             # Optional TEXT (/tmp/gnb.log). File␣
↪path for logs. Logs can be redirected to stdout by setting the filename to
↪"stdout".
  all_level: warning                    # Optional TEXT (warning). Sets a␣
```

(continues on next page)

(continued from previous page)

```
↪common log level across PHY, MAC, RLC, PDCP, RRC, SDAP, NGAP and GTPU␣
↪layers.
  phy_level: warning                    # Optional TEXT (warning). Sets PHY␣
↪log level.
  mac_level: warning                    # Optional TEXT (warning). Sets MAC␣
↪log level.
  rlc_level: warning                    # Optional TEXT (warning). Sets RLC␣
↪log level.
  pdcp_level: warning                   # Optional TEXT (warning). Sets PDCP␣
↪log level.
  rrc_level: warning                    # Optional TEXT (warning). Sets RRC␣
↪log level.
  sdap_level: warning                   # Optional TEXT (warning). Sets SDAP␣
↪log level.
  ngap_level: warning                   # Optional TEXT (warning). Sets NGAP␣
↪log level.
  gtpu_level: warning                   # Optional TEXT (warning). Sets GTPU␣
↪log level.
  radio_level: info                     # Optional TEXT (info). Sets radio␣
↪log level.
  fapi_level: warning                   # Optional TEXT (warning). Sets FAPI␣
↪log level.
  ofh_level: warning                    # Optional TEXT (warning). Sets Open␣
↪Fronthaul log level.
  f1ap_level: warning                   # Optional TEXT (warning). Sets F1AP␣
↪log level.
  f1u_level: warning                    # Optional TEXT (warning). Sets F1u␣
↪log level.
  du_level: warning                     # Optional TEXT (warning). Sets DU␣
↪log level.
  cu_level: warning                     # Optional TEXT (warning). Sets CU␣
↪log level.
  sec_level: warning                    # Optional TEXT (warning). Sets␣
↪security functions level.
  lib_level: warning                    # Optional TEXT (warning). Sets␣
↪generic log level.
  hex_max_size: 0                       # Optional UINT (0). Sets maximum␣
↪number of bytes to print for hex messages. Supported: [0 - 1024]
  broadcast_enabled: 0                  # Optional BOOL (0). Enables logging␣
↪in the PHY and MAC layer of broadcast messages and all PRACH opportunities.␣
↪Supported: [0, 1].
  phy_rx_symbols_filename:              # Optional TEXT. Print received␣
↪symbols to file. Symbols will be printed if a valid path is set. Format:␣
↪file path. This file can be used in the srsRAN_matlab project.
  phy_rx_symbols_port: 0                # Optional TEXT. Set to a valid␣
↪receive port number to dump the IQ symbols from that port only, or set to
↪"all" to dump the IQ symbols from all UL receive ports. Only works if "phy_
↪rx_symbols_filename" is set. Supported: [NON-NEGATIVE or all].
  phy_rx_symbols_prach: 0               # Optional BOOLEAN (0). Set to true␣
```

(continues on next page)

(continued from previous page)

```
↪to dump the IQ symbols from all the PRACH ports. Only works if "phy_rx_
↪symbols_filename" is set. Supported: [0, 1].
  tracing_filename:                     # Optional TEXT. Set to a valid file␣
↪name to enable tracing log.

pcap:
  ngap_filename: /tmp/gnb_ngap.pcap     # Optional TEXT (/tmp/gnb_ngap.pcap).␣
↪Path for NGAP PCAPs.
  ngap_enable: false                    # Optional BOOL (false). Enable/
↪disable NGAP packet capture.
  e1ap_filename: /tmp/gnb_e1ap.pcap     # Optional TEXT (/tmp/gnb_e1ap.pcap).␣
↪Path for E1AP PCAPs.
  e1ap_enable: false                    # Optional BOOL (false). Enable/
↪disable E1AP packet capture.
  f1ap_filename: /tmp/gnb_f1ap.pcap     # Optional TEXT (/tmp/gnb_f1ap.pcap).␣
↪Path for F1AP PCAPs.
  f1ap_enable: false                    # Optional BOOL (false). Enable/
↪disable F1AP packet capture.
  rlc_filename: /tmp/gnb_rlc.pcap       # Optional TEXT (tmp/gnb_rlc.pcap).␣
↪Path for RLC PCAPs.
  rlc_rb_type: all                      # Optional TEXT. Sets the RLC PCAP RB␣
↪type. Supported: [all, srb, drb].
  rlc_enable: false                     # Optional BOOL (false). Enable/
↪disable RLC packet capture.
  mac_filename: /tmp/gnb_mac.pcap       # Optional TEXT (/tmp/gnb_mac.pcap).␣
↪Path for MAC PCAPs.
  mac_type: udp                         # Optional TEXT (udp). Sets the MAC␣
↪PCAP pcap type. Supported: [DLT or UDP].
  mac_enable: false                     # Optional BOOL (false). Enable/
↪disable MAC packet capture.
  e2ap_filename: /tmp/gnb_e2ap.pcap     # Optional TEXT (/tmp/gnb_e2ap.pcap).␣
↪Path for E2AP PCAPs.
  e2ap_enable: false                    # Optional BOOL (false). Enable/
↪disable E2AP packet capture.
  gtpu_filename: /tmp/gnb_gtpu.pcap     # Optional TEXT (/tmp/gnb_gtpu.pcap).␣
↪Path for GTPU PCAPs.
  gtpu_enable: false                    # Optional BOOL (false). Enable/
↪disable GTPU packet capture.

metrics:
  rlc_report_period: 1000               # Optional UINT (1000). Sets the RLC␣
↪metrics report period in milliseconds. Supported [0 - inf]
  rlc_json_enable: 0                    # Optional BOOLEAN (0). Enable RLC␣
↪JSON metrics reporting. Supported: [0, 1].
  pdcp_report_period: 0                 # Optional UINT (0). Sets the PDCP␣
↪metrics report period.
  cu_cp_statistics_report_period: 1     # Optional UINT (1). Sets the CU-CP␣
↪statistics report period in seconds. Set this value to 0 to disable this␣
↪feature. Supported: [0 - inf].
```

(continues on next page)

(continued from previous page)

```
  cu_up_statistics_report_period: 1      # Optional UINT (1). Sets the CU-UP␣
↪statistics report period in seconds. Set this value to 0 to disable this␣
↪feature. Supported: [0 - inf].
  enable_json_metrics: 0                 # Optional BOOLEAN (0). Enables JSON␣
↪metrics reporting. Supported: [0, 1].
  addr:                                  # Optional TEXT:IPV4. Sets the␣
↪metrics address. Supported: IPV4 address.
  port: 55555                            # Optional UINT. Sets the metrics UPD␣
↪port. Supported: [0 - 65535].
  autostart_stdout_metrics: 0            # Optional BOOLEAN (0). Sets whether␣
↪or note to autostart stdout metrics reporting. Supported [0, 1].
```

CHAPTER

ELEVEN

GRAFANA METRICS GUI

srsRAN allows the reporting and visualization of the CU/DU metrics to a Grafana WebUI. This is done through the use of a Docker container that comes as standard with the srsRAN code base, located in the `~/srsRAN/docker/` folder. This container allows users to bring up the Grafana dashboard in a single command.

Further Reading:

- Grafana Docs

11.1 Configuration

To use the Grafana webUI, you will first need to have Docker installed on your system, you will also need to modify the srsRAN CU/DU configuration file to allow the reporting of the metrics to the necessary JSON format for use in the webUI.

11.1.1 Docker

Using the Docker Containers included with srsRAN requires `docker compose` to be installed on your system. You can read about `docker compose` here. There are multiple ways to install this, but the most basic way to do so is to install Docker Desktop. For installing Docker Desktop on linux, take a look at the Docker documentation.

Note: We recommend using a Docker Compose V2 or later.

11.1.2 srsRAN

To enable the correct reporting of metrics to the Grafana UI, the srsRAN configuration files needs to be updated to allow the metrics to be output in the correct JSON format and then sent through a udp-socket to the metrics-server, where it can be parsed and displayed correctly by the GUI.

```
metrics:
    enable_json_metrics: true       # Enable reporting metrics in JSON format
    addr: 172.19.1.4                # Metrics-server IP
    port: 55555                     # Metrics-server Port
```

The `addr` and `port` values defined above mirror those set in the `docker-compose.yml` file found in `~/srsRAN_Project/docker`. Any changes in these values must be kept consistent across both files.

11.2 Launching GUI

To launch the docker image for the Grafana UI, run the following command from the main folder containing srsRAN:

```
sudo docker compose -f docker/docker-compose.yml up grafana
```

The following output should be observed:

```
Creating network "docker_ran" with the default driver
Starting metrics_server ...
Starting metrics_server ... done
Creating grafana        ... done
Attaching to grafana
```

Navigating to http://localhost:3300/ in your preferred web browser will allow you to view the UI.

You can then run srsRAN as normal. As the UE(s) connect to the network you will begin to see an output for each. These figures and graphics will update automatically during runtime, showing plots for each UE on the network.

11.3 GUI Output

A sample of the UI output can be seen here:

The above figure shows a single COTS UE connected to the network, with different traffic bursts of varying bandwidth being generated using iPerf. The cell bandwidth is 20 MHz.

CHAPTER

TWELVE

TROUBLESHOOTING

For support and help using srsRAN Project, check out the community driven discussion forum.

12.1 Performance Tuning

The following sections outline key steps to improve gNB performance.

12.1.1 CPU Performance Mode

The CPU governor of the PC should be set to performance mode to allow for maximum compute power and throughput. This can be configured for e.g. Ubuntu using:

```
echo "performance" | sudo tee /sys/devices/system/cpu/cpu*/cpufreq/scaling_
↪governor
```

It is also recommended that users running on a laptop keep the PC connected to a power-source at all times while running the gNB, as this will avoid performance loss due to CPU frequency scaling on the machine.

12.1.2 Performance Configuration Script

Before running the gNB application, we recommend tuning your system for best performance. We provide a script to configure known performance parameters:

- srsran_performance

The script does the following:

1. Sets the scaling governor to performance
2. Disables DRM KMS polling
3. Tunes network buffers (Ethernet based USRPs only)

Run the script as follows from the main project folder:

```
sudo ./scripts/srsran_performance
```

12.2 USRP Configuration

Users should always ensure that the USRP they are using is running over USB 3.0 or ethernet and correctly configured. If UHD is built from source, users will have multiple example applications available in `/usr/lib/uhd/examples/`. User can verify their USRP is correctly configured by running the `uhd_benchmark` application as follows:

```
sudo ./benchmark_rate --rx_rate [rate in Hz] --tx_rate [rate in Hz]
```

More details can be found in this guide on the Ettus Knowledge Base.

12.2.1 USRP Time Calibration

Incorrect time calibration of a USRP can lead to preventing the gNB from receiving PRACH transmissions. The TX/RX time calibration adjusts the offset between TX and RX processing chains delay in the USRP. This value varies as a function of sampling rate, USRP model and UHD version. Users experiencing issues with incorrect time calibration will see a message similar to the following in the logs:

```
2023-03-08T08:38:34.130365 [UL-PHY0 ] [I] [ 1001.18] PRACH:  rssi=+0.5dB␣
↪detected_preambles=[{idx=55 ta=-7.29us power=+85.8dB snr=0.0dB}] t=351.3us
2023-03-08T08:38:34.130377 [FAPI    ] [E] [     0.0] Detected 1 errors in␣
↪RACH.indication message at slot=1001.18:
   - Property=Timing advance offset in nanoseconds, value=-7291, expected␣
↪value=[0-2005000]
```

In the above log the PRACH arrived 7.29 us early to the gNB, which at a sampling rate of 23.04 MHz is approximately 168 samples. The TX/RX offset needs to be adjusted to a higher value than this so that the PRACH arrives within the detection window. This can be done by adding the following to the config file under the `ru_sdr` options:

```
ru_sdr:
   time_alignment_calibration: -170    # This will set an offset of -170␣
↪samples
```

In general, a larger negative value is better as it will make sure the PRACH falls within the detection window. The consequence of increasing this too much is that the effective cell size is reduced (this is not important for a lab set up).

By default the `time_alignment_calibration` parameter is set to 0. This means that in most SDR frontends the PRACH will arrive a few samples late within the window. With preamble format 0, there is enough space in the detection window and this does not cause any problem. However, if you are trying to set up a very large cell, or using different preamble formats, you might want to set a positive `time_alignment_calibration` value such that there is space in the window for UEs far in the cell.

CHAPTER

THIRTEEN

SOFTWARE ARCHITECTURE

A primer on the O-RAN gNB architecture has already been outlined in the Knowledge Base, this can be found *here*.

This documentation aims to outline how this architecture is implemented the srsRAN Project codebase. The function and implementation of each component will be discussed in subsequent sections.

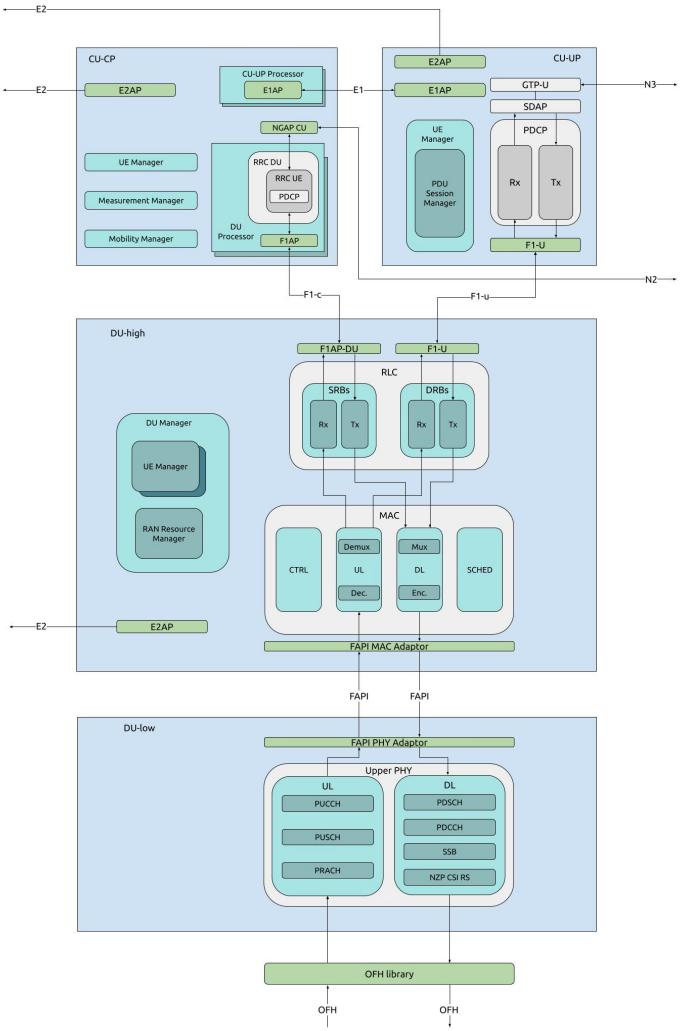


Fig. 1: High level overview of the srsRAN Project codebase architecture showing all main components and interfaces.

The code implements all of the components and interfaces in the above diagram. All of these elements have been implemented in software and are fully performant, customizable and compliant with the O-

RAN standard. Users can also integrate 3rd-party RICs, RUs, and gNB components with the srsRAN Project components.

13.1 CU-CP

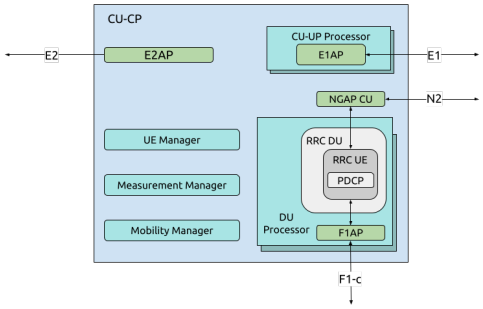


Fig. 2: srsRAN Project CU-CP implementation.

The CU-CP, or Central Unit - Control Plane, is responsible for the handling of control plane messaging, specifically, the control plane part of the PDCP protocol. In the CU-CP, there are five main components and four main interfaces. The CU-CP communicates directly with the 5G Core (via the N2 interface), the CU-UP (via the E1 interface), the DU-high (via the F1-c interface) and can also be connected to the near-RT RIC (via the E2 interface). This implementation takes a UE-centric approach.

Return to top level architecture diagram.

Components:

- CU-UP Processor: Custom component for handling each CU-UP connected to the CU-CP. Multiple CU-UPs can be connected to one CU-CP, as a result the E1AP is created inside the CU-UP processor for each connected CU-UP.
- DU Processor: Custom component for handling DUs for connected to the CU-CP. Each connected DU has its own DU processor with bundled F1AP, PDCP, and RRC procedures. Each UE connected to the DU also has its own RRC UE containing the PDCP processes.
- UE Manager: Custom component for managing connected UEs in the CU-CP. Responsible for adding and removing UEs and providing relevant UE information to other processes. Communicates information to/from the CU-UP, DU and Core.
- Measurement Manager: Custom component for managing cell measurement within the gNB.
- Mobility Manager: Custom component for managing UE mobility in the CU-CP.

Interfaces:

- E2: Interface with the near-RT RIC.
- E1: Interface with the CU-UP.
- F1-c: Control plane interface with the DU.
- N2: Control plane interface with the 5G Core (AMF).

13.2 CU-UP

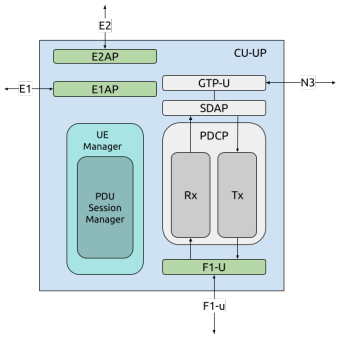


Fig. 3: srsRAN Project CU-UP implementation.

The CU-UP, or Central Unit - User Plane, is responsible for the handling of user plane messaging. Specifically the user plane aspect of the PDCP and SDAP protocols. In the implementation of the CU-UP, there are four main components and four main interfaces. The CU-UP communicates directly with the UPF in the 5G Core (via the N3 interface), the CU-UP (via the E1 interface), the DU-high (via the F1-u interface) and can also be connected to the near-RT RIC (via the E2 interface).

Return to top level architecture diagram.

Components:

- UE Manager: Custom component for managing connected UEs in the CU-UP. Responsible for adding and removing UEs and providing relevant UE information to other processes. Communicates information to/from the CU-CP, DU and Core.
- GTP-U: The GPRS Tunneling Protocol - User Plane (GTP-U) is responsible for transporting User Plane traffic to and from the UPF of the 5G Core via the N3 interface.
- SDAP: The Service Data Adaptation Protocol (SDAP) is responsible for adaption and mapping of Quality of Service (QoS) requirements on User Plan traffic.
- PDCP: The Packet Data Convergence Protocol (PDCP) is responsible for handling the User Plane data before/ after it enters the DU-high.

Interfaces:

- E2: Interface with the near-RT RIC.
- E1: Interface with the CU-CP.
- F1-u: User plane interface with the DU.
- N3: User plane interface with the 5G Core (UPF).

13.3 DU-high

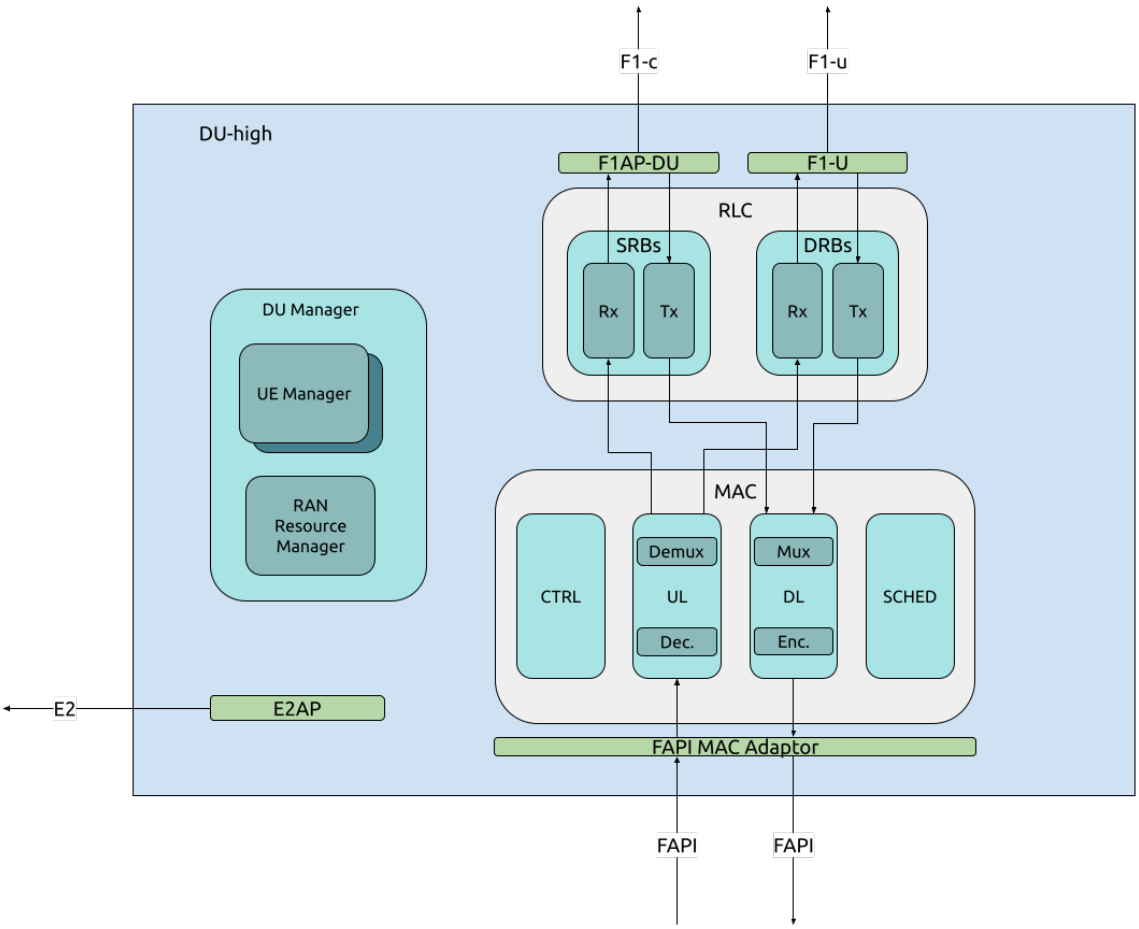


Fig. 4: srsRAN Project DU-high implementation

The DU-high, or Distributed Unit - High, is responsible for the handling of both uplink and downlink traffic. Specifically the MAC and RLC processing of these signals. The DU-high, has three main components and three main interfaces. The DU-high communicates directly with the CU-CP and CU-UP via the F1-c and F1-u interfaces, the DU-low via the FAPI interface, and can also be connected to the near-RT RIC via the E2 interface.

Return to top level architecture diagram.

Components:

- *DU Manager*: The DU Manager is responsible for managing the DU, specifically the connected UEs and RAN Resources.
- *RLC*: The Radio Link Control (RLC) layer sits between the MAC in the DU-high and the F1AP/PDCP in the CU, and provides the transport services for SRBs and PRBs between these layers. It operates in three modes: transparent, unacknowledged and acknowledged.
- *MAC*: The Medium Access Control (MAC) is responsible for the encoding and decoding of MAC PDUs, scheduling uplink and downlink grants and other control services.

Interfaces:

- E2: Interfaces with the nRT RIC.

- F1: Interfaces with the CU control and user plane.
- FAPI: Interface between the DU-high and DU-low.

Contents:

13.3.1 DU Manager

The DU manager acts as a mediator in the process of configuring different layers of the DU. It is particularly important in the pre-operational stage, when it provides system information to the FAPI, MAC and F1AP that is essential to their operation. The DU manager is also essential during the operational stage, in the creation/reconfiguration/deletion of UEs and in the dynamic connection between bearers across different layers.

The main responsibilities of the DU Manager include:

- Configuring the MAC, FAPI and F1AP during the pre-operational stage with relevant DU system information, such as supported DU cells and their configuration
- Orchestration of the creation/reconfiguration/removal of UEs in the DU, in particular, the setting up of UE contexts in the MAC, FAPI and F1AP.
- Orchestration of the creation/reconfiguration/removal of UE bearers in MAC, RLC and F1AP, and connection of each bearer with its respective upper and lower layers.
- Management of commands received through E2.

The DU Manager contains the following components:

- UE Manager: Manages the UEs connected to the DU
- RAN Resource Manager: Manages the physical RAN resources associated with the DU

13.3.2 Radio Link Control (RLC)

RLC layer connects the MAC and the F1AP/PDCP, as shown in `rlcstack`, and provides that transfer services to these layers.

It does this, by providing three different modes:

- **Transparent Mode (TM)**

  TM is used only in control signaling (SRB0 only) and provides data transfer services without modifying the SDUs/PDUs at all.

- **Unacknowledged Mode (UM)**

  UM can only be used in data traffic (DRBs only) and provides data transfer services with segmentation/reassembly. It is usually used in delay-sensitive and loss-tolerant traffic, as it does not provide re-transmissions.

- **Acknowledged Mode (AM)**

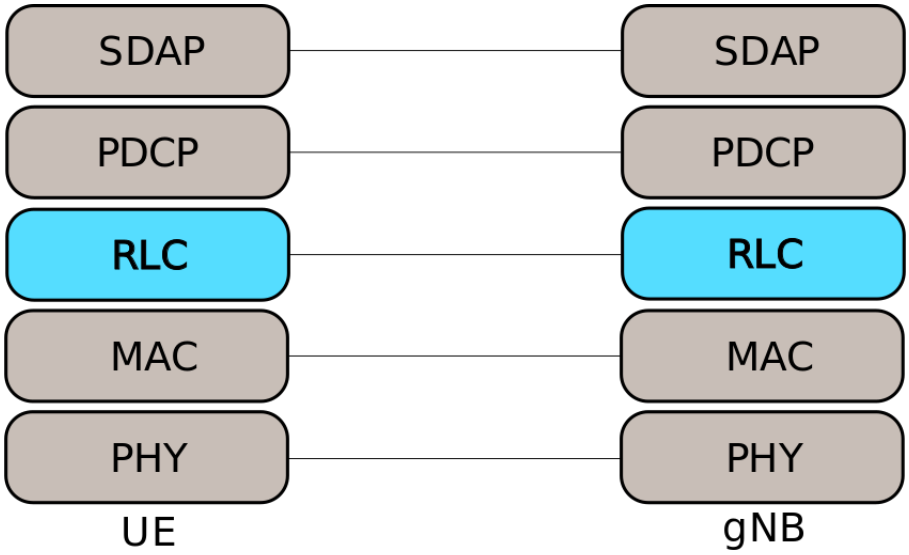


Fig. 5: RLC stack

AM can be used in data and control traffic (mandatory for SRBs, optional for DRBs) and provides data transfer services with segmentation and ARQ procedures. This mode is usually used for traffic that is more loss-sensitive, but more delay-tolerant.

RLC Acknowledged Mode (AM)

The RLC AM provides data transfer services with segmentation/concatenation and ARQ. It does this by keeping two windows, one for TX and another for RX, which store information regarding the transmitted/received SDUs. Unlike LTE, each window entry will contain information regarding a – single – SDU and its associated Sequence Number (SN).

Depending on the provided space in each transmission opportunity that is indicated by lower layer (MAC), the RLC may split SDUs into smaller SDU segments. The RX entity is responsible to store the received SDU segments in the RX window, and to pass the assembled SDU to the upper layer when it is fully received.

To provide ARQ services, the RX entity will generate *Status Reports* with the information of which SDUs/SDU segments have been received. This will be transmitted to the peer RLC entity, which will use the NACK information to trigger re-transmissions. The Tx entity will keep information regarding the transmitted SDUs on its TX window, so that it can retransmit the SDUs requested by its peer. This information is freed when the SDUs have been sequentially ACKed.

Data transmission

In `rlctxentity`, we can see a simplified illustration of the RLC AM Tx entity. There, we can see two threads do the work of pushing SDUs into the RLC and pulling PDUs from the RLC.

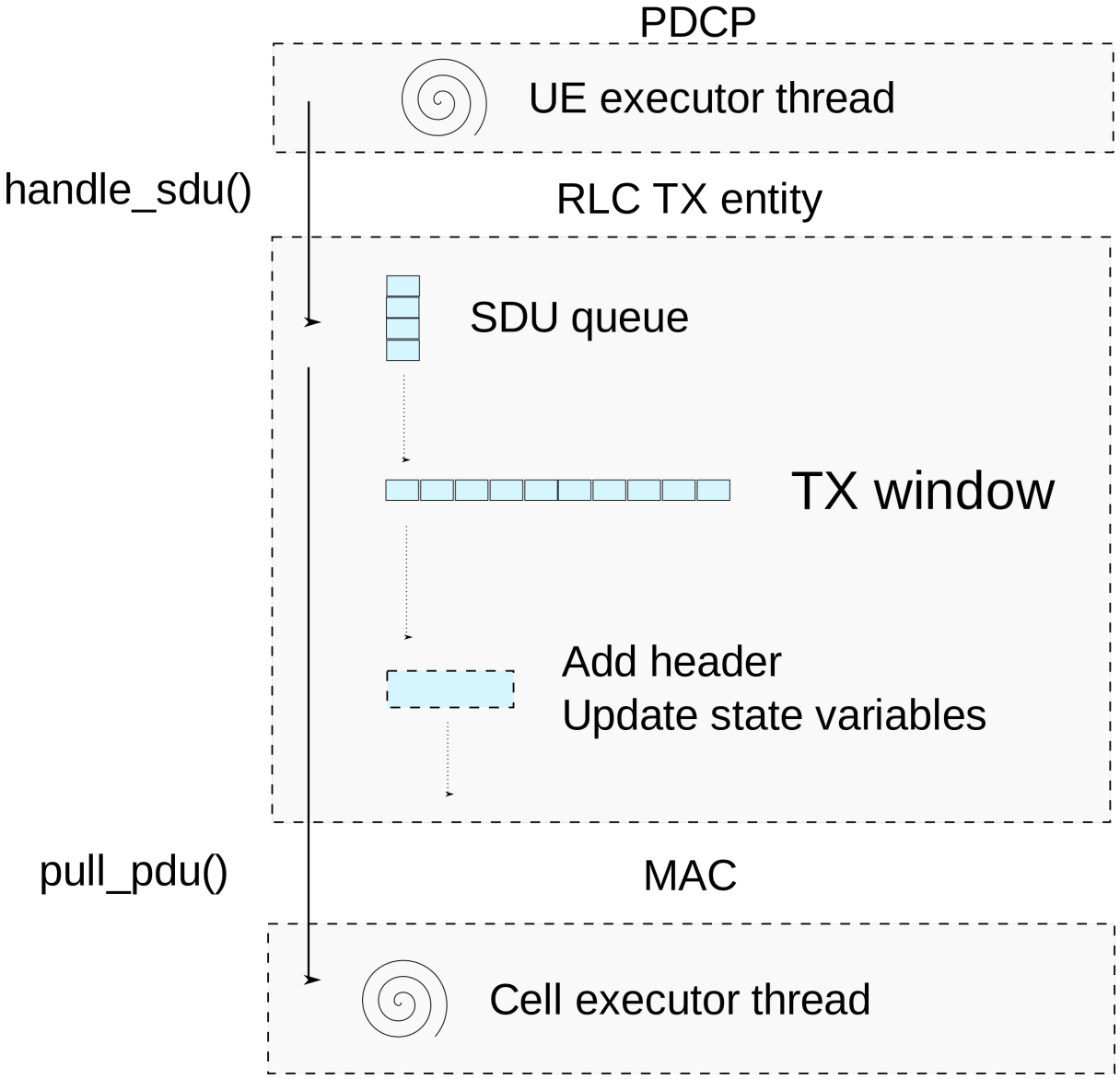


Fig. 6: Tx entity data path

The thread from the *UE executor* will push SDUs from the F1/PDCP into a thread-safe queue. The thread from the *Cell Executor* will call the *pull_pdu()* method to pull PDUs from the RLC. The MAC will let know the RLC the size of the grant that it can fill, and the TX entity of the RLC will fill the payload with a new PDU, a retransmission or a status report. The *pull_pdu()* can be called multiple times in a single slot.

When a PDU is pulled, the RLC entity will generate either a status report, an RETX, an SDU segment, or a new PDU, in that order of priority. When generating a new PDU, the RLC will keep the information about the RLC SDU in the TX window, most notably copy of the SDU bytes, and the next Segment Offset (SO), i.e the progress in case of segmentation.

This will be kept until the transmitted PDUs are ACKed in sequence. This is illustrated in `rlctxwindow`.

There, we can see that the TX window will maintain two state variables: *Tx_Next_Ack* and *Tx_Next*. *Tx_Next_Ack* refers to the bottom of the TX window, i.e., the SN next to the last PDU that has been sequentially ACKed. *Tx_Next* is the SN that will be assigned to the next PDU.

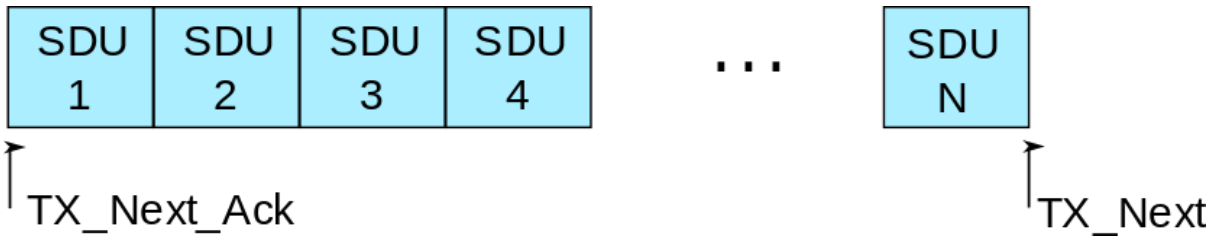


Fig. 7: Tx window

We will detail the RETX and Status Report generation later on, but first it is important to detail the reception of RLC data PDUs.

Data reception

We can see in figure `rlcrxentity` a simplified illustration of the RLC AM Rx entity. There we can see that only a single thread will push PDUs into the RLC Rx entity, the *UE executor* thread. When receiving a PDU, first the thing the entity will do is to check whether this is a Data PDU or a Control PDU (i.e. a status report.)

If it is a Data PDU, the SDU or SDU segment will be added/appended to the RX window, using the received SN. If the SDU has been fully received, SDU will be passed to the upper layers. The RX window will release the SDU information, when all SDUs have been received in order.

This is illustrated in `rlcrxwindow`, where we can see four state variables: *RX_Next*, *RX_Highest_Status*, *RX_Next_Status_Trigger* and *RX_Next_Highest*. There *RX_Next*, the lower edge of the RX window, will contain the first PDU that has not been fully received. *RX_Next_Higest*, the higher edge of the Rx window, is the highest PDU received. The RX window will also keep *RX_Highest_Status*, which is the SN of the first SDU that is considered lost, as determined by the *t-Reassembly* timer. Finally, the *RX_Next_Status_trigger*, will keep the SN that triggered the *t-Reassembly*. This is for updating *RX_Highest_Status* to the first known lost PDU, when *t-Reassembly* expires and new losses are detected.

ARQ and status reporting

When the RLC receives a PDU with the header's *Polling Bit* set, and *t-Status-Prohibit* is not running, the RLC entity must transmit a status report to its peer in the next transmit opportunity. The TX entity will check whether the status report is required, by checking a boolean in the RX entity, that is set upon receiving the polling bit.

If the status report is required, the TX entity will retrieve a cached status report from the RX entity. This cached status report is updated at the reception of every PDU, to avoid blocking blocking the MAC generating a status report during the *pull_pdu()*.

An illustration of the process of generating the Status Report can be found in `rlcstatusgeneration`.

When the RLC receives a Status Report, it must be passed to the TX entity for processing. The TX entity will use the received status report to update the TX window and the RETX queue. Because both the *UE executor* thread and *Cell Executor* thread can update both the TX window and RETX queue, both

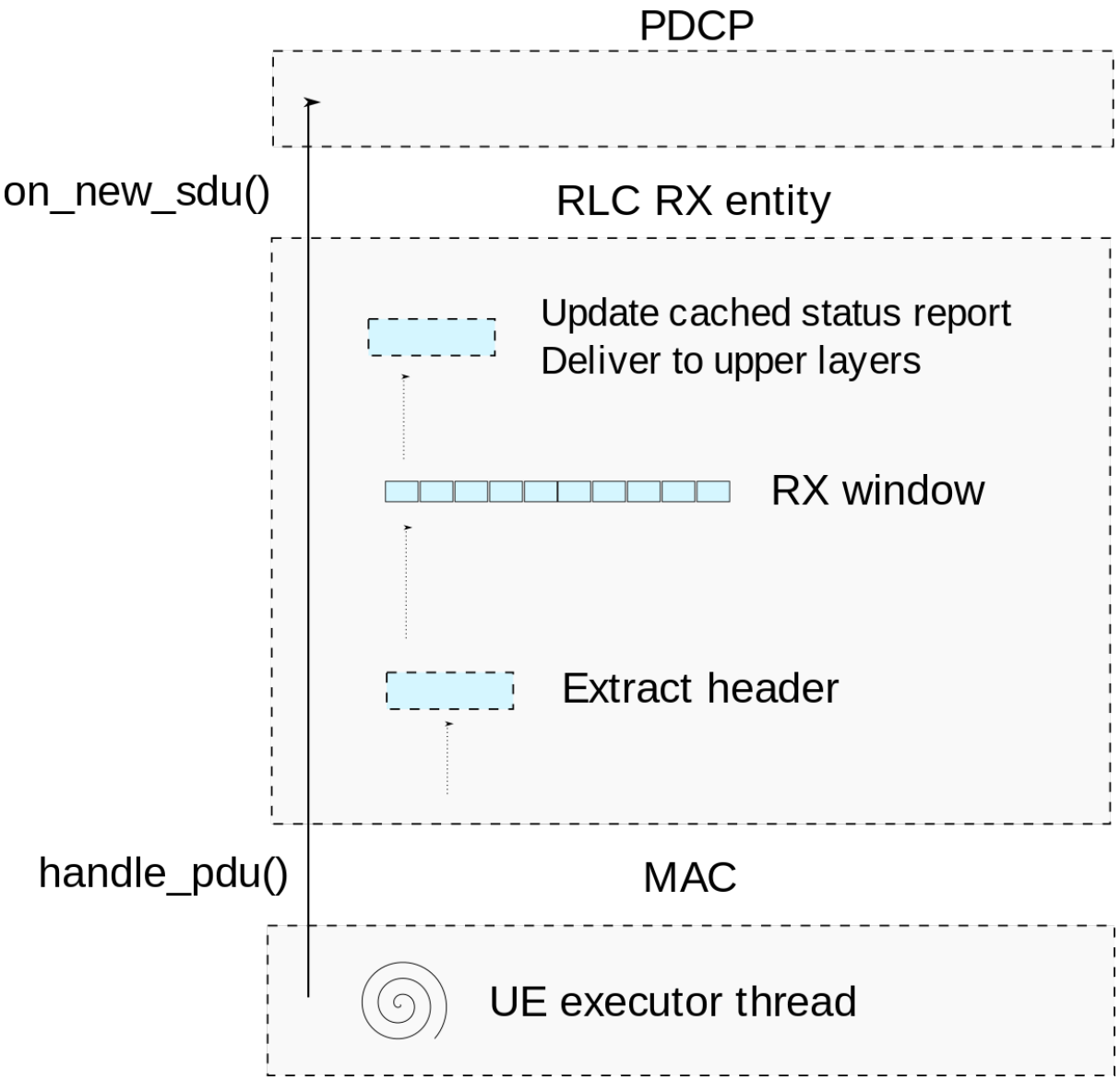


Fig. 8: Rx entity data path

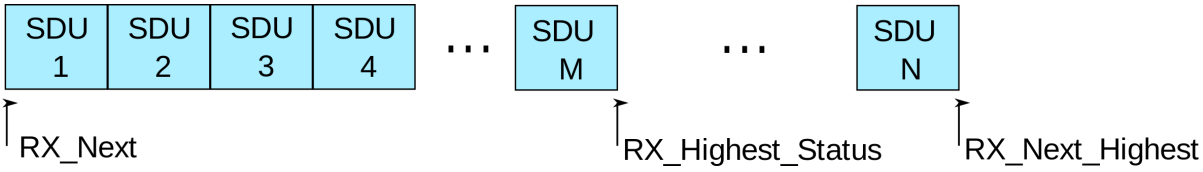


Fig. 9: Rx window

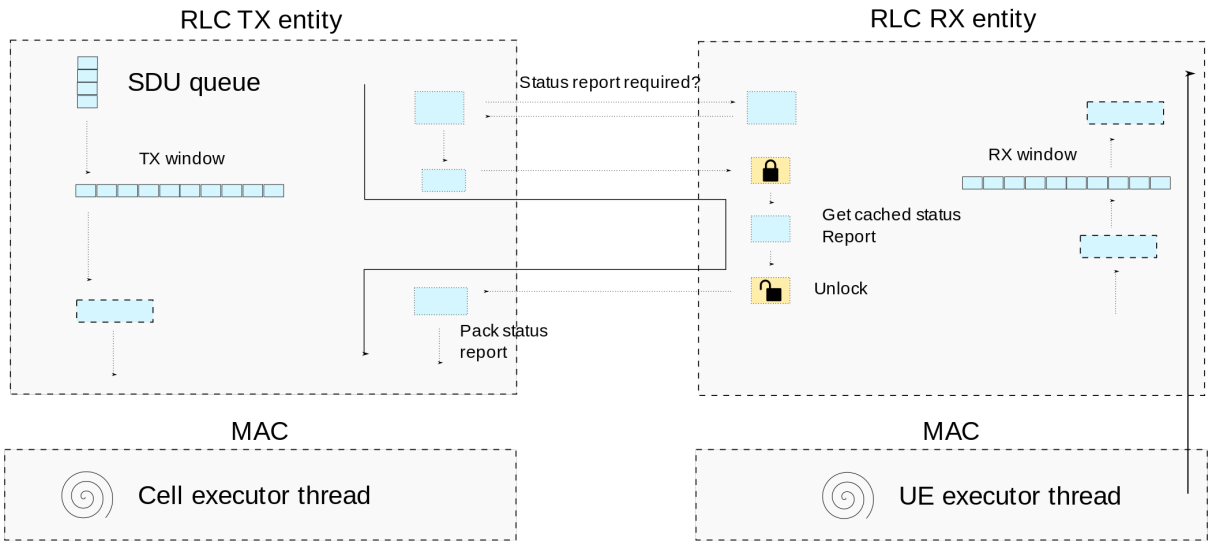


Fig. 10: Status Report transmission

of these variables need to be protected with a lock. An illustration of the process of handling the Status Report can be found in `rlcstatushandling`.

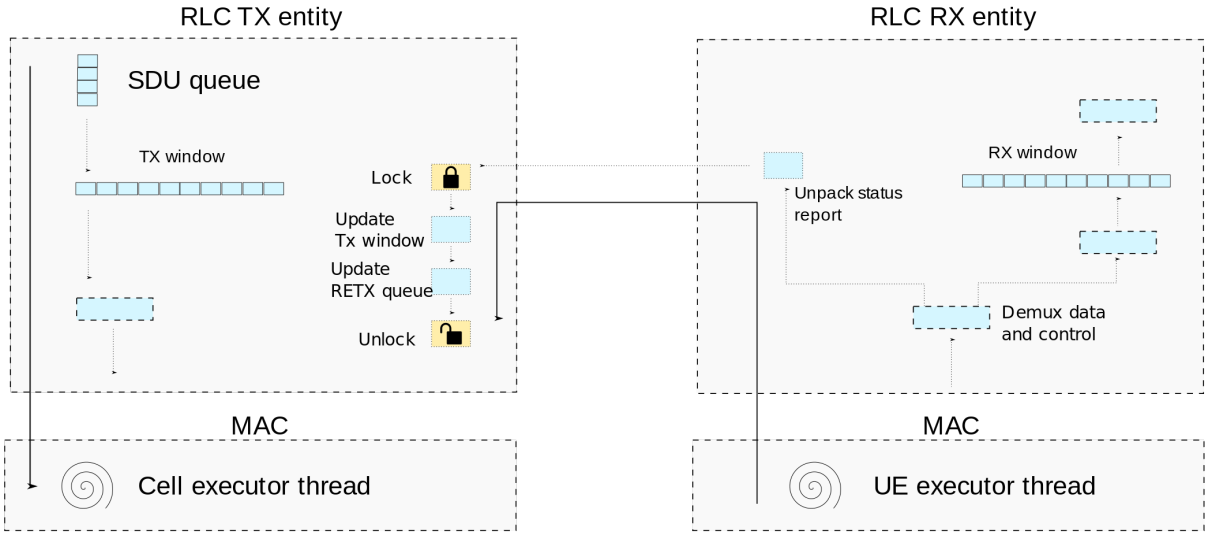


Fig. 11: Status Report handling

MAC buffer status reporting

The MAC needs to know the size of the buffer in the RLC TX entity to give appropriate grants. As polling by the MAC is inefficient when there are many inactive UEs and bearers, the RLC will push the new state of the buffer whenever this value is updated. This can be done when a PDU is pulled, when a PDU is received, or when a timer expires that, for example, requires an update to the status report.

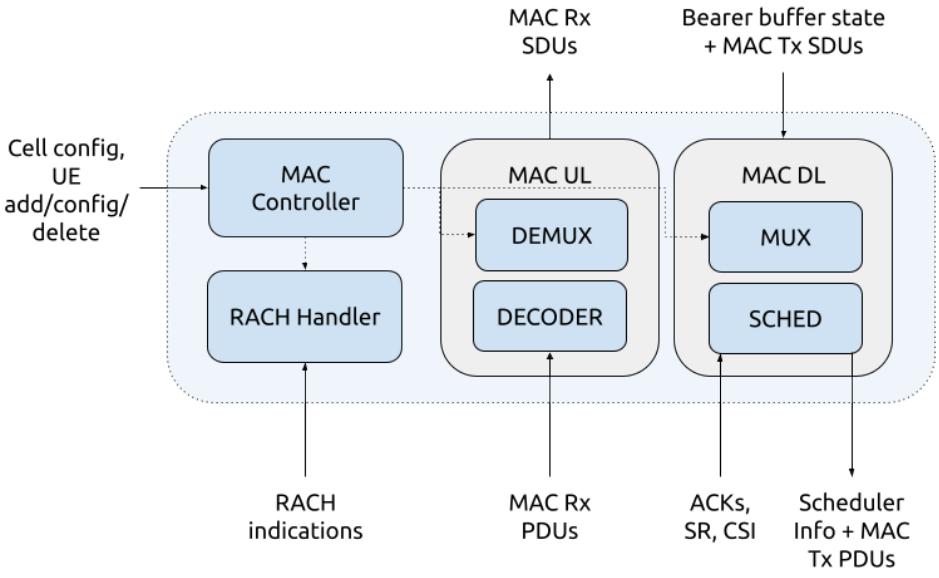
As this can be done from any thread, cached sizes of all queues and any pending status report are kept, so we can update the buffer state with minimal blocking to the MAC.

13.3.3 MAC

The main responsibilities of the MAC are:

- Encoding and decoding of MAC PDUs that are sent/received to the PHY via the FAPI interface
- Contains the gNB scheduler, which allocates DL/UL grants for System Information, Paging, UE data (RLC PDUs + MAC CEs) and Random Access handling.
- Demultiplexing and forwarding of decoded MAC Rx SDUs to their respective logical channel.
- Handle received MAC CEs
- PRACH handling and RNTI allocation

Implementation



The MAC is divided into the following sub-components:

- **MAC Controller:**

  Translates DU configuration requests into configuration commands that can be understood by the remaining MAC sub-components. The MAC controller ensures that the other components are configured with minimal service disruption in terms of traffic latency and avoiding any race conditions. The configuration commands that the DU manager sends to the MAC controller include the addition of new DU cells and addition/reconfiguration/removal of UEs. This is implemented in the `mac_controller` class.

- **RACH Handler:**

  Manages the allocation of RNTIs for the received PRACH preambles and association of reach RNTI to a DU UE Index. Which is implemented in the `rach_handler` class.

- **MAC UL Processor:**

  Decodes the received MAC PDUs and forwards the resulting MAC SDUs to their respective logical channels using the DEMUX component and forwards the UL Buffer Status Reports to the Scheduler. This is implemented in the `mac_ul_processor` class.

- **MAC DL processor:**

  Manages the MAC scheduler. This is implemented in the `mac_dl_processor` class.

RA Procedure

The Random Access procedure is handled primarily by the DU MAC in the following steps:

1. The RACH handler manages the allocation of temporary RNTIs (TC-RNTIs) for each of the detected PRACH preambles in a given slot, and forwarding of these preambles and TC-RNTIs to the scheduler.
2. The scheduler is then responsible for allocating the RAR and Msg3 grant for each detected PRACH preamble.
3. The MAC DL processor has the role of encoding the MAC DL PDUs sent to the PHY.

This leads to the following messaging flow graph:

It is worth noting, that the creation of a new UE in the DU is deferred until Msg3 is received. This design has the following advantages:

- UE Resources are not allocated unnecessarily for the cases when a phantom PRACH is detected, the UE fails to detect the RAR, or the Msg3 is not correctly decoded by the gNB.
- UE Resources are allocated after the RAR window has passed. With this design, any latency in the UE resource allocation thus won't affect the ability of the scheduler to timely schedule RARs. Notice that the RAR window in NR can be relatively short, depending on the type of service provided.

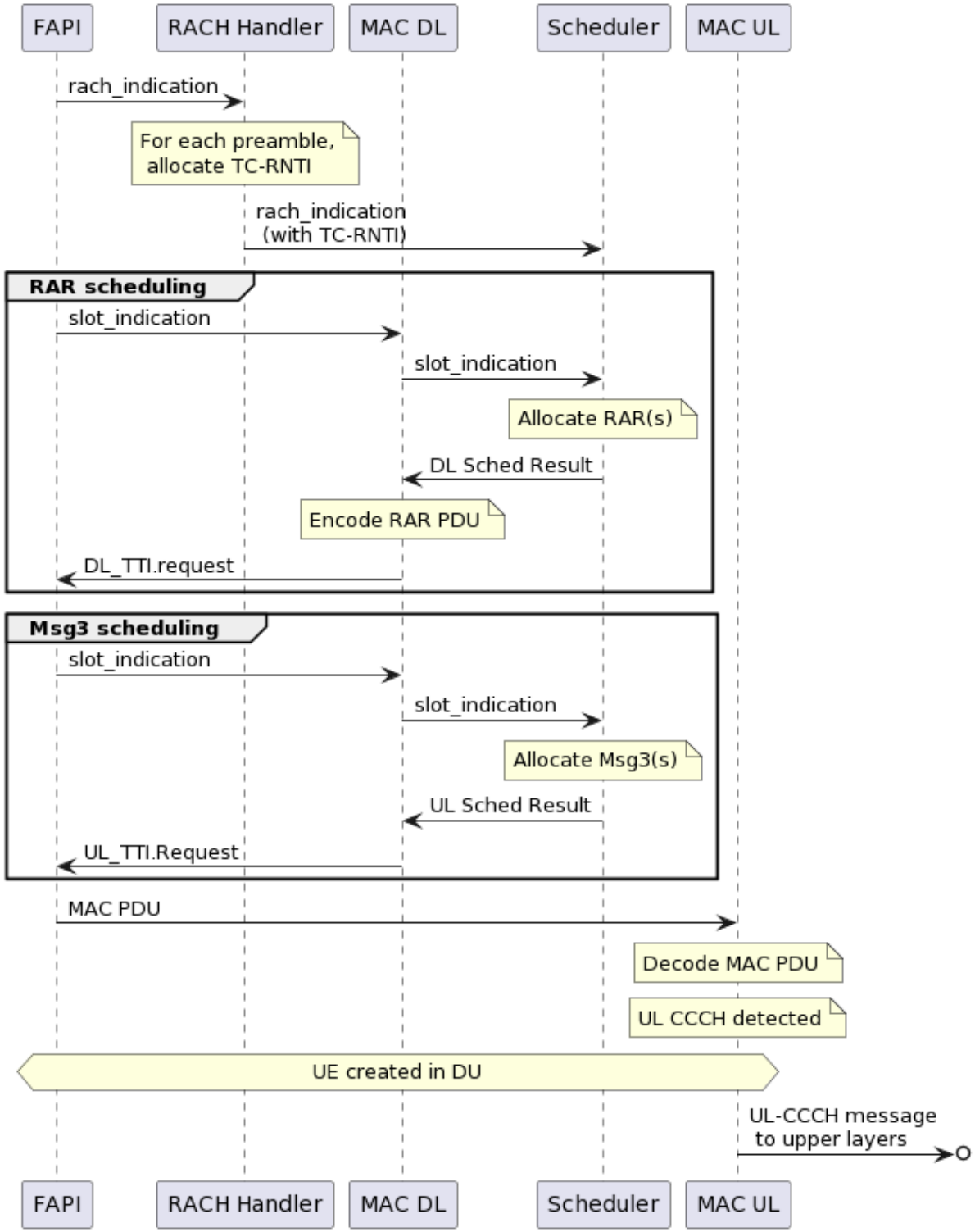
13.4 DU-low

The DU-low, or Distributed Unit - Low, is responsible for the handling of both uplink and downlink traffic. Specifically, it handles the Upper PHY processing related to these signals. The DU-low, contains only the Upper PHY and has two main interfaces. The DU-low communicates directly with the DU-high via the FAPI interface, and with the RU via the Open FrontHaul (OFH) interface. The lower PHY processing is carried out in the RU. This architecture is specific to ORAN Split 7.2. You can read more about split 7.2 *here* .

Return to top level architecture diagram.

Components:

- Upper PHY: The Upper PHY handles the processing of UL and DL signals coming from and to the RU.


FAPI
RACH Handler
MAC DL
Scheduler
MAC UL
rach_indication
For each preamble,
allocate TC-RNTI
rach_indication
(with TC-RNTI)
RAR scheduling
slot_indication
slot_indication
Allocate RAR(s)
DL Sched Result
Encode RAR PDU
DL_TTI.request
Msg3 scheduling
slot_indication
slot_indication
Allocate Msg3(s)
UL Sched Result
UL_TTI.Request
MAC PDU
Decode MAC PDU
UL CCCH detected
UE created in DU
UL-CCCH message
to upper layers
FAPI
RACH Handler
MAC DL
Scheduler
MAC UL

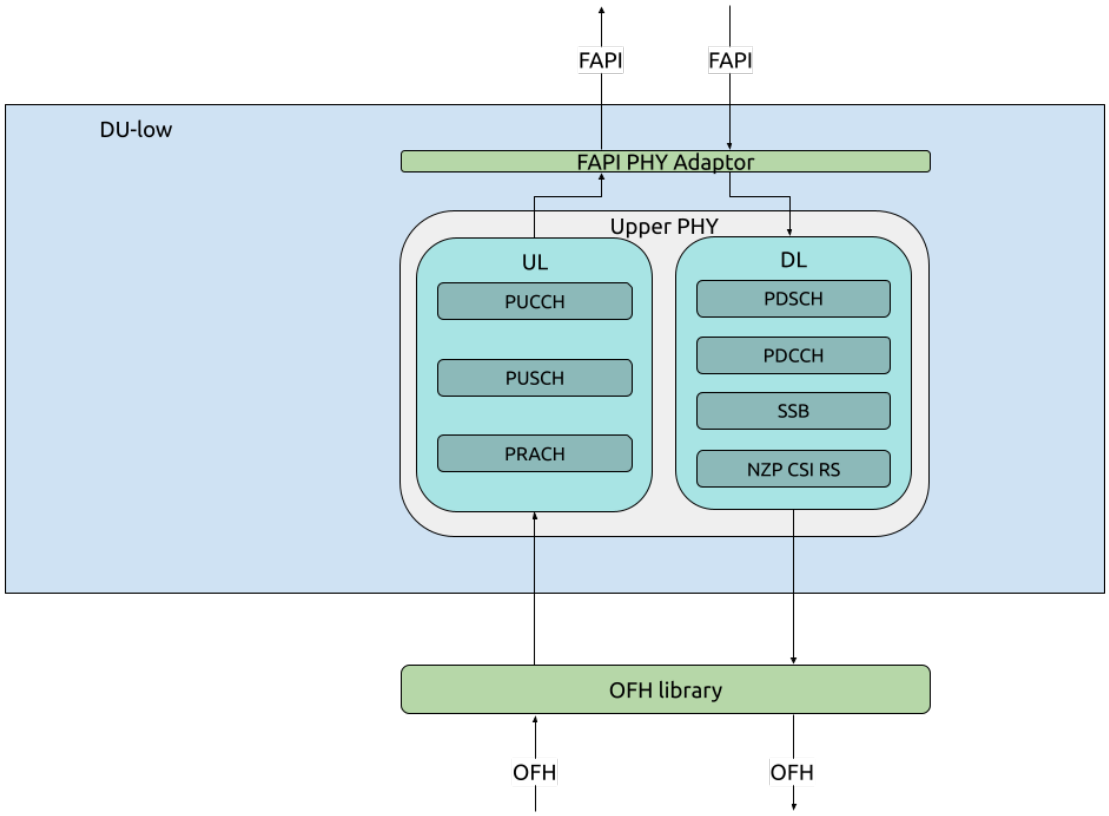


Fig. 12: srsRAN Project DU-low implementation

Interfaces:

- FAPI: Interfaces with the MAC in the DU-high.
- OFH: Interfaces with the lower PHY in the Radio Unit (RU).

CHAPTER

FOURTEEN

CODE STYLE GUIDE

This document establishes the basic guidelines and recommendations for C and C++ programming styles within the srsRAN Project code base. Many of the guidelines are C++ specific, however some can still be applied to C. The main goal of this document is to help establish a consistent programming style throughout the srsRAN Project code base, and improve the readability and maintainability of code committed by users.

This document is heavily inspired in LLVM's coding standards, please refer to this if you would like to take a further look into coding standards and best practices.

Contents:

14.1 Language and libraries

14.1.1 C & C++ version

The srsRAN Project project is written in standard C++14, thus it is important to avoid the use of compiler specific extensions to ease code portability and adding support for additional toolchains in the future.

If you need to use custom features such as builtin functions, guard them behind a macro or a function and always provide a fallback mechanism for toolchains where this feature is not available.

srsRAN Project is compiled and tested using the following toolchains with the following minimum requirements:

- GCC 9.4.0
- Clang 10.0.0

Note: We recommend using Ubuntu 22.04, with GCC 11.3.0 and Clang 14.0.0

14.1.2 Use of the C++ standard library

It is really encouraged to use the C++ standard library to avoid reinventing the wheel for many common programming tasks and utilities.

When writing C++ code, it is preferable to use the C++ library functions instead of falling back to the C library if possible. For example: `std::this_thread::sleep_for()` vs `::usleep()`.

Additionally, the srsRAN Project code base provides many additional support utilities and custom abstract data types. So before implementing a new utility or container, please check if it has been already implemented or if an existing one could be extended.

When both C++ and the srsRAN Project support libraries provide similar functionality, and there is a reason to not favor the C++ implementation, it is generally preferable to use the srsRAN Project support library, especially for custom containers found in the ADT folder.

14.2 Mechanical source aspects

14.2.1 Source Code Formatting

The srsRAN Project project uses a defined set of formatting rules, and heavily relies on clang-format for automatic source code formatting. You must use the custom clang-format file in the root of the repository, see here for more information about the options found in this file.

If you are interested in specific rules regarding spaces, maximum width, etc., then please check the contents of the above file.

When adding new commits to the repository, please make sure you have formatted your new changes with clang-format **before** submitting them.

14.2.2 Comments

Comments have to be regarded as an essential part of the code, as they greatly facilitate reviewers' and maintainers' tasks. From this perspective, comments should describe what the code is trying to do and why, without getting into the implementation details at a micro level and, most importantly, targeting an audience that may not be as familiar as you are with the topic at hand.

The following brief sections provide a very rough introduction to how developers are expected to comment code. A more thorough guide can be found *here*.

Comment Guidelines

- Write comments as English prose, using proper capitalization, punctuation, etc., ending them with a period sign. Words are spelled in American English, as given by the main entry of The Merriam-Webster Dictionary.
- Comment lines should start with a double slash for *normal* comments and with a triple slash for *documentation* comments. The double/triple-slash format should be used always, also for multiline comments.

```
/// \brief Does something.
///
/// More details about the function.
void some_func(){
  // Now we print a simple text.
  fmt::print("Some comment.");
}
```

- Comments should always start on a new line above the code that is being commented. Specifically, documentation comments should precede the class or function declarations and not go inside their body. Normal comments cannot be placed next to the code or outside the body of a function.

```
// Some general comment outside the body of a function - WRONG!

/// Does something. - OK
void some_func(){
  // Now we print a simple text. - OK
  fmt::print("Some comment.");

  fmt::print("Something else."); // Side comment. - WRONG!
}
```

- In cases where indentation is helpful to support the content of the comment, the suggested form is to start the line with one or more "greater than" (>) signs, depending on the indentation level. For instance, the comments in the example below suggest that the PUCCH configuration is a step within the UL configuration.

```
uplink_config srsran::config_helpers::make_default_ue_uplink_config(const␣
↪cell_config_builder_params& params)
{
  // > UL configuration.
  uplink_config ul_config{};
  ul_config.init_ul_bwp.pucch_cfg.emplace();

  // >> PUCCH configuration.
  auto& pucch_cfg = ul_config.init_ul_bwp.pucch_cfg.value();

  // ... more code ...
}
```

- C-style comments `/* */` are, in general, not allowed. The only exceptions are file headers (see below) and the documentation of parameters in a function call (very useful when, e.g., passing a `bool` or a `nullptr` as arguments).

```
// Hard to see what "true" and "nullptr" actually refer to.
obj.method(a, b, true, nullptr);
// Easy to see what the input values actually do.
obj.method(a, b, /*enable_X=*/true, /*options=*/nullptr);
```

- Merge requests should **not** contain lines of code that have been commented out. If you really need to do it for documentation purposes or maybe for debug printing, use `#if 0` and `#endif`. These nest properly and are better behaved in general than C-style comments.

File Headers

Every source file of the project should have a copyright header and a short description of the basic purpose of the file. The standard header looks like the example below.

```
 /*
 *
 * Copyright 2021-2023 Software Radio Systems Limited
 *
 * This file is part of srsRAN.
 *
 * srsRAN is free software: you can redistribute it and/or modify
 * it under the terms of the GNU Affero General Public License as
 * published by the Free Software Foundation, either version 3 of
 * the License, or (at your option) any later version.
 *
 * srsRAN is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU Affero General Public License for more details.
 *
 * A copy of the GNU Affero General Public License can be found in
 * the LICENSE file in the top-level directory of this distribution
 * and at http://www.gnu.org/licenses/.
 *
*/
```

14.2.3 Header Guards

All header files should have an include guard to prevent double inclusion. The srsRAN Project codebase uses the `#pragma once` directive, which is widely supported by common compilers. Unlike conventional include guards (via `#ifndef` and `#define`), neither a unique identifier nor a closing expression (`#endif`) is required.

The following example shows this:

```
/*
 * File header...
 */

#pragma once

#include "foo.h"
#include <file.h>

namespace srsgnb {
// ...
} // srsgnb
```

14.2.4 #include Style

Try to include a minimal list of `#include` and keep it clean of redundant header files as dependencies change. To that end, it is OK to exploit the fact that includes propagate transitively. So, if for instance *foo.h* already includes *bar.h*, only *foo.h* needs to be included when using functions or classes from both files.

The include list should be immediately after the header file comment, and after the include guards if working on a header file. Include files should be listed in the following order:

1. Main module header.
2. Local and private headers.
3. SRSRAN project or subproject headers (srsgnb/..., srsue/..., srsran/..., etc).
4. System library includes.

Keep each category sorted lexicographically by the full path and avoid adding newlines between categories or include directives. The main module header should be always the first in the list. Sub-project headers should be included before srsran headers (from most specific to least specific eg:srsgnb before srsran).

```
#include "my_class_header.h"             // category 1
#include "private_module_utils.h"        // category 2
#include "srsenb/hdr/public_header.h"    // category 3
#include "srsran/adt/bounded_vector.h"   // category 3
#include <string>                        // category 4
#include <vector>                        // category 4
```

Use C++ library header files in C++ files instead of using the C library headers, eg: `cstring`, `cassert`, `cstdint`, etc.

As a final note, `clang-format` will lexicographically sort all includes files automatically for you.

14.2.5 Language and compiler aspects

Treat compiler warnings as errors

Avoid submitting code that generates compiler warnings. Sometimes you may get a false positive warning, in this case find a way to suppress it.

Code Portability

Try to write portable code under all circumstances. If you are under the situation where you need to do something that is not portable, then put it behind a documented interface so it is centralized in a single place and not scattered in different places.

Avoid RTTI and Exceptions

In C++ code, **do not** use exceptions or RTTI (runtime type information eg:`dynamic_cast<>`).

Use of auto

In general, use auto when it makes the code more readable and easier to maintain. Typical cases would be for iterators or complex template types. Some examples:

```
auto *p = std::unique_ptr<int>(new int);  // good: easy to infer that p is an␣
↪int*
auto i = my_map.begin();                  // good, easy to see it is an␣
↪iterator and avoids writing std::unordered_map<std::string, int>::iterator
auto a = my_func();                       // bad: what type does my_func␣
↪return?
```

14.3 Style Aspects: High Level Issues

14.3.1 Self-contained Headers

Header files should be self-contained, this means that they can compile on their own.

In general, a header may be implemented by one or more source files. Each of these source files will include the header file that defines the interface first in the include list (see include style).

14.3.2 Using `#include` Sparingly

Headers included in other header files propagate transitively. For instance, if header file `foo.h` includes header file `bar.h`, any file that includes `foo.h` will also include `bar.h` implicitly. To avoid long compilation times, includes of large header files should be taken with care.

When you are about to include a large header file `large_include.h` in another header file `my_module.h`, consider the following improvements:

1. Can `large_include.h` be included by the respective source file `my_module.cpp` instead? This will avoid the transitive propagation of `large_include.h` by any file that includes `my_module.h`.
2. Can `my_module.h` just include a subset of the `large_include.h` included headers? E.g. `my_module.h` may just need access to interface definitions, rather than derived class implementations.
3. If 1. is not possible without altering code, consider making forward declarations of `large_include.h` types and use reference/pointers in `my_module.h`. Then, place the `large_include.h` header in `my_module.cpp` instead.

14.3.3 Using "Internal" Headers

When writing a module, avoid adding implementation details, helper classes and utility functions in the public interface header that are only meant to be used by the module internally. Instead, create a private header file in the same directory as the source files and include it locally. This will avoid polluting the public interface with unnecessary dependencies reducing recompilations.

14.3.4 Use of `namespace`

Avoid opening namespace blocks inside a .cpp file that spans for the complete file.

When adding a free function implementation in a .cpp file, avoid opening namespace blocks. Instead, use the namespace qualifier to ensure that the function definition matches the new declaration.

```
// foo.h
namespace srsran {
int foo(const char *s);
}

// foo.cpp
#include "foo.h"
using namespace srsran;
int srsran::foo(const char *s) {
  // ...
}
```

Avoid this:

```
// foo.cpp

#include "foo.h"
namespace srsran { // Namespace block.
int foo(char *s) { // Mismatch between "const char *" and "char *"
    // ...
}
} // end namespace srsran
```

The above code snippet will add a new overload of srsran::foo instead of providing a definition of the existing function declared in the header. To make things a bit worse, this error will be detected by the linker towards the end of the build, instead of being detected instantly by the compiler.

A final word about namespace qualification: when you need to call a C library or system function like `read()`, `socket()`, etc, you should prefix the calls with `::` such as `::socket()`, this will avoid potential name clashes and unexpected behavior.

14.3.5 Using "early exits" and `continue`

When reading code, keep in mind how much state information and how many previous decisions have to be remembered by the reader to understand a block of code. Aim to reduce indentation where possible, unless doing so would make the code more readable. One great way to do this is by making use of early exits and the `continue` keyword in long loops.

This example does not use early exits:

```
int *my_function(my_class *p) {
  if (!p->f1() && p->f2() &&
      my_other_function(p)) {
    // ... some long code ....
  }

  return nullptr;
}
```

This can be transformed to:

```
int *my_function(my_class *p) {
  // f1 for this p is true because ...
  if (p->f1()) {
    return nullptr;
  }

  // f2 for this p is false because ...
  if (!p->f2()) {
    return nullptr;
  }

  // Something else ...
  if (!my_other_function(p)) {
    return nullptr;
  }

  // ... some long code ....
}
```

Similarly, in loops:

```
for (int i : my_vector) {
  if (i % 7 == 0) {
    if (check_something(i)) {
      do_something(i);
    } else {
        // ... some long code ....
    }
  }
}
```

Can be transformed to:

```
for (int i : my_vector) {
  if (i % 7 != 0) {
    continue;
  }
  if (check_something(i)) {
    do_something(i);
    continue;
  }
  // ... some long code ....
}
```

14.3.6 Avoid `else` after a `return` statement

Following the same reasoning as the previous point, avoid using `else` or *else if* after a statement that interrupts control flow like return, break, continue, goto, etc.

For example:

```
case 2: {
  if (something) {
    v = get_buffer_type_1();
    if (!v.is_valid()) {
      error_string = "Invalid buffer type 1";
      return -1;
    } else {
      break;
    }
  } else {
    v = get_buffer_type_2();
    if (!v.is_valid()) {
      error_string = "Invalid buffer type 2";
      return -1;
    } else {
      break;
    }
  }
}
```

Can be transformed to:

```
case 2: {
  if (something) {
    v = get_buffer_type_1();
    if (!v.is_valid()) {
      error_string = "Invalid buffer type 1";
      return -1;
    }
  } else {
    v = get_buffer_type_2();
    if (!v.is_valid()) {
```

(continues on next page)

(continued from previous page)

```
      error_string = "Invalid buffer type 2";
      return -1;
    }
  }
  break;
}
```

Or optimally to:

```
case 2: {
  if (something) {
    v = get_buffer_type_1();
  } else {
    v = get_buffer_type_2();
  }

  if (!v.is_valid()) {
      error_string = (something) ? "Invalid buffer type 1"
                                 : "Invalid buffer type 2";
      return -1;
  }
  break;
}
```

This way helps to understand the code better as you need to keep track of less context and reduces indentation.

14.3.7 Use of Static Helper Functions

It is very common to write small loops that just compute a boolean value, for example:

```
bool found_valid = false;
for (unsigned i = 0, e = vector.size(); i != e; **i)
  if (vector[i]->is_valid()) {
    found_valid = true;
    break;
  }

if (found_valid) {
  ...
}
```

Instead of writing this loop inside a bigger function, extract it to a predicate function (usually static) that also uses early exits:

```
/// Returns true if the input vector contains a valid buffer.
static bool has_valid_buffer(const std::vector<buffer *> &vector) {
  for (unsigned i = 0, e = vector.size(); i != e; **i) {
    if (vector[i]->is_valid()) {
```

(continues on next page)

(continued from previous page)

```
      return true;
    }
  }
  return false;
}
...
if (has_valid_buffer(vector)) {
  ...
}
```

Writing it this way has many benefits:

1. Reduces indentation.
2. Factors code that can be potentially reused by other places.
3. Forces the programmer to give a name for the function, converting a piece of code into a concept which is easier to understand by the reader (a raw loop vs a function call with name `has_valid_buffer()`).
4. Includes a documentation block for the function.

Please extrapolate this example to more complex cases where the predicate is not as obvious. Instead of being faced with all the inline details of how the predicate is checked, we can trust the function and continue reading with better locality.

14.4 Style Aspects: Low Level Issues

14.4.1 Naming Conventions

Names should be coded following the snake_case pattern. For variables, functions and classes the name schema will always be lowercase and separate the words by an underscore. Exceptions are SI units and other common units with uppercase letters (dB, dBm) which should be kept in their original form. Names should be descriptive and easy to read and avoid abbreviations unless they are well known.

```
int n_pdu          // Not so good, it's difficult to imagine what represents.
int number_of_pdu  // Good, descriptive name.
int pdu_count      // Good, descriptive name.
int path_loss_dB   // Good, descriptive name with unit suffix.
```

Non-descriptive names should be avoided except for iterators in `for` loops or variables within a very small scope.

```
for (unsigned i = 0, e = size; i != e; ++i) { // i and e are iterators inside␣
↪a for loop.
  std::lock_guard<std::mutex> lck(mutex);     // Good as the scope is only 2␣
↪lines long.
  detach(i);
}
```

Names should follow these rules (always use snake_case):

- **Type names** (classes, structs, enums, typedefs, ...) should be nouns.
- **Variable names** should be nouns (they represent state).
- **Function names** should be verb phrases (they represent actions) and command-like functions should be imperative.
- **Enum declarations** are types, so they should follow the naming conventions for types.
- **Enumerators** (eg: *enum { red, green }*) should follow the naming conventions for types. Enumerators that are convenience constants should be written in uppercase. For instance:

```
enum {
  MAX_ELEMENTS = 32,
  BYTES_PER_ELEMENT = 64
};
```

14.4.2 Assert and expect

The srsRAN project provides a custom assert macro called `srsran_assert`. Use it as much as you can to check all your preconditions and assumptions. This will help to reduce debugging times as the assert may be triggered by your code or even by external faulty code.

Make sure to write a descriptive error message in the assert statement, which will be printed when the assertion is triggered. For example:

```
char *get(unsigned idx) {
  srsran_assert(idx < v.size() && "get() out of range!");
  return v[idx];
}
```

Other additional examples:

```
srsran_assert(buffer->is_valid() && "Buffer should always be valid!");
srsran_assert((channel == DLSCH || channel == ULSCH) && "Channel type is␣
↪invalid!");
srsran_assert(idx < get_num_ues() && "UE index value is out of range!");
srsran_assert(v1.size() == v2.size() && "vector sizes must be identical!");
```

Note: If an error condition can be triggered by user input then do not use an assert, instead, use a recoverable error mechanism.

Warning: which?

Another nice side effect of using assertions is that you can apply the "design by contract" approach for many interfaces, helping to reduce a lot of error checks and error handling clutter. As software is usually layered, you may validate user inputs in a certain layer, keeping it centralized in a single place and then let inner layer interfaces be designed by contract by using assertions. Using this approach will benefit the appearance of simpler interfaces, reduces corner cases and the implementation will have less states to validate which will reduce bugs in untested corner cases.

When using assertions you may get warnings for *"unused value"* if assertions are disabled (mainly in release builds). For example:

```
unsigned size = v.size();
srsran_assert(size > 42 && "Vector smaller than it should be");

bool is_value_new = set.insert(x);
srsran_assert(is_value_new && "The value shouldn't be in the set yet");
```

In the first case, the call to `v.size()` is only useful in the assert, and we don't want it executed if assertions are disabled. In this case the code should be moved inside the assertion. In the second case, the side effects of the call must happen whether the assert is enabled or not. In this case, the value should be cast to void to disable the warning.

```
srsran_assert(v.size() > 42 && "Vector smaller than it should be");

bool is_value_new = set.insert(x);
(void)is_new_value;
srsran_assert(is_value_new && "The value shouldn't be in the set yet");
```

Warning: write something about expect

14.4.3 Do not use `using namespace std`

When you need to refer to identifiers in the standard library then prefer to explicitly use a `std::` prefix rather than relying in `using namespace std;`. In header files, adding a using namespace directive will pollute the namespace of any source file that includes this header, causing maintenance issues.

The exception to this rule (not for the `std` namespace) is for implementation files (`.cpp`). For example, all the code in the srsRAN project implements code that lives in the srsran namespace. In this case, it is clearer for the .cpp files to have a using namespace srsran directive at the top of the file, just after the include list. This will reduce indentation in the body of the file.

The general form of this rule is that any .cpp file that implement code in any namespace may use that namespace, including its parents, but should not use any others.

14.4.4 Using Range for Loops

Since the introduction of range-based for loops in C++11, it is rarely necessary to do any kind of explicit manipulation of iterators. Try to use range-based for loops wherever possible, for example:

```
for (const auto &ue : ue_db)
  ... use ue ...
```

14.4.5 Loop Structure

In cases where a range-based for loop cannot be used and it is necessary to write an explicit iterator-based loop, pay attention to the whether end() is re-evaluated on each loop iteration. The most common mistake is writing a loop this way:

```
for (auto i = x.begin(); i != x.end(); ++i)
  ... use i ...
```

The problem is that it evaluates `x.end()` on each iteration. Instead, use loops that evaluate it once before the loops starts. This can be done using this form:

```
for (auto i = x.begin(), e = x.end(); i != e; ++i)
  ... use i ...
```

Note: These two loops have different semantics: if the container is being mutated inside the loop, `x.end()` may change its value every time through the loop, so the second form may not be correct. If you actually depend on this behavior, please write the loop in the first form and add a comment indicating you did it intentionally.

By consistently using the second form, the reader will implicitly see that the loop is not mutating the container without needing to analyse the loop body, making easier to read and understand what the code does.

In general, the C++ syntax for iterator comparison in loops is to use the `!=` operator instead of `<`. This should be used consistently, even if the iterator is a plain integer.

For example, the preferred form looks like this:

```
for (unsigned i = 0, e = x.size(); i != e; ++i)
  ... use i ...
```

Instead of the C-style way:

```
for (unsigned i = 0, e = x.size(); i < e; ++i)
  ... use i ...
```

14.4.6 Using Pre-increment

In C++ (does not apply to C code), pre-increment (++x) may be no slower than post-increment (x++) and could be a lot faster than it. As a result, it is preferential to use pre-increment whenever possible.

14.4.7 Use of Anonymous Namespaces

Use an anonymous namespace for making the contents surrounding it private to the file, just like `static` is used for C functions and global variables.

The problem with anonymous namespaces is that they encourage indentation of their body, and they reduce locality of reference: if you see a random function definition in a C++ file, it is easy to see if it is marked as static, but seeing if it is inside an anonymous namespace may require scanning a big chunk of the file.

For these reasons, follow this simple rule: make anonymous namespaces as small as possible and only use them for class declarations, **not** for functions.

```
namespace {
class foo { // foo class has internal linkage
  ...
public:
  int bar(); // define this method outside the anonymous namespace.
  ...
};
} // end anonymous namespace

static void some_helper() { // static function marked as static outside␣
↪anonymous namespace.
  ...
}

int foo::bar() { // method definition outside of anonymous namespace to keep␣
↪namespace as compact as possible.
  return 42;
}
```

14.4.8 Using C++ Casts

Avoid using C-style casts in C++ code.

C++ casts are more explicit to read in code than the C counterpart. Another reason is that C++ has four different types of casts, allowing the programmer to choose the right tool depending on the circumstances. While the C cast may be used for any case and potentially disabling any warnings from the compiler.

14.5 Recommendations

14.5.1 Function and Class Length

Avoid writing classes or functions that are too large.

As a rule of thumb, functions should fit in a window and no scrolling should be needed to review them. Try to write functions that only do one thing or action. If you write a function that does more than one thing, try to split it into multiple functions, each of which do a single thing. This will dramatically improve code maintainability. By using concise function names, and by having each function well documented, code will be much easier to understand.

For classes, it is harder to set a maximum number of lines, so instead, give a class a single responsibility. Although the concept of responsibility may sound abstract, another way of thinking about it is that a class should only have "one reason to change". If a class does multiple things or responsibilities, it is better to split it into multiple classes. This way we may avoid falling into the *god class* anti-pattern which can be a maintenance nightmare.

14.5.2 Scope

Define variables where they are going to be used, making their scope as short as possible. This applies to both C++ and C code.

For example:

```
int i;
float f;
...
... other unrelated code ...
...
f = get_float();
a = f * get_multiplier();
```

Should be transformed to:

```
...
... other unrelated code ...
...
float f = get_float();
int i = f * get_multiplier();
```

14.5.3 Logical Operators

The codebase uses the standard form of logical operators:

-&&, || and !

As opposed to the alternative form:

- and, or and not

The following code block shows the correct and incorrect use of logical operators:

```
if (!a && (b == 8))     // Correct, only used the standard form.

if (not a && (b == 8))  // Incorrect, used 'not' instead of the standard form '!
↪'.
```

14.5.4 Using References(&) Over Pointers(*)

In the general, it is better to use references instead of pointers. C++ references have a simpler syntax than pointers and always refer to a valid object (unless you do some undefined behavior).

Pointers are very useful when the pointed object may optionally be *nullptr*, however managing this case may require adding additional checks to manage this situation.

```
static bool check_objects_are_equal(foo *left, foo *right);              //␣
↪Bad, left or right may be nullptr, prefer references.
static bool check_objects_are_equal(foo left, foo right);                //␣
↪Bad, copies objects.
static bool check_objects_are_equal(foo &left, foo &right);              //␣
↪Bad, missing const, checking the objects should not modify them.
static bool check_objects_are_equal(const foo &left, const foo &right);  //␣
↪Good.
static void fill_optional_object(foo *x);                                //␣
↪Good, x is optional, function checks for nullptr.
```

14.5.5 Const Correctness

Write code that is `const` correct.

```
const object& get_object() const; // Return value cannot be modified, method␣
↪does not modify the contents of the class.
```

14.5.6 Avoid Complex Expressions

Try to use local variables to store intermediate results or to cache the result of a function call when calling it multiple times and it is known not to change. This will help to avoid complex expressions.

For example:

```
for (unsigned i = 0; i != get_bar_count(); ++i) {
  if (get_bar_count() < i) {
    do_something();
  }
  do_other();
}
```

Can be transformed to this code, saving many calls to `get_bar_count()`

```
unsigned bar_count = get_bar_count();
for (unsigned i = 0; i != bar_count; ++i) {
  if (bar_count < i) {
    do_something();
  }
  do_other();
}
```

There may be cases where you *need* to work with complex and deep nested structures. In these cases try to store intermediate members as pointers or references (if writing C++ code) and remember to use `const` if only reading the values. This will avoid needless typing, having more compact code and potentially improving performance for unneeded indirection chasing pointers.

For example:

```
for (uint32_t j = 0, je = metrics_tmp.size(); j != je; ++j) {
  metrics[j].dl.n_samples += metrics_tmp[j].dl.n_samples;
  metrics[j].dl.mcs += metrics_tmp[j].dl.n_samples * metrics_tmp[j].dl.mcs;
  metrics[j].ul.n_samples += metrics_tmp[j].ul.n_samples;
}
```

Can be transformed into:

```
for (uint32_t j = 0, je = metrics_tmp.size(); j != je; ++j) {
  downlink_metrics &dl = metrics[j].dl;  // store non const reference to␣
↪access the struct
  const downlink_metrics &dl_tmp = metrics_tmp[j].dl;  // const ref to access␣
↪the struct

  dl.n_samples += dl_tmp.n_samples;
  dl.mcs += dl_tmp.n_samples * dl_tmp.mcs;

  const uplink_metrics &ul_tmp = metrics_tmp[j].dl;
  ul.n_samples += ul_tmp.n_samples;
}
```

14.5.7 Magic numbers

Avoid the use of magic numbers. Instead, use a variable (`static`, `constexpr`, etc.): this way, you will be forced to give it a descriptive name, improving code readability.

```
if (b < 2.0)  // Bad
  ...

static constexpr float detection_threshold_dB = 2.0;
if (b < detection_threshold_dB)  // Good - we now know this is a detection␣
↪threshold value in dBs
  ...
```

14.5.8 Fixed Width Integer Types

Use the fixed width integer types (`uint32_t`, etc...) found in the `cstdint` and `stdint.h` header files when you need to ensure that a variable needs to be of fixed size, independent of the target architecture.

Usually you will need to use these types when defining message structures, interfacing hardware, network programming, etc... However, **avoid** using them when the type width is not important because the use of a fixed size type is a declaration done by the programmer that the affected code needs special treatment and careful handling.

Notice that all 32-bit architectures and above define `int` and `unsigned int` to be of 32-bits, so it is safe to use the generic types safely unless you know you will need a wider type.

> **Warning:** discussion - bring this point up

```
for (uint32_t cc_idx = 0; cc_idx != num_cc; ++cc_idx)
    // here we know we're never going to exceed 4 billion cc's - safe to use a␣
↪generic unsigned int (32 bits)

for (uint32_t i = 0, e = vector.size(); i != e; ++i)
    // how many elements will the vector hold?
    // a) if we know it is going to hold less than 4 billion, use unsigned␣
↪int (32 bits)
    // b) if in doubt, use size_t.
    // but as you can see there is no need to use a fixed width type, generic␣
↪types can be safely used here.
```

Avoid using the `long` type since it can be either 32 or 64-bits depending on the architecture. See the standard definition for more information.

14.5.9 Function signatures

When declaring a function, the list of arguments shall be ordered as follows:

1. Arguments passed by reference or pointer, if any, that are only used to store the function output. These arguments are documented by the Doxygen command `\param[out]`.
2. Arguments passed by reference or pointer, if any, that contain inputs to the function and will also be modified by the function itself. These arguments are documented by the Doxygen command `\param[in,out]`.
3. Arguments passed by value, constant pointer or constant reference, if any, that represent inputs to the function. These arguments are documented by the Doxygen command `\param[in]`. In particular, structures gathering a number of configuration parameters, shall appear last in the argument list.

For example:

```
/// \brief Does something.
///
/// This function writes a span, reads and modifies a vector, only reads the␣
```

(continues on next page)

(continued from previous page)

```
↪rest of arguments.
/// \param[out]    fnct_out       Main function output (recall that \c span␣
↪behaves like a pointer).
/// \param[in,out] my_vector      A vector of integers the function may read␣
↪and modify.
/// \param[in]     another_vector A vector of integers the function may only␣
↪read.
/// \param[in]     quantity       Another value the function needs to know␣
↪about.
/// \param[in]     cfg_struct     A structure describing the context the␣
↪function works in.
void my_function(span<int> fnct_out, std::vector<int>& my_vector, const␣
↪std::vector<int>& another_vector, int quantity, const config_t& cfg_struct);
```

14.5.10 Class Layout Example

This section describes the recommended layout of a class declaration, recommendations found inline.

```
/// Class description block.
class my_class : public interface_1 {
    // Define private constant configurable parameters first - easy to find␣
↪by class maintainers to be always in the
    // same place.
    /// Constant 1 explanation.
    static constexpr unsigned some_constant_1 = 2;
    /// Constant 2 explanation.
    static constexpr unsigned some_constant_2 = 7;

public:
    // Declare/define special members first (constructors, destructor, etc...)
    // No need to document special members unless something not trivial␣
↪requires an explanation.
    my_class() { ... }  // If the ctor body is big, move it down to the cpp␣
↪file.
    ~my_class() { ... }  // Likewise, if the dtor body is big, move it down␣
↪to the cpp file.

    /// Method 1 explanation block.
    void method1();
    // Separate method declarations with a single newline.
    /// Method 2 explanation block.
    void method2();

    // Try to group methods of the same category, accessors, modifiers,␣
↪utilities...

    /// Getter explanation block.
    something& get_something();
```

(continues on next page)

(continued from previous page)

```
    const something& get_something() const; // Feel free to provide two␣
↪identical explanation blocks or declare
                                            // both methods in a row, no need␣
↪of newline here.

    /// Getter explanation block.
    another_thing& get_anotherthing();  // Here we separated the const and␣
↪non const method declarations.
                                        // Please be consistent per class␣
↪basis.

    /// Getter explanation block.
    const another_thing& get_anotherthing() const;

    // No need to add documentation for overridden interface methods, as they␣
↪are already documented in the interface,
    // we don't want to keep two copies of documentation blocks since they can␣
↪easily diverge creating maintenance issues.
    void interface_method1() override;
    // Remember to add a newline between methods.
    void interface_method2() override;

    /// Enum description block.
    // Define the enum near the site where it is going to be used to improve␣
↪locality when reading the interface.
    enum class my_enum {
        enumerator1,
        enumerator2
    };

    /// Method using above enum description block.
    void method_uses_enum(my_enum e); // my_enum is declared above - easy to␣
↪find.

protected:  // After the public method section is done, declare the rest if␣
↪applicable (protected, private).
    // The audience here has changed to class developers, not public␣
↪interface users anymore, we avoid polluting the public
    // interface with implementation details
    /// Protected method explanation block.
    void prot_method();

private:  // Private method section.
    // This will be only read by users who need to understand implementation␣
↪details. Keep away of public interface.
    /// Private method explanation block.
    void private_method();

protected: // Declare member variables at the end. Keep them separated from␣
```

(continues on next page)

(continued from previous page)

```
↪methods.
    /// Protected member variable.
    unsigned protected_member_var;

private: // Last, declare private member variables.
    // Variable names should be descriptive, so most of the times there is no␣
↪need to document them.
    unsigned memb_var_1;
    void* memb_var_2;
    /// Protects access to memb_var_2.
    // In the case of a mutex it helps to add a short explanation of what is␣
↪protecting...
    mutable std::mutex m;
};
```

14.6 Self Generating Documentation

The srsRAN Project repository uses **Doxygen** to generate API documentation directly from the annotated C++ source files. In order to contribute to the API documentation with an homogeneous style, the following sections provide some general guidelines focusing on the most common code elements. Please refer to the Doxygen Documentation for a complete overview of the Doxygen commands and features.

14.6.1 General Aspects

The documentation must be written in English, with American English spelling as given by the main entry of The Merriam-Webster Dictionary. For all editorial matters (e.g., acronyms, plurals, capitalization, equations), the suggested reference is the IEEE Editorial Style Manual for Authors , especially *Section E* thereof.

All code entities are documented with a comment block just before the declaration of the entity. Placing the documentation block on the same line as the code element is not allowed, even for short, one-line comments. As mentioned *here*, all lines of a documentation comment block start with a triple slash. Also, the srsRAN Project documentation prefers the `\command` form of Doxygen commands (as opposed to `@command`).

Generally, a documentation block consists of a brief description (ideally, not more than one line) that starts with the command `\brief` and one or more paragraphs separated by empty lines. When no detailed description is provided, the `\brief` command in the brief description can be omitted. The brief descriptions should go directly to the point, without expressions like "The `myclass` class provides/ specifies…" (more on this below).

```
/// \brief Brief description of the code element.
///
/// Detailed description starts here. This can span several lines, randomly
/// filled in this example. Lorem ipsum dolor sit amet consectetur adipiscing
/// elit lacinia maecenas, hendrerit luctus libero vivamus elementum feugiat
/// torquent accumsan eleifend, diam orci aptent tincidunt a iaculis sed nisi.
///
```

(continues on next page)

(continued from previous page)

```
/// More detailed description. Paragraphs are separated by empty lines. Curae
/// arcu tempor urna convallis pulvinar conubia rutrum auctor, rhoncus nam
/// faucibus montes velit non molestie, nostra proin metus senectus sem tempus
/// tincidunt.
class example1;

/// Brief and only description of the code element.
class example2;
```

All code entities should be documented when declared. It is not required (actually, it is discouraged) to repeat the documentation block when defining the code entity. In particular, virtual methods are only documented when declaring the interface/base class and the documentation is not repeated when defining the implementation.

Other general recommendations are as follows

- When referencing technical specifications or similar documents, only reference specific sections and tables, since they do not change between releases, and avoid using page numbers. Consider including extracts of the document if they provide clarification.

```
/// The class implements TS38.211 Section 4.2.1.
```

- Look for examples inside the code about the spelling of acronyms and references. For instance, the recommended forms are TS38.211 (note the absence of space between TS and the number) or DM-RS.
- Whenever possible, use rigorous mathematical notation for formulas, equations and related matters. For instance, $(0, 1)$ denotes the open interval between 0 and 1 (that is, all real numbers x such that $0 < x < 1$), while $[0, 1]$ stands for the closed interval, with both endpoints included. Half-open intervals, e.g., $[0, 1)$ or $(0, 1]$, are also possible. Discrete sets are written in enumerated notation (e.g., $\{1, 4, 23\}$), with possible ellipsis when the meaning is obvious (e.g., $\{1, ..., 10\}$ for the first ten natural numbers, or $\{1, 1.2, 1.4, ..., 2.4\}$ for all numbers between 1 and 2.4 with increments of 0.2).
- Wrap code examples between `\code` and `\endcode` commands.

```
/// \code
class example_class {
  int useless_field;
  };
/// \endcode
```

- Use the `\c` command or the `<tt>...</tt>` tags for short inline bits of code, e.g. `\c variable_name` or `<tt> variable_name = 5 </tt>`.
- Start a paragraph with `\note`, `\warning` or `\remark` if you want to put extra emphasis (with the obvious meaning) on it.

```
/// \warning An exception will be thrown if the input sequences are empty.
```

14.6.2 Files

A description of the contents of a file can be provided just after the the copyright header. This is particularly recommended for source files of tests and applications. For files, the `\brief` command is always required.

```
/*
 *
 * Copyright header...
 *
 */

/// \file
/// \brief Unit test for LDPC encoder and decoder.
///
/// For all possible base graphs and lifting sizes, the test extracts from a
/// file a small set of messages and corresponding codeblocks. The messages␣
↪are
/// fed to the encoder, whose output is compared to the codeblocks. Similarly,
/// the codeblocks are fed to the decoder and the resulting messages are
/// compared to the example ones.
```

14.6.3 Classes and Structures

Class and structure documentation should provide enough information about what it represents and how an instantiation of the class should be used. In the brief description, do not write expressions like "The class/structure represents/defines" but provide directly more meaningful information. Use the `\brief` command when the brief description is followed by a detailed one (optional if the documentation is limited to the brief description).

It is a good practice to explicitly describe edge cases, side effects and less evident aspects of the class (see last example).

```
/// LDPC rate dematcher interface.
class ldpc_rate_dematcher;

/// Decoder configuration.
struct configuration;

/// \brief PHY&ndash;FAPI bidirectional adaptor interface.
///
/// This adaptor is a collection of interfaces to translate FAPI messages into
/// their PHY layer counterpart and vice versa.
///
/// \note All implementations of this public interface must hold the ownership
/// of all its internal components.
class phy_fapi_adaptor;
```

14.6.4 Class Methods and Free Functions

Methods and functions correspond to actions. As such, the brief description typically starts with a verb (in the third singular person).

```
/// \brief Finds the smallest prime number greater than \c n.
unsigned prime_greater_than(unsigned n);
```

The free function in the example above is very simple, with an input and an output that are clear at first sight. For more complex cases, prefer providing more information by means of the `\param` and `\return` commands. Argument description should follow the same guidelines as general variables (see next section).

Also, similarly to what explained for class and structures, edge cases and unpredictable behaviors should be pointed out.

```
/// \brief Decodes a codeblock.
///
/// By passing a CRC calculator, the CRC is verified after each iteration␣
↪allowing,
/// when successful, an early stop of the decoding process.
///
/// \param[out] output  Reconstructed message of information bits.
/// \param[in]  input   Log-likelihood ratios of the codeblock to be decoded.
/// \param[in]  crc     Pointer to a CRC calculator for early stopping. Set
///                     to \c nullptr for disabling early stopping.
/// \param[in]  cfg     Decoder configuration.
/// \return If the decoding is successful, returns the number of LDPC␣
↪iterations
///         needed by the decoder. Otherwise, no value is returned.
/// \note A codeblock of all zero-valued log-likelihood ratios will␣
↪automatically
/// return an empty value (i.e., failed CRC) and set all the output bits to␣
↪one.
virtual optional<unsigned> decode(bit_buffer& output,
                                  span<const log_likelihood_ratio> input,
                                  crc_calculator* crc,
                                  const configuration& cfg) = 0;
```

14.6.5 Class Data Members, Objects, Variables

Objects and class data members, both variable and constant, static or not, are treated as if they were the concept they represent. As such, their brief description should not show terms like "represents", "indicates", "denotes" or similar. Also, there is no need to repeat the type of the object, since Doxygen repeats the declaration of the object together with its description. Although not common, documentation of this type of entities may also require one or more paragraphs of detailed description. Some good documentation examples are reported below.

```
/// Maximum number of iterations.
unsigned max_iterations = 6;
```

(continues on next page)

(continued from previous page)

```
/// New data flag (\c true if first HARQ transmission).
bool new_data = true;

/// \brief LDPC decoding statistics.
///
/// Provides access to LDPC decoding statistics such as the number of decoded
/// codeblocks (via <tt>ldpc_stats->get_nof_observations()</tt>) or the␣
↪average
/// number of iterations for correctly decoded codeblocks (via
/// <tt>ldpc_stats->get_mean()</tt>).
sample_statistics<unsigned> ldpc_decoder_stats = {};
```

14.7 Commit Formatting

When committing code to the srsRAN Project codebase it is important that commit messages are clear and succinct. This means having clear guidelines for how commits should be structured, both in the title and body of the commit messages.

The preferred commit message style is as follows:

```
component: brief change description
```

Here, component refers to the section of the codebase being modified by the commit. Some examples of this could be:

- `mac_test`, `equalizer` - changes are limited to a specific function/class
- `phy`, `mac`, `rlc`, etc - multiple changes across a specific layer
- `cmake` - changes to the cmake file
- `all` - a commit modifies something across the whole codebase
- `misc` - a miscellaneous change

Multiple components can be used if a commit touches two or more components. Simply separate the components with a comma. For example:

```
rlc, mac: some commit message
```

The brief change description should be just that, *brief*. This means no more than 50 characters, including the component(s). This ensures that the commit message will be displayed properly with no cuts. If you need to create a commit message with more information, the body of the commit message may be used. To do this, separate the heading from the body with a line break in your editor. For example:

```
component: brief commit outline less than 50 char

Body of commit message giving more detail about the commit. This should
only be used when further explanation is needed. This can span multiple
lines if necessary, but keep it as succinct as possible.
```

Note that all words in the subject line are lower case and there is no trailing period.

To summarize, here are a few brief points to follow when committing to the srsRAN codebase:

- Each commit should have a `component` and a `brief outline`
- Commit subject lines should be limited to 50 characters in total
- The body of a commit should be separated by a line break
- Keep all text as succinct as possible
- No upper case letters or trailing period in subject line

If you would like more information on writing "good" commits, this guide is a useful resource.

CHAPTER

FIFTEEN

TESTING POLICY

This section of our knowledge base outlines our comprehensive testing strategy, which adheres to the Shift-left development approach. Unlike the traditional Waterfall model, where testing occurs after coding, our strategy emphasizes testing at the earliest possible stages of development. This proactive approach ensures that the development team considers testing from the project's inception.

Throughout the development and validation stages, we employ various types of tests to ensure the robustness and reliability of our software:

15.1 Unit Tests

Unit tests are the foundational layer of our testing strategy. They scrutinize software behavior at a granular level, examining individual components in isolation. We prioritize unit testing because it is faster, more straightforward, and cost-effective compared to other types of tests. Our development team writes unit tests during development. In most cases, new features in the code and their associated unit tests are in the same Pull Request, even in the same commit. In order to merge a Pull Request into the main development branch, all unit tests (old and new) must success.

Key aspects of our Unit Testing approach:

- **Focus on requirements and behavior:**

  Our unit tests primarily assess whether the software meets its specified requirements and behaves correctly, instead of internal design validation. Most of our unit tests are designed as "sociable unit tests," which means they remain agnostic of implementation details and solely depend on interfaces. This approach allows us to refactor code without affecting the tests and minimizes test instability. However, we also maintain some traditional unit tests for legacy components.

- **Testing happy, bad and edge Paths:**

  Our unit tests include test cases that explore various scenarios, including corner cases, in addition to testing the expected behavior. Besides that, We encourage testing not only the "happy path" but also adverse and error-prone paths (bad paths and death paths).

- **AAA Pattern (Arrange-Act-Assert):**

  Whenever possible, our unit tests adhere to the AAA pattern for improved readability and maintainability.

- **Multiple Small Tests vs. Large Test Cases:**

We prefer multiple small unit tests over large test cases with multiple assertions. This approach enhances test debugging and comprehension.

- **Mandatory Test Additions:**

  Whenever developers introduce new features or bug fixes, they are required to create corresponding unit tests. We enforce this through code reviews and code coverage.

- **Component Labeling:**

  We label unit tests by component, making it easy to assess testing coverage for each software component.

We mainly use Google Test for unit and integration tests, although we have some tests written in pure C++ without the help of a testing framework

---

15.2 Integration and Component Tests

Integration tests encompass a few sub-components or complete ORAN items, verifying input-output relationships as black-box tests. These tests play a crucial role in the initial development stages of (sub)components, serving as skeleton tests to validate general scenarios.

15.3 End-to-End (E2E) Tests

Given the complexity of our ecosystem, we still need E2E and system tests to evaluate the behavior and integration of our CU/DU solutions with real and simulated UEs and 5G cores. These tests are primarily managed by the testing team and follow two distinct approaches:

- **Tests without Hardware:**

  Using ZMQ and simulators, we have developed a test suite encompassing basic scenarios such as attaches, reattaches, and various types of traffic. These tests are easily executable and do not require hardware devices, making them suitable for developers to assess the software's behavior after major changes. They run nightly and can be triggered inside Pull Requests.

- **Tests with Hardware:**

  For real-world scenarios, we employ commercial (COTS) UE devices and/or RUs to trigger complex tests. These tests evaluate behavior, performance, and integration with external components. They run nightly and weekly and provide valuable metrics and Key Performance Indicators (KPIs).

To facilitate these tests, we've developed an in-house E2E testing framework, containerized for deployment flexibility, ensuring testing replicates production environments. Additionally, we utilize testing solutions from VIAVI to validate our product.

While our E2E test suite does not grow with every code change like unit tests, we continually add new tests when significant features are introduced or when exploratory testing identifies noteworthy scenarios to automate.

15.4 Exploratory Tests

Exploratory testing holds particular importance in our strategy due to the complexity of the real-world scenarios we aim to support. Whenever an issue or unexpected behavior is identified, we endeavor to create a test at the lowest possible level, ideally as a unit test.

CHAPTER

SIXTEEN

O-RAN GNB OVERVIEW

This document aims to provide a basic understanding of O-RAN compliant gNBs, and how we have implemented this with srsGNB. A further deep dive can be found in our Developer Guide.

16.1 Introduction

5G Radio Access Networks (RANs) introduce the gNodeB (gNB), the next evolution of the eNodeB found in LTE networks. With Release 15, 3GPP introduced a flexible architecture to the 5G RAN. This splits the gNB into the Centralized Unit (CU), the Distributed Unit (DU) and the Radio Unit (RU).

This new architecture brings multiple new interfaces: the CU and DU communicate over the F1 interface, gNBs communicate with each other over the Xn interface and the RAN and the Core communicate over NG interface. Within this new release there are 8 functional splits, which dictate how and where the RAN is split in the CU, DU and RU.

The Open-RAN (O-RAN) Alliance defined option 7.2x, which is a low-level split for Ultra Reliable Low Latency Communication (URLLC) and near-edge deployments. O-RAN standards also introduce the non-Real Time and near-Real Time RAN Intelligence Controller, or nonRT-RIC and nearRT-RIC, to the 5G RAN. This splits the gNB further, into the DU-low, DU-high, CU-CP and CU-UP

The aim of O-RAN is to promote virtualized and disaggregated RANs, with a highly interoperable, multi-vendor approach.

16.2 Split 7.2x Architecture

As outlined in the introduction, Release 15 introduces the “Functional Split” to the 5G gNB. This separates the gNB into the CU and DU. A simplified outline of this change is illustrated in the following figure.

O-RAN standards further define split 7.2x. This informs how the CU and DU are separated, and which layers of the stack are contained in each. Before looking into Split 7.2x further, it is worth examining the CU/ DU split at a higher level.

At a basic level the CU and DU layer functions are split as follows:

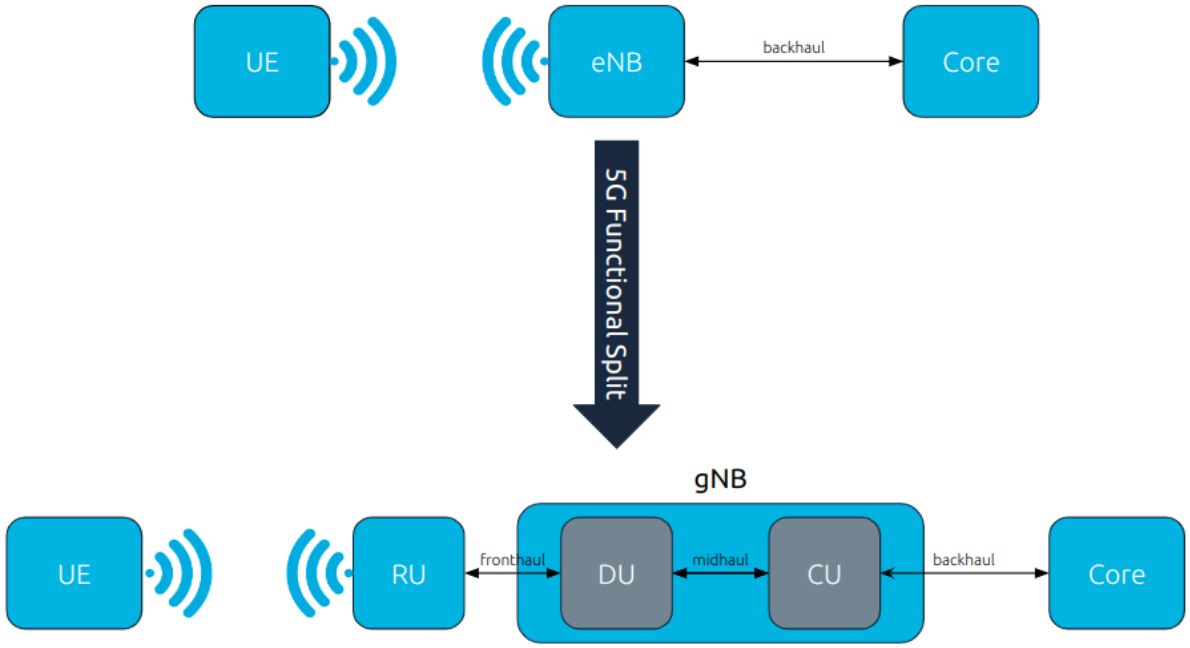


Fig. 1: Simplified overview of move from 4G architecture to 5G architecture with functional split.

Table 1: CU/ DU Split

CU	DU
RRC	RLC
SDAP	MAC
PDCP	PHY

The above shows which layers are located in the CU and DU. This is shown graphically below.

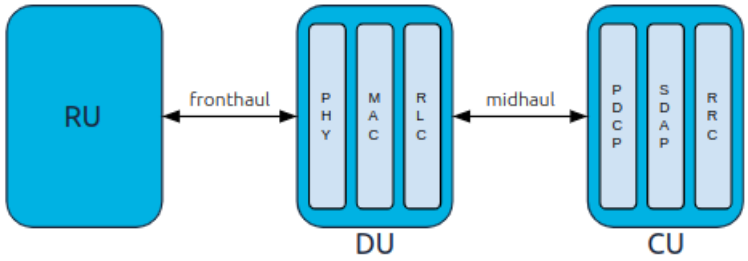


Fig. 2: Simplified overview of O-RAN gNB architecture.

This is an overly simplified view of how the CU and DU might look. In practice, the CU and DU are split further, with multiple interfaces between them.

We will now look specifically at Split 7.2x. In this, the CU is split into the control plane and the user plane, resulting in two elements - the CU-CP and CU-UP. The ORAN-DU is split into the DU-high and DU-low. Not only are the CU and DU split, but the layers they contain are also separated to allow for further control and acceleration of certain procedures - e.g. PDCP and PHY.

The further separation of the RAN components means more interfaces must be introduced to facilitate communication between them. These interfaces are as follows:

- The CU-CP, CU-UP and DU-high will be connected to the nearRT-RIC via the **E2** interface.
- Backhaul from the CU-UP/CP will be done via the **NG** and **XN** and interfaces.
- The CU elements will communicate via the **E1** interface
- The CU-UP and CU-CP will communicate with the DU-high via the **F1-u** and **F1-c** interfaces respectively.
- The DU-high and DU-low communicate via a **FAPI+** interface.

The following figure shows the complete architecture of an O-RAN compliant gNB, implementing Split 7.2x.

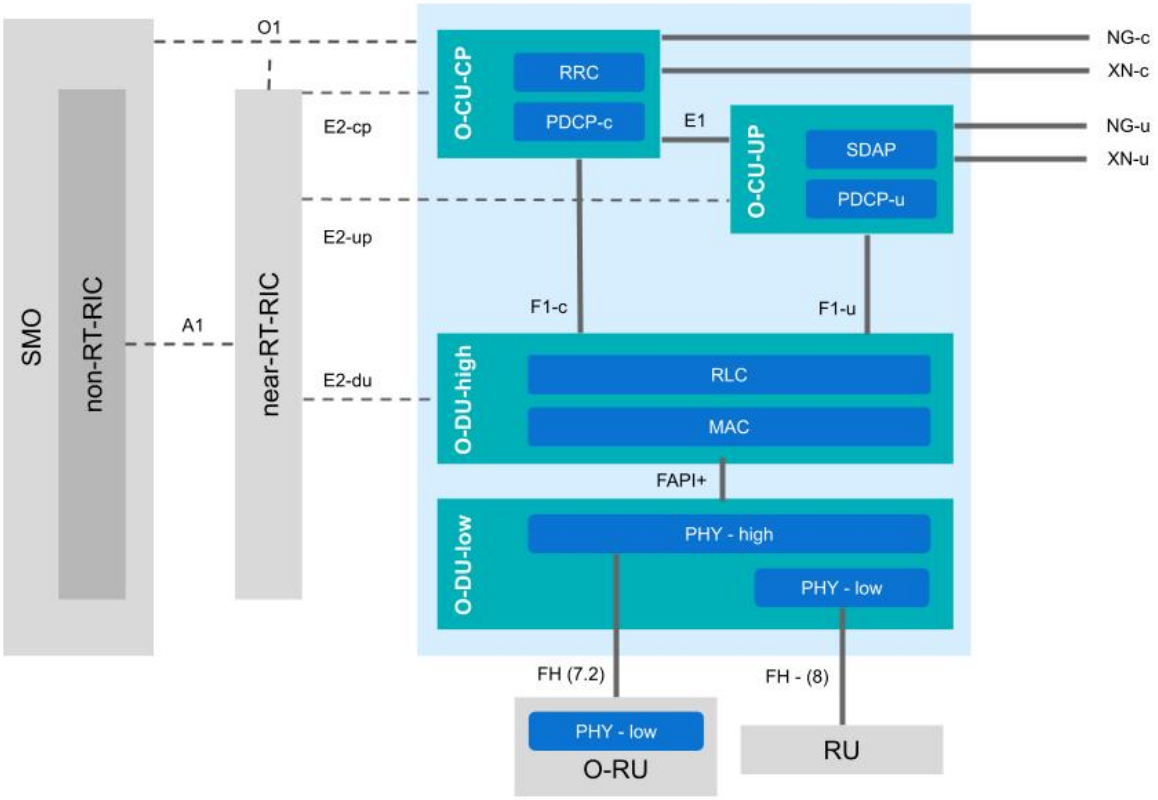


Fig. 3: O-RAN compliant gNB architecture, showing all main components and interfaces. The gNB-specific components are contained in the blue area.

The gNB is shown here in the blue shaded area. The CU-UP and CU-CP are clearly illustrated, as is the split between the high and low DU, and the interfaces between them. This diagram also shows how the nearRT-RIC is connected to the gNB and the nonRT-RIC. The RU is shown here in two versions, the RU (Split 7.2x) and a non-ORAN RU (Split 8). Split 7.2 moves to low PHY out to the RU , while Split 8 keeps it within the DU-low. The frontHaul interface used here is typically eCPRI.

To be O-RAN compliant a gNB must implement the CU and DU as shown above, while also having support for the interfaces needed for the nearRT-RIC, nonRT-RIC, and the RU and RU.

16.3 srsRAN Project gNB

The srsRAN Project gNB is fully O-RAN compliant. While users will still be able to stand up a monolithic gNodeB on a single machine in minutes, our new gNB will also eventually enable users to distribute the RAN functionality across the machines and geography of their choice. All of the elements in the blue shaded area above are implemented in the srsRAN Project gNB, along with all of the interfaces. This allows users to easily use third party RICs, PHY solutions and other O-RAN compliant hardware and applications with the gNB.

16.4 Further Reading

For further information and reading, you can take a look at the following resources:

- ShareTechnote: Open RAN
- O-RAN SC Docs
- O-RAN Alliance website

CHAPTER

SEVENTEEN

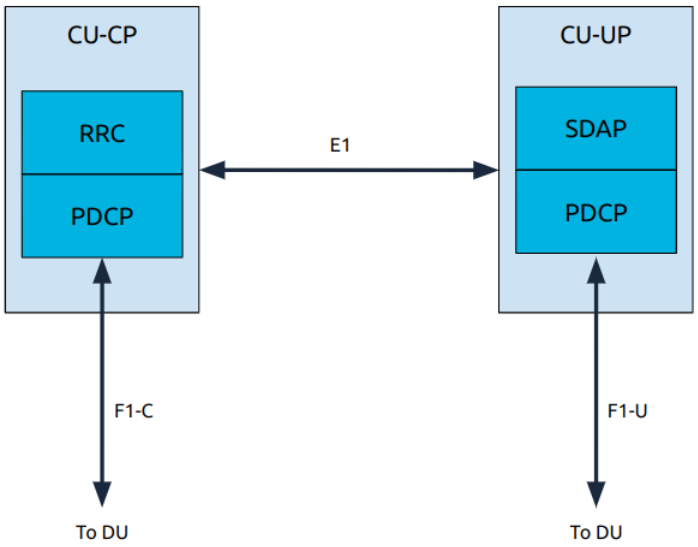
O-RAN GNB COMPONENTS

As outlined in the *O-RAN gNB Overview* the 5G NR gNB introduces a flexible architecture. Splitting the gNB into the CU, DU and RU.

This section of our knowledge base aims to go through these components in further detail. Outlining the main functions of each, their layers and the interfaces between them.

17.1 Centralized Unit (CU)

The CU has the following architecture:



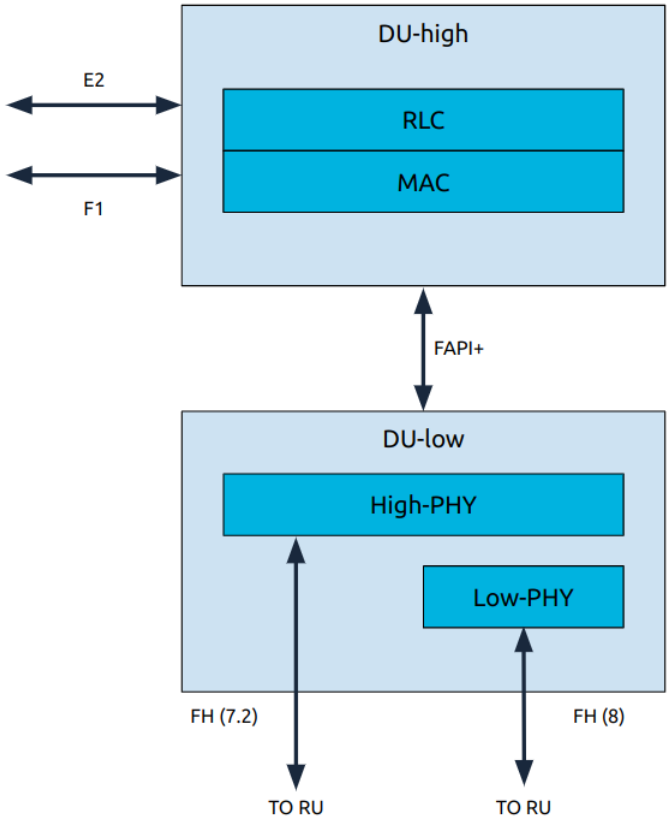
In the CU the user plane and control plane are split into two new submodules: the CU-CP and the CU-UP.

The key points are:

- The user plane contains the SDAP and PDCP layers.
- The control plane contains the RRC and PDCP layers.
- CU-UP and CU-CP are interconnected through the E1 interface; this interface is used for control signalling.
- CU-UP is connected to the DU(s) through the F1-U interface.
- CU-CP is connected to the DU(s) through the F1-C interface.

17.2 Distributed Unit (DU)

The DU has the following architecture:



The DU is composed of L1 and L2 functional blocks which communicate using the FAPI+ interface.

There are two approaches to how the low-PHY is implemented, it can either be in the DU-low or the RU. If split 8 is being used then the low-PHY will be located in the DU-low, if split 7.2x is being used it will be located in the RU.

17.2.1 DU-high

The DU-high contains L2 functional blocks, and communicates with the nearRT-RIC and CUs.

The main interfaces are:

- E2 Handler
  - Terminates the E2 interface from the near-RT RIC. Handles the following tasks:
  - E2/SCTP interface termination
  - E2AP protocol message send and receive
  - Configure KPI measurements in L2 and L1 modules
  - Metrics reporting over E2

  - Relays actions to other modules based on E2 messages
  - Relaying response over E2
- F1 Control plane
  - Performs tasks related to cell management, UE management and semi-static air interface resource management at the cell level.
- F1 User plane
  - Perform tasks related to tunnel management, DL and UL data transmission and DL flow control (RLC feedback).

The main components are:

- RLC
  - Transfers the control and data messages between MAC layer and CU.
- MAC
  - Uses the services of the PHY to send and receive, and multiplexing and de-multiplexing of data on the various logical channels.
  - Responsible for scheduling UL and DL resources for the cell and UE procedures.
  - Interfaces with the DU-low and handles L1 messaging.

17.2.2 DU-low

The DU-low executes the following tasks within L1 functional blocks:

- **L2 FAPI processing:** handle L2 interface request/response FAPI messages from the DU-high.
- **Timing events processing:** Handle timing related operations and triggers scheduled periodic tasks.
- **FEC acceleration processing:** handle FEC related operations and pass the FEC requests to hardware and invokes callback functions when the hardware finishes.
- **FrontHaul processing:** handle all of the front haul related TX/RX operations.
- UL Task scheduling
- DL Task scheduling
- PUSCH, PUCCH, PRACH processing (UL)
- PDSCH, PDCCH, PBCH processing (DL)

The high-PHY in particular is responsible for:

- **UL:** PUSCH, PUCCH, PRACH
- **DL:** PDSCH, PDCCH, PBCH
- UL/DL reference signals (DMRS, PTRS, SRS, PSS, SSS)
- Fronthaul handler
- FAPI handler
- Hardware accelerator handler

- Timing synchronization
- L1 Task control module

CHAPTER

EIGHTEEN

GNB INTERFACES

As outlined in the *O-RAN gNB Overview* the 5G NR gNB introduces a flexible architecture. Splitting the gNB and introducing multiple new interfaces.

This section of our knowledge base aims to go through these interfaces in further detail. Outlining the components they connect, the types of messages they carry and their purpose.

18.1 E1

The E1 interface connects the CU Control Plane (CU-CP) to the CU User Plane (CU-UP). The application protocol (E1AP) is implemented on top of SCTP using ASN1 messages. Its main function is the bearer context management.

18.2 E2

The E2 interface connects the near-RT RIC to an E2 node (E2-du, E2-cp, E2-up). The application protocol (E2AP) is implemented on top of SCTP using ASN1 messages (similar to E1 interface defined by 3GPP). Its main functions are:

- Provide Near-RT RIC services:
  - Report
  - Insert
  - Control
  - Policy

Support functions include:

- Interface management
- RIC service updates

The main functions outlined above include the following procedures:

- **Report:** Subscription request to the E2 node with information to configure how reports should be sent by the node back to the RIC.

- **Insert:** Subscription request to the E2 node with information to configure an insert message which will be used to suspend an ongoing procedure in the node. The RIC then decides how to deal with the suspended procedure (override, cancel, ...).
- **Control:** Control message to the E2 node to instantiate a procedure or resume a previously suspended one.
- **Policy:** RIC requests a node to execute a specific policy during the normal functioning of the E2 node.

The E2 service model defines the functions in the E2 node which may be controlled by the RIC. For each exposed function in the service model, the RIC may monitor, suspend, stop, override or control via different policies the behavior of the E2 node.

The current specification (as of March 2021 specs) defines two service models which are specific implementations of the above points:

1. **Network Interface (NI):** exposure of network interfaces and modification of the messages.
2. **Key Performance Measurement (KPM):** Performance monitoring of set metrics within the RAN.

The KPM service model requires the following metrics to be exposed:

- **CU-UP:** PDCP UL/DL bytes, QCI.
- **DU:** DL/UL PRB usage.
- **CU-CP:** number of active UEs.

18.3 F1

The F1 interface connects the CU to the DU. Its is split into two interfaces. The F1-C connects the CU-CP to the DU, while the F1-U conntects the the CU-UP to the DU.

18.3.1 F1-C

The F1-C interface connects the CU-CP to the DU. The application protocol (F1AP) is implemented on top of SCTP using ASN1 messages. Its main functions are:

- UE context management
- RRC message transfer
- Warning message transmission
- System information
- Paging

18.3.2 F1-U

18.4 FAPI

18.5 NG

The NG interface connects the CU to the Core (AMF). Its is split into two interfaces. The NG-C connects the CU-CP to the AMF, while the NG-U conntects the the CU-UP to the AMF.

18.5.1 NG-C

The NG-C interface connects the CU-CP to the AMF. The application protocol (NGAP) is implemented on top of SCTP using ASN1 messages. Its main functions are:

- PDU session management
- UE context management
- UE mobility management
- Paging
- Transport of NAS messages
- Configuration transfer
- Warning message transmission
- NRPPa transport
- Trace
- Location reporting
- UE TNLA binding
- UE radio capability management
- Data usage reporting

18.5.2 NG-U

18.6 O1

The O1 interface is aligned to the 3GPP specifications for RAN element management. It connects the multiple elements of the gNB to the Service Management and Orchestration (SMO) platform. The following Fault, Configuration, Accounting, Performance and Security (FCAPS) functions are supported through the O1 interface:

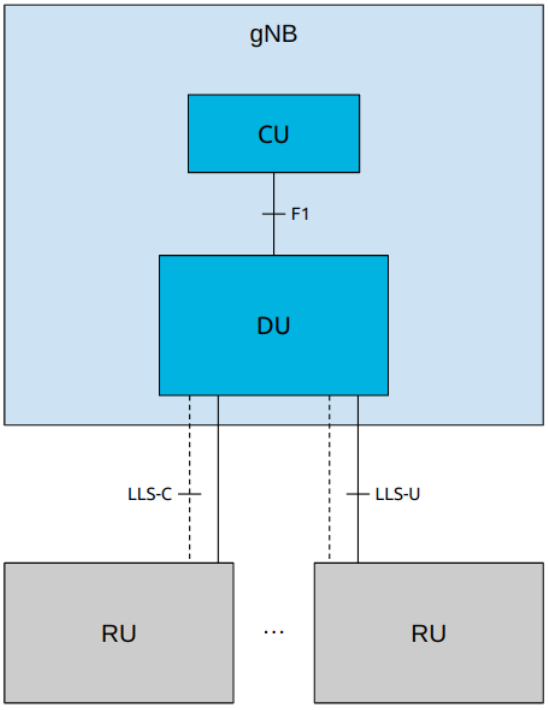
- Performance management (PM)
- Configuration management (CM)
- Fault management (FM)
- File management

- Communication surveillance (heartbeat)
- Tracing
- Physical network function (PNF) discovery
- PNF software management

O1 provisioning operations use NETCONF for communication using YANG files for data exchange, while for CM notifications a RESTFUL HTTP-based solution is used. O1 is the communication protocol to support OAM.

18.7 Open FrontHaul

The architecture of a gNB with CU, DU and RUs is shown in the following figure, using split 7.2x. Here, the term Lower Layer Split (LSS) replaces FrontHaul. LLS-C and LLS-U provide the control and user planes over the LLS interface respectively.



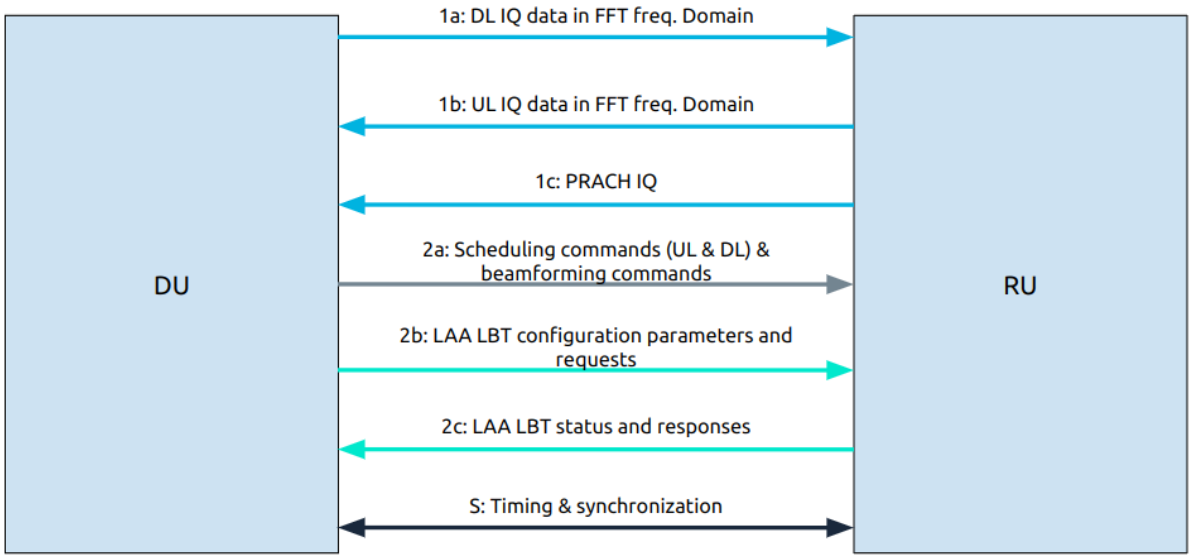
The FrontHaul interface is divided into two main planes:

- **CUS plane:** control, user and synchronization
- **M plane:** management

The CUS plane of the FrontHaul interface uses different data flows to exchange data between the DU and the RU. They are:

1. User plane:
    - **1a:** flow of IQ data in FFT frequency domain on DL
    - **1b:** flow of IQ data in FFT frequency domain on UL
    - **1c:** flow of PRACH IQ data in FFT frequency domain

2. Control plane:
    - **2a:** Scheduling commands (DL and UL) & beamforming commands
    - **2b:** LAA listen-before-talk (LBT) configuration commands and requests
    - **2c:** LAA LBT status and response messages
3. Synchronization plane:
    - **S:** timing and synchronization data



The Management plane, not included in the previous figure, uses a similar interface to O1 to set up, configure and perform maintenance tasks on the FrontHaul interface.

The following tables show the protocol architecture for each plane:

Control Plane	User Plane	Sync Plane	
eCPRI / ROE	eCPRI / ROE		
UDP (optional)	UDP (optional)		
IP (optional)	IP (optional)	PTP	SyncE
Eth L2 + VLAN	Eth L2 + VLAN	ETH L2	
ETH L1	ETH L1	ETH L1	

We can see that for user and control planes we can choose, for transport purposes, between eCPRI or IEEE 1914.3 (Radio over Ethernet).

CHAPTER

NINETEEN

COTS UES

This section of our knowledge base aims to list all COTS UEs known to be working with the srsRAN Project gNB including tested bands and configurations.

19.1 Tested COTS UEs

This is a list of COTS UEs that have been confirmed to be working with the srsRAN Project gNB tested by SRS.

Table 1: SRS tested COTS UEs

Make	Model	Code-name	Base-band	FDD	TDD	Config-uration	SDR	Notes
Google	Pixel 6	GB7N6		n71 - 10 MHz		IMSI 00101 gNB PLMN: 00101	USRP B210[3]	Test SIM: Sysmo-com SJS1[2], SJA2[5] (must disable service 124[6]) Requires disabled NR_TIMER_WAIT_IMS_ or SUP-PORT_IMS_NR_REGIST in hidden IMS menu by dialing *#*#0702#*#*
Mo-torola	Edge 30 Pro	XT2201-1	SM8450 Snap-dragon 8 Gen 1	n3 - 5, 10, 20 MHz	n78 - 10, 20, 40 MHz		USRP B200mini GPSDO	Test SIM: Sysmo-com SJS1[Page 113, 2]
OnePlus	Nord 5G	A2003	SM7250 Snap-dragon 765G	n3 - 5, 10, 20 MHz	n78 - 20, 40, 80, 100 MHz	IMSI 00101 gNB PLMN 90170	B210[Page 1] Leo Bodnar GPSDO N310[4], External Clock	Seems to not like Test PLMN Has been tested in the shield-ing box and on a desk

continues on next page

Table 1 – continued from previous page

Make	Model	Code-name	Base-band	FDD	TDD	Config-uration	SDR	Notes
OnePlus	8T	KB2003	SM8250 Snap-dragon 865 5G		n78 - 20 MHz	IMSI 00101 gNB PLMN 90170	B210, Leo Bodnar GPSDO	Requires use of the "roam-ing hack" Requires GPSDO to keep stable connec-tion.
OnePlus	9 Pro	LE2123	Snap-dragon 888 Octa-core	n3 - 10 MHz	n78 - 20 MHz	IMSI 00101 gNB PLMN: 00101, 99970, 46007		Test SIM: Sysmo-com SJA2[Page 113, 5] (must disable service 124[Page 113, 6]) Requires root[7] NSG must force 5G-NR only[8]
Quectel	RM500q		Snap-dragon X55					

continues on next page

Table 1 – continued from previous page

Make	Model	Code-name	Base-band	FDD	TDD	Config-uration	SDR	Notes
Simcom	SIM8262E-M2		Snap-dragon X62		n78 - 20 MHz			Test SIM: works with test SIM Sysmocom IMSI doesn't work
Telit	FN990A2		Snap-dragon X62	n7 - 5, 10, 20 MHz n71 - 5, 10, 20 MHz	n78 - 20, 30 MHz	IMSI 00101, 99970 gNB PLMN: 00101, 99970	USRP B210[Page 1]	Test SIM: Sysmo-com SJS1[Page 113, 2], SJA2[Page 113, 5] (must disable service 124[Page 113, 6]) For Symo-com Break-out Board requires SIMIN to be "Active High" MS op-eration mode: normal or type_approval (both work)

continues on next page

Table 1 – continued from previous page

Make	Model	Code-name	Base-band	FDD	TDD	Config-uration	SDR	Notes
Xiaomi	11 Lite 5G NE		SM7325 Snap-dragon 778G 5G	n3 - 10, 20 MHz	n78 - 20 MHz	gNB PLMN: 00101	B210, Leo Bodnar GPSDO	Test SIM: Sysmo-com SJA2[Page 113, 5] Activate hidden SA option by dialing *#*#726633#*#*
Xiaomi	12		SM8450 Snap-dragon 8 Gen 1	n3 - 20 MHz	n78 - 20 MHz	gNB PLMN: 00101	B210, Leo Bodnar GPSDO	Test SIM: Sysmo-com SJA2[Page 113, 5] Initially unstable connec-tion, fixed by forcing NR only RAT Activate hidden SA option by dialing *#*#726633#*#*

continues on next page

Table 1 – continued from previous page

Make	Model	Code-name	Base-band	FDD	TDD	Config-uration	SDR	Notes
iPhone	13 Pro		A15 Bionic		n78 - 30 MHz	gNB PLMN: 00101	B210, Leo Bodnar GPSDO	iPhones require IMS to be enable UE to stay con-nected. An Open5GS and Ka-mailio IMS set up can be found here. SUPI conceal-ment is required to attempt an attach, this is outlined in this post.

[3] https://www.ettus.com/all-products/ub210-kit/
[2] https://osmocom.org/projects/cellular-infrastructure/wiki/SysmoUSIM-SJS1
[5] https://osmocom.org/projects/cellular-infrastructure/wiki/SysmoISIM-SJA2
[6] https://docs.srsran.com/projects/4g/en/latest/app_notes/source/5g_sa_COTS/source/index.html?highlight=cots#g-sim
[1] https://www.ettus.com/all-products/usrp-b200mini/
[4] https://www.ettus.com/all-products/usrp-n310/
[7] https://docs.srsran.com/projects/4g/en/latest/app_notes/source/5g_sa_COTS/source/index.html?highlight=cots#rooting-cots-ue
[8] https://docs.srsran.com/projects/4g/en/latest/app_notes/source/5g_sa_COTS/source/index.html?highlight=cots#network-signal-guru

19.2 User reported working UEs

This is a list of COTS UEs that have been reported to be working with the srsRAN Project gNB by users.

Table 2: User reported COTS UEs

Make	Model	Code-name	Base-band	FDD	TDD	Config-uration	SDR	Notes
OnePlus	8	IN2013						
OnePlus	Nord N10	BE2029	Snap-dragon 690					
OnePlus	Nord CE 5G	EB2103	Snap-dragon 750G 5G	n3	n78 - 20 MHz			
OnePlus	10 Pro		SM8450 Snap-dragon 8 Gen 1	n3	n78 - 100 MHz		USRP X410[9]	
Quectel	RM510Q-GL							
Sierra	EM9191		Snap-dragon X55		n78 - 40 MHz			
Xiaomi	Redmi Note 10 5G		Me-diatek MT6833 Dimen-sity 700	n28	N77 and 78			Activate hidden SA option by dialing *#*#726633#*#*
ZTE	MU5002/I		SDX55+Q					
OnePlus	Nord CE 2 Lite 5G				n78 - 20 MHz and 40 MHz		B210	Same PLMN for both SIM and RAN
OnePlus	Nord CE 3 Lite 5G				n78 - 20 MHz and 40 MHz		B210	Same PLMN for both SIM and RAN

continues on next page

Table 2 – continued from previous page

Make	Model	Code-name	Base-band	FDD	TDD	Config-uration	SDR	Notes
Asus	Zenfone 8				n78 - 20 MHz and 40 MHz		B210	Same PLMN for both SIM and RAN
Cross-call	Core Z5				n78 - 20MHz			
Google	Pixel 7 Pro				n78 - 40 MHz to 60 MHz		X310	SISO

19.3 UE Capability Messages

This is a list of PCAP messages containing COTS UE capability messages. Information on viewing PCAP files can be found *here*

Table 3: UE Capability PCAP

Make	Model	UE Capability PCAP
Simcom	SIM8262E-M2	`pcap`
Motorola	Edge 30 Pro	`pcap`
Google	Pixel 6a	`pcap`

19.4 References

[9] https://www.ettus.com/all-products/usrp-x410/

CHAPTER

TWENTY

SRSRAN GNB WITH SRSUE

20.1 Overview

srsRAN Project is a 5G CU/DU solution and does not include a UE application. However, srsRAN 4G does include a prototype 5G UE (srsUE) which can be used for testing. This application note shows how to create an end-to-end fully open-source 5G network with srsUE, the srsRAN Project gNodeB and Open5GS 5G core network.

First, we show how to connect srsUE to the gNodeB over-the-air using USRPs. In the second part, we outline how to use virtual radios based on ZeroMQ instead of SDR hardware. Virtual radios allow the UE to be connected to the gNodeB over sockets. This approach can be very useful for development, testing, debugging, CI/CD or for teaching and demonstrating.

20.2 Hardware and Software Overview

For this application note, the following hardware and software are used:

- PC with Ubuntu 22.04.1 LTS
- srsRAN Project
- srsRAN 4G (23.11 or later)
- Two Ettus Research USRP B210s (connected over USB3)
- Open5GS 5G Core
- ZeroMQ

Ideally the USRPs would be connected to a 10 MHz external reference clock or GPSDO, although this is not a strict requirment. We recommend the Leo Bodnar GPSDO.

20.2.1 srsRAN 4G

If you have not already done so, install the latest version of srsRAN 4G and all of its dependencies. This is outlined in the installation guide.

Please check our srsRAN 4G ZeroMQ Application Note for information on installing ZMQ and using it with srsRAN 4G.

Limitations

The current srsUE implementation has a few feature limitations when running in 5G SA mode. The key feature limitations are as follows:

- Limited to 15 kHz Sub-Carrier Spacing (SCS), which means only FDD bands can be used.
- Limited to 5, 10, 15 or 20 MHz Bandwidth (BW)

20.2.2 Open5GS

For this example, we are using Open5GS as the 5G Core.

Open5GS is a C-language Open Source implementation for 5G Core and EPC. The following links will provide you with the information needed to download and set-up Open5GS so that it is ready to use with srsRAN:

- GitHub
- Quickstart Guide

For the purpose of this application note, we will use a dockerized Open5GS version provided in srsRAN Project at `srsgnb/docker`.

20.2.3 ZeroMQ

On Ubuntu, ZeroMQ development libraries can be installed with:

```
sudo apt-get install libzmq3-dev
```

Alternatively, ZeroMQ can also be built from source.

First, one needs to install libzmq:

```
git clone https://github.com/zeromq/libzmq.git
cd libzmq
./autogen.sh
./configure
make
sudo make install
sudo ldconfig
```

Second, install czmq:

```
git clone https://github.com/zeromq/czmq.git
cd czmq
./autogen.sh
./configure
make
sudo make install
sudo ldconfig
```

Finally, you need to compile srsRAN Project and srsRAN 4G (assuming you have already installed all the required dependencies). Note, if you have already built and installed srsRAN 4G and srsRAN Project prior to installing ZMQ and other dependencies you will have to re-build both to ensure the ZMQ drivers have been recognized correctly.

For srsRAN Project, the following commands can be used to download and build from source:

```
git clone https://github.com/srsran/srsRAN_Project.git
cd srsRAN_Project
mkdir build
cd build
cmake ../ -DENABLE_EXPORT=ON -DENABLE_ZEROMQ=ON
make -j`nproc`
```

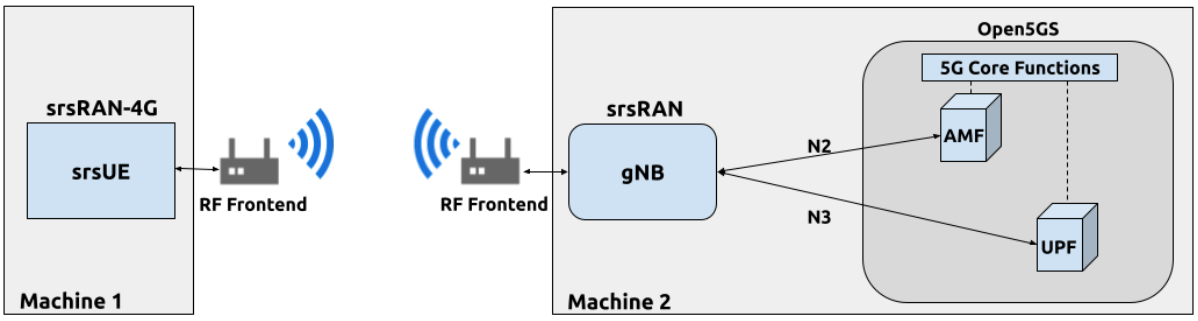
ZeroMQ is disbaled by default, this is enabled when running `cmake` by including `-DENABLE_EXPORT=ON -DENABLE_ZEROMQ=ON`.

Pay extra attention to the cmake console output. Make sure you read the following line:

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

20.3 Over-the-air Setup

The following diagram presents the setup architecture:



20.3.1 Configuration

You can find the srsRAN Project gNB configuration file for this example in the `configs` folder of the srsRAN Project source files. You can also find it here:

- gNB FDD srsUE config

You can download the srsUE config here:

- `srsUE`

It is recommended you use these files to avoid errors while changing configs manually. Any configuration files not included here do not require modification from the default settings. Details of the modifications made are outlined in following sections.

gNB

The following changes need to be made to the gNB configuration file.

The gNB has to connect to AMF in the 5G core network, therefore we need to provide two IP addresses:

```
amf:
  addr: 10.53.1.2                 # The address of the AMF. Check Open5GS␣
↪config -> amf -> ngap -> addr
  bind_addr: 10.53.1.1            # A local IP that the gNB binds to for␣
↪traffic from the AMF.
```

Next, we have to configure the RF front-end device:

```
ru_sdr:
  device_driver: uhd              # The RF driver name.
  device_args: type=b200          # Optionally pass arguments to the␣
↪selected RF driver.
  clock: external                 # Specify the clock source used by the RF.
  sync: external                  # Specify the sync source used by the RF.
  srate: 23.04                    # RF sample rate might need to be␣
↪adjusted according to selected bandwidth.
  tx_gain: 75                     # Transmit gain of the RF might need to␣
↪adjusted to the given situation.
  rx_gain: 75                     # Receive gain of the RF might need to␣
↪adjusted to the given situation.
```

Finally, we configure the 5G cell parameters:

```
cell_cfg:
  dl_arfcn: 368500                # ARFCN of the downlink carrier (center␣
↪frequency).
  band: 3                         # The NR band.
  channel_bandwidth_MHz: 20       # Bandwith in MHz. Number of PRBs will be␣
↪automatically derived.
  common_scs: 15                  # Subcarrier spacing in kHz used for data.
  plmn: "00101"                   # PLMN broadcasted by the gNB.
  tac: 7                          # Tracking area code (needs to match the␣
```

(continues on next page)

(continued from previous page)

```
↪core configuration).
  pdcch:
    common:
      ss0_index: 0                    # Set search space zero index to match␣
↪srsUE capabilities
      coreset0_index: 12              # Set search CORESET Zero index to match␣
↪srsUE capabilities
    dedicated:
      ss2_type: common                # Search Space type, has to be set to␣
↪common
      dci_format_0_1_and_1_1: false # Set correct DCI format (fallback)
  prach:
    prach_config_index: 1             # Sets PRACH config to match what is␣
↪expected by srsUE
```

srsUE

The following changes need to be made to the UE configuration file to allow it to connect to the gNB in SA mode.

First, the following parameters need to be changed under the **[rf]** options so that the B210 is configured optimally:

```
[rf]
freq_offset = 0
tx_gain = 50
rx_gain = 40
srate = 23.04e6
nof_antennas = 1

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

The next set of changes need to be made to the **[rat.eutra]** options. The LTE carrier is disabled, to force the UE to use a 5G NR carrier:

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

Then, the **[rat.nr]** options need to be configured for 5G SA mode operation:

```
[rat.nr]
bands = 3
nof_carriers = 1
max_nof_prb = 106
nof_prb = 106
```

The *max_nof_prb* and *nof_prb* parameters have to be adapted for the used bandwidth according to the following table:

BW	PRBs
5	25
10	52
15	79
20	106

Lastly, set the release and ue_category:

```
[rrc]
release = 15
ue_category = 4
```

Note that the following (default) USIM Credentials are used:

```
[usim]
mode = soft
algo = milenage
opc  = 63BFA50EE6523365FF14C1F45F88737D
k    = 00112233445566778899aabbccddeeff
imsi = 001010123456780
imei = 353490069873319
```

The APN is enabled with the following configuration:

```
[nas]
apn = srsapn
apn_protocol = ipv4
```

20.3.2 Running the Network

The following order should be used when running the network:

1. 5GC
2. gNB
3. UE

Open5GS Core

srsRAN Project provides a dockerized version of the Open5GS. It is a convenient and quick way to start the core network. You can run it as follows:

```
cd ./srsRAN_Project/docker
docker compose up --build 5gc
```

Note that we have already configured Open5GS to operate correctly with srsRAN Project gNB. Moreover, the UE database is populated with the credentials used by our srsUE.

gNB

We run gNB directly from the build folder (the config file is also located there) with the following command:

```
sudo ./gnb -c ./gnb.yaml
```

The console output should be similar to:

```
---== srsRAN gNB (commit 374200dee) ==---

Connecting to AMF on 10.53.1.2:38412
[INFO] [UHD] linux; GNU C++ version 9.2.1 20200304; Boost_107100; UHD_3.15.0.
↪0-2build5
[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 23.040000 MHz...
[INFO] [B200] Actually got clock rate 23.040000 MHz.
Cell pci=1, bw=20 MHz, dl_arfcn=368500 (n3), dl_freq=1842.5 MHz, dl_ssb_
↪arfcn=368410, ul_freq=1747.5 MHz
```

The `Connecting to AMF on 10.53.1.2:38412` message indicates that gNB initiated a connection to the core. If the connection attempt is successful, the following (or similar) will be displayed on the Open5GS console:

```
Open5GS    | 04/17 10:00:43.567: [amf] INFO: gNB-N2 accepted[10.53.1.1]:41578␣
↪in ng-path module (../src/amf/ngap-sctp.c:113)
Open5GS    | 04/17 10:00:43.567: [amf] INFO: gNB-N2 accepted[10.53.1.1] in␣
↪master_sm module (../src/amf/amf-sm.c:706)
Open5GS    | 04/17 10:00:43.567: [amf] INFO: [Added] Number of gNBs is now 1␣
↪(../src/amf/context.c:1034)
Open5GS    | 04/17 10:00:43.567: [amf] INFO: gNB-N2[10.53.1.1] max_num_of_
↪ostreams : 30 (../src/amf/amf-sm.c:745)
```

srsUE

Finally, we start srsUE. This is also done directly from within the build folder, with the config file in the same location:

```
sudo ./srsue ue_rf.conf
```

If srsUE connects successfully to the network, the following (or similar) should be displayed on the console:

```
Reading configuration file ./ue_rf.conf...

Built in Release mode using commit eea87b1d8 on branch master.

Opening 1 channels in RF device=default with args=clock=external
Supported RF device list: UHD zmq file
Trying to open RF device 'UHD'
[INFO] [UHD] linux; GNU C++ version 9.2.1 20200304; Boost_107100; UHD_3.15.0.
↪0-2build5
[INFO] [LOGGING] Fastpath logging disabled at runtime.
[INFO] [MPMD FIND] Found MPM devices, but none are reachable for a UHD␣
↪session. Specify find_all to find all devices.
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] 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] Asking for clock rate 23.040000 MHz...
[INFO] [B200] Actually got clock rate 23.040000 MHz.
RF device 'UHD' successfully opened
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=2094
Random Access Complete.     c-rnti=0x4602, ta=0
RRC Connected
PDU Session Establishment successful. IP: 10.45.1.2
RRC NR reconfiguration successful.
```

It is clear that the connection has been made successfully once the UE has been assigned an IP, this is seen in `PDU Session Establishment successful. IP: 10.45.1.2`. The NR connection is then confirmed with the `RRC NR reconfiguration successful.` message.

20.3.3 Testing the Network

Here, we demonstrate how to use ping and iPerf3 tools to test the connectivity and throughput in the network.

Routing Configuration

Before being able to ping UE, you need to add a route to the UE on the **host machine** (i.e. the one running the Open5GS docker container):

```
sudo ip ro add 10.45.0.0/16 via 10.53.1.2
```

Check the host routing table:

```
route -n
```

It should contain the following entries (note that Iface names might be different):

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.0.1     0.0.0.0         UG    100    0        0 eno1
10.45.0.0       10.53.1.2       255.255.0.0     UG    0      0        0 br-
↪dfa5521eb807
10.53.1.0       0.0.0.0         255.255.255.0   U     0      0        0 br-
↪dfa5521eb807
...
```

Next, add a default route for the UE as follows:

```
sudo ip route add default via 10.45.1.1 dev tun_srsue
```

Ping

Ping is the simplest tool to test the end-to-end connectivity in the network, i.e., it tests whether the UE and core can communicate.

- **Uplink**

To test the connection in the uplink direction, run the following command from the UE machine:

```
ping 10.45.1.1
```

- **Downlink**

To test the connection in the downlink direction, run the following command from the machine running the core network (i.e., Open5GS docker container):

```
ping 10.45.1.2
```

The IP for the UE can be taken from the UE console output. This might change each time a UE reconnects to the network, so it is best practice to always double-check the latest IP assigned by reading it from the console before running the downlink traffic.

- **Ping Output**

Example **ping** output:

```
# ping 10.45.1.1 -c 4
PING 10.45.1.1 (10.45.1.1) 56(84) bytes of data.
64 bytes from 10.45.1.1: icmp_seq=1 ttl=64 time=39.9 ms
64 bytes from 10.45.1.1: icmp_seq=2 ttl=64 time=38.9 ms
64 bytes from 10.45.1.1: icmp_seq=3 ttl=64 time=37.0 ms
64 bytes from 10.45.1.1: icmp_seq=4 ttl=64 time=36.1 ms

--- 10.45.1.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 36.085/37.952/39.859/1.493 ms
```

iPerf3

iPerf3 is a tool that generates (TCP and UDP) traffic and measures parameters (e.g., throughput and packet loss) of the traffic flow.

In this example, we generate traffic in the uplink direction. To this end, we run an iPerf3 client on the UE side and a server on the network side. UDP traffic will be generated at 10Mbps for 60 seconds. It is important to start the server first, and then the client.

- **Network-side**

Start the iPerf3 server the machine running the core network (i.e., Open5GS docker container):

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

The server listens for traffic coming from the UE. After the client connects, the server prints flow measurements every second.

- **UE-side**

With the network and the iPerf3 server up and running, the client can be run from the UE's machine with the following command:

```
# TCP
iperf3 -c 10.53.1.1 -i 1 -t 60
# or UDP
iperf3 -c 10.53.1.1 -i 1 -t 60 -u -b 10M
```

Traffic will now be sent from the UE to the network. This will be shown in both the server and client consoles. Additionaly, we will observe console traces of the UE and the gNB.

Note: All routes have to be configured as presented in *Routing Configuration for USRP-based setup* section.

- **Iperf3 Output**

Example **server** iPerf3 output:

```
# iperf3 -s -i 1
-----------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 10.45.1.2, port 40544
[  5] local 10.45.1.1 port 5201 connected to 10.45.1.2 port 40546
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-1.00   sec  1.20 MBytes  10.1 Mbits/sec
[  5]   1.00-2.00   sec  1.22 MBytes  10.2 Mbits/sec
[  5]   2.00-3.00   sec  1.16 MBytes  9.71 Mbits/sec
[  5]   3.00-4.00   sec  1.12 MBytes  9.44 Mbits/sec
[  5]   4.00-5.00   sec  1.25 MBytes  10.5 Mbits/sec
[  5]   5.00-6.00   sec  1.25 MBytes  10.5 Mbits/sec
```

Example **client** iPerf3 output:

```
# iperf3 -c 10.53.1.1 -i 1 -t 60 -u -b 10M
Connecting to host 10.45.1.1, port 5201
[  5] local 10.45.1.2 port 40546 connected to 10.45.1.1 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec  1.20 MBytes  10.1 Mbits/sec    0    117 KBytes
[  5]   1.00-2.00   sec  1.25 MBytes  10.5 Mbits/sec    0    130 KBytes
[  5]   2.00-3.00   sec  1.25 MBytes  10.5 Mbits/sec    0    130 KBytes
[  5]   3.00-4.00   sec  1.12 MBytes  9.44 Mbits/sec    0    130 KBytes
[  5]   4.00-5.00   sec  1.25 MBytes  10.5 Mbits/sec    0    130 KBytes
[  5]   5.00-6.00   sec  1.12 MBytes  9.44 Mbits/sec    0    130 KBytes
```

- **Console Traces**

The following example trace was taken from the **srsUE console** while running the above iPerf3 test:

```
---------Signal-----------|-----------------DL-----------------|-----------UL-
↪----------
rat  pci  rsrp   pl   cfo | mcs  snr  iter  brate  bler  ta_us | mcs   buff ␣
↪brate  bler
 nr    1     0    0 -457m |  27   43   1.3   274k    0%    0.0 |  27   136k ␣
↪ 13M    0%
 nr    1     0    0 -122m |  27   43   1.4   285k    0%    0.0 |  27    0.0 ␣
↪ 13M    0%
 nr    1     0    0 -282m |  27   43   1.3   267k    0%    0.0 |  27    47k ␣
↪ 13M    0%
 nr    1     0    0  -14m |  27   43   1.4   274k    0%    0.0 |  27    3.0 ␣
↪ 13M    0%
 nr    1     0    0 -373m |  27   43   1.4   268k    0%    0.0 |  27    47k ␣
↪ 13M    0%
 nr    1     0    0  244m |  27   43   1.3   274k    0%    0.0 |  27    0.0 ␣
↪ 13M    0%
```

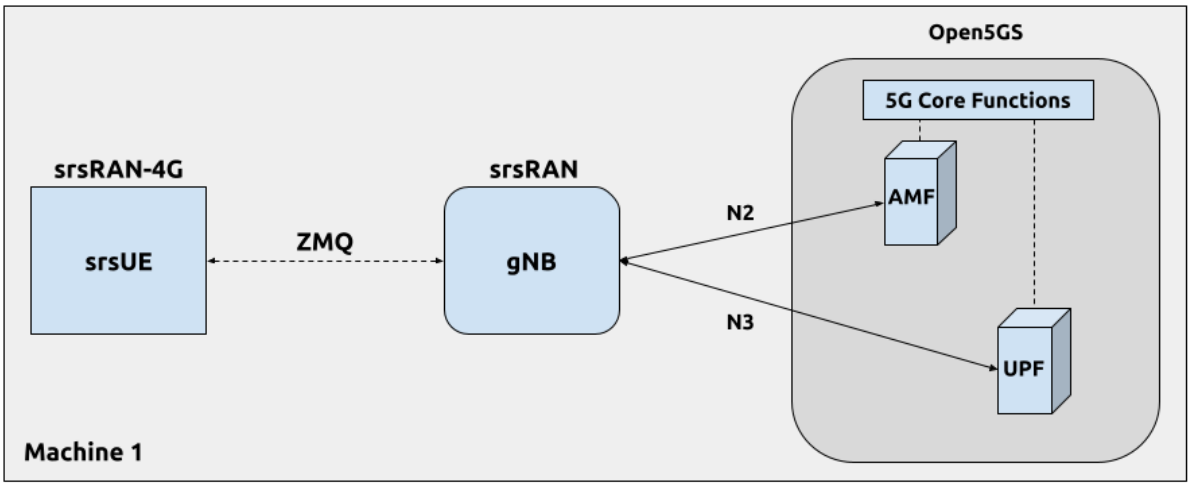
To read more about the UE console trace metrics, see the UE User Manual.

The following example trace was taken from the **gNB console** at the same time period as the srsUE trace shown above:

```
          -------------DL-----------------|------------------UL----------------
↪----
pci rnti  cqi  mcs  brate   ok  nok  (%) | pusch  mcs  brate   ok  nok  (%)  ␣
↪ bsr
  1 4601   15   27   275k  328    0   0% |  23.2   28    13M  398    0   0% ␣
↪55.5k
  1 4601   15   27   266k  336    0   0% |  23.1   28    13M  387    0   0%  ␣
↪ 0.0
  1 4601   15   27   284k  349    0   0% |  23.1   28    13M  410    1   0%  ␣
↪ 0.0
  1 4601   15   27   258k  315    0   0% |  23.1   28    12M  371    0   0%  ␣
↪ 0.0
  1 4601   15   27   275k  330    0   0% |  23.2   28    13M  394    0   0% ␣
↪55.5k
  1 4601   15   27   265k  332    0   0% |  23.2   28    13M  386    1   0%  ␣
↪ 0.0
```

20.4 ZeroMQ-based Setup

In this section, we describe the steps required to configure the ZMQ-based RF driver in both gNB and srsUE. The following diagram presents the setup architecture:



20.4.1 Configuration

The following config files were modified to use ZMQ-based RF driver:

- `gNB config`
- `UE config`

Details of the modifications made are outlined in following sections.

gNB

Replacing the UHD driver with the ZMQ-based RF driver requires changing only **ru_sdr** sections of the gNB 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=23.04e6
  srate: 23.04
  tx_gain: 75
  rx_gain: 75
```

srsUE

When using the ZMQ-based RF driver in the srsUE, it is important to create an appropriate network namespace in the host machine. This is achieved with the following command:

```
sudo ip netns add ue1
```

To verify the new "ue1" network namespace exists, run:

```
sudo ip netns list
```

Then, the **[rf]** section in the srsUE config file has to be changed as follows:

```
[rf]
freq_offset = 0
tx_gain = 50
rx_gain = 40
srate = 23.04e6
nof_antennas = 1

device_name = zmq
device_args = tx_port=tcp://127.0.0.1:2001,rx_port=tcp://127.0.0.1:2000,base_
↪srate=23.04e6
```

In addition, the srsUE must be configured to use the created network namespace. This is achieved by updating the **[gw]** section of the config file:

```
[gw]
netns = ue1
ip_devname = tun_srsue
ip_netmask = 255.255.255.0
```

20.4.2 Running the Network

Once the config files are updated, the network can be set up on a single host machine, using the same commands as in the case of the over-the-air setup.

20.4.3 Testing the Network

Routing Configuration

Before being able to ping UE, you need to add a route to the UE on the **host machine** (i.e. the one running the Open5GS docker container):

```
sudo ip ro add 10.45.0.0/16 via 10.53.1.2
```

Check the host routing table:

```
route -n
```

It should contain the following entries (note that Iface names might be different):

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.0.1     0.0.0.0         UG    100    0        0 eno1
10.45.0.0       10.53.1.2       255.255.0.0     UG    0      0        0 br-
↪dfa5521eb807
10.53.1.0       0.0.0.0         255.255.255.0   U     0      0        0 br-
↪dfa5521eb807
...
```

Next, add a default route for the UE as follows:

```
sudo ip netns exec ue1 ip route add default via 10.45.1.1 dev tun_srsue
```

Check the routing table of ue1:

```
sudo ip netns exec ue1 route -n
```

The output should be as follows:

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.45.1.1       0.0.0.0         UG    0      0        0 tun_
↪srsue
10.45.1.0       0.0.0.0         255.255.255.0   U     0      0        0 tun_
↪srsue
```

Ping

- **Uplink**

To test the connection in the uplink direction, use the following:

```
sudo ip netns exec ue1 ping 10.45.1.1
```

- **Downlink**

To run ping in the downlink direction, use:

```
ping 10.45.1.2
```

The IP for the UE can be taken from the UE console output. This might change each time a UE reconnects to the network, so it is best practice to always double-check the latest IP assigned by reading it from the console before running the downlink traffic.

- **Ping Output**

Example **ping** output:

```
# sudo ip netns exec ue1 ping 10.45.1.1 -c4
PING 10.45.1.1 (10.45.1.1) 56(84) bytes of data.
64 bytes from 10.45.1.1: icmp_seq=1 ttl=64 time=26.6 ms
64 bytes from 10.45.1.1: icmp_seq=2 ttl=64 time=56.9 ms
64 bytes from 10.45.1.1: icmp_seq=3 ttl=64 time=45.2 ms
64 bytes from 10.45.1.1: icmp_seq=4 ttl=64 time=34.9 ms

--- 10.45.1.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3002ms
rtt min/avg/max/mdev = 26.568/40.907/56.878/11.347 ms
```

iPerf3

In this example, we generate traffic in the uplink direction. To this end, we run an iPerf3 client on the UE side and a server on the network side. UDP traffic will be generated at 10Mbps for 60 seconds. It is important to start the server first, and then the client.

- **Network-side**

Start the iPerf3 server the machine running the core network (i.e., Open5GS docker container):

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

The server listens for traffic coming from the UE. After the client connects, the server prints flow measurements every second.

- **UE-side**

With the network and the iPerf3 server up and running, the client can be run from the UE's machine with the following command:

```
# TCP
sudo ip netns exec ue1 iperf3 -c 10.53.1.1 -i 1 -t 60
# or UDP
sudo ip netns exec ue1 iperf3 -c 10.53.1.1 -i 1 -t 60 -u -b 10M
```

Traffic will now be sent from the UE to the network. This will be shown in both the server and client consoles. Additionaly, we will observe console traces of the UE and the gNB.

Note: All routes have to be configured as presented in *Routing Configuration for ZMQ-based setup* section.

- **Iperf3 Output**

Example **server** iPerf3 output:

```
# iperf3 -s -i 1
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 10.45.1.2, port 39176
[  5] local 10.45.1.1 port 5201 connected to 10.45.1.2 port 39184
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-1.00   sec  1.18 MBytes  9.91 Mbits/sec
[  5]   1.00-2.00   sec  1.25 MBytes  10.5 Mbits/sec
[  5]   2.00-3.00   sec  1.12 MBytes  9.44 Mbits/sec
[  5]   3.00-4.00   sec  1.17 MBytes  9.85 Mbits/sec
[  5]   4.00-5.00   sec  1.20 MBytes  10.1 Mbits/sec
[  5]   5.00-6.00   sec  1.25 MBytes  10.5 Mbits/sec
```

Example **client** iPerf3 output:

```
#sudo ip netns exec ue1 iperf3 -c 10.53.1.1 -i 1 -t 60 -u -b 10M
Connecting to host 10.45.1.1, port 5201
[  5] local 10.45.1.2 port 39184 connected to 10.45.1.1 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec  1.31 MBytes  11.0 Mbits/sec    0    119 KBytes
[  5]   1.00-2.00   sec  1.12 MBytes  9.44 Mbits/sec    0    132 KBytes
[  5]   2.00-3.00   sec  1.25 MBytes  10.5 Mbits/sec    0    132 KBytes
[  5]   3.00-4.00   sec  1.12 MBytes  9.44 Mbits/sec    0    132 KBytes
[  5]   4.00-5.00   sec  1.25 MBytes  10.5 Mbits/sec    0    132 KBytes
[  5]   5.00-6.00   sec  1.12 MBytes  9.44 Mbits/sec    0    132 KBytes
```

- **Console Traces**

The following example trace was taken from the **srsUE console** while running the above iPerf3 test:

```
---------Signal-----------|-----------------DL-----------------|-----------UL-
↪----------
rat  pci  rsrp   pl   cfo | mcs  snr  iter  brate  bler  ta_us | mcs   buff ␣
↪brate  bler
 nr    1     9    0 -1.8u |  27   69   1.3   282k    0%    0.0 |  27   136k  ␣
↪ 13M    0%
 nr    1     8    0  505n |  27   73   1.3   299k    0%    0.0 |  27    0.0  ␣
```

(continues on next page)

(continued from previous page)

```
↪ 14M    0%
 nr    1     9    0  499n |  27  n/a   1.2   276k    0%    0.0 |  27   110k ␣
↪ 13M    0%
 nr    1     9    0  1.8u |  27   66   1.3   295k    0%    0.0 |  27    3.0 ␣
↪ 14M    0%
 nr    1     9    0  759n |  27   69   1.3   277k    0%    0.0 |  27    68k ␣
↪ 13M    0%
 nr    1     9    0  188n |  27   71   1.3   290k    0%    0.0 |  27    0.0 ␣
↪ 13M    0%
```

To read more about the UE console trace metrics, see the UE User Manual.

The following example trace was taken from the **gNB console** at the same time period as the srsUE trace shown above:

```
          -------------DL-----------------|------------------UL----------------
↪----
pci rnti  cqi  mcs  brate   ok  nok  (%) | pusch  mcs  brate   ok  nok  (%)  ␣
↪ bsr
  1 4601   15   27   281k  335    0   0% |  65.5   28    13M  396    0   0% ␣
↪39.8k
  1 4601   15   27   288k  353    0   0% |  65.5   28    14M  412    0   0% ␣
↪39.8k
  1 4601   15   27   293k  346    0   0% |  65.5   28    13M  410    0   0% ␣
↪39.8k
  1 4601   15   27   273k  332    0   0% |  65.5   28    13M  384    0   0%  ␣
↪ 0.0
  1 4601   15   27   286k  345    0   0% |  65.5   28    14M  414    0   0%  ␣
↪ 0.0
  1 4601   15   27   288k  349    0   0% |  65.5   28    14M  410    0   0%  ␣
↪ 0.0
```

20.5 Multi-UE Emulation

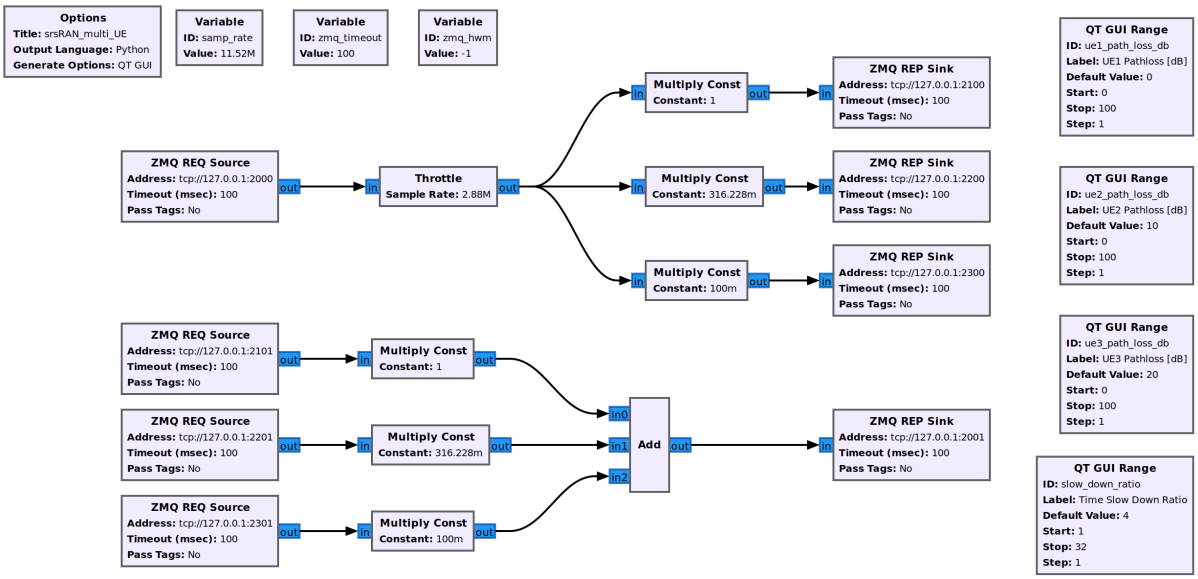
To connect multiple UEs to a single gNB using ZMQ-based RF devices, we need a **broker** that:

- receives DL signal from the gNB and sends its copy to each connected UE,
- receives UL signal from each connected UE, aggregates them, and sends the aggregated signal to the gNB.

In this tutorial, we use **GNU-Radio Companion** to implement such a broker, as it allows easy integration with our ZMQ-based RF devices. Specifically, GNU-Radio Companion is a free and open-source software development toolkit that provides signal processing blocks, which can be simply connected to form a signal **flow graph**. By default, it also provides ZMQ-compatible blocks that allow connecting to external processes (acting as signal sources or sinks) over TCP sockets. We exploit the ZMQ blocks to connect with gNB and srsUE processes and transfer signal samples between them.

The following figure shows the signal flow graph of our broker:

The upper graph is responsible for handling Downlink signal samples, while the lower graph for Uplink signal samples.



Note that here we only provide a simple broker that allows changing path loss separately for each connected UE. But thanks to the rich signal processing block library available in the GNU-Radio Companion, the provided flow graph can be easily extended with any signal processing, manipulation, and/or visualization.

As already mentioned, the GNU-Radio Companion connects over ZMQ sockets with the gNB and srsUE processes (i.e., their ZMQ-based RF devices). The following table lists port numbers are used in the example flow graph:

Table 1: Ports Used in the GNU-Radio flow graph

Port Direction	gNB	srsUE1	srsUE2	srsUE3
TX	2000	2101	2201	2301
Rx	2001	2100	2200	2300

20.5.1 Configuration

GNU-Radio Companion

Please install GNU-Radio Companion following the instructions available here. On Ubuntu it can be installed with the following command:

```
sudo apt-get install gnuradio
```

GNU-Radio flow graph can be downloaded here:

- `GNU-Radio flow graph`

Note that the flow graph allows connecting **three UEs**, but it can be easily adapted to support any (but reasonable) UE number.

Open5GS Core

By default, the subscriber database of the dockerized Open5GS Core is populated with only one UE. A file with a list of all UEs used in this example can be downloaded here:

- `subscriber_db.csv`

The file needs to be saved at: `srsRAN_Project/docker/open5gs/`

Then, edit the `srsRAN_Project/docker/open5gs/open5gs.env` file as follow:

```
 MONGODB_IP=127.0.0.1
 OPEN5GS_IP=10.53.1.2
 UE_IP_BASE=10.45.0
 DEBUG=false
-SUBSCRIBER_DB=001010123456780,00112233445566778899aabbccddeeff,opc,
 ↪63bfa50ee6523365ff14c1f45f88737d,8000,9,10.45.1.2
+SUBSCRIBER_DB="subscriber_db.csv"
```

Alternatively, you can download already edited `open5gs.env` file `here`.

gNB

The following gNB config files was modified to operate with a channel bandwidth of 10 MHz and use the ZMQ-based RF driver.

- `gNB config`

In addition, the total number of available PRACH preambles was set to 63 to mitigate contention among UEs:

```
  prach:
    prach_config_index: 1           # Sets PRACH config to match what is␣
↪expected by srsUE
+   total_nof_ra_preambles: 63      # Sets number of available PRACH preambles
```

srsUE

The following srsUE config files were modified to operate with a channel bandwidth of 10 MHz and use the ZMQ-based RF driver. In addition, the config files were modified to allow the execution of multiple srsUE processes on the same host PC. Specifically, each config file has different:

- ports for the ZMQ connections, that match the those listed in *Ports Used in the GNU-Radio flow graph*.
- pcap and log filenames,
- USIM data (IMSI, etc),
- network namespace name.

You can download the srsUE configs here:

- `UE1 config`
- `UE2 config`

- `UE3 config`

20.5.2 Running the Network

The following order must be used when running the network with multiple UEs:

1. Open5GS
2. gNB
3. **All** UEs
4. GNU-Radio flow graph.

Open5GS Core

Run Open5GS as follows:

```
cd ./srsRAN_Project/docker
docker compose up --build 5gc
```

gNB

Run gNB directly from the build folder (the config file is also located there) with the following command:

```
cd ./srsRAN_Project/build/apps/gnb
sudo ./gnb -c gnb_zmq.yaml
```

srsUE

First, a correct network namespace must be created for each UE:

```
sudo ip netns add ue1
sudo ip netns add ue2
sudo ip netns add ue3
```

Next, start three srsUE instances, each in a separate terminal window. This is also done directly from within the build folder, with the config files in the same location:

```
cd ./srsRAN_4G/build/srsue/src
sudo ./srsue ./ue1_zmq.conf
sudo ./srsue ./ue2_zmq.conf
sudo ./srsue ./ue3_zmq.conf
```

Note, UEs will not connect to the gNB until the GNU-Radio flow graph has been started, as the UL and DL channels are not directly connected between the UE and gNB.

GNU-Radio Companion

Run the GRC Flowgraph associated with the broker.

```
sudo gnuradio-companion ./multi_ue_scenario.grc
```

When gnuradio-companion is started, click on the play button (i.e., `Execute the flow graph`). Now, the signal samples are transferred between gNB and UEs. After a few seconds, all UE should attach and get an IP address. If a srsUE connects successfully to the network, the following (or similar) should be displayed on the console:

```
...
Random Access Transmission: prach_occasion=0, preamble_index=45, ra-rnti=0x39,
↪ tti=174
Random Access Complete.     c-rnti=0x4602, ta=0
RRC Connected
PDU Session Establishment successful. IP: 10.45.1.2
RRC NR reconfiguration successful.
```

It is clear that the connection has been made successfully once the UE has been assigned an IP, this is seen in `PDU Session Establishment successful. IP: 10.45.1.2`. The NR connection is then confirmed with the `RRC NR reconfiguration successful.` message.

Now, you can start traffic (e.g., ping/iperf) to/from each UE using the commands described in the previous section.

Notes:

- The gNB and **all** UEs have to be started before executing the GNU-Radio flow graph.
- You will also need to restart the GNU-Radio flow graph each time the network is restarted (i.e., click `Kill the flow graph`, and then `Execute the flow graph`)

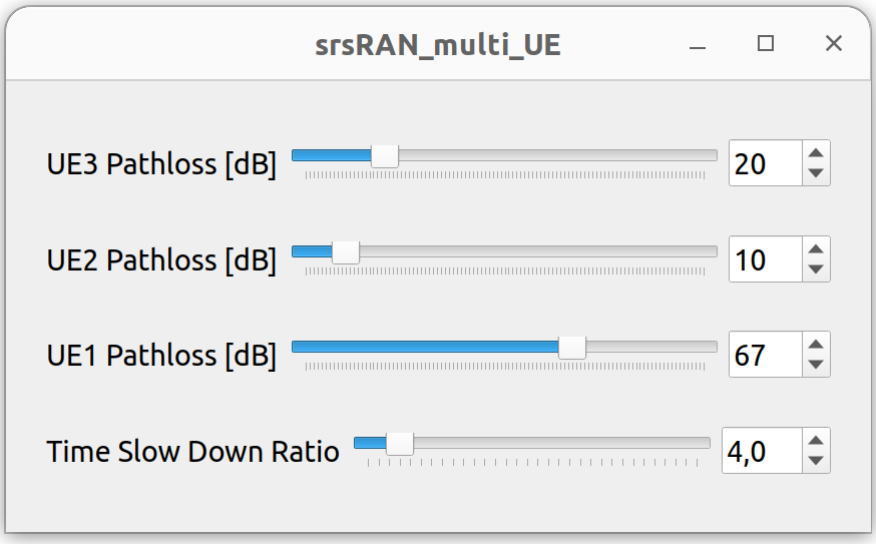
Path-loss Control

The path loss can be controlled for each UE separately via sliders in the flow graph control panel (note, that the control panel pops up after starting the flow graph). The following figure shows the path loss control panel:

You can check the impact of the path loss on the RSRP in each UE. To this end, please activate trace logging in the **srsUE console** by typing `t`. The following example trace shows the changing RSRP that was measured by the UE when setting diverse values of the path loss in the flow graph control panel:

```
t
Enter t to stop trace.
---------Signal-----------|-----------------DL-----------------|-----------UL-
↪----------
rat  pci  rsrp   pl   cfo | mcs  snr  iter  brate  bler  ta_us | mcs   buff ␣
↪brate  bler
 nr    1    43    0 -4.5u |   0   65   0.0    0.0    0%    0.0 |   0    0.0 ␣
↪ 0.0    0%
 nr    1    42    0 -1.4u |   0   65   0.0    0.0    0%    0.0 |   0    0.0 ␣
↪ 0.0    0%
```

(continues on next page)



(continued from previous page)

```
nr     1     36     0   -2.3 |    0   n/a    0.0    0.0     0%     0.0 |    0     0.0 ␣
↪ 0.0     0%
nr     1      8     0   -12u |    0   n/a    0.0    0.0     0%     0.0 |    0     0.0 ␣
↪ 0.0     0%
nr     1      8     0   -16u |    0    84    0.0    0.0     0%     0.0 |    0     0.0 ␣
↪ 0.0     0%
nr     1     25     0   -20u |    0    82    0.0    0.0     0%     0.0 |    0     0.0 ␣
↪ 0.0     0%
nr     1     42     0   -11u |    0    65    0.0    0.0     0%     0.0 |    0     0.0 ␣
↪ 0.0     0%
```

In addition, the control panel allows setting the `Time Slow Down Ratio` parameter, which controls how fast the samples are transferred between gNB and UEs (i.e., the higher the parameter value, the slower the samples are transferred). Specifically, GNU-Radio allows to throttle how fast samples are passed between entities (using the **Throttle** object). By default, the Sample Rate of the Throttle object is set to `samp_rate/slow_down_ratio`, where `samp_rate = 11.52 MHz` and `slow_down_ratio=4`. Therefore, if a connected ZMQ-based RF device generates (and consumes) samples with a sampling rate of 11.52 MHz, it takes 4 seconds to pass them through the GNU-Radio flow graph.

Note that when samples are delivered at slower speeds, gNB and UE have more time to process them (hence lower average CPU load). Therefore, controlling the `Time Slow Down Ratio` parameter might be helpful when running the emulation on a slower host machine and/or with a high number of UEs. You can check the impact of the `Time Slow Down Ratio` parameter on CPU load and network traffic volume on the loopback interface using `htop` and `nload lo`, respectively. Also, a ping round-trip time (RTT) between the core network and UEs is impacted when changing this parameter.

20.5.3 Testing the Network

Once the setup is ready, you can start data traffic by following *Testing the Network* section for the ZMQ-based setup. Note that here we have three UEs (hence, three network namespaces, namely ue1,ue2,ue3), and the default route has to be configured for each of them.

20.6 Troubleshooting

20.6.1 Performance Issues

If you experience some performance-related issues (e.g., RF underflows/lates), please run the srsran_performance script on all PCs used in your setup. The script configures the host machine (CPU, etc.) to run with the best possible performance.

20.6.2 Reference clock

If you encounter issues with the srsUE not finding the cell and/or not being able to stay connected it might be due to inaccurate clocks at the RF frontends. Try to use an external 10 MHz reference or use a GPSDO oscillator.

20.6.3 5G QoS Identifier

By default, Open5GS uses 5QI = 9. If the **qos** section is not provided in the gNB config file, the default one with 5QI = 9 will be generated and the UE should connect to the network. If one needs to change the 5QI, please harmonize these settings between gNB and Open5GS config files, as otherwise, a UE will not be able to connect.

20.6.4 UE does not get IP address

If the UE successfully performs RACH procedure, gets into *RRC Connected* state, but finally disconnects with *RRC Release*, this might indicate that the UE database in the core network is not filled properly. Specifically, in such case, the srsUE console output will look similar to this:

```
Attaching UE...
Random Access Transmission: prach_occasion=0, preamble_index=8, ra-rnti=0x39,␣
↪tti=174
Random Access Complete.     c-rnti=0x4601, ta=0
RRC Connected
Received RRC Release
```

You can also check core network logs, for more information on the cause of this event. For example, Open5gs might show the following information in its log output:

```
open5gs_5gc  | 02/02 09:06:13.742: [amf] INFO: InitialUEMessage (../src/amf/
↪ngap-handler.c:372)
open5gs_5gc  | 02/02 09:06:13.742: [amf] INFO: [Added] Number of gNB-UEs is␣
```

(continues on next page)

(continued from previous page)

```
↪now 2 (../src/amf/context.c:2327)
open5gs_5gc  | 02/02 09:06:13.742: [amf] INFO:     RAN_UE_NGAP_ID[2] AMF_UE_
↪NGAP_ID[3] TAC[7] CellID[0x19b0] (../src/amf/ngap-handler.c:533)
open5gs_5gc  | 02/02 09:06:13.742: [amf] INFO: [suci-0-001-01-0000-0-0-
↪0123456791] Unknown UE by SUCI (../src/amf/context.c:1634)
open5gs_5gc  | 02/02 09:06:13.742: [amf] INFO: [Added] Number of AMF-UEs is␣
↪now 2 (../src/amf/context.c:1419)
open5gs_5gc  | 02/02 09:06:13.742: [gmm] INFO: Registration request (../src/
↪amf/gmm-sm.c:985)
open5gs_5gc  | 02/02 09:06:13.742: [gmm] INFO: [suci-0-001-01-0000-0-0-
↪0123456791]    SUCI (../src/amf/gmm-handler.c:149)
open5gs_5gc  | 02/02 09:06:13.743: [dbi] INFO: [imsi-001010123456791] Cannot␣
↪find IMSI in DB (../lib/dbi/subscription.c:56)
```

From the above, it is clear that UE data is not present in the subscriber database.

Please check and populate the UE database if needed.

In case you are using Open5gs, you can open http://localhost:3000/ in your browser and log in to its WebUI (user: admin, passwd: 1423). You should see an entry for each IMSI present in a UE database. If the required IMSI is not present, you can add it manually by clicking on the **+** button in the lower right corner. **Note:** the WebUI is available when running Open5gs using the docker image we provide. If installed manually, one needs to install the WebUI separately.

In the case of the ZMQ-based setup, please check if you have properly added network namespace for each emulated UE, i.e.:

```
sudo ip netns add ue1
```

To verify the new "ue1" network namespace exists, run:

```
sudo ip netns list
```

20.6.5 UE IP Forwarding

To ensure that the UE traffic is sent correctly to the internet the correct IP forwarding must be enabled. IP Forwarding should be enabled on the **host machine**, i.e. the one running the Open5GS docker container. This can be done with the following command:

```
sudo sysctl -w net.ipv4.ip_forward=1
sudo iptables -t nat -A POSTROUTING -o <IFNAME> -j MASQUERADE
```

Where <IFNAME> is the name of the interface connected to the internet.

To check that this has been configured correctly run the following command:

```
sudo ip netns exec ue1 ping -i 1 8.8.8.8
```

If the UE can ping the Google DNS, then the internet can be successfully accessed.

20.6.6 2nd Open5GS instance (installed manually)

The routing entries on the host PC for IPs: *10.45.0.0* and *10.53.1.0* should use the same interface, e.g.:

```
route -n

Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.0.1     0.0.0.0         UG    100    0        0 eno1
10.45.0.0       10.53.1.2       255.255.0.0     UG    0      0        0 br-
↪dfa5521eb807
10.53.1.0       0.0.0.0         255.255.255.0   U     0      0        0 br-
↪dfa5521eb807
...
```

However, if a second instance of Open5GS (that was installed manually) is running on the host PC, the route to *10.45.0.0* goes to *ogstun* interface. For this reason, a UE cannot access the Internet, as the host will send packets to the manually installed Open5GS version. To solve this routing issue, you can disable (or even remove) the manually installed Open5GS – please check sections 6 and/or 7 of the Open5GS tutorial. In addition, you might need to disable the *ogstun* interface with the following command:

```
sudo ifconfig ogstun 0.0.0.0 down
```

20.6.7 USRP X300/X310

The following config files changes are required to run the above setup with USRP X300/X310.

In the gnb config file, the following parameters in *ru_sdr* section have to changed:

```
ru_sdr:
  ...
  device_args: type=x300,addr=X.X.X.X,master_clock_rate=184.32e6,send_frame_
↪size=8000,recv_frame_size=8000
  ...
  srate: 30.72
```

In the srsUE config file, the following parameters have to changed:

```
[rf]
...
srate = 30.72e6
...
device_args = type=x300,addr=X.X.X.X,master_clock_rate=184.32e6,send_frame_
↪size=8000,recv_frame_size=8000,clock=external,sync=external
...
[expert]
lte_sample_rates = true
```

20.7 Limitations

20.7.1 srsUE

Multiple TACs

- srsUE does not support the use of multiple TACs. Using multple TACs will result in errors parsing NAS messages from the core, resulting in the UE not connecting correctly.

CHAPTER

TWENTYONE

SRSRAN GNB WITH COTS UES

Warning: Operating a private 5G SA network on cellular frequency bands may be tightly regulated in your jurisdiction. Seek the approval of your telecommunications regulator before doing so.

21.1 Overview

This tutorial demonstrates how to configure and connect a 5G capable COTS UE to a 5G SA network using the srsRAN Project gNB and a 3rd-party 5G core (Open5GS in this example).

To connect a COTS UE to a 5G network using the srsRAN Project gNB users will need the following requirements:

- A PC with a linux based OS
- srsRAN Project CU/DU
- RF-frontend compatible with srsRAN Project
- A third-party 5G Core
- A 5G SA capable COTS UE
- A test USIM/ISIM/SIM card (must be a test sim or programmable card with known keys)
- We also recommend the use of an external clock source such as an octoclock or GPSDO

This specific implementation uses the following:

- Dell XPS-13 (Intel i7-10510U CPU) with Ubuntu 20.04
- srsRAN Project CU/DU
- B210 USRP
- Open5GS 5G Core
- OnePlus 8T
- SysmoISIM-SJA2 with reprogrammed credentials
- Leo Bodnar GPS Reference Clock

21.2 Setup Considerations

21.2.1 Open5GS

For this example we are using Open5GS as the 5G Core.

Open5GS is a C-language Open Source implementation for 5G Core and EPC. The following links will provide you with the information needed to download and set-up Open5GS so that it is ready to use with the srsRAN Project gNB:

- GitHub
- Quickstart Guide

21.2.2 COTS UE

You should make sure your device is capable of operating in 5G SA mode and that it operates in the bands supported by the srsRAN Project gNB.

The following bands are not currently supported by srsRAN:

- Band 79 (both 15 and 30 kHz sub-carrier-spacing)
- Band 34, 38, 39 (15 kHz sub-carrier-spacing)

21.2.3 External Clock

A Leo Bodnar GPS reference clock was used as the USRP clock source for this set-up.

The use of an external clock is not essential, but it mitigates possible errors occurring as result of inaccuracies in the internal clock of the RF front-end being used. The issue comes from the CFOs and timings tolerated by 5G handsets - the onboard clocks within SDRs are not always accurate enough to operate within the limits allowed by the handsets. Using an external clock, which is much more accurate than an SDR onboard clock, reduces the likelihood that a COTS UE will be unable to see/ connect to the network due to synchronization issues.

21.3 Configuration

The following srsRAN configuration files were modified for this set-up:

- `gnb_b210_20MHz_oneplus_8t.yml`

The following Open5GS configuration files were modified for this set-up:

- `amf.yaml`
- `upf.yaml`

The aspects of the network should be configured in the following order:

1. gNB
2. ISIM

3. Core
4. COTS UE

21.3.1 gNB

The gNB is configured using the base configuration example found here, with some slight modifications.

The configuration has the following modifications from the original example:

- Set the clock source of the USRP to `external`.
- Set the ARFCN of the cell to `627340`. This ensured the gNB was broadcasting in a free area of spectrum with a BW of 20 MHz in our location.
- The PLMN is changed to `90170`. Using the OnePlus 8t, we have found that by forcing a roaming scenario, the phone sees the network and attaches more reliably. A roaming scenario is forced by setting different PLMNs in the cell and the ISIM in the phone.

The above modifications are specific to the set-up being used here. It is recommended to adjust the ARFCN to your local set-up so that you are transmitting in a free area of spectrum.

To configure the connection between the core and the gNB, you need to set the AMF `addr` to the address of the AMF. In this example, the default value is used in the gNB configuration file, and the relevant changes are made in the Open5GS configuration files.

```
    amf:
      addr: 127.0.1.100                                             # The␣
↪address or hostname of the AMF.
      bind_addr: 127.0.0.1                                          # A local␣
↪IP that the gNB binds to for traffic from the AMF.

    ru_sdr:
      device_driver: uhd                                            # The RF␣
↪driver name.
      device_args: type=b200,num_recv_frames=64,num_send_frames=64  #␣
↪Optionally pass arguments to the selected RF driver.
-     clock:                                                        # Specify␣
↪the clock source used by the RF.
+     clock: external
      sync:                                                         # Specify␣
↪the sync source used by the RF.
      srate: 23.04                                                  # RF␣
↪sample rate might need to be adjusted according to selected bandwidth.
      otw_format: sc12
      tx_gain: 50                                                   #␣
↪Transmit gain of the RF might need to adjusted to the given situation.
      rx_gain: 60                                                   # Receive␣
↪gain of the RF might need to adjusted to the given situation.

    cell_cfg:
-     dl_arfcn: 632628                                              # ARFCN␣
↪of the downlink carrier (center frequency).
+     dl_arfcn: 627340
```

(continues on next page)

(continued from previous page)

```
      band: 78                                                          # The NR␣
↪band.
      channel_bandwidth_MHz: 20                                         #␣
↪Bandwith in MHz. Number of PRBs will be automatically derived.
      common_scs: 30                                                    #␣
↪Subcarrier spacing in kHz used for data.
-     plmn: "00101"                                                     # PLMN␣
↪broadcasted by the gNB.
+     plmn: "90170"
      tac: 7                                                            #␣
↪Tracking area code (needs to match the core configuration).
      pci: 1                                                            #␣
↪Physical cell ID.
```

21.3.2 ISIM

SIM Programming

As outlined previously, this set-up uses the OnePlus 8t, during internal tests it was found that this phone (and other OnePlus devices) sometimes connect to the network more easily in a roaming scenario. This is achieved by setting different PLMNs for the cell and the ISIM in the phone.

The MMC, MNC, IMSI and other credentials in the ISIM can be set by reprogramming. We reprogrammed our SysmoISIM-SJA2 using the following steps.

Download pySim :

```
git clone https://github.com/osmocom/pysim
cd pysim
sudo apt-get install --no-install-recommends \
    pcscd libpcsclite-dev \
    python3 \
    python3-setuptools \
    python3-pyscard \
    python3-pip
pip3 install -r requirements.txt
```

You can then run the following commands from within the `pysim` directory.

Check the current ISIM configuration:

```
./pySim-read.py -p0
```

Reconfigure the ISIM:

```
./pySim-prog.py -p0 -s <ICCID> --mcc=<MCC> --mnc=<MNC> -a <ADM-KEY> --imsi=
↪<IMSI> -k <KI> --opc=<OPC>
```

You need to at least set the PLMN to 00101, optionally you can also reconfigure other aspects of the ISIM. For this set-up the following command was used:

```
./pySim-prog.py -p0 -s 8988211000000689615 --mcc=001 --mnc=01 -a 77190612 --
↪imsi=001010123456789  -k 41B7157E3337F0ADD8DA89210D89E17F --
↪opc=1CD638FC96E02EBD35AA0D41EB6F812F
```

Note: You will need to get the ICCID, ADM-KEY and other security information from the SIM provider.

SUCI Configuration

If you are using a sysmoISIM-SJA2 ISIM (5G-enabled) as in this example, then you will need to modify the 5G-related fields of the sim card. In particular you need to configure or disable SUPI concealment (SUCI).

SUPI concealment can be disabled using the following commands. You should replace <ADM-KEY> with the ADM key of the respective SIM card.

Note: `verify_adm` does not print any output on success. If you see something like *"SW Mismatch: Expected 9000 and got 6982"* the ADM key is not correct. Keep in mind that after 3 failed write attempts due to a wrong ADM key the SIM is blocked and cannot be rewritten again.

```
pySIM-shell (MF)> select MF
pySIM-shell (MF)> select ADF.USIM
pySIM-shell (MF/ADF.USIM)> select EF.UST
pySIM-shell (MF)> verify_adm <ADM-KEY>
pySIM-shell (MF/ADF.USIM/EF.UST)> ust_service_deactivate 124
pySIM-shell (MF/ADF.USIM/EF.UST)> ust_service_deactivate 125
```

After these steps **UST service 124** and **125** should be disabled. You can verify the ISIM configuration using the following command:

```
./pySim-read.py -p0
```

More information on pySim and SUCI configuration can be found in this guide in the pySim documentation.

21.3.3 Open5GS

For this set-up Open5GS is running as a service on the machine, this is the "default" way of running Open5GS as described in their documentation. If you are running open5GS in a docker container, or other environment, your configuration will vary slightly.

The Open5GS 5G Core Quickstart Guide provides a comprehensive overview of how to configure Open5GS to run as a 5G Core.

To configure the core correctly the following steps need to be taken:

- Configure the core to connect to the gNB.
- Configure the PLMN and TAC values so that they are the same as those present in the gNB configuration.

- Register the ISIM credentials to the list of subscribers through the Open5GS WebUI.

amf.yml

In the AMF configuration file the following modifications need to be made:

- Set the NGAP addr, this should be the same as the AMF addr as seen in the gNB configuration file
- Set the MCC, MNC and TAC values such that they are the same as the PLMN and TAC used in the gNB configuration file, and different to that of the ISIM

```
    ngap:
-     - addr: 127.0.0.5
+     - addr: 127.0.1.100
     metrics:
         addr: 127.0.0.5
         port: 9090
     guami:
       - plmn_id:
-          mcc: 999
-          mnc: 70
+          mcc: 901
+          mnc: 70
         amf_id:
           region: 2
           set: 1
     tai:
       - plmn_id:
-          mcc: 999
-          mnc: 70
+          mcc: 901
+          mnc: 70
-        tac: 1
+        tac: 7
     plmn_support:
       - plmn_id:
-          mcc: 999
-          mnc: 70
+          mcc: 901
+          mnc: 70
```

upf.yml

In the UPF configuration file the following modifications need to be made:

- Set the GTPU addr, this should be the same as the AMF addr as seen in the gNB configuration file

```
    upf:
     pfcp:
       - addr: 127.0.0.7
     gtpu:
```

(continues on next page)

(continued from previous page)

```
-       - addr: 127.0.0.7
+       - addr: 127.0.1.100
      subnet:
        - addr: 10.45.0.1/16
        - addr: 2001:db8:cafe::1/48
      metrics:
        - addr: 127.0.0.7
          port: 9090
```

User Database

You can see how to register subscriber information with the core here.

You will need to at least fill the IMSI, AMF, K and OPc related to the subscriber, as well as the APN.

Note: Set the APN to IPv4, not IPv4/6 or IPv6.

21.3.4 COTS UE

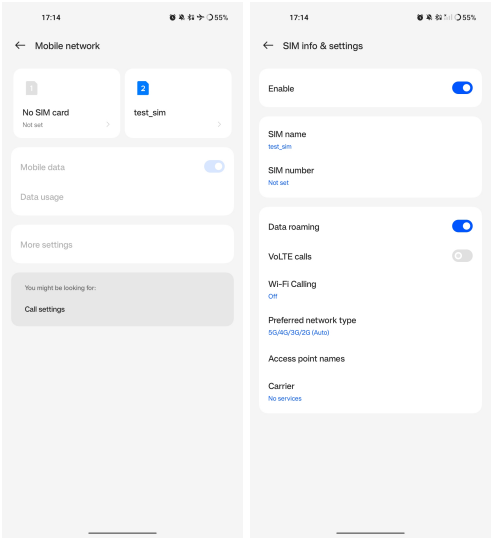
To configure the OnePlus 8t to connect to the network the following steps must be taken:

1. Enable the ISIM
2. Enable 5G SA Mode
3. Enable data roaming
4. Disable VoLTE and/or VoNR
5. Configure the APN
6. Force NR only

Enable ISIM, 5G and data roaming

The first step in configuring the UE is to make sure the SIM and the use of a 5G NR carrier is enabled. In this example the ISIM is placed in SIM tray 1, and there is no other SIM present.

In the first image, you can see that the ISIM is correctly found, and that mobile data is enabled. In the second image you can see that the ISIM is enabled, data roaming is enabled and that 5G is set as the preferred network type.

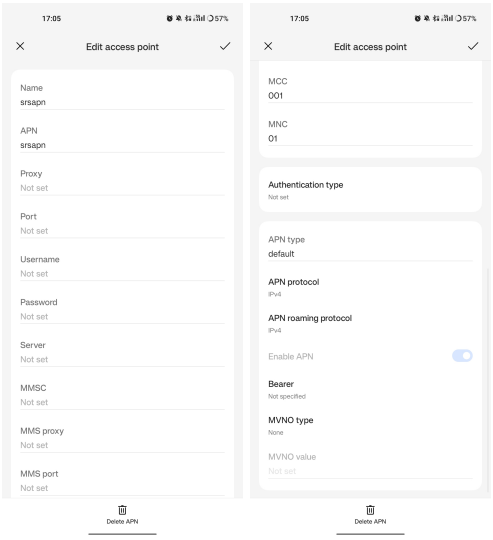


If you cannot see the `5G` option in `Preferred network type`, then you may need to activate it. This can be enabled under the Developer Options, if you do not have access to Developer Options see this guide. In `Developer Options` go to `Networking` and enable `5G`, you may also need to set `5G network mode` to `NSA + SA Mode`

The final option that needs to be enabled here is `data roaming`, this is shown in the second image.

In some phones there may also be an option to configure `VoNR` and/or `VoLTE`, it is important to make sure that this is **disabled**.

Configure APN



The above images show the APN configuration used in this example. The key points to note are the following:

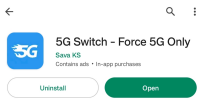
- The APN `Name` is arbitrary, and can have any string value.

- The `APN` option needs to be set to the same as the `DNN/APN` option as set in the Open5GS subscriber registration.
- The `APN protocol` and `APN roaming protocol` are both set to **IPv4** as in the Open5GS subscriber registration. Setting to IPv6 or IPv4/6 can lead to issues connecting the internet.
- All other options are left to the default values.

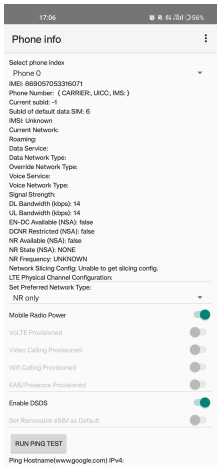
Force NR

The application `5G Switch - Force 5G Only` can be used to force your device to only see 5G NR networks. This works with devices that are not rooted, and was used as part of this setup to ensure the device could see and attach to the network. Although it was not a requirement to get the phone to connect it made it easier to consistently connect to the network.

The apps Play Store page looks like the following:



When you run the app you can select `NR Only` from the `Set Preferred Network Type` menu. This looks like the following:



When you select this option, you may see the `Preferred Network Type` field in the SIM configuration menu change to `4G/3G/2G (Auto)` as seen in the screenshot in the *Connecting to the Network section*. This is fine, and can be ignored. Once NR is selected in the app, you do not have to select 5G from the SIM configuration menu.

21.4 Connecting the COTS UE

To connect the COTS UE to the network the following steps must be taken once the phone and network have been correctly configured:

1. Run the gNB and ensure it is correctly connected to the core
2. Search for the network from the UE
3. Select and connect to the network

4. Verify the attach
5. Stream data

21.4.1 Setting-up the Network

Check that the Core is running correctly

First it is good to check that Open5GS is running correctly, this can be done with the following command:

```
ps aux | grep open5gs
```

If the core is running correctly the following should be given as the output:

```
open5gs     1601  0.0  0.0 141680 15872 ?        Ssl  10:36   0:00 /usr/bin/
↪open5gs-bsfd -c /etc/open5gs/bsf.yaml
open5gs     1606  0.0  0.1 134452 24840 ?        Ssl  10:36   0:01 /usr/bin/
↪open5gs-nrfd -c /etc/open5gs/nrf.yaml
open5gs     1613  0.0  0.2 147068 41720 ?        Ssl  10:36   0:02 /usr/bin/
↪open5gs-scpd -c /etc/open5gs/scp.yaml
open5gs     2663  0.0  0.1 2801740 16788 ?       Ssl  10:36   0:02 /usr/bin/
↪open5gs-hssd -c /etc/open5gs/hss.yaml
open5gs     2675  0.0  0.1 2800268 16568 ?       Ssl  10:36   0:02 /usr/bin/
↪open5gs-pcrfd -c /etc/open5gs/pcrf.yaml
open5gs     2676  0.0  0.1 185572 21584 ?        Ssl  10:36   0:00 /usr/bin/
↪open5gs-pcfd -c /etc/open5gs/pcf.yaml
open5gs     2690  0.0  0.1 169668 20768 ?        Ssl  10:36   0:00 /usr/bin/
↪open5gs-udrd -c /etc/open5gs/udr.yaml
open5gs     3065  0.0  0.1 155984 20136 ?        Ssl  10:36   0:00 /usr/bin/
↪open5gs-amfd -c /etc/open5gs/amf.yaml
open5gs     3067  0.0  0.0 136052 15960 ?        Ssl  10:36   0:00 /usr/bin/
↪open5gs-ausfd -c /etc/open5gs/ausf.yaml
open5gs     3071  0.0  0.0 2778684 14404 ?       Ssl  10:36   0:02 /usr/bin/
↪open5gs-mmed -c /etc/open5gs/mme.yaml
open5gs     3074  0.0  0.0 134300 15416 ?        Ssl  10:36   0:00 /usr/bin/
↪open5gs-nssfd -c /etc/open5gs/nssf.yaml
open5gs     3079  0.0  0.1 260852 19656 ?        Ssl  10:36   0:00 /usr/bin/
↪open5gs-sgwcd -c /etc/open5gs/sgwc.yaml
open5gs     3081  0.0  0.1 249660 17840 ?        Ssl  10:36   0:00 /usr/bin/
↪open5gs-sgwud -c /etc/open5gs/sgwu.yaml
open5gs     3084  0.0  0.2 3127048 44456 ?       Ssl  10:36   0:02 /usr/bin/
↪open5gs-smfd -c /etc/open5gs/smf.yaml
open5gs     3091  0.0  0.1 136072 17136 ?        Ssl  10:36   0:00 /usr/bin/
↪open5gs-udmd -c /etc/open5gs/udm.yaml
open5gs     3099  0.0  0.1 274176 24588 ?        Ssl  10:36   0:00 /usr/bin/
↪open5gs-upfd -c /etc/open5gs/upf.yaml
```

In total there should be 16 processes running.

Once the core is running it is helpful to view the AMF logs for the duration of testing. This makes is clear when the gNB attaches, and when the COTS UE successfully attaches to the network.

To view this you can run this command:

```
tail -f /var/log/open5gs/amf.log
```

You should see an output similar to the following:

```
04/03 10:36:52.012: [sctp] INFO: AMF initialize...done (../src/amf/app.c:33)
04/03 10:36:52.049: [sbi] INFO: [aea4db10-d1fa-41ed-916b-e56218b693e5] (NRF-
↪notify) NF registered (../lib/sbi/nnrf-handler.c:632)
04/03 10:36:52.049: [sbi] INFO: [aea4db10-d1fa-41ed-916b-e56218b693e5] (NRF-
↪notify) NF Profile updated (../lib/sbi/nnrf-handler.c:642)
04/03 10:36:52.049: [sbi] WARNING: [aea4db10-d1fa-41ed-916b-e56218b693e5]␣
↪(NRF-notify) NF has already been added (../lib/sbi/nnrf-handler.c:636)
04/03 10:36:52.049: [sbi] INFO: [aea4db10-d1fa-41ed-916b-e56218b693e5] (NRF-
↪notify) NF Profile updated (../lib/sbi/nnrf-handler.c:642)
04/03 10:36:52.049: [sbi] WARNING: NF EndPoint updated [127.0.0.12:80] (../
↪lib/sbi/context.c:1618)
04/03 10:36:52.049: [sbi] WARNING: NF EndPoint updated [127.0.0.12:7777] (../
↪lib/sbi/context.c:1527)
04/03 10:36:52.238: [app] INFO: SIGHUP received (../src/main.c:58)
04/03 10:36:52.350: [sbi] INFO: [aea6bae8-d1fa-41ed-904f-f78f7a58f5f3] (NRF-
↪notify) NF registered (../lib/sbi/nnrf-handler.c:632)
04/03 10:36:52.350: [sbi] INFO: [aea6bae8-d1fa-41ed-904f-f78f7a58f5f3] (NRF-
↪notify) NF Profile updated (../lib/sbi/nnrf-handler.c:642)
```

Run the gNB

To run the gNB using the configuration file above, navigate to `srsRAN_Project/build/apps/gnb` and use the following command:

```
sudo ./gnb -c gnb_b210_20MHz_oneplus_8t.yml
```

This above command assumes the configuration file is located in the same folder.

Once the gNB is running you should see the following output:

```
---- srsRAN gNB (commit fbe73a49c) ----

Connecting to AMF on 127.0.1.100:38412
[INFO] [UHD] linux; GNU C++ version 9.3.0; Boost_107100; UHD_4.0.0.0-666-
↪g676c3a37
[INFO] [LOGGING] Fastpath logging disabled at runtime.
Making USRP object with args 'type=b200,num_recv_frames=64,num_send_frames=64'
[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] 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.
```

(continues on next page)

(continued from previous page)

```
[INFO] [MULTI_USRP] Setting master clock rate selection to 'manual'.
[INFO] [B200] Asking for clock rate 23.040000 MHz...
[INFO] [B200] Actually got clock rate 23.040000 MHz.
Cell pci=1, bw=20 MHz, dl_arfcn=627340 (n78), dl_freq=3410.1 MHz, dl_ssb_
↪arfcn=627264, ul_freq=3410.1 MHz

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

If the connection to the core is successful you should see the following from the AMF log:

```
04/03 13:25:13.469: [amf] INFO: gNB-N2 accepted[127.0.0.1]:47633 in ng-path␣
↪module (../src/amf/ngap-sctp.c:113)
04/03 13:25:13.469: [amf] INFO: gNB-N2 accepted[127.0.0.1] in master_sm␣
↪module (../src/amf/amf-sm.c:706)
04/03 13:25:13.469: [amf] INFO: [Added] Number of gNBs is now 1 (../src/amf/
↪context.c:1034)
```

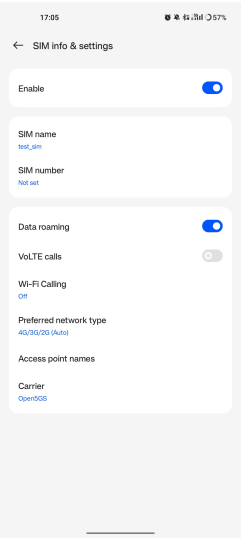
21.4.2 Connecting to the Network

The COTS UE can now search for the network. To do this, navigate to *Mobile Network* > *SIM 1* > *Carrier* and search for the network.

When you enter the *Carrier* menu your device may automatically search for available carriers, if not you can manually select the search option from the top right of the screen.

If the device can successfully receive SIBs (specifically SIB1) and "see" the network it will appear of the list of available carriers. It will be displayed as `Open5GS 5G` or `90170 5G`. If your PLMN is something else it may be displayed as `[PLMN] 5G`.

The following image shows what this may look like:



Select the carrier for the network, in this instance `Open5GS 5G`, the UE should then attach to the network.

To confirm the attach is successful you can monitor both the AMF log and gNB console output.

The AMF log should look similar to the following:

```
04/27 13:16:31.746: [amf] INFO: InitialUEMessage (../src/amf/ngap-handler.
↪c:361)
04/27 13:16:31.746: [amf] INFO: [Added] Number of gNB-UEs is now 1 (../src/
↪amf/context.c:2036)
04/27 13:16:31.746: [amf] INFO:     RAN_UE_NGAP_ID[0] AMF_UE_NGAP_ID[78]␣
↪TAC[7] CellID[0x0] (../src/amf/ngap-handler.c:497)
04/27 13:16:31.746: [amf] INFO: [suci-0-001-01-0-0-0-0000068960] Known UE by␣
↪5G-S_TMSI[AMF_ID:0x20040,M_TMSI:0xdd00ff1a] (../src/amf/context.c:1402)
04/27 13:16:31.746: [gmm] INFO: Registration request (../src/amf/gmm-sm.c:134)
04/27 13:16:31.746: [gmm] INFO: [suci-0-001-01-0-0-0-0000068960]    5G-S_
↪GUTI[AMF_ID:0x20040,M_TMSI:0xdd00ff1a] (../src/amf/gmm-handler.c:169)
04/27 13:16:31.913: [gmm] INFO: [imsi-001010000068960] Registration complete␣
↪(../src/amf/gmm-sm.c:1063)
04/27 13:16:31.913: [amf] INFO: [imsi-001010000068960] Configuration update␣
↪command (../src/amf/nas-path.c:389)
04/27 13:16:31.913: [gmm] INFO:     UTC [2023-04-27T13:16:31] Timezone[0]/
↪DST[0] (../src/amf/gmm-build.c:502)
04/27 13:16:31.913: [gmm] INFO:     LOCAL [2023-04-27T13:16:31] Timezone[0]/
↪DST[0] (../src/amf/gmm-build.c:507)
04/27 13:16:32.105: [gmm] INFO: UE SUPI[imsi-001010000068960] DNN[srsapn] S_
↪NSSAI[SST:1 SD:0xffffff] (../src/amf/gmm-handler.c:1042)
```

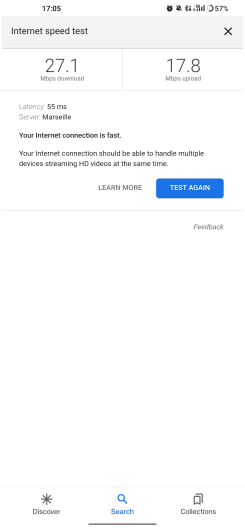
The gNB trace should show the following:

```
          -------------DL-----------------|------------------UL----------------
↪----
pci rnti  cqi  mcs  brate   ok  nok  (%) | pusch  mcs  brate   ok  nok  (%)  ␣
↪ bsr
  1 4601   15   15   4.3k    7    0   0% |  21.3   23    17k    4    0   0%  ␣
↪ 0.0
  1 4601   15   27   287k   84    0   0% |  23.1   27   233k   39    0   0%  ␣
↪ 0.0
  1 4601   15   28   1.2k    1    0   0% |  21.8   28   8.7k    2    0   0%  ␣
↪ 0.0
  1 4601   15    0      0    0    0   0% |   n/a    0      0    0    0   0%  ␣
↪ 0.0
  1 4601   15    0      0    0    0   0% |   n/a    0      0    0    0   0%  ␣
↪ 0.0
  1 4601   15    0      0    0    0   0% |   n/a    0      0    0    0   0%  ␣
↪ 0.0
  1 4601   12    0      0    0    0   0% |   n/a    0      0    0    0   0%  ␣
↪ 0.0
  1 4601   15    0      0    0    0   0% |   n/a    0      0    0    0   0%  ␣
↪ 0.0
  1 4601   15   28    53k   10    0   0% |  24.6   26    55k   32    0   0%  ␣
↪ 0.0
  1 4601   15   28   7.7k    4    0   0% |  22.7   28    17k    4    0   0%  ␣
↪ 0.0
  1 4601   15    0      0    0    0   0% |   n/a    0      0    0    0   0%  ␣
↪ 0.0
```

21.5 Traffic and Testing

21.5.1 Speed Test

Running a speedtest directly from google gives the following results:



While running this test, the following was observed on the gNB console:

Uplink Test

```
          -------------DL-----------------|------------------UL----------------
↪----
pci rnti  cqi  mcs  brate   ok  nok  (%) | pusch  mcs  brate   ok  nok  (%)  ␣
↪ bsr
  1 4601   15   28    23M  820    8   0% |  24.3   27   376k   90    0   0%  ␣
↪ 0.0
  1 4601   15   28    31M 1070    6   0% |  22.4   28   141k   33    0   0%  ␣
↪ 0.0
  1 4601   15   28    31M 1068    8   0% |  23.7   27   155k   39    0   0%  ␣
↪ 0.0
  1 4601   15   28    31M 1064    6   0% |  23.3   28   134k   29    0   0%  ␣
↪ 0.0
  1 4601   15   28    31M 1060    9   0% |  22.5   28   150k   32    0   0%  ␣
↪ 0.0
  1 4601   15   28    31M 1071    6   0% |  23.1   27   323k   68    0   0%  ␣
↪ 0.0
```

Downlink Test

```
          -------------DL-----------------|------------------UL----------------
↪----
pci rnti  cqi  mcs  brate   ok  nok  (%) | pusch  mcs  brate   ok  nok  (%)  ␣
↪ bsr
  1 4601   15   27   548k  447    3   0% |  17.1   25    17M  596    4   0%  ␣
↪150k
  1 4601   15   27   598k  456    6   1% |  17.4   25    17M  596    4   0%  ␣
```

(continues on next page)

(continued from previous page)

```
↪150k
  1 4601    15    27    502k   468     2    0% |   17.5    25     17M   600     0    0% ␣
↪150k
  1 4601    15    27    544k   449     2    0% |   18.2    26     18M   598     2    0% ␣
↪150k
  1 4601    15    27    470k   448     2    0% |   18.7    27     19M   595     5    0% ␣
↪150k
  1 4601    15    27    485k   455     6    1% |   18.6    27     19M   594     6    1% ␣
↪150k
```

21.5.2 Video Test

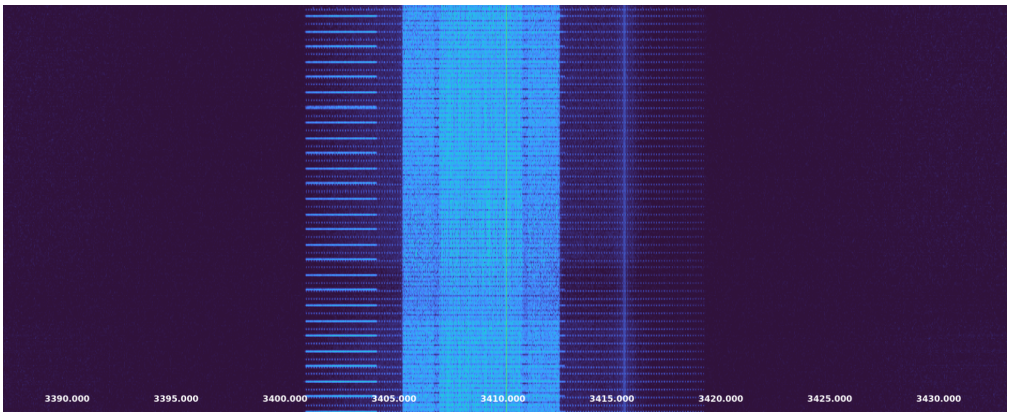
The following shows an example trace output seen while streaming video from the internet:

```
          -------------DL-----------------|------------------UL-----------------
↪----
pci rnti  cqi  mcs  brate   ok  nok  (%) | pusch  mcs  brate   ok  nok  (%)  ␣
↪ bsr
  1 4601   14   27   1.3M  111   15  11% |  22.6   28   109k   25    0    0%  ␣
↪ 0.0
  1 4601   15   27   1.9M  180    4   2% |  22.4   28   109k   25    0    0%  ␣
↪ 0.0
  1 4601   15   28   3.3M  302    0   0% |  22.7   28   109k   25    0    0%  ␣
↪ 0.0
  1 4601   15   28   5.5M  489    0   0% |  22.5   28   109k   25    0    0%  ␣
↪ 0.0
  1 4601   15   28   7.6M  553    0   0% |  22.5   28   109k   25    0    0%  ␣
↪ 0.0
  1 4601   15   28   9.7M  630    0   0% |  22.8   28   109k   25    0    0%  ␣
↪ 0.0
  1 4601   15   28    12M  651    0   0% |  22.7   28   109k   25    0    0%  ␣
↪ 0.0
  1 4601   15   28    12M  656    1   0% |  22.8   28   112k   27    0    0%  ␣
↪ 0.0
  1 4601   15   28    12M  679    0   0% |  22.8   28   109k   25    0    0%  ␣
↪ 0.0
  1 4601   15   28    12M  634    1   0% |  22.6   28   109k   25    0    0%  ␣
↪ 0.0
  1 4601   15   28   7.8M  464    0   0% |  22.3   28    83k   19    0    0%  ␣
↪ 0.0
```

21.6 Troubleshooting

21.6.1 Network Not Visible

- If you are not using a GPSDO or other external clock, you may need to use one. As explained previously, the onboard clocks within SDRs are not always accurate enough to operate within the limits allowed by the handsets.
- For this device, the ISIM needed to be in SIM tray 2. If your device is dual SIM capable and you cannot see the network, try placing the ISIM in the other slot.
- If you were previously able to see the network, but now cannot, you should eject the ISIM and insert it again. The device may be blacklisting the gNB if the device has previously tried to connect and failed.
- You should check that the gNB is transmitting correctly. This can be done with a spectrum analyzer or tools like gr-fosphor and Maia SDR. An example of a "healthy" gNB broadcast from Maia SDR can be seen here:



21.6.2 Unable to Attach

If you can see the network, but cannot attach, here are some things to test:

- Check that the subscriber has been added correctly to the Open5GS list of users. If you did not restart the Open5Gs services after making modifications, then do so and retry connecting the UE. Open5GS does not support on-the-fly modifications to subscribers or config files.
- The device may not be able to PRACH. If you are using NSG, then you will be able to see the control messages being exchanged between the UE and the gNB, check this to see whether or not the PRACH was successful. If not, here are a list of things to check:
  - The signal quality (use Maia SDR, Fosphor or some other tool); you can adjust the Tx and Rx gains to compensate for this. If there are any commercial cells broadcasting in the same area of spectrum this could also be causing RF issues.
  - Timing issues; if there are discrepencies in timing then the UE will not be able to connect. Use an external clock to overcome this.

21.6.3 No Internet Access

If your device is connected to the network but cannot access the internet it is most likely an issue with the APN configuration. Make sure that the credentials and info are the same across both the UE and the APN configuration in the Core. The main things to check are:

- The APN should have the same ID in both the phone and core
- Set the protocol to IPv4
- Make sure VoNR/ VoLTE is disabled on the UE
- Restart all Open5GS services and try again

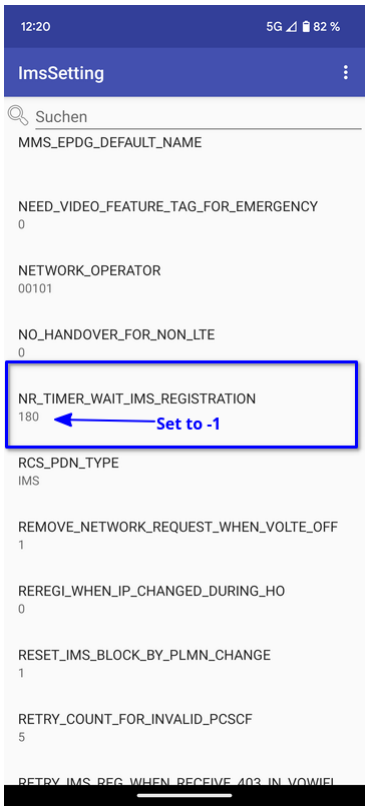
21.6.4 UE Disconnects after a few Minutes

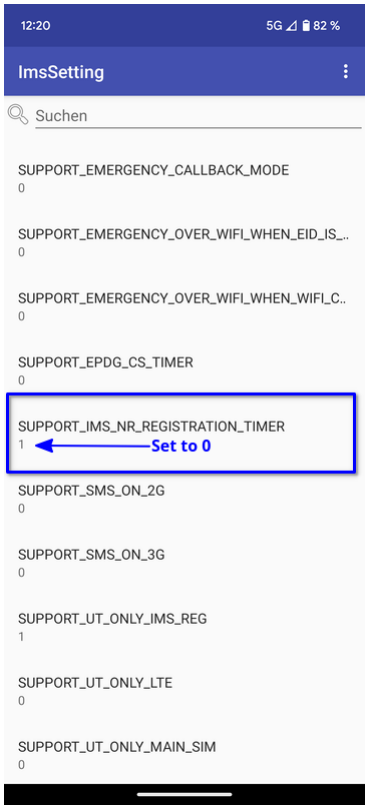
Some Android smartphones silently drop the network connection if IMS is not configured within a couple of minutes after attach. We confirmed this behavior with Google Pixel 6 having a timeout of 180s (3 minutes), but it may apply to other devices and vendors as well.

A possible solution without the need to configure IMS is to either set an infinite timeout or disable the feature in smartphone. For this purpose, open a hidden IMS settings menu by dialing `*#*#0702#*#*`, then change one of the two following settings:

- Infinite timeout: Set `NR_TIMER_WAIT_IMS_REGISTRATION` from default `180` to `-1`
- Disable timeout: Set `SUPPORT_IMS_NR_REGISTRATION_TIMER` from default `1` to `0`

Examples:





The smartphone stores these setting persistently across reboots on a per-IMSI basis, i.e. if you change the SIM the UE remembers the settings for each SIM separately.

21.7 Tested Devices

You can find a list of all of the devices that have been tested by the SRS team and reported by the community *here*. This list contains information about the devices being used, and the configuration of the network.

CHAPTER

TWENTYTWO

SRSRAN GNB WITH AMARISOFT UE

22.1 Overview

The Amarisoft UE simulator (AmariUE) is a commercial software radio solution that supports the simluation of up to 1000s of UE devices. This tutorial outlines how the AmariUE can be used to test and exercise srsRAN gNB. In this example, we will use the open5GS core network to complete the end-to-end 5G network. Usually the gNB and UE connect with each other using physical radios for over-the-air transmissions. However, srsRAN gNB also includes support for virtual radios, which use the ZeroMQ networking library to transfer radio samples between applications. This approach is very useful for development, testing, debugging, CI/CD or for teaching and demonstrating.

This tutorial details two example setups: connecting the srsRAN gNB to AmariUE using virtual radios (ZMQ), and using physical radios (USRP B210).

22.2 Hardware and Software Overview

For this application note, the following hardware and software are used:

- PC with Ubuntu 22.04.2
- srsRAN Project
- Amarisoft UE (2021-09-18 or later)
- Two Ettus Research USRP B210s (connected over USB3)
- Open5GS 5G Core
- ZeroMQ

Note: Ideally the USRPs would be connected to a 10 MHz external reference clock or GPSDO, although this is not a strict requirement. We recommend the Leo Bodnar GPSDO.

22.2.1 Amarisoft UE

Amarisoft UE simulator (AmariUE) is a commercial solution that supports functional and performance testing of 5G networks. Acting as a 3GPP compliant LTE, NB-IOT and NR UE, it can simulate hundreds of UEs sharing the same spectrum. For more information visit the Amarisoft website.

22.2.2 Open5GS

For this example we are using Open5GS as the 5G Core.

Open5GS is a C-language Open Source implementation for 5G Core and EPC. The following links will provide you with the information needed to download and setup Open5GS so that it is ready to use with srsRAN 4G:

- GitHub
- Quickstart Guide

Note: This tutorial assumes that the 5G Core Network is running on a different machine than the machine running Amarisoft UE.

22.3 Installation

22.3.1 ZeroMQ

First thing is to install ZeroMQ and build srsRAN. On Ubuntu, ZeroMQ development libraries can be installed with:

```
sudo apt-get install libzmq3-dev
```

Alternatively, installing from sources can also be done.

First, one needs to install libzmq:

```
git clone https://github.com/zeromq/libzmq.git
cd libzmq
./autogen.sh
./configure
make
sudo make install
sudo ldconfig
```

Second, install czmq:

```
git clone https://github.com/zeromq/czmq.git
cd czmq
./autogen.sh
./configure
```

(continues on next page)

(continued from previous page)

```
make
sudo make install
sudo ldconfig
```

22.3.2 srsRAN Project gNB

Once ZeroMQ is installed, you need to compile srsRAN (assuming you have already installed all the required dependencies). Note, if you have already built and installed srsRAN prior to installing ZMQ and other dependencies you will have to re-run the make command to ensure srsRAN recognizes the addition of ZMQ:

```
git clone https://github.com/srsran/srsRAN_Project.git
cd srsRAN_Project
mkdir build
cd build
cmake ../ -DENABLE_EXPORT=ON -DENABLE_ZEROMQ=ON
make -j`nproc`
```

ZeroMQ is disabled by default, this is enabled when running `cmake` by including `-DENABLE_EXPORT=ON -DENABLE_ZEROMQ=ON`.

Pay extra attention to the cmake console output. Make sure you read the following line:

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

22.3.3 Amarisoft UE

Download the appropriate version of Amarisoft UE and install as per steps provided in its install guide.

This tutorial uses version 2023-02-06 of Amarisoft UE, but it can be any version above 2021-09-18.

22.3.4 ZeroMQ driver for Amarisoft UE

Note: These steps should only be completed **after** compiling srsRAN Project gNB as mentioned above, as they require the build files of srsRAN Project gNB and Amarisoft UHD RF frontend driver.

Interfacing the Amarisoft UE with srsRAN Project requires a custom TRX driver implemented by SRS, which can be found in the srsRAN Project source files in `srsRAN_Project/utils/trx_srsran`.

The Amarisoft UE release folder, `amarisoft.2023-02-06.tar.gz`, should contain a file called `trx_uhd-linux-2023-02-06.tar.gz`. The release folder and the sub-file in question should be uncompressed before proceeding.

First, the driver needs to be compiled, do this by running the following commands from `srsRAN_Project/build`:

```
cmake ../ -DENABLE_EXPORT=TRUE -DENABLE_ZEROMQ=TRUE -DENABLE_TRX_DRIVER=TRUE -
↪DTRX_DRIVER_DIR=<PATH TO trx_uhd-linux-2023-02-06>
make trx_srsran_test
ctest -R trx_srsran_test
```

Make sure CMake finds the file `trx_driver.h` in the specified folder. CMake should print the following:

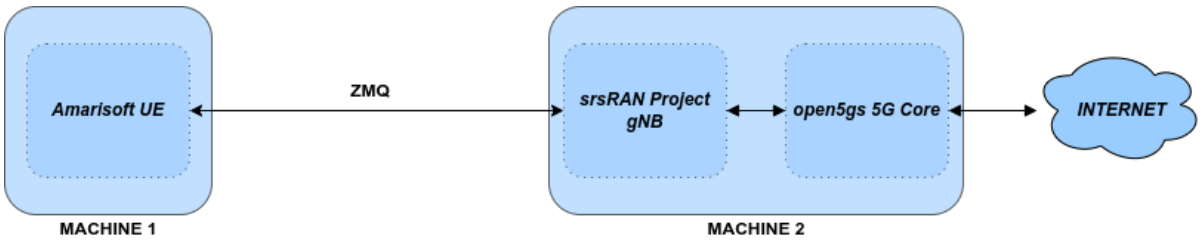
```
-- Found trx_driver.h in TRX_DRIVER_DIR=/home/user/amarisoft/2021-03-15/trx_
↪uhd-linux-2021-03-15/trx_driver.h
```

A symbolic link must be done for the UE application to load the driver. From the Amarisoft UE build folder run the following command:

```
ln -s srsRAN_Project/build/utils/trx_srsran/libtrx_srsran.so trx_srsran.so
```

22.4 ZeroMQ-based Setup

In this section, we describe the steps required to configure the ZMQ-based RF driver in both gNB and AmariUE. The following diagram presents the setup architecture:



22.4.1 Configuration

The following config files were modified to use ZMQ-based RF driver:

- `gNB FDD band 7 config`
- `gNB TDD band 78 config`
- `Single UE config FDD band 7`
- `Single UE config TDD band 78`
- `Multiple UEs config FDD band 7`
- `Multiple UEs config TDD band 78`

Details of the modifications made are outlined in following sections.

srsRAN Project gNB

Modify the `amf` section with IP of AMF and IP to which gNB need to bind in order to connect to AMF:

```
amf:
  addr:      172.22.0.10
  bind_addr: 172.22.0.1
```

Modify the `ru_sdr` section with IPs from which gNB sends and receives radio samples via ZMQ driver:

```
ru_sdr:
  device_driver: zmq
  device_args:   tx_port=tcp://10.53.1.1:5000,rx_port=tcp://10.53.1.2:6000
```

Amarisoft UE

Modify the `ru_sdr` section of Amarisoft UE configuration with IPs from which UE sends and receives radio samples via ZMQ driver:

```
ru_sdr: {
  name:      "srsran",
  args:      "",
  tx_port0:  "tcp://10.53.1.2:6000",
  rx_port0:  "tcp://10.53.1.1:5000",
  log_level: "info"
},
```

Then, set the gain parameters as follows:

```
tx_gain: 0,
rx_gain: 0,
```

Make sure the CELL_BANDWIDTH matches that on gNB configuration file:

```
#define CELL_BANDWIDTH 10
```

Add `sample_rate:  61.44,` in the cell entry under `cells` section:

```
sample_rate: 61.44,
```

Then, set `N_ANTENNA_DL` to 1:

```
#define N_ANTENNA_DL 1
```

Note that the following (default) SIM Credentials and APN are used:

```
sim_algo: "milenage",
imsi: "001019123456799",
K: "00112233445566778899aabbccddeeff",
opc: "63bfa50ee6523365ff14c1f45f88737d",
apn: "internet",
```

Open5GS 5G Core

As highlighted above, the Open5GS Quickstart Guide provides a comprehensive overview of how to configure Open5GS to run as a 5G Core.

The main modifications needed are:

- Change the TAC in the AMF config to 7
- Check that the NGAP, and GTPU addresses are all correct. This is done in the AMF and UPF config files.
- It is also a good idea to make sure the PLMN values are consistent across all of the above files and the UE config file.

The final step is to register the UE to the list of subscribers through the Open5GS WebUI. The values for each field should match what is in the Amarisoft UE config file, under the `ue_list` section.

Note: Make sure to correctly configure the APN, if this is not done correctly the UE will not connect to the internet.

It is important to run Amarisoft UE and 5G Core in different machine or at least in different network namespaces. This is because both the 5GC and UE will be sharing the same network configuration, i.e. routing tables etc. Because the UE receives an IP address from the 5GC's subnet, the Linux kernel would bypass the TUN interfaces when routing traffic between both ends.

22.4.2 Running the 5G Network

The following order should be used when running the network:

1. 5GC
2. gNB
3. UE

Open5gs 5G Core

Once the steps from the Open5GS Quickstart Guide are followed you do not need to do any more to bring the core online. It will run in the background. Make sure to restart the relevant daemons after making any changes to the config files.

srsRAN Project gNB

Run the gNB from its build directory, using the configuration file provided:

```
sudo ./apps/gnb/gnb -c gnb_rf_zmq_fdd_n7_10mhz.yml
```

The console output should be similar to the following:

```
Available radio types: uhd and zmq.

---== srsRAN gNB (commit 05beac11e) ==---

Connecting to AMF on 172.22.0.10:38412
Cell pci=1, bw=10 MHz, dl_arfcn=536020 (n7), dl_freq=2680.1 MHz, dl_ssb_
↪arfcn=535930, ul_freq=2560.1 MHz

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

The `Connecting to AMF on 172.22.0.10:38412` message indicates that gNB initiated a connection to the core.

If the connection attempt is successful, the following (or similar) will be displayed on the Open5GS console:

```
amf      | 04/23 14:38:26.459: [amf] INFO: gNB-N2 accepted[172.22.0.1]:49428␣
↪in ng-path module (../src/amf/ngap-sctp.c:113)
amf      | 04/23 14:38:26.459: [amf] INFO: gNB-N2 accepted[172.22.0.1] in␣
↪master_sm module (../src/amf/amf-sm.c:741)
amf      | 04/23 14:38:26.459: [amf] INFO: [Added] Number of gNBs is now 1 (..
↪/src/amf/context.c:1178)
amf      | 04/23 14:38:26.459: [amf] INFO: gNB-N2[172.22.0.1] max_num_of_
↪ostreams : 30 (../src/amf/amf-sm.c:780)
```

Amarisoft UE

Lastly we can launch the UE, with root permissions to create the TUN device.

```
/root/ue/lteue /root/ue/config/ue-nr-sa-fdd-n7-zmq-single-ue.cfg
```

The above command should start the UE and attach it to the core network.

If UE connects successfully to the network, the following (or similar) should be displayed on the console:

```
amariue  | RF0: sample_rate=61.440 MHz dl_freq=2680.100 MHz ul_freq=2560.100␣
↪MHz (band n7) dl_ant=1 ul_ant=1
amariue  | (ue) Cell 0: SIB found
amariue  | UE PDN TUN iface requested: ue_id: ue1, pdn_id: 0, ifname: ue1-
↪pdn0, ipv4_addr: 192.168.100.2, ipv4_dns: 8.8.4.4, ipv6_local_addr: , ipv6_
↪dns:
amariue  | Created iface ue1-pdn0 with 192.168.100.2
```

It is clear that the connection has been made successfully once the UE has been assigned an IP, this is seen in `PDU Session Establishment successful. IP: 192.168.100.2`. The NR connection is then confirmed with the `RRC NR reconfiguration successful.` message.

22.4.3 Testing the Network

Here, we demonstrate how to use ping and iPerf3 tools to test the connectivity and throughput in the network.

Ping

To exchange traffic in the `downlink` direction, i.e. from the the 5GC (UPF), just run ping as usual on the command line, e.g.:

```
ping 192.168.100.2
```

In order to generate traffic in the `uplink` direction it is important to run the ping command using the TUN interface of UE (e.g. tun0 created once Amarisoft UE attaches to 5G network).

```
ip netns exec ue0 ping 192.168.100.1 -I tun0
```

iPerf

Downlink

For generating downlink traffic, we run iPerf client on the 5GC (UPF) and server on the UE as follows:

In the UE,

```
ip netns exec ue0 iperf -u -s -i 1
```

In the UPF,

```
iperf -u -c 192.168.100.2 -b 5M -i 1 -t 30
```

Uplink

And, for uplink traffic, we run iPerf server on the 5GC (UPF) and client on the UE as follows.

In the UPF,

```
iperf -u -s -i 1
```

In the UE,

```
ip netns exec ue0 iperf -u -c 192.168.100.1 -b 5M -i 1 -t 30
```

iPerf Output

Example `server` iPerf3 output:

```
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 192.168.100.1 port 5001 connected with 192.168.100.3 port 36039
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total␣
↪Datagrams
[  3]  0.0- 1.0 sec   629 KBytes  5.15 Mbits/sec   6.188 ms    0/  438 (0%)
[  3]  1.0- 2.0 sec   431 KBytes  3.53 Mbits/sec   2.656 ms    0/  300 (0%)
[  3]  2.0- 3.0 sec   866 KBytes  7.09 Mbits/sec   2.690 ms    0/  603 (0%)
[  3]  3.0- 4.0 sec   589 KBytes  4.82 Mbits/sec   8.544 ms    0/  410 (0%)
[  3]  4.0- 5.0 sec   693 KBytes  5.68 Mbits/sec   2.879 ms    0/  483 (0%)
[  3]  5.0- 6.0 sec   637 KBytes  5.22 Mbits/sec   2.583 ms    0/  444 (0%)
[  3]  6.0- 7.0 sec   500 KBytes  4.09 Mbits/sec  15.044 ms    0/  348 (0%)
[  3]  7.0- 8.0 sec   784 KBytes  6.42 Mbits/sec   2.792 ms    0/  546 (0%)
[  3]  8.0- 9.0 sec   637 KBytes  5.22 Mbits/sec   9.773 ms    0/  444 (0%)
[  3]  9.0-10.0 sec   600 KBytes  4.92 Mbits/sec  10.407 ms    0/  418 (0%)
[  3] 10.0-11.0 sec   683 KBytes  5.60 Mbits/sec   2.029 ms    0/  476 (0%)
[  3] 11.0-12.0 sec   637 KBytes  5.22 Mbits/sec  11.048 ms    0/  444 (0%)
[  3] 12.0-13.0 sec   643 KBytes  5.27 Mbits/sec   2.563 ms    0/  448 (0%)
[  3] 13.0-14.0 sec   626 KBytes  5.13 Mbits/sec   3.593 ms    0/  436 (0%)
[  3] 14.0-15.0 sec   593 KBytes  4.86 Mbits/sec   8.771 ms    0/  413 (0%)
[  3] 15.0-16.0 sec   669 KBytes  5.48 Mbits/sec   1.940 ms    0/  466 (0%)
[  3] 16.0-17.0 sec   501 KBytes  4.10 Mbits/sec   9.604 ms    0/  349 (0%)
[  3] 17.0-18.0 sec   813 KBytes  6.66 Mbits/sec   2.372 ms    0/  566 (0%)
[  3] 18.0-19.0 sec   637 KBytes  5.22 Mbits/sec   2.671 ms    0/  444 (0%)
[  3] 19.0-20.0 sec   632 KBytes  5.17 Mbits/sec   3.903 ms    0/  440 (0%)
[  3] 20.0-21.0 sec   606 KBytes  4.96 Mbits/sec   2.835 ms    0/  422 (0%)
[  3] 21.0-22.0 sec   678 KBytes  5.55 Mbits/sec   2.643 ms    0/  472 (0%)
[  3] 22.0-23.0 sec   649 KBytes  5.32 Mbits/sec   2.577 ms    0/  452 (0%)
[  3]  0.0-23.6 sec  14.7 MBytes  5.25 Mbits/sec   2.420 ms    0/10510 (0%)
```

Example `client` iPerf3 output:

```
------------------------------------------------------------
Client connecting to 192.168.100.1, UDP port 5001
Sending 1470 byte datagrams, IPG target: 2243.04 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 192.168.100.3 port 36039 connected with 192.168.100.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec   642 KBytes  5.26 Mbits/sec
[  3]  1.0- 2.0 sec   640 KBytes  5.24 Mbits/sec
[  3]  2.0- 3.0 sec   640 KBytes  5.24 Mbits/sec
[  3]  3.0- 4.0 sec   640 KBytes  5.24 Mbits/sec
[  3]  4.0- 5.0 sec   640 KBytes  5.24 Mbits/sec
```

(continues on next page)

(continued from previous page)

```
[  3]  5.0- 6.0 sec   639 KBytes  5.23 Mbits/sec
[  3]  6.0- 7.0 sec   640 KBytes  5.24 Mbits/sec
[  3]  7.0- 8.0 sec   640 KBytes  5.24 Mbits/sec
[  3]  8.0- 9.0 sec   640 KBytes  5.24 Mbits/sec
[  3]  9.0-10.0 sec   640 KBytes  5.24 Mbits/sec
[  3] 10.0-11.0 sec   640 KBytes  5.24 Mbits/sec
[  3] 11.0-12.0 sec   639 KBytes  5.23 Mbits/sec
[  3] 12.0-13.0 sec   640 KBytes  5.24 Mbits/sec
[  3] 13.0-14.0 sec   640 KBytes  5.24 Mbits/sec
[  3] 14.0-15.0 sec   640 KBytes  5.24 Mbits/sec
[  3] 15.0-16.0 sec   640 KBytes  5.24 Mbits/sec
[  3] 16.0-17.0 sec   639 KBytes  5.23 Mbits/sec
[  3] 17.0-18.0 sec   640 KBytes  5.24 Mbits/sec
[  3] 18.0-19.0 sec   640 KBytes  5.24 Mbits/sec
[  3] 19.0-20.0 sec   640 KBytes  5.24 Mbits/sec
[  3] 20.0-21.0 sec   640 KBytes  5.24 Mbits/sec
```

gNB Console Output

Following console output was taken while performing UL iperf test:

```
----------------------DL-----------------|-------------------UL-----------------
↪----
pci rnti  cqi  mcs  brate   ok  nok  (%) | pusch  mcs  brate   ok  nok  (%)  ␣
↪ bsr
  1 4601   15   27   5.8k   10    0   0% |  65.5   28    11M  396    0   0% ␣
↪5.45k
  1 4601   15   27   5.8k    9    0   0% |  65.5   28   5.7M  253    0   0% ␣
↪10.6k
  1 4601   15   27   5.2k   10    0   0% |  65.5   28   5.8M  247    0   0% ␣
↪1.04k
  1 4601   15   27   5.8k   10    0   0% |  65.5   28   8.2M  351    0   0%  ␣
↪ 0.0
  1 4601   15   27   5.8k   10    0   0% |  65.5   28   6.4M  265    0   0% ␣
↪5.45k
  1 4601   15   27   5.8k   10    0   0% |  65.5   28   5.8M  255    0   0% ␣
↪5.45k
  1 4601   15   27   5.8k   10    0   0% |  65.5   28   5.0M  246    0   0%  ␣
↪108k
  1 4601   15   27   5.8k   10    0   0% |  65.5   28    11M  412    0   0%  ␣
↪ 0.0
  1 4601   15   27   5.8k   10    0   0% |  65.5   28   4.9M  234    0   0% ␣
↪7.59k
  1 4601   15   27   5.8k   10    0   0% |  65.5   28   9.3M  347    0   0% ␣
↪10.6k
  1 4601   15   27   5.8k   10    0   0% |  65.5   28   6.7M  278    0   0% ␣
↪5.45k
```

Packet Capture

- `NGAP packet capture of UE attach scenario`

22.4.4 Configuring Amarisoft for multiple UEs

The following config files were modified for simulating multiple UEs:

- `Multiple UEs config FDD band 7`
- `Multiple UEs config TDD band 78`

The main modifications needed are:

- Set `multi_ue` to true.
- Add multiple entries with UE details (IMSI, K, OPc etc.) under `ue_list`.
- Add `global_timing_advance: 0` in the cell entry under `cells` section.
- Add the following configuration under each entry in `ue_list` and increase the value of `start_time` for each UE to have a staggered UE bring up.

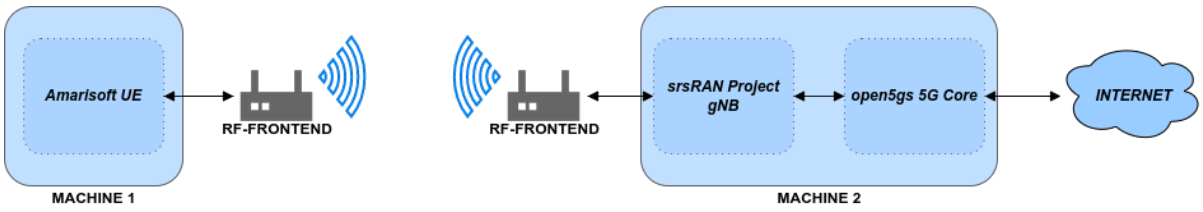
```
sim_events: [
  {
    event: "power_on",
    start_time: 0,
  },
]
```

Note: When `multi_ue` to false, make sure to have only one entry in `ue_list`.

Note: Make sure to configure all the subscribers through the Open5GS WebUI, before attempting to attach UEs.

22.5 Over-the-air Setup

In this section, we describe the steps required to configure the SDR RF driver in both gNB and Amarisoft UE. The following diagram presents the setup architecture:



22.5.1 Configuration

You can find the srsRAN Project gNB configuration file for this example in the `configs` folder of the srsRAN Project source files. You can also find it here.

- gNB TDD band 78 config

You can download the AmariUE configs here:

- `Single UE config FDD band 7`
- `Single UE config TDD band 78`

Details of the modifications made compared to ZMQ setup are outlined in following sections.

srsRAN Project gNB

Modify the `ru_sdr` section to send and receive radio samples via UHD driver:

```
ru_sdr:
  device_driver: uhd
  device_args: type=b200,num_recv_frames=64,num_send_frames=64
  srate: 11.52
  otw_format: sc12
  tx_gain: 80
  rx_gain: 40
```

Amarisoft UE

Modify the `ru_sdr` section of Amarisoft UE configuration to send and receive radio samples via UHD driver:

```
ru_sdr: {
    name: "uhd",
    sync: "none",
#if CELL_BANDWIDTH < 5
    args: "send_frame_size=512,recv_frame_size=512",
#elif CELL_BANDWIDTH == 5
    args: "send_frame_size=1024,recv_frame_size=1024",
#elif CELL_BANDWIDTH == 10
    args: "",
#elif CELL_BANDWIDTH > 10
    args: "num_recv_frames=64,num_send_frames=64",
    dl_sample_bits: 12,
    ul_sample_bits: 12,
#endif
    rx_antenna: "rx",
},
```

Then, set the gain parameters and timing offset as follows:

```
tx_gain: 75.0, /* TX gain (in dB) B2x0: 0 to 89.8 dB */
rx_gain: 40.0, /* RX gain (in dB) B2x0: 0 to 73 dB */
tx_time_offset: -150, /* in samples */
```

Make sure the CELL_BANDWIDTH matches that on gNB configuration file:

```
#define CELL_BANDWIDTH 10
```

Then, set `N_ANTENNA_DL` to 1:

```
#define N_ANTENNA_DL 1
```

22.5.2 Running the 5G Network

The following order should be used when running the network:

1. 5GC
2. gNB
3. UE

Open5gs 5G Core

Running the 5GC is same as in the ZMQ based setup.

srsRAN Project gNB

Let's now launch the gNB from its build directory.

```
sudo ./apps/gnb/gnb -c gnb_rf_b210_tdd_n78_10mhz.yml
```

The console output should be similar as follows:

```
Available radio types: uhd and zmq.

--== srsRAN gNB (commit 87c3fe355) ==--

Connecting to AMF on 172.22.0.10:38412
[INFO] [UHD] linux; GNU C++ version 11.3.0; Boost_107400; UHD_4.4.0.0-
↪0ubuntu1~jammy1
[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...
```

(continues on next page)

(continued from previous page)

```
[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.
Cell pci=1, bw=10 MHz, dl_arfcn=632628 (n78), dl_freq=3489.42 MHz, dl_ssb_
↪arfcn=632640, ul_freq=3489.42 MHz

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

Amarisoft UE

Lastly we can launch the UE, with root permissions to create the TUN device.

```
/root/ue/lteue /root/ue/config/ue-nr-sa-tdd-n78-b210-single-ue.cfg
```

The above command should start the UE and attach it to the core network. If UE connects successfully to the network, the following (or similar) should be displayed on the console:

```
[INFO] [UHD] linux; GNU C++ version 9.2.1 20200304; Boost_107100; UHD_3.15.0.
↪0-2build5
[INFO] [MPMD FIND] Found MPM devices, but none are reachable for a UHD␣
↪session. Specify find_all to find all devices.
[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.
RF0: sample_rate=11.520 MHz dl_freq=3489.420 MHz ul_freq=3489.420 MHz (band␣
↪n78) dl_ant=1 ul_ant=1
(ue) Warning: config/ru_sdr/config_tdd.cfg:25: unused property 'tx_time_offset
↪'
[INFO] [MULTI_USRP]     1) catch time transition at pps edge
[INFO] [MULTI_USRP]     2) set times next pps (synchronously)
Chan Gain(dB)   Freq(MHz)
 TX1     70.0 3489.420000
```

(continues on next page)

(continued from previous page)

```
 RX1      40.0 3489.420000
Cell 0: SIB found
```

And, once connected you can use the Amarisoft UE command line tool to verify whether its attached to cell as follows:

```
(ue) cells
Cell #0 / NR:
  PCI:    1
  TDD:    config=0, ssf=0
  EARFCN: DL=632628 UL=632628
  RB:     DL=24 UL=24
```

It is clear that the connection has been made successfully once the UE has been assigned an IP, this is seen in `PDU Session Establishment successful. IP: 192.168.100.2`. The NR connection is then confirmed with the `RRC NR reconfiguration successful.` message.

22.5.3 Testing the Network

Here, we demonstrate how to use ping and iPerf3 tools to test the connectivity and throughput in the network.

Ping

To exchange traffic in the `downlink` direction, i.e. from the the 5GC (UPF), just run ping as usual on the command line, e.g.:

```
ping 192.168.100.2
```

In order to generate traffic in the `uplink` direction it is important to run the ping command using the TUN interface of UE (e.g. tun0 created once Amarisoft UE attaches to 5G network).

```
ip netns exec ue0 ping 192.168.100.1 -I tun0
```

iPerf

For generating `downlink` traffic, we run iperf client on the 5GC (UPF) and server on the UE as follows:

In the UE,

```
ip netns exec ue0 iperf -u -s -i 1
```

In the UPF,

```
iperf -u -c 192.168.100.2 -b 10M -i 1 -t 60
```

And, for `uplink` traffic, we run iperf server on the 5GC (UPF) and client on the UE as follows.

In the UPF,

```
iperf -u -s -i 1
```

In the UE,

```
ip netns exec ue0 iperf -u -c 192.168.100.1 -b 3M -i 1 -t 60
```

iPerf Output

Example `server` iPerf3 output:

```
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 32.0 MByte (default)
------------------------------------------------------------
[  3] local 10.45.0.52 port 5001 connected with 10.45.0.1 port 33894
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total␣
↪Datagrams
[  3]  0.0- 1.0 sec  1.25 MBytes  10.5 Mbits/sec   0.838 ms    0/  893 (0%)
[  3]  1.0- 2.0 sec  1.25 MBytes  10.5 Mbits/sec   0.909 ms    0/  892 (0%)
[  3]  2.0- 3.0 sec  1.25 MBytes  10.5 Mbits/sec   0.839 ms    0/  891 (0%)
[  3]  3.0- 4.0 sec  1.25 MBytes  10.5 Mbits/sec   0.905 ms    0/  892 (0%)
[  3]  4.0- 5.0 sec  1.25 MBytes  10.5 Mbits/sec   0.871 ms    3/  892 (0.34%)
[  3]  5.0- 6.0 sec  1.25 MBytes  10.5 Mbits/sec   0.878 ms    0/  891 (0%)
[  3]  6.0- 7.0 sec  1.25 MBytes  10.5 Mbits/sec   0.922 ms    0/  892 (0%)
[  3]  7.0- 8.0 sec  1.25 MBytes  10.5 Mbits/sec   0.921 ms    0/  892 (0%)
[  3]  8.0- 9.0 sec  1.25 MBytes  10.5 Mbits/sec   0.842 ms    0/  891 (0%)
[  3]  0.0-10.0 sec  12.5 MBytes  10.5 Mbits/sec   0.932 ms    3/ 8917 (0.034
↪%)
```

Example `client` iPerf3 output:

```
------------------------------------------------------------
Client connecting to 10.45.0.52, UDP port 5001
Sending 1470 byte datagrams, IPG target: 1121.52 us (kalman adjust)
UDP buffer size: 32.0 MByte (default)
------------------------------------------------------------
[  3] local 10.45.0.1 port 33894 connected with 10.45.0.52 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec  1.25 MBytes  10.5 Mbits/sec
[  3]  1.0- 2.0 sec  1.25 MBytes  10.5 Mbits/sec
[  3]  2.0- 3.0 sec  1.25 MBytes  10.5 Mbits/sec
[  3]  3.0- 4.0 sec  1.25 MBytes  10.5 Mbits/sec
[  3]  4.0- 5.0 sec  1.25 MBytes  10.5 Mbits/sec
[  3]  5.0- 6.0 sec  1.25 MBytes  10.5 Mbits/sec
[  3]  6.0- 7.0 sec  1.25 MBytes  10.5 Mbits/sec
[  3]  7.0- 8.0 sec  1.25 MBytes  10.5 Mbits/sec
[  3]  8.0- 9.0 sec  1.25 MBytes  10.5 Mbits/sec
[  3]  0.0-10.0 sec  12.5 MBytes  10.5 Mbits/sec
[  3] Sent 8917 datagrams
```

(continues on next page)

(continued from previous page)

```
[  3] Server Report:
[  3]  0.0-10.0 sec  12.5 MBytes  10.5 Mbits/sec   0.931 ms    3/ 8917 (0.034
↪%)
```

gNB Console Output

Following console output was taken while performing DL iperf test:

```
----------------------DL----------------|-----------------UL----------------
↪----
pci rnti  cqi  mcs  brate   ok  nok  (%) | pusch  mcs  brate   ok  nok  (%)  ␣
↪ bsr
  1 4601    4    0      0    0    0   0% | -21.3    0   3.0k    0    5   0%  ␣
↪ 0.0
  1 4602   15   28    11M  973    0   0% |   n/a    0      0    0    0   0%  ␣
↪ 0.0
  1 4601    4    0      0    0    0   0% |   n/a    0      0    0    0   0%  ␣
↪ 0.0
  1 4602   15   28    11M  976    0   0% |   n/a    0      0    0    0   0%  ␣
↪ 0.0
  1 4601    4    0      0    0    0   0% |   n/a    0      0    0    0   0%  ␣
↪ 0.0
  1 4602   15   28    11M  976    0   0% |   n/a    0      0    0    0   0%  ␣
↪ 0.0
  1 4601    4    0      0    0    0   0% |   n/a    0      0    0    0   0%  ␣
↪ 0.0
  1 4602   15   28    11M  973    9   0% |  27.4   28   4.4k    1    0   0%  ␣
↪ 0.0
  1 4601    4    0      0    0    0   0% |   n/a    0      0    0    0   0%  ␣
↪ 0.0
  1 4602   15   28    11M  973    1   0% |  28.5   28   4.4k    1    0   0%  ␣
↪ 0.0
  1 4601    4    0      0    0    0   0% | -21.0    0   3.0k    0    5   0%  ␣
↪ 0.0
  1 4602   15   28    11M  980    0   0% |  30.3   28   4.4k    1    0   0%  ␣
↪ 0.0
  1 4601    4    0      0    0    0   0% |   n/a    0      0    0    0   0%  ␣
↪ 0.0
  1 4602   15   28    11M  972    0   0% |   n/a    0      0    0    0   0%  ␣
↪ 0.0
  1 4601    4    0      0    0    0   0% |   n/a    0      0    0    0   0%  ␣
↪ 0.0
  1 4602   15   28    11M  976    0   0% |   n/a    0      0    0    0   0%  ␣
↪ 0.0
  1 4601    4    0      0    0    0   0% |   n/a    0      0    0    0   0%  ␣
↪ 0.0
  1 4602   15   28   8.1M  732    0   0% |  25.9   28   414k   43    4   8%  ␣
↪ 0.0
```

Packet Capture

- `NGAP packet capture of UE attach scenario`

22.6 Troubleshooting

22.6.1 ZMQ setup

Amarisoft UE in ZMQ based setup may not connect to srsRAN Project gNB if the time between gNB bring up and UE bring up too large. So its advised to run immediately after running gNB for better results.

22.6.2 Reference clock for over-the-air

If you encounter issues with the srsUE not finding the cell and/or not being able to stay connected it might be due to inaccurate clocks at the RF frontends. Try to use an external 10 MHz reference or use a GPSDO oscillator.

22.6.3 5G QoS Identifier

By default, Open5GS uses 5QI = 9. If the **qos** section is not provided in the gNB config file, the default one with 5QI = 9 will be generated and the UE should connect to the network. If one needs to change the 5QI, please harmonize these settings between gNB and Open5GS config files, as otherwise, a UE will not be able to connect.

CHAPTER

TWENTYTHREE

SRSRAN GNB ON KUBERNETES

23.1 Introduction

This tutorial outlines the steps required to launch the srsRAN CU/DU with Kubernetes. This is a useful tool for use-cases that require users to deploy and manage networks over a prolonged period, with high levels of guaranteed service ability and enhanced fault tolerance.

In short, Kubernetes can be described as the following:

> *When it comes to managing test networks, the adoption of Kubernetes is pivotal for streamlined and efficient operations. Kubernetes provides a unified platform for orchestrating diverse functions within the network, optimizing resource utilization, and automating scaling based on demand. Its inherent capability to enhance fault tolerance ensures consistent service availability, while the ease of continuous deployment supports rapid innovation. Kubernetes also excels in simplifying the deployment process across various environments, fostering adaptability. In essence, Kubernetes emerges as a practical solution, offering a cohesive and scalable framework for effective network function management.*

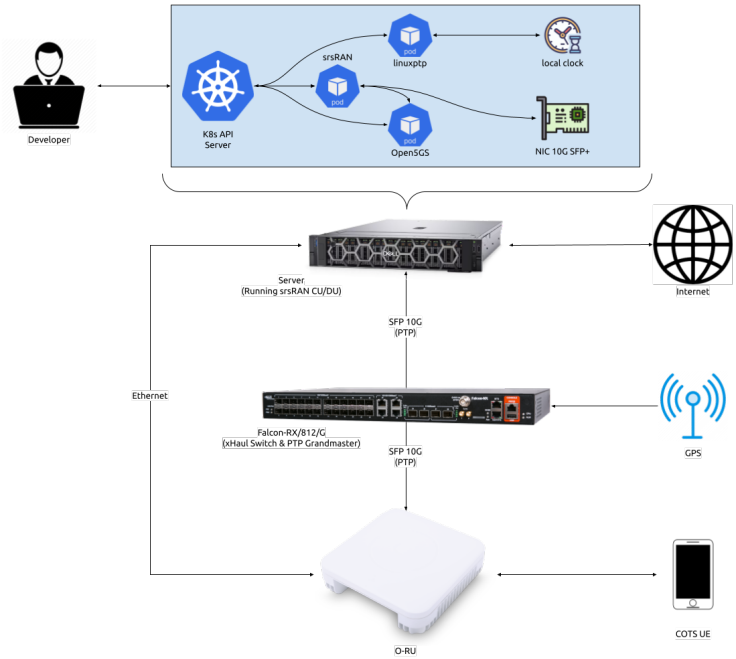
Such deployments may not be suited to more research and development focused use-cases that require fine-tuning of configuration files and development of source-code. Iterative development and testing is more suited to "bare metal" deployments.

23.1.1 Further Reading

We recommend unexperienced users read the following two articles before starting with this tutorial:

- Getting started with Kubernetes
- Getting started with Helm

23.2 Setup Considerations



This tutorial uses the following hardware:

- Server (Running srsRAN Project CU/DU)
  - CPU: AMD Ryzen 7 5700G
  - MEM: 64GB
  - NIC: Intel Corporation 82599ES 10-Gigabit
  - OS: Ubuntu 22.04 (5.15.0-1037-realtime)
- Falcon-RX/812/G xHaul Switch (w/ PTP grandmaster)
- Foxconn RPQN-7800E (v3.1.13q.551p1)
- COTS UE

This tutorial requires a functional Kubernetes cluster v1.24 or newer. We are using Ubuntu 22.04 on the worker node. If you chose to use another operating system, please refer to the operating system specific documentation for the installation of Kubernetes and configuration of the system. As previously stated, this tutorial requires a basic understanding of Kubernetes and Helm.

23.2.1 CU/DU

The CU/DU is provided by the srsRAN Project gNB. The Open Fronthaul (OFH) Library provides the necessary interface between the DU and the RU. The DU is connected to the Falcon switch via SFP+ fiber cable.

23.2.2 RU

The Foxconn RPQN-7800E RU is used as the RU in this setup. This is a Split 7.2x indoor O-RU. The RU is connected to the Falcon-RX via SFP+ fiber cable through the main fronthaul interface.

23.2.3 5G Core

For this tutorial we use the Open5GS 5G Core.

Open5GS is a C-language open-source implementation for 5G Core and EPC. The following links will provide you with the information needed to download and setup Open5GS so that it is ready to use with srsRAN:

- Open5GS GitHub
- Open5GS Quickstart Guide

23.2.4 Clocking & Synchronization

The split 7.2 interface requires tight timing synchronization between the DU and RU. O-RAN WG 4 has defined various synchronization methods for use with Open Fronthaul. These are outlined in O-RAN.WG4.CUS.0-R003-v11.00 Section 11.

In this setup we use LLS-C3. The LLS-C3 configuration enables the distribution of network timing between central sites and remote sites from PRTC/T-GM to RU. In simpler terms, it allows the synchronization of one or more PRTC/T-GM devices (serving as PTP masters) in the fronthaul network to transmit network timing signals to DU and RU components as seen in the figure above. In our setup the Falcon switch is acting as the PTP grandmaster (which is synchronized via GPS), providing timing to the RU and the DU. These are connected to the SFP+ 10G ports on the switch via Ethernet.

Switch

The Falcon-RX/812/G switch is a 5G xHaul timing-aware O-RAN switch & PTP grandmaster. This is used to provide timing synchronization to both the DU and RU.

23.3 Configuration

23.3.1 Kubernetes Worker Node

In this section we will cover the configuration of the Kubernetes worker node.

NIC Configuration

The NIC used for OFH traffic needs to support jumbo frames. In our example the NIC port has the name `eth0`. To configure the `MTU` to `9600`, use the following command:

```
ifconfig eth0 mtu 9600 up
```

Depending on the network configuration you we advice to use DPDK. For 2x2 MIMO DPDK is not required, while for 4x4 MIMO DPDK is recommended. This tutorial will cover configuration for a 2x2 MIMO set up.

A tutorial outlining the use of DPDK with srsRAN will be available soon.

23.3.2 PTP Grandmaster

In this tutorial we will be using the Falcon-RX/812/G switch as the PTP grandmaster in a LLS-C3 configuration. The switch is synchronized via GPS. The switch is connected to the DU and RU via SFP+ fiber cable.

For more information on configuring the Falcon xHaul switch, see *this section* of the RU tutorial.

23.3.3 Helm Charts

Install the srsRAN Helm repositories with the following command:

```
helm repo add srsran https://srsran.github.io/srsRAN_Project_helm/
```

To deploy Open5GS on Kubernetes we use the Helm charts from the Openverso Project. Install the Openverso Helm repository with the following command:

```
helm repo add openverso https://gradiant.github.io/openverso-charts/
```

LinuxPTP Helm Chart

Download and adjust the LinuxPTP `values.yaml` file to configure the deployment based on your cluster. For more information on the explicit configuration of the LinuxPTP container, have look at the LinuxPTP Documentation.

```
wget https://raw.githubusercontent.com/srsran/srsRAN_Project_helm/main/charts/
↪linuxptp/values.yaml
```

In this tutorial we will be using the `G.8275.1` profile with the default configuration of LinuxPTP. Therefore, we adjust the config section of the `values.yaml` file as follows:

```
config:
   dataset_comparison: "G.8275.x"
   G.8275.defaultDS.localPriority: "128"
   maxStepsRemoved: "255"
   logAnnounceInterval: "-3"
   logSyncInterval: "-4"
```

(continues on next page)

(continued from previous page)

```
  logMinDelayReqInterval: "-4"
  serverOnly: "0"
  G.8275.portDS.localPriority: "128"
  ptp_dst_mac: "01:80:C2:00:00:0E"
  network_transport: "L2"
  domainNumber: "24"
```

Apart from that, we need to set the `interface_name` parameter to the interface name of the NIC used for the PTP traffic.

For more information on the parameters of the `values.yaml` file refer to `charts/linuxptp/README.md` in this repository.

Docker images are available on Docker Hub. The Dockerfile for the LinuxPTP container can be found in `images/linuxptp` in this repository.

srsRAN Project Helm Chart

Download and adjust the srsRAN Project `values.yaml` file to configure the deployment based on your cluster. For more information on the explicit configuration of the srsRAN CU/DU for various RUs, see *this tutorial*.

```
wget https://raw.githubusercontent.com/srsran/srsRAN_Project_helm/main/charts/
↪srsran-project/values.yaml
```

One of the necessary parameters to configure are `bind_addr` and `amf_addr`. `bind_addr` is the IP address of the Kubernetes worker node. The `amf_addr` is the IP address of the Open5GS AMF. Obtain the IP address of the Kubernetes worker node with the following command:

```
kubectl get nodes -o wide
```

You should see an output similar to the following:

```
$ kubectl get nodes -o wide
NAME        STATUS   ROLES           AGE   VERSION   INTERNAL-IP   EXTERNAL-
↪IP   OS-IMAGE             KERNEL-VERSION         CONTAINER-RUNTIME
preskit-1   Ready    control-plane   52d   v1.26.5   10.12.1.223   <none>      ␣
↪   Ubuntu 22.04.3 LTS   5.15.0-1032-realtime   containerd://1.7.1
```

Set `bind_addr` to the `INTERNAL-IP` value of the Kubernetes worker node where you want to deploy the srsRAN Project CU/DU. Obtain the IP address of the Open5GS AMF with the following command:

```
kubectl get pods -A -o wide
```

You should see an output similar to the following:

```
$ kubectl get pods -A -o wide
NAME                                        READY   STATUS    RESTARTS      ␣
↪AGE   IP              NODE        NOMINATED NODE   READINESS GATES
open5gs-amf-9d5788cdc-qhzkb                 1/1     Running   0             ␣
↪18m   10.233.124.24   preskit-1   <none>           <none>
```

(continues on next page)

(continued from previous page)

```
open5gs-ausf-66547fc8bd-4p4r5                1/1     Running   0              ␣
↪18m   10.233.124.51   preskit-1   <none>           <none>
open5gs-bsf-59966bf9cb-rz262                 1/1     Running   0              ␣
↪18m   10.233.124.59   preskit-1   <none>           <none>
open5gs-mongodb-749b78fd9f-v96gg             1/1     Running   0              ␣
↪18m   10.233.124.4    preskit-1   <none>           <none>
open5gs-nrf-7995d7c5bf-rxh7s                 1/1     Running   0              ␣
↪18m   10.233.124.15   preskit-1   <none>           <none>
open5gs-nssf-86c9f9d5cd-97lr6                1/1     Running   0              ␣
↪18m   10.233.124.17   preskit-1   <none>           <none>
open5gs-pcf-74c8468bcc-jwn7s                 1/1     Running   2 (18m ago)    ␣
↪18m   10.233.124.26   preskit-1   <none>           <none>
open5gs-populate-6444478f56-kb8cd            1/1     Running   0              ␣
↪18m   10.233.124.55   preskit-1   <none>           <none>
open5gs-smf-779bc766ff-gcps9                 1/1     Running   0              ␣
↪18m   10.233.124.11   preskit-1   <none>           <none>
open5gs-udm-5ffc6fc456-9g9n2                 1/1     Running   0              ␣
↪18m   10.233.124.30   preskit-1   <none>           <none>
open5gs-udr-7b5499cd89-2kb8l                 1/1     Running   2 (18m ago)    ␣
↪18m   10.233.124.58   preskit-1   <none>           <none>
open5gs-upf-796655fbcc-5sd6s                 1/1     Running   0              ␣
↪18m   10.233.124.19   preskit-1   <none>           <none>
open5gs-webui-c6c949568-fp2r9                1/1     Running   0              ␣
↪18m   10.233.124.1    preskit-1   <none>           <none>
```

Search for a Pod called `open5gs-amf-*` and set `amf_addr` to the `IP` value of the Pod.

For more information on the configuration of the CU/DU please refer to the srsRAN gNB with COTS UEs Tutorial. For more information on the parameters of the `values.yaml` file refer to `charts/srsran-project/README.md` in this repository.

Docker images are available on Docker Hub. Dockerfiles for the srsRAN Project container can be found in `images/srsran-project` in this repository.

A detailed breakdown on connecting the CU/DU to open5GS can be found in *this tutorial*.

Core

Download and adjust the following Open5GS `values.yaml` file to configure the deployment based on your needs.

```
wget https://gradiant.github.io/openverso-charts/docs/open5gs-ueransim-gnb/
↪5gSA-values.yaml
```

For information on the initCommands section to insert subscribers please refer to this file. For the Open5GS configuration, please refer to the Open5GS Documentation.

23.4 Initializing the Network

23.4.1 RU

Bring up the RU and ensure it is running correctly before attempting to connect the DU.

23.4.2 Core

By default the MongoDB helm chart involved in this deployment uses persistent volumes through a persistent volume claim. In this example we will deploy Open5GS without persistent storage for the sake of simplicity. That means that the user database will be reset when you restart the MongoDB container. If you want to have persistent storage for MongoDB you need to configure a PV provisioner like OpenEBS.

Obtain and configure the `values.yaml` file like described above. After that, deploy Open5GS with the following command:

```
helm install open5gs openverso/open5gs --version 2.0.8 --values=5gSA-values.
↪yaml -n open5gs --create-namespace --set mongodb.persistence.enabled=false
```

You should see the following output:

```
NAME: open5gs
LAST DEPLOYED: Tue Nov 21 16:55:12 2023
NAMESPACE: open5gs
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Wait until all Pods are running. You can check the status with the following command:

```
kubectl get pods -n open5gs
```

Once all components are started you can edit the subscribers via the webui. For that, you need to forward port `3000` of the `open5gs-webui` service to your local machine:

```
kubectl port-forward svc/open5gs-webui 3000:3000 -n open5gs
```

You should see the following output:

```
Forwarding from 127.0.0.1:3000 -> 3000
Forwarding from [::1]:3000 -> 3000
```

Don't close the shell and open your browser at http://localhost:3000. `Username: admin, Password: 1423` Once you are logged in you can edit the subscribers.

23.4.3 DU

PTP

Obtain and configure the `values.yaml` file like described above. After that, deploy LinuxPTP container with the following command:

```
helm install linuxptp srsran/linuxptp --values=values.yaml -n ptp --create-
↪namespace
```

You should see the following output:

```
NAME: linuxptp
LAST DEPLOYED: Tue Nov 21 16:53:36 2023
NAMESPACE: ptp
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

To verify that the deployment is working properly we need to get the name of our LinuxPTP Pod. Therefore, use `kubectl get pods -A` to list all Pods in the cluster. You should see an output similar to the following:

```
NAMESPACE     NAME                                      READY   STATUS    ␣
↪RESTARTS       AGE
ptp           linuxptp-854f797f64-vnmdt                 1/1     Running   0␣
↪             23d
kube-system   calico-kube-controllers-6dfcdfb99-xmdmx   1/1     Running   0␣
↪             22d
kube-system   calico-node-292sr                         1/1     Running   0␣
↪             23d
kube-system   coredns-645b46f4b6-lggfs                  1/1     Running   0␣
↪             22d
kube-system   dns-autoscaler-756ff885f8-7d6hr           1/1     Running   0␣
↪             22d
kube-system   kube-apiserver-preskit-1                  1/1     Running   0␣
↪             22d
kube-system   kube-controller-manager-preskit-1         1/1     Running   0␣
↪             22d
kube-system   kube-proxy-5mbbm                          1/1     Running   0␣
↪             23d
kube-system   kube-scheduler-preskit-1                  1/1     Running   0␣
↪             22d
kube-system   nodelocaldns-bfx77                        1/1     Running   0␣
↪             23d
```

The name of the Pod in our example is `linuxptp-854f797f64-vnmdt`. Now we can get logs from that Pod with this command:

```
kubectl logs linuxptp-854f797f64-vnmdt -n srsran
```

You should see an output similar to the following:

```
ptp4l[1328454.803]: rms   15 max   24 freq  -3622 +/-  19 delay   135 +/-   1
phc2sys[1328454.902]: CLOCK_REALTIME phc offset        -4 s2 freq   +2045␣
↪delay    431
phc2sys[1328455.027]: CLOCK_REALTIME phc offset        -7 s2 freq   +2040␣
↪delay    429
phc2sys[1328455.152]: CLOCK_REALTIME phc offset        11 s2 freq   +2056␣
↪delay    437
phc2sys[1328455.278]: CLOCK_REALTIME phc offset        -1 s2 freq   +2048␣
↪delay    431
phc2sys[1328455.403]: CLOCK_REALTIME phc offset        -1 s2 freq   +2047␣
↪delay    431
phc2sys[1328455.528]: CLOCK_REALTIME phc offset         6 s2 freq   +2054␣
↪delay    429
phc2sys[1328455.653]: CLOCK_REALTIME phc offset       -11 s2 freq   +2039␣
↪delay    430
phc2sys[1328455.778]: CLOCK_REALTIME phc offset        -9 s2 freq   +2037␣
↪delay    422
ptp4l[1328455.803]: rms   17 max   32 freq  -3622 +/-  21 delay   135 +/-   2
phc2sys[1328455.903]: CLOCK_REALTIME phc offset        -7 s2 freq   +2037␣
↪delay    430
phc2sys[1328456.028]: CLOCK_REALTIME phc offset         7 s2 freq   +2049␣
↪delay    430
phc2sys[1328456.154]: CLOCK_REALTIME phc offset        -6 s2 freq   +2038␣
↪delay    431
phc2sys[1328456.279]: CLOCK_REALTIME phc offset        11 s2 freq   +2053␣
↪delay    430
phc2sys[1328456.404]: CLOCK_REALTIME phc offset        -1 s2 freq   +2044␣
↪delay    431
phc2sys[1328456.529]: CLOCK_REALTIME phc offset         1 s2 freq   +2046␣
↪delay    430
phc2sys[1328456.654]: CLOCK_REALTIME phc offset         1 s2 freq   +2046␣
↪delay    430
phc2sys[1328456.779]: CLOCK_REALTIME phc offset         2 s2 freq   +2048␣
↪delay    430
```

CU/DU

Obtain and configure the `values.yaml` file like described above. PTP sync needs to be established on RU and DU, and the Kubernetes worker node needs to be configured before the deployment. Furthermore, the 5G core must be up and running.

Deploy the srsRAN Project with the following commands:

```
helm install srsran-project srsran/srsran-cu-du --values=values.yaml -n␣
↪srsran --create-namespace
```

You should see the following output:

```
NAME: srsran-project
LAST DEPLOYED: Tue Nov 21 16:54:12 2023
```

(continues on next page)

(continued from previous page)

```
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

To verify that the srsRAN Project Pod is working properly you can extract logs using the following steps:

1. First get the name of the Pod

   ```
   kubectl get pods -A
   ```

   You should see an output similar to the following:

   ```
   NAMESPACE     NAME                                       READY   STATUS
   ↪  RESTARTS      AGE
   default       srsran-project-l647c                       1/1     Running
   ↪  0             23d
   kube-system   calico-kube-controllers-6dfcdfb99-xmdmx    1/1     Running
   ↪  0             22d
   kube-system   calico-node-292sr                          1/1     Running
   ↪  0             23d
   kube-system   coredns-645b46f4b6-lggfs                   1/1     Running
   ↪  0             22d
   kube-system   dns-autoscaler-756ff885f8-7d6hr            1/1     Running
   ↪  0             22d
   kube-system   kube-apiserver-preskit-1                   1/1     Running
   ↪  0             22d
   kube-system   kube-controller-manager-preskit-1          1/1     Running
   ↪  0             22d
   kube-system   kube-proxy-5mbbm                           1/1     Running
   ↪  0             23d
   kube-system   kube-scheduler-preskit-1                   1/1     Running
   ↪  0             22d
   kube-system   nodelocaldns-bfx77                         1/1     Running
   ↪  0             23d
   ```

2. Find the Pod, in this example it is `srsran-project-l647c`. The logs can now be extracted with:

   ```
   kubectl logs srsran-project-l647c -n srsran
   ```

   You should see an output similar to the following:

   ```
   EAL: Detected CPU lcores: 16
   EAL: Detected NUMA nodes: 1
   EAL: Detected shared linkage of DPDK
   EAL: Multi-process socket /var/run/dpdk/rte/mp_socket
   EAL: Selected IOVA mode 'PA'
   EAL: VFIO support initialized
   EAL: Probe PCI driver: net_ice (8086:159b) device: 0000:01:00.1 (socket 0)
   ice_load_pkg(): failed to search file path
   ```

(continues on next page)

(continued from previous page)

```
ice_dev_init(): Failed to load the DDP package,Entering Safe Mode
TELEMETRY: No legacy callbacks, legacy socket not created

--== srsRAN gNB (commit 374200dee) ==--

Connecting to AMF on 10.44.50.221:38412
Initializing Open Fronthaul Interface sector=0: ul_comp=[BFP,9], dl_
↪comp=[BFP,9], prach_comp=[BFP,9] prach_cp_enabled=true, downlink_
↪broadcast=false.
ice_init_rss(): RSS is not supported in safe mode

ice_set_rx_function(): Using AVX512 Vector Scattered Rx (port 0).
ice_set_tx_function(): Using AVX512 Vector Tx (port 0).
ice_vsi_config_outer_vlan_stripping(): Single VLAN mode (SVM) does not␣
↪support qinq
Cell pci=1, bw=40 MHz, dl_arfcn=649980 (n78), dl_freq=3749.7 MHz, dl_ssb_
↪arfcn=649248, ul_freq=3749.7 MHz

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

23.5 Connecting to the Network

Once the network has been configured and is running, connecting the UE to the network is the same as in a bare-metal set-up.

The UE does not require any further modification or configuration outside of the normally required steps. You can follow the COTS UE tutorial to learn how to connect a COTS UE to the network. From the UEs perspective, it does not matter if an RU or USRP is being used as the frontend or if the CU/DU and 5GC are running on bare-metal or in Kubernetes containers.

23.6 Supported O-RUs

For more information on supported O-RUs, see this section of the RU tutorial.

CHAPTER

TWENTYFOUR

SRSRAN GNB WITH DPDK

Note: This tutorial assumes the machine being used has an Intel processor. For AMD processors some of the commands used will change. The overall steps will remain mostly the same.

24.1 Introduction

This tutorial outlines how to configure DPDK for use with the srsRAN CU/DU.

DPDK, which stands for Data Plane Development Kit, is a set of software libraries and drivers that is used to improve the performance of packet processing in the CU/DU.

Specifically, in the case of the srsRAN CU/DU, this can enable users to achieve higher throughput and a more stable performance with certain configurations. For instance, usecases that require users to run an O-RU in a 4x4 MIMO configuration can greatly benefit from using DPDK.

24.1.1 Further Reading

- DPDK Build Guide
- Linux Drivers
- Using Hugepages in a Linux Environment
- Running Hugepages
- EAL Parameters

24.2 Installing DPDK

Note: In this tutorial we will use `vfio-pci` but you can also use `igb_uio` or `uio_pci_generic`. For more information on the drivers please refer to this document.

As mentioned in the DPDK documentation:

VFIO kernel is usually present by default in all distributions, however please consult your distributions documentation to make sure that is the case.

To make use of full VFIO functionality, both kernel and BIOS must support and be configured to use IO virtualization (such as Intel® VT-d).

Please make sure that your system meets the above requirements before continuing. In case your system doesn't meet the requirements you can continue with the `igb_uio` module, as described in the DPDK documentation.

First install the following requirements:

```
sudo apt install build-essential tar wget python3-pip libnuma-dev meson ninja-
↪build python3-pyelftools
```

It is recommended to use DPDK 23.11 with srsRAN CU/DU. This can be installed with the following commands:

```
wget https://fast.dpdk.org/rel/dpdk-23.11.tar.xz
tar xvf dpdk-23.11.tar.xz dpdk-23.11/
cd dpdk-23.11/
meson setup build
cd build
ninja
sudo meson install
sudo ldconfig
```

In order to use the `vfio-pci module`, we need to enable IOMMU in the BIOS. This is usually done by default, but should still be checked. Furthermore, IOMMU needs to be activated in the kernel. Depending on your CPU, you may need to add `intel_iommu=on iommu=pt` for Intel CPUs or `amd_iommu=on iommu=pt` for AMD CPUs to the `GRUB_CMDLINE_LINUX_DEFAULT` line in `/etc/default/grub`. After this, update grub and reboot the system:

```
sudo update-grub
sudo reboot
```

The next step is to ensure the `vfio-pci` module is loaded correctly. This can be done with the following command:

```
sudo modprobe vfio-pci
```

Verify that `vfio-pci` was loaded correctly with the following command:

```
sudo ./usertools/dpdk-devbind.py -s
```

You should see an output similar to the following:

```
Network devices using DPDK-compatible driver
============================================

Network devices using kernel driver
===================================
0000:01:00.0 '82599ES 10-Gigabit SFI/SFP+ Network Connection 10fb'␣
```

(continues on next page)

(continued from previous page)

```
 ↪if=enp1s0f0 drv=ixgbe unused=igb_uio,vfio-pci *Active*
0000:01:00.1 '82599ES 10-Gigabit SFI/SFP+ Network Connection 10fb' ␣
 ↪if=enp1s0f1 drv=ixgbe unused=igb_uio,vfio-pci *Active*
0000:05:00.0 'Ethernet Controller E810-XXV for SFP 159b' if=enp5s0f0 drv=ice ␣
 ↪unused=igb_uio,vfio-pci *Active*
0000:05:00.1 'Ethernet Controller E810-XXV for SFP 159b' if=enp5s0f1 drv=ice ␣
 ↪unused=igb_uio,vfio-pci *Active*
0000:09:00.0 'RTL8125 2.5GbE Controller 8125' if=enp9s0 drv=r8169 unused=igb_
 ↪uio,vfio-pci
```

`unused=vfio-pci *Active*` confirms that the `vfio_pci` module was loaded correctly.

If the `vfio-pci` module is not present, it can have multiple issues which are out of scope of this tutorial. Please refer to your OS maintainer's or CPU vendor's documentation for more information if this is the case.

You can continue with the `igb_uio` module, as described below if necessary.

> **Warning:** Only do this if you were unable to correctly load the `vfio_pci` module.

You can install and load `igb_uio` with the following commands:

```
git clone http://dpdk.org/git/dpdk-kmods
cd dpdk-kmods/linux/igb_uio
make
sudo modprobe uio # Ensure uio module is loaded
sudo insmod igb_uio.ko
```

Ensure that the module was loaded correctly with the following command:

```
lsmod | grep uio
```

You should see an output similar to the following:

```
igb_uio                36864  0
uio                    24576  1 igb_uio
```

If the module is not present use `dmesg` to check for potential errors:

```
sudo dmesg -T
```

For more information and troubleshooting tips please refer to back to the DPDK documentation and that of your OS maintainers.

24.3 Configuring DPDK

24.3.1 Configure Hugepages

DPDK requires `hugepages` to be configured to run correctly. THe `dpdk-hugepages.py` helper script can be used to configure this correctly:

```
sudo ./dpdk-hugepages.py -p 1G --setup 8G
```

To make these changes persistent across boot-cycles, run the following:

```
sudo mkdir -p /mnt/huge
```

Then add the following line at the end of `/etc/fstab`:

```
nodev /mnt/huge hugetlbfs pagesize=1G 0 0
```

and edit this line in `/etc/default/grub`:

```
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash intel_iommu=on iommu=pt␣
↪hugepagesz=1G hugepages=8 hugepagesz=2M hugepages=0 default_hugepagesz=1G"
```

After that, update the grub config and reboot the system:

```
sudo update-grub
sudo reboot
```

After reboot verify that the hugepages are configured correctly with the following commands:

```
cat /proc/cmdline
cat /proc/meminfo
```

You should see an output similar to the following:

```
cat /proc/cmdline

BOOT_IMAGE=/vmlinuz-5.15.0-1037-realtime root=/dev/mapper/ubuntu--vg-ubuntu--
↪lv quiet splash intel_iommu=on iommu=pt hugepagesz=1G hugepages=8␣
↪hugepagesz=2M hugepages=0 default_hugepagesz=1G

cat /proc/meminfo

[...]
HugePages_Total:       8
HugePages_Free:        8
HugePages_Rsvd:        0
HugePages_Surp:        0
[...]
```

Once the driver and hugepages are set up successfully the desired interface can then be bound to DPDK.

24.3.2 Binding to DPDK

We use `dpdk-devbind.py` helper script to find interface name and bus ID:

```
sudo ./dpdk-devbind.py -s
```

You should see the following output or similar:

```
Network devices using kernel driver
===================================
0000:01:00.0 82599ES 10-Gigabit SFI/SFP+ Network Connection 10fb if=enp1s0f0␣
↪drv=ixgbe unused=igb_uio,vfio-pci,uio_pci_generic
0000:01:00.1 82599ES 10-Gigabit SFI/SFP+ Network Connection 10fb if=enp1s0f1␣
↪drv=ixgbe unused=igb_uio,vfio-pci,uio_pci_generic
0000:03:00.0 RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller 8168␣
↪if=enp3s0 drv=r8169 unused=igb_uio,vfio-pci,uio_pci_generic *Active*
```

The network card we want to use is `82599ES 10-Gigabit SFI/SFP+ Network Connection`, port 1. Interface name `enp1s0f1`, bus ID `0000:01:00.1`. The next step is to bind the desired port to `vfio-pci`. Some NICs require deactivating the interface before binding. Use the following commands to achieve this:

```
sudo ifconfig enp1s0f1 down
sudo ./dpdk-devbind.py --bind vfio-pci 0000:01:00.1
```

To test that the device has been bound successfully the following command can be used:

```
sudo ./dpdk-devbind.py -s
```

You should see the following output, or similar:

```
Network devices using DPDK-compatible driver
============================================
0000:01:00.1 82599ES 10-Gigabit SFI/SFP+ Network Connection 10fb if=enp1s0f1␣
↪drv=vfio-pci unused=igb_uio,uio_pci_generic,ixgbe
```

If the bind was successful, the output will show `drv=vfio-pci`.

24.3.3 EAL Parameters

EAL (Environmental Abstraction Layer) Parameters are used in DPDK to provide a set of functions and abstractions for common environment-related tasks such as memory allocation, thread management, and initialization. The DPDK documentation covers EAL parameters here.

In the context of srsRAN we use the `eal_args` parameter in the configuration file to instruct DPDK which cores to use for certain processes. If not configured correctly, then the CU/DU will not exploit the improvements in performance that comes with using DPDK. This is because DPDK will only see a single core and will not be able to run concurrent processes correctly.

The EAL parameters, as defined in the above document, are simply passed as as argument to the `eal_args` in the CU/DU config. Any of the parameters mentioned in the document can be set in this way.

An example configuration is as follows:

```
hal:
   eal_args: "--lcores '(0-1)@(0-23)'"
```

This will tell DPDK that EAL threads [0 - 1] should use cores [0 - 23]. This is assuming the CU/DU is running on a machine with 24 cores.

Another example is:

```
hal:
   eal_args: "--lcores (0-1)@(0-23) -a 0000:52:00.0"
```

This configuration tells DPDK that EAL threads [0 - 1] should use cores [0 - 23] for the device bound to the address [0000:52:00.0].

24.4 Running srsRAN with DPDK

Once DPDK has been installed and configured, running srsRAN is the same as a vanilla configuration. If DPDK is running with srsRAN correctly you should see the following console output:

```
EAL: Detected CPU lcores: 24
EAL: Detected NUMA nodes: 1
EAL: Detected shared linkage of DPDK
EAL: Multi-process socket /var/run/dpdk/rte/mp_socket
EAL: Selected IOVA mode 'VA'
EAL: VFIO support initialized
EAL: Using IOMMU type 1 (Type 1)
EAL: Probe PCI driver: net_bnxt (14e4:1751) device: 0000:18:00.2 (socket 0)
TELEMETRY: No legacy callbacks, legacy socket not created

---== srsRAN gNB (commit ) ==---

Connecting to AMF on 192.168.20.100:38412
Initializing the Open FrontHaul Interface for sector#0: ul_compr=[BFP,9], dl_
↪compr=[BFP,9], prach_compr=[BFP,9] prach_cp_enabled=true, downlink_
↪broadcast=false.
Cell pci=1, bw=100 MHz, dl_arfcn=625000 (n78), dl_freq=3375.0 MHz, dl_ssb_
↪arfcn=622272, ul_freq=3375.0 MHz

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

The first lines beginning with `EAL` tell us that the CU/DU is successfully running with DPDK, specifically the third line which reads `Detected shared linkage of DPDK`.

CHAPTER

TWENTYFIVE

O-RAN 7.2 RU GUIDE

25.1 Overview

The srsRAN Project supports both split 7.2 and split 8 fronthaul interfaces to Radio Units (RUs). This tutorial outlines the general steps required to interface an RU with the srsRAN CU/DU via split 7.2. For more detailed instructions on specific O-RUs select an RU from *this list*.

Split 7.2 is an open specification published by the O-RAN Alliance aiming to ensure interoperability between different DU and RU solutions.

The split 7.2 interface is supported in srsRAN through the Open Fronthaul (OFH) Library. Developed by the SRS team, OFH is an open-source, portable library with minimal 3rd-party dependencies. It has been designed to minimize the integration and configuration burden associated with using srsRAN with 3rd-party O-RUs.

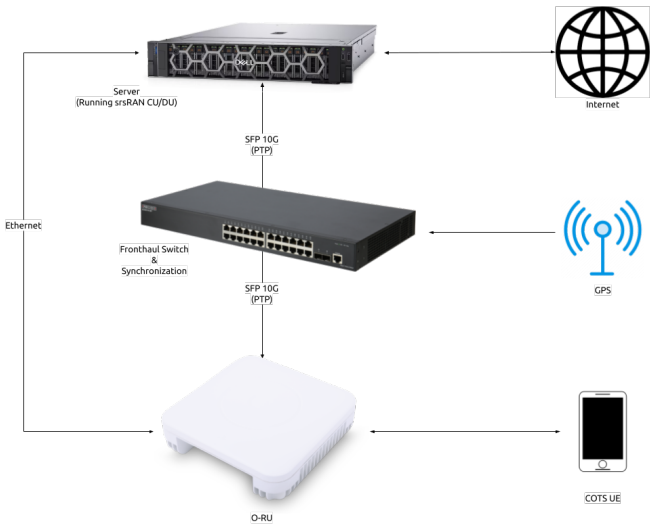
25.2 Setup Considerations

This tutorial uses the following hardware:

- Server (Running srsRAN Project CU/DU)
  - CPU: AMD Ryzen 7 5700G
  - MEM: 64GB
  - NIC: Intel Corporation 82599ES 10-Gigabit
  - OS: Ubuntu 22.04 (5.15.0-1037-realtime)
- Fronthaul switch & synchronization
- RU
- COTS UE (OnePlus 9)

With the following software:

- srsRAN Project
- Open5GS 5G Core



25.2.1 CU/DU

The CU/DU is provided by the srsRAN Project gNB. The Open Fronthaul (OFH) Library provides the necessary interface between the DU and the RU. The DU is connected to the fronthaul switch via SFP+ fiber cable.

25.2.2 RU

Users should chose the RU that best suits their usecase and specific requirements. There are various O-RUs available with specific implementations for indoor and outdoor use, various price points, and various hardware capabilities. A list of tested O-RUs can be found *here*, along with details on using them with the srsRAN Project gNB.

For all set-ups the RU should be connected to the fronthaul switch via SFP+ fiber cable through the main fronthaul interface.

25.2.3 5G Core

For this example we use the Open5GS 5G Core.

Open5GS is a C-language open-source implementation for 5G Core and EPC. The following links will provide you with the information needed to download and setup Open5GS so that it is ready to use with srsRAN:

- GitHub
- Quickstart Guide

25.2.4 Synchronization

The split 7.2 interface requires tight timing synchronization between the DU and RU. O-RAN WG 4 has defined various synchronization methods for use with Open Fronthaul. These are outlined in O-RAN.WG4.CUS.0-R003-v11.00 Section 11.

In this setup we use LLS-C3. The LLS-C3 configuration enables the distribution of network timing between central sites and remote sites from PRTC/T-GM to RU. In simpler terms, it allows the synchronization of one or more PRTC/T-GM devices (serving as PTP master) in the fronthaul network to transmit network timing signals to DU and RU components as seen in the figure above. In our setup the fronthaul switch is acting as the PTP grandmaster (which is synchronized via GPS), providing timing to the RU and the DU. These are connected to the SFP+ 10G ports on the switch via Ethernet.

Note: The OFH library supports all of the defined clock model and synchronization topologies defined by O-RAN WG4. The use of LLS-C3 is specific to this example.

Switch

The chosen switch should be a timing-aware O-RAN switch & PTP grandmaster. This is used to provide both clocking and timing synchronization to both the DU and RU.

25.3 Configuration

25.3.1 Switch

Refer to the specific Switch documentation to correctly configure it. Specifically any timing and routing options that may need to be configured.

We recommend using the manufacturers documentation as well as the specific switch guide from SRS if it is available in the *Integration Guide*.

25.3.2 CU/DU

NIC configuration

The DU machine should have jumbo frames enabled in the NIC and the PTP process should be checked to make sure it is synchronized correctly.

To set the jumbo frames in the NIC use the following command for a temporary configuration:

```
ifconfig <eth0> mtu 9600 up
```

Where `eth0` is the ethernet port for the SFP+ fiber cable that connects the DU to the fronthaul switch.

PTP configuration

To start the PTP process in the DU, use the command below. The configuration file can be downloaded here

```
./ptp4l -2 -i enp1s0f0 -f ./configs/default.cfg -m
```

You should then see the following output:

```
ptp4l[4321.966]: rms    6 max   14 freq -25784 +/-   9 delay   172 +/-   1
ptp4l[4323.091]: rms    5 max   10 freq -25778 +/-   8 delay   170 +/-   1
ptp4l[4324.216]: rms    6 max   11 freq -25781 +/-   9 delay   169 +/-   1
ptp4l[4325.341]: rms    5 max   10 freq -25783 +/-   8 delay   170 +/-   1
```

In the above output, the rms value can be used to determine if the PTP sync is correct, for this we look for a value < 10.

Next, run:

```
./phc2sys -s enp1s0f0 -w -m -R 8 -f ./configs/default.cfg
```

You should then see the following output:

```
phc2sys[4348.303]: CLOCK_REALTIME phc offset       -25 s2 freq   +8026 delay ␣
↪ 1467
phc2sys[4348.428]: CLOCK_REALTIME phc offset       -11 s2 freq   +8033 delay ␣
↪ 1466
phc2sys[4348.553]: CLOCK_REALTIME phc offset       -25 s2 freq   +8016 delay ␣
↪ 1396
phc2sys[4348.678]: CLOCK_REALTIME phc offset        -5 s2 freq   +8028 delay ␣
↪ 1397
```

The first value here is used to determine if the PTP sync is correct, for this we look for a value < 100.

In both of the above commands enp1s0f0 is the network interface on our DU that gets the PTP sync.

srsRAN configuration

Sample configuration files for the CU/DU can be downloaded from here. There is an associated configuration file for each of the tested RUs.

The main configuration steps for the CU/DU occur in the ru_ofh field. Here the CU/DU is configured to match the capabilities of the RU being used. All parameters should be configured specifically for each RU.

See the specific RU guides in the *Integration Guide* for more information on configuring the CU/DU.

25.3.3 RU

Refer to the specific RU documentation to correctly configure the RU. Ensure the RU is running before trying to make any configuration changes.

We recommend using the manufacturers documentation as well as the specific RU guide from SRS if it is available in the *Integration Guide*.

25.3.4 Core

For this setup Open5GS is used as the core, it is running in a docker.

The Open5GS 5G Core Quickstart Guide provides a comprehensive overview of how to configure Open5GS to run as a 5G Core.

To configure the core correctly the following steps need to be taken:

- Configure the core to connect to the gNB, ensuring the correct AMF address for both.
- Configure the PLMN and TAC values so that they are the same as those present in the gNB configuration.
- Register the ISIM credentials of the UE to the list of subscribers through the Open5GS WebUI.

25.4 Initializing the Network

25.4.1 RU

To bring up the RU simply boot it and ensure it is running correctly before attempting to connect the DU. Check for RU synchronization and that the PTP process is running correctly.

These exact steps will vary depend on the RU being used.

25.4.2 CU/DU

Before running the CU/DU, make sure you have used the commands outlined in the configuration section above to confirm the PTP sync between the DU and the fronthaul switch.

We can now run the CU/DU. First, navigate to *srsRAN_Project/build/apps/gnb*, and then run the gNB with the following command:

```
sudo ./gnb -c <RU_CONFIG>
```

Where <RU_CONFIG> is the configuration file associated with the RU being used.

If the DU connects to the RU successfully, you will see the following output or similar:

```
---== srsRAN gNB (commit ) ==---

Connecting to AMF on 10.53.1.2:38412
Initializing Open Fronthaul Interface with ul_comp=[BFP,9], dl_comp=[BFP,9],␣
```

(continues on next page)

(continued from previous page)

```
↪prach_cp_enabled=false, downlink_broadcast=true.
Operating a 20MHz cell over a RU with instantaneous bandwidth of 100MHz.
Warning: Configured PRACH occasion collides with PUCCH RBs ([0..1) intersects␣
↪[0..3)). Some interference between PUCCH and PRACH is expected.
Warning: Configured PRACH occasion collides with PUCCH RBs ([0..1) intersects␣
↪[0..3)). Some interference between PUCCH and PRACH is expected.
Cell pci=1, bw=20 MHz, dl_arfcn=634548 (n78), dl_freq=3518.22 MHz, dl_ssb_
↪arfcn=634464, ul_freq=3518.22 MHz

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

25.5 Connecting to the Network

The following sections will outline two different approaches for connecting to the network. The first will show how to connect to the network using a COTS UE, the second will show how to connect using the AmariUE UE emulator from Amarisoft.

25.5.1 COTS UE

For full details on configuring and connecting a COTS UE to the srsRAN Project gNB see *this tutorial*.

For this setup a OnePlus 9 5G UE was used to connect to the network. The set-up and configuration of the device is the same as in the above tutorial.

25.5.2 AmariUE

Additionally, third party UEs, such as AmariUE can be used to connect to the network.

For full details on configuring and connecting AmariUE to the srsRAN Project gNB see *this tutorial*.

Sending Traffic

Once connected to the network you can use iPerf to generate traffic. The following console trace was taken from the gNB during bi-directional testing:

```
          -------------DL----------------|------------------UL----------------
↪----
pci rnti  cqi  mcs  brate   ok  nok  (%) | pusch  mcs  brate   ok  nok  (%)  ␣
↪ bsr
  1 4601   15   28    38M 1200    0   0% |  17.8   26    15M  493  107  17%  ␣
↪300k
  1 4601   15   28    38M 1186   14   1% |  17.7   26    14M  488  112  18%  ␣
↪300k
  1 4601   15   28    38M 1196    4   0% |  17.8   26    15M  506   94  15%  ␣
```

(continues on next page)

(continued from previous page)

```
↪300k
  1 4601   15   28    38M 1200    0   0% |  17.8   26    15M  501   99  16%  ␣
↪300k
  1 4601   15   28    38M 1200    0   0% |  17.9   26    15M  498  102  17%  ␣
↪300k
  1 4601   15   28    38M 1200    0   0% |  17.9   26    15M  497  103  17%  ␣
↪300k
  1 4601   15   28    38M 1198    2   0% |  17.8   26    15M  497  103  17%  ␣
↪300k
  1 4601   15   28    38M 1194    6   0% |  17.8   26    15M  495  105  17%  ␣
↪300k
  1 4601   15   28    38M 1195    5   0% |  17.8   26    15M  510   89  14%  ␣
↪300k
  1 4601   15   28    38M 1200    0   0% |  17.8   26    15M  503   98  16%  ␣
↪300k
  1 4601   15   28    38M 1200    0   0% |  17.8   26    15M  495  105  17%  ␣
↪300k
```

25.6 Integration Guide

The *Integration Guide* aims to show the specific steps needed to successfully integrate COTS O-RUs and switches with the srsRAN CU/DU for use in an O-RAN split 7.2 compliant network. All of the hardware items listed here have been tested in-house.

These documents aim to show the specific steps needed to configure RUs and Switches for use with the srsRAN CU/DU in an O-RAN Split 7.2 compliant network. For information on the overall network configuration and architecture, see the *O-RAN 7.2 Guide*.

The following guides contain sample configuration and the necessary supplementary files where relevant. These documents are intended to be used as guides, users should always refer back to the manufacturer's documentation for specific information regarding stable firmware versions, drivers, configuration files and other hardware specific information. Questions relating to hardware specific issues should be directed to the manufacturer of that hardware.

Tip: For the most up-to-date configuration changes, troubleshooting and information on potential issues check the GitHub Discussions.

25.6.1 RUs

Benetel R550

Warning: This document is intended to be used as a guide. Variances in firmware and software versions in local setups may require the sample configuration files provided to be changed. As a result please closely follow the specific users guides of your RU in conjunction with this guide.

Overview

This guide provides further details on connecting the srsRAN CU/DU to an RU using the OFH Lib. Specifically the RAN550 RU from Benetel. This is a Split 7.2x indoor RU.

Note: Please refer to the Benetel User Guide Documentation for up-to-date configuration and usage guidelines, along with disclaimer and warranty information. Contact Benetel (sales@benetel.com) for more information.

Configuration

Tip: For the most up-to-date configuration changes, troubleshooting and information on potential issues check the GitHub Discussions

CU/DU

A sample configuration file for the DU can be downloaded from here. This configuration is **specific** to the firmware version of the RU being used for this guide. As a result, you may need to modify this slightly for your local setup.

The following excerpt shows how the DU is configured to communicate with the RU:

```
ru_ofh:
  ru_bandwidth_MHz: 100                 # RU instantaneous bandwidth.
  t1a_max_cp_dl: 500                    # Maximum T1a on Control-Plane for␣
↪Downlink in microseconds.
  t1a_min_cp_dl: 250                    # Minimum T1a on Control-Plane for␣
↪Downlink in microseconds.
  t1a_max_cp_ul: 465                    # Maximum T1a on Control-Plane for␣
↪Uplink in microseconds.
  t1a_min_cp_ul: 250                    # Minimum T1a on Control-Plane for␣
↪Uplink in microseconds.
  t1a_max_up: 250                       # Maximum T1a on User-Plane in␣
↪microseconds.
```

(continues on next page)

(continued from previous page)

```
  t1a_min_up: 80                        # Minimum T1a on User-Plane in␣
↪microseconds.
  is_prach_cp_enabled: false            # Configures if Control-Plane messages␣
↪should be used to receive PRACH messages.
  is_dl_broadcast_enabled: true         # Set to true for a workaround over a␣
↪firmware bug in the RAN550 when operating in SISO mode.
  compr_method_ul: bfp                  # Uplink compression method.
  compr_bitwidth_ul: 9                  # Uplink IQ samples bitwidth after␣
↪compression.
  compr_method_dl: bfp                  # Downlink compression method.
  compr_bitwidth_dl: 9                  # Downlink IQ samples bitwidth after␣
↪compression.
  compr_method_prach: bfp               # PRACH compression method.
  compr_bitwidth_prach: 9               # PRACH IQ samples bitwidth after␣
↪compression.
  enable_ul_static_compr_hdr: true      # Configures if the compression header␣
↪is present for uplink User-Plane messages (false) or not present (true).
  enable_dl_static_compr_hdr: true      # Configures if the compression header␣
↪is present for downlink User-Plane messages (false) or not present (true).
  iq_scaling: 1.5                       # IQ samples scaling factor applied␣
↪before compression, should be a positive value smaller than 10.
  cells:
    - network_interface: enp1s0f0       # Ethernet interface name used to␣
↪communicate with the RU.
      ru_mac_addr: 70:b3:d5:e1:5b:06    # RU MAC address.
      du_mac_addr: 80:61:5f:0d:df:aa    # DU MAC address.
      vlan_tag: 5                       # VLAN tag value.
      prach_port_id: [4]                # PRACH eAxC port value.
      dl_port_id: [0]                  # Downlink eAxC port values.
      ul_port_id: [0]                  # Uplink eAxC port values.
```

To expand on this, the following parameters are set in the `cells` field:

- `network_interface` : Network interface used to send the OFH packets.
- `ru_mac_addr` : MAC address of the RAN550.
- `du_mac_addr` : MAC address of the interface used by the gNB (it should be connected directly to the RU or using a smart switch).
- `vlan_tag` : V-LAN identifier, should be set to the value configured in the switch settings

RU

Refer to the Benetel User Guide documentation to apply the following configuration changes so that they match your local setup. The following information is purely a guide and may not work for your specific set up.

Ensure the RU is running before trying to make any configuration changes.

`MAC Address` : The MAC address of the DU must be configured in the RU for Control-Plane and User-Plane traffic. In our configuration we use the same MAC address for both planes.

`VLAN tag` : In our setup the same VLAN ID is used for all network traffic, as only one MAC address is used.

`Compression` : Currently only static compression headers are supported for this setup. We use BFP9 compression for all uplink and downlink channels. Refer to the Benetel User Guide for details on how to configure compression in the RU.

`Transmission Power` : Depending on your setup, you may need to alter the transmission power of the RU. For example, in a lab setting with the UE in close proximity to the RU, the default power settings may result in UE saturation.

`PRACH format` : We recommend using long PRACH format. Use *long_form_prach.sh* script on the RU or apply the required changes manually.

`DL scaling` : We use downlink scaling of 6dB.

`TDD pattern` : The TDD pattern should be changed to 6-3 format (DDDDDDSUUU). Download it `here`.

The full init script we used for this set up can be found `here`.

UE

You can download the specific amariUE configuration used for this set up `here`.

Initializing and connecting to the network

Initializing and connecting to the network is done in the same way as outlined in the general 7.2 RU guide.

Initializing the network

The following steps should be taken to initialize the network:

1. Ensure the R550 is online and that both the PTP process and RU synchronization are running correctly.

2. Run the CU/DU, making sure that the PTP sync between the DU and the Falcon switch is successful as previously outlined.

```
sudo ./gnb -c du_R550_rf.yml
```

If the DU connects to the RU successfully, you will see the following output:

```
---== srsRAN gNB (commit ) ==---

Connecting to AMF on 10.53.1.2:38412
Initializing Open Fronthaul Interface with ul_comp=[BFP,9], dl_comp=[BFP,9],␣
↪prach_cp_enabled=false, downlink_broadcast=true.
Operating a 20MHz cell over a RU with instantaneous bandwidth of 100MHz.
Warning: Configured PRACH occasion collides with PUCCH RBs ([0..1) intersects␣
↪[0..3)). Some interference between PUCCH and PRACH is expected.
Warning: Configured PRACH occasion collides with PUCCH RBs ([0..1) intersects␣
↪[0..3)). Some interference between PUCCH and PRACH is expected.
Cell pci=1, bw=20 MHz, dl_arfcn=634548 (n78), dl_freq=3518.22 MHz, dl_ssb_
↪arfcn=634464, ul_freq=3518.22 MHz

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

Connecting to the network

You can now connect a UE to the network. This can either be done using a third party UE, such as amariUE, or a COTS UE. See the main RU guide for details on this.

The specific amariUE configuration file for this setup was provided above.

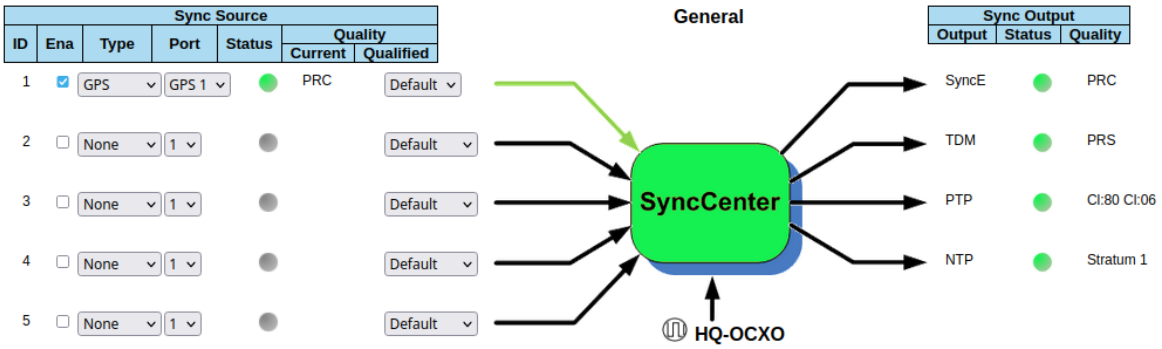
25.6.2 Switches

Falcon-RX Switch

The Falcon-RX/812/G Switch is a 5G xHaul timing-aware O-RAN switch & PTP grandmaster. This is used to provide both clocking and timing synchronization to both the DU and RU.

SyncCenter

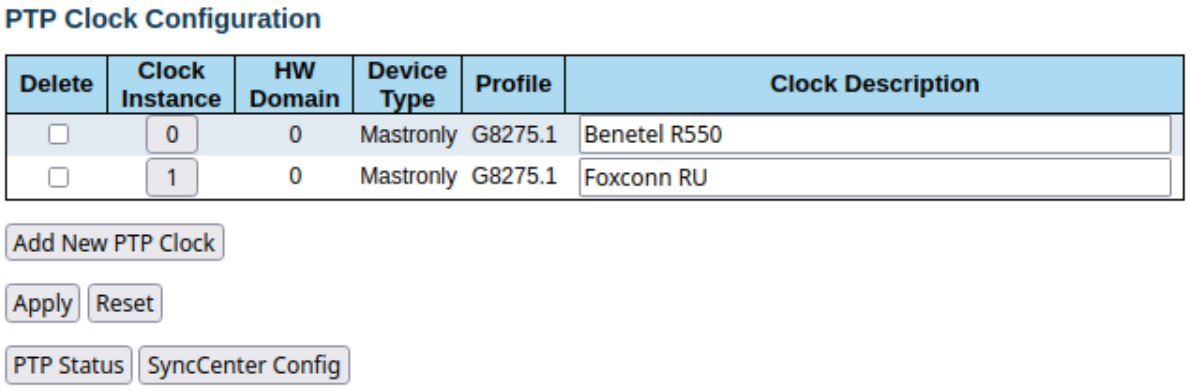
The switch must be connected to an external clock source to ensure the PTP grandmaster is synchronized correctly. Once connected it is important to check that the GPS has been locked correctly and an accurate clock source is being provided. In this example a GPS reference is used.



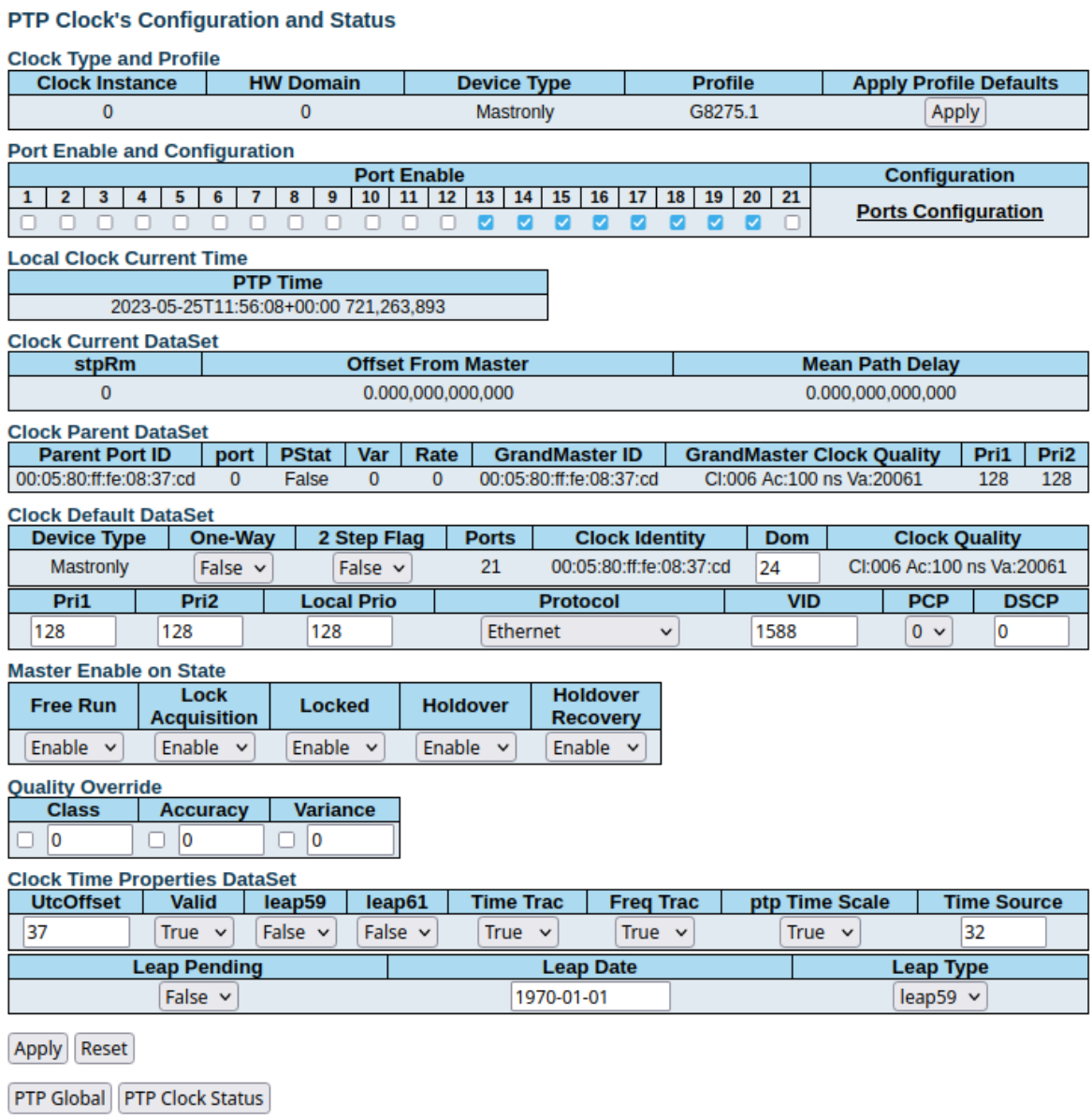
To do this, navigate to the FalconRX configuration GUI and go to *Configuration* > *Timing* > *SyncCenter* and select GPS as the `Sync Source Type`. Once this is done, wait for the GPS to lock and synchronize correctly. The SyncCenter will display green once it has successfully locked to the GPS signal. This is shown in the above image.

PTP Clocks

Once the PTP grandmaster is successfully synchronized it must be configured correctly for use with the DU and RU.



First, go to *Configuration* > *Timing* > *PTP* and add a new PTP Clock. Select `Device Type: Master Only` and `Profile: G8275.1`. This is shown in the above image. After adding the `PTP clock`, click on the Clock Instance that you want to edit.



PTP Clock's Configuration and Status

Clock Type and Profile

Clock Instance	HW Domain	Device Type	Profile	Apply Profile Defaults
0	0	Mastronly	G8275.1	Apply

Port Enable and Configuration

Port Enable																			Configuration
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	Ports Configuration
☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☑	☑	☑	☑	☑	☑	☑	☑	☐	

Local Clock Current Time

PTP Time
2023-05-25T11:56:08+00:00 721,263,893

Clock Current DataSet

stpRm	Offset From Master	Mean Path Delay
0	0.000,000,000,000	0.000,000,000,000

Clock Parent DataSet

Parent Port ID	port	PStat	Var	Rate	GrandMaster ID	GrandMaster Clock Quality	Pri1	Pri2
00:05:80:ff:fe:08:37:cd	0	False	0	0	00:05:80:ff:fe:08:37:cd	Cl:006 Ac:100 ns Va:20061	128	128

Clock Default DataSet

Device Type	One-Way	2 Step Flag	Ports	Clock Identity	Dom	Clock Quality
Mastronly	False	False	21	00:05:80:ff:fe:08:37:cd	24	Cl:006 Ac:100 ns Va:20061

Pri1	Pri2	Local Prio	Protocol	VID	PCP	DSCP
128	128	128	Ethernet	1588	0	0

Master Enable on State

Free Run	Lock Acquisition	Locked	Holdover	Holdover Recovery
Enable	Enable	Enable	Enable	Enable

Quality Override

Class	Accuracy	Variance
☐ 0	☐ 0	☐ 0

Clock Time Properties DataSet

UtcOffset	Valid	leap59	leap61	Time Trac	Freq Trac	ptp Time Scale	Time Source
37	True	False	False	True	True	True	32

Leap Pending	Leap Date	Leap Type
False	1970-01-01	leap59

Apply Reset

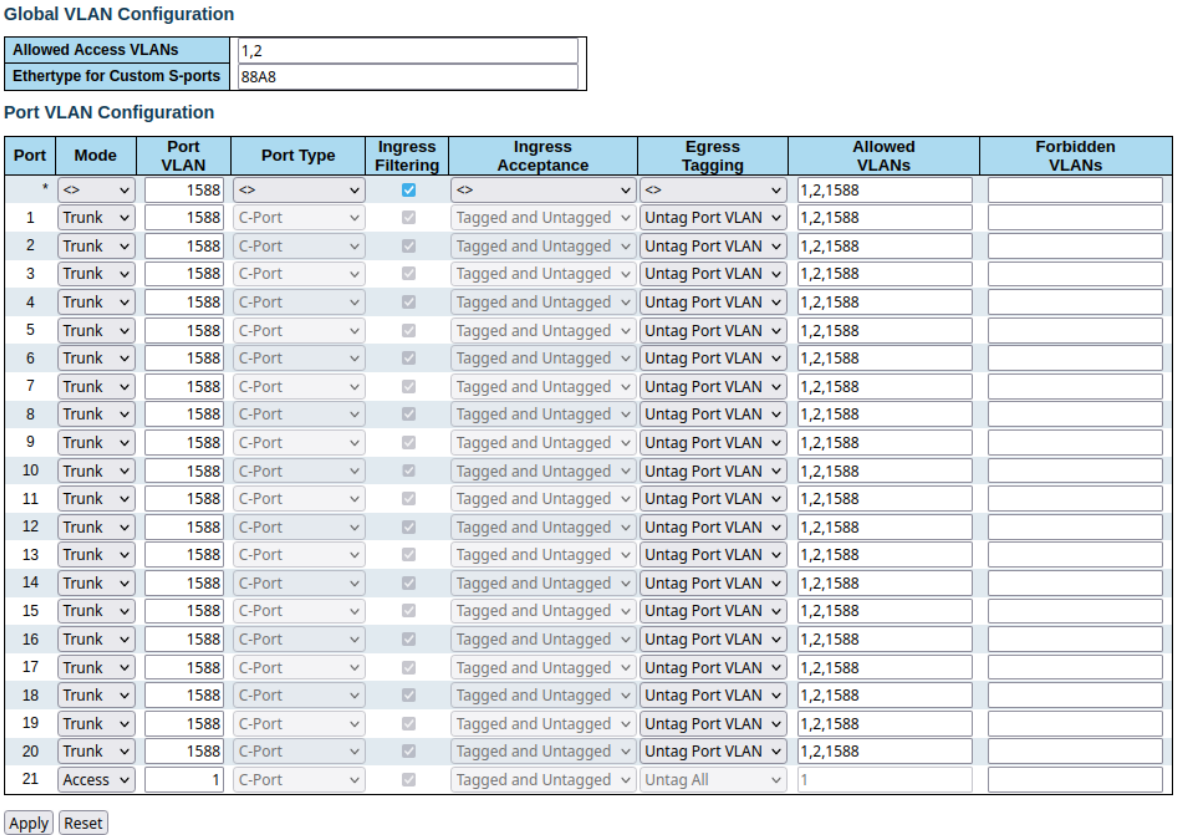
PTP Global PTP Clock Status

Once you have selected the `Clock Instance` you want to edit, set the `VLAN ID` to `1588` and activate all ports that you want to serve with PTP. From now on the PTP is sent with VLAN ID 1588.

You should now save your configuration.

VLAN

Next, the VLANs must be configured correctly so as to allow the DU and RU to receive the PTP sync from the grandmaster.



Global VLAN Configuration

Allowed Access VLANs	1,2
Ethertype for Custom S-ports	88A8

Port VLAN Configuration

Port	Mode	Port VLAN	Port Type	Ingress Filtering	Ingress Acceptance	Egress Tagging	Allowed VLANs	Forbidden VLANs
*	<>	1588	<>	☑	<>	<>	1,2,1588	
1	Trunk	1588	C-Port	☑	Tagged and Untagged	Untag Port VLAN	1,2,1588	
2	Trunk	1588	C-Port	☑	Tagged and Untagged	Untag Port VLAN	1,2,1588	
3	Trunk	1588	C-Port	☑	Tagged and Untagged	Untag Port VLAN	1,2,1588	
4	Trunk	1588	C-Port	☑	Tagged and Untagged	Untag Port VLAN	1,2,1588	
5	Trunk	1588	C-Port	☑	Tagged and Untagged	Untag Port VLAN	1,2,1588	
6	Trunk	1588	C-Port	☑	Tagged and Untagged	Untag Port VLAN	1,2,1588	
7	Trunk	1588	C-Port	☑	Tagged and Untagged	Untag Port VLAN	1,2,1588	
8	Trunk	1588	C-Port	☑	Tagged and Untagged	Untag Port VLAN	1,2,1588	
9	Trunk	1588	C-Port	☑	Tagged and Untagged	Untag Port VLAN	1,2,1588	
10	Trunk	1588	C-Port	☑	Tagged and Untagged	Untag Port VLAN	1,2,1588	
11	Trunk	1588	C-Port	☑	Tagged and Untagged	Untag Port VLAN	1,2,1588	
12	Trunk	1588	C-Port	☑	Tagged and Untagged	Untag Port VLAN	1,2,1588	
13	Trunk	1588	C-Port	☑	Tagged and Untagged	Untag Port VLAN	1,2,1588	
14	Trunk	1588	C-Port	☑	Tagged and Untagged	Untag Port VLAN	1,2,1588	
15	Trunk	1588	C-Port	☑	Tagged and Untagged	Untag Port VLAN	1,2,1588	
16	Trunk	1588	C-Port	☑	Tagged and Untagged	Untag Port VLAN	1,2,1588	
17	Trunk	1588	C-Port	☑	Tagged and Untagged	Untag Port VLAN	1,2,1588	
18	Trunk	1588	C-Port	☑	Tagged and Untagged	Untag Port VLAN	1,2,1588	
19	Trunk	1588	C-Port	☑	Tagged and Untagged	Untag Port VLAN	1,2,1588	
20	Trunk	1588	C-Port	☑	Tagged and Untagged	Untag Port VLAN	1,2,1588	
21	Access	1	C-Port	☑	Tagged and Untagged	Untag All	1	

Apply Reset

Go to *Configuration* > *VLANs* > *Configuration* to correctly configure the VLAN settings. First, set `Allowed Access VLANs:` as `1,2`. Next, configure the ports you want to use as `Trunk` ports, set the `Port VLAN` as `1588`, and set `Egress Tagging` as `Untag Port VLAN`. In the `Allowed VLANs` field you can set a range or specify specific VLANs. For example, here we are specifying `1,2,1588`. You **must** include `1588` otherwise the DU and RU will not correctly receive the PTP sync.

CHAPTER

TWENTYSIX

O-RAN NEARRT-RIC AND XAPP

26.1 Overview

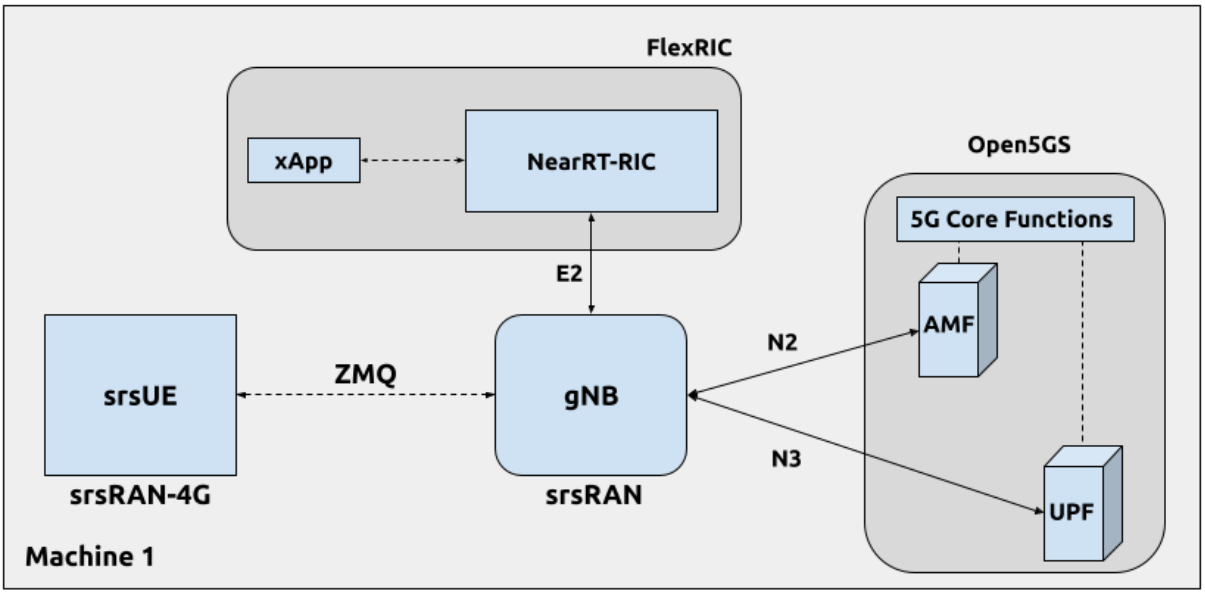
This application note shows how to use the E2 interface exposed by the srsRAN Project gNodeB. For this purpose, we use third-party O-RAN Alliance compliant NearRT-RIC and xApp provided in FlexRIC framework.

Our E2 interface implementation is based on the following O-RAN technical specifications:

- O-RAN.WG3.E2AP-R003-v03.00
- O-RAN.WG3.E2SM-R003-v03.00
- O-RAN.WG3.E2SM-KPM-R003-v03.00
- O-RAN.WG3.E2SM-RC-R003-v03.00

26.2 Setup Overview

The following diagram presents the setup architecture used in this application note:



26.3 Hardware and Software Overview

For this application note, the following hardware and software are used:

- PC with Ubuntu 22.04.1 LTS
- srsRAN Project
- srsRAN UE (srsRAN 4G 23.04 or later)
- ZeroMQ
- FlexRIC
- Open5GS 5G Core
- Wireshark (Version 4.0.7 or later)

26.3.1 Limitations

While our ultimate goal is to fully support the E2 interface, it is still under development and its current version is limited in features and operation. Specifically, the current E2 interface implementation supports only E2SM_KPM and E2SM_RC service models with the following limitations:

- E2SM_RC service model:
  - Only Control Service Style 2 is supported
- E2SM_KPM service model:
  - All Report Service Styles (1 - 5) are supported
  - Monitoring period limited to 1s
  - The following 3 'dummy' DU metrics are exposed (they will be removed in future releases):
    - CQI
    - RSRP
    - RSRQ
  - The following 6 ORAN defined metrics are expose:
    - `DRB.UEThpDl` - DL throughput
    - `DRB.UEThpUl` - UL throughput
    - `DRB.RlcPacketDropRateDl` - UL packet success rate
    - `DRB.PacketSuccessRateUlgNBUu` - RLC DL packet drop rate
    - `DRB.RlcSduTransmittedVolumeDL` - RLC DL transmitted SDU volume
    - `DRB.RlcSduTransmittedVolumeUL` - RLC UL transmitted SDU volume

26.4 Installation

26.4.1 FlexRIC

The FlexRIC framework provides O-RAN Alliance compliant E2 node Agent emulators, a NearRT-RIC and xApps written in C/C++ and Python. For the purpose of presenting the usage of E2 interface exposed by srsRAN Project gNodeB, we use the NearRT-RIC and an example KPM monitoring xApp from the FlexRIC framework.

The O-RAN specifications are evolving and FlexRIC is under development with multiple tracks (i.e. git branches). In this tutorial, we use `br-flexric` branch (commit: `1a3903a7`).

FlexRIC requires the following dependencies to be installed (see FlexRIC Installation Guide for details):

```
sudo apt-get update
sudo apt-get install swig libsctp-dev python3 cmake-curses-gui python3-dev␣
↪pkg-config libconfig-dev libconfig++-dev
```

The FlexRIC installation is performed as follows:

```
git clone https://gitlab.eurecom.fr/mosaic5g/flexric.git
cd flexric
git checkout br-flexric
mkdir build
cd build
cmake -DKPM_VERSION=KPM_V2 -DXAPP_DB=NONE_XAPP ../
make
sudo make install
```

Note that while by default Ubuntu 22.04.1 uses *gcc-11*, the used FlexRIC version can be built only with *gcc-10*. One possible way to switch *gcc* version is to use *update-alternatives* tool, for example:

```
sudo update-alternatives --config gcc

There are 3 choices for the alternative gcc (providing /usr/bin/gcc).

  Selection    Path             Priority   Status
------------------------------------------------------------
  0            /usr/bin/gcc-11   11        auto mode
* 1            /usr/bin/gcc-10   10        manual mode
  2            /usr/bin/gcc-11   11        manual mode
  3            /usr/bin/gcc-9    9         manual mode

Press <enter> to keep the current choice[*], or type selection number:
```

26.4.2 Open5GS

For this example we are using Open5GS as the 5G Core.

Open5GS is a C-language Open Source implementation for 5G Core and EPC. The following links will provide you with the information needed to download and set-up Open5GS so that it is ready to use with srsRAN:

- GitHub
- Quickstart Guide

For the purpose of this application note, we will use a dockerized Open5GS version provided in srsRAN Project at `srsgnb/docker`.

26.4.3 ZeroMQ

On Ubuntu, ZeroMQ development libraries can be installed with:

```
sudo apt-get install libzmq3-dev
```

Alternatively, ZeroMQ can also be built from source.

First, one needs to install libzmq:

```
git clone https://github.com/zeromq/libzmq.git
cd libzmq
./autogen.sh
./configure
make
sudo make install
sudo ldconfig
```

Second, install czmq:

```
git clone https://github.com/zeromq/czmq.git
cd czmq
./autogen.sh
./configure
make
sudo make install
sudo ldconfig
```

Finally, you need to compile srsRAN Project and srsRAN 4G (assuming you have already installed all the required dependencies).

Note: If you have already built and installed srsRAN 4G and srsRAN Project prior to installing ZMQ and other dependencies you will have to re-build both to ensure the ZMQ drivers have been recognized correctly.

26.4.4 srsRAN Project

For srsRAN Project, the following commands can be used to download and build from source:

```
git clone https://github.com/srsran/srsRAN_Project.git
cd srsRAN_Project
mkdir build
cd build
cmake ../ -DENABLE_EXPORT=ON -DENABLE_ZEROMQ=ON
make -j`nproc`
```

ZeroMQ is disabled by default, this is enabled when running `cmake` by including `-DENABLE_EXPORT=ON -DENABLE_ZEROMQ=ON`.

Pay extra attention to the cmake console output. Make sure you read the following line:

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

26.4.5 srsUE

If you have not already done so, install the latest version of srsRAN 4G and all of its dependencies. This is outlined in the installation guide.

Please check our srsRAN 4G ZeroMQ Application Note for information on installing ZMQ and using it with srsRAN 4G/ srsUE.

26.5 Configuration

Here, we use ZMQ-based setup, and hence the configuration files are based on those introduced in *srsRAN gNB with srsUE* application note.

The following config files were modified to use ZMQ-based RF driver and enable E2 interface in the srsRAN Project gNodeB:

- `gNB config`
- `UE config`

Details of the modifications made are outlined in the following sections. The description of the remaining config parameters is available in *srsRAN gNB with srsUE* application note.

It is recommended you use these files to avoid errors while changing configs manually. Any configuration files not included here do not require modification from the default settings.

26.5.1 gNB

Here, we describe the gNB configuration parameters related to the E2 agent.

Enable E2 agents in all DUs and enable E2SM_KPM service module:

```
e2:
  enable_du_e2: true                # Enable DU E2 agent (one for each DU␣
↪instance)
  e2sm_kpm_enabled: true            # Enable KPM service module
  addr: 127.0.0.1                   # RIC IP address
  port: 36421                       # RIC port
```

Enable E2AP packet captures and set the name of the output pcap file:

```
pcap:
  e2ap_enable: true                 # Set to true to enable E2AP PCAPs.
  e2ap_filename: /tmp/gnb_e2ap.pcap # Path where the E2AP PCAP is stored.
```

Enable Enable RLC metrics reporting that will feed E2SM_KPM service model with measurements data:

```
metrics:
  rlc_json_enable: 1                # Enable RLC metrics reporting
  rlc_report_period: 1000           # Set reporting period to 1s
```

26.6 Running the Network

The following order should be used when running the network:

1. Open5GS
2. NearRT-RIC
3. gNB
4. UE
5. Start IP traffic (e.g., ping)
6. xApp

26.6.1 Open5GS Core

srsRAN Project provides a dockerized version of the Open5GS. It is a convenient and quick way to start the core network. You can run it as follows:

```
cd ./srsRAN_Project/docker
docker compose up --build 5gc
```

Note that we have already configured Open5GS to operate correctly with srsRAN Project gNB. Moreover, the UE database is populated with the credentials used by our srsUE.

26.6.2 NearRT-RIC

Start example NearRT-RIC provided in FlexRIC framework:

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

The NearRT-RIC console output should be similar to:

```
Setting the config -c file to /usr/local/etc/flexric/ric.conf
[LibConf]: loading service models from SM_DIR: /usr/local/lib/flexric/
[LibConf]: reading configuration for NearRT_RIC
[LibConf]: NearRT_RIC IP: 127.0.0.1
[LibConf]: E2_Port Port: 36421
[LibConf]: E42_Port Port: 36422
[NEAR-RIC]: nearRT-RIC IP Address = 127.0.0.1, PORT = 36421
[NEAR-RIC]: Initializing
[NEAR-RIC]: Loading SM ID = 3 with def = ORAN-E2SM-RC
[NEAR-RIC]: Loading SM ID = 142 with def = MAC_STATS_V0
[NEAR-RIC]: Loading SM ID = 148 with def = GTP_STATS_V0
[NEAR-RIC]: Loading SM ID = 146 with def = TC_STATS_V0
[NEAR-RIC]: Loading SM ID = 145 with def = SLICE_STATS_V0
[NEAR-RIC]: Loading SM ID = 143 with def = RLC_STATS_V0
[NEAR-RIC]: Loading SM ID = 2 with def = ORAN-E2SM-KPM
[NEAR-RIC]: Loading SM ID = 144 with def = PDCP_STATS_V0
[iApp]: Initializing ...
[iApp]: nearRT-RIC IP Address = 127.0.0.1, PORT = 36422
fd created with 6
[NEAR-RIC]: Initializing Task Manager with 2 threads
```

26.6.3 gNB

We run gNB directly from the build folder (the config file is also located there) with the following command:

```
sudo ./gnb -c gnb_zmq.yaml
```

The gNB console output should be similar to:

```
---== srsRAN gNB (commit 0b2702cca) ==---

Connecting to AMF on 10.53.1.2:38412
Available radio types: zmq.
Connecting to NearRT-RIC on 127.0.0.1:36421
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
```

The `Connecting to AMF on 10.53.1.2:38412` message indicates that gNB initiated a connection to the core. While, the `Connecting to NearRT-RIC on 127.0.0.1:36421` message indicates that

gNB initiated a connection to the NearRT-RIC.

If the connection attempt is successful, the following (or similar) will be displayed on the NearRT-RIC console:

```
Received message with id = 411, port = 1715
[E2AP] Received SETUP-REQUEST from PLMN   1. 1 Node ID 411 RAN type ngran_gNB
[NEAR-RIC]: Accepting RAN function ID 2 with def = ORAN-E2SM-KPM
[iApp]: no xApp connected, no need to generate E42 UPDATE-E2-NODE
```

26.6.4 srsUE

First, the correct network namespace must be created for the UE:

```
sudo ip netns add ue1
```

Next, we start srsUE. This is also done directly from within the build folder, with the config file in the same location:

```
sudo ./srsue ue_zmq.conf
```

If srsUE connects successfully to the network, the following (or similar) should be displayed on the console:

```
Built in Release mode using commit fa56836b1 on branch master.

Opening 1 channels in RF device=zmq with args=tx_port=tcp://127.0.0.1:2001,rx_
↪port=tcp://127.0.0.1:2000,base_srate=11.52e6
Supported RF device list: UHD zmq file
CHx base_srate=11.52e6
Current sample rate is 1.92 MHz with a base rate of 11.52 MHz (x6 decimation)
CH0 rx_port=tcp://127.0.0.1:2000
CH0 tx_port=tcp://127.0.0.1:2001
Current sample rate is 11.52 MHz with a base rate of 11.52 MHz (x1 decimation)
Current sample rate is 11.52 MHz with a base rate of 11.52 MHz (x1 decimation)
Waiting PHY to initialize ... done!
Attaching UE...
Random Access Transmission: prach_occasion=0, preamble_index=0, ra-rnti=0x39,␣
↪tti=334
Random Access Complete.     c-rnti=0x4601, ta=0
RRC Connected
PDU Session Establishment successful. IP: 10.45.1.2
RRC NR reconfiguration successful.
```

It is clear that the connection has been made successfully once the UE has been assigned an IP, this is seen in `PDU Session Establishment successful. IP: 10.45.1.2`. The NR connection is then confirmed with the `RRC NR reconfiguration successful.` message.

26.6.5 IP Traffic with ping

Ping is the simplest tool to test the end-to-end connectivity in the network, i.e., it tests whether the UE and core can communicate. Here, we use it to generate traffic from UE, hence the gNB can measure data transmission-related metrics (e.g., throughput).

To run ping from UE to the core, use:

```
sudo ip netns exec ue1 ping -i 0.1 10.45.1.1
```

Note that we set the ping interval to 0.1s to increase the traffic volume.

Example **ping** output:

```
PING 10.45.1.1 (10.45.1.1) 56(84) bytes of data.
64 bytes from 10.45.1.1: icmp_seq=1 ttl=64 time=32.2 ms
64 bytes from 10.45.1.1: icmp_seq=2 ttl=64 time=35.3 ms
64 bytes from 10.45.1.1: icmp_seq=3 ttl=64 time=38.2 ms
64 bytes from 10.45.1.1: icmp_seq=4 ttl=64 time=71.5 ms
64 bytes from 10.45.1.1: icmp_seq=5 ttl=64 time=32.9 ms
```

You can also ping the from core to the UE. First add a route to the UE on the **host machine** (i.e. the one running the Open5GS docker container):

```
sudo ip ro add 10.45.0.0/16 via 10.53.1.2
```

Check the host routing table:

```
route -n
```

It should contain the following entries (note that Iface names might be different):

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.0.1     0.0.0.0         UG    100    0        0 eno1
10.45.0.0       10.53.1.2       255.255.0.0     UG    0      0        0 br-
↪dfa5521eb807
10.53.1.0       0.0.0.0         255.255.255.0   U     0      0        0 br-
↪dfa5521eb807
...
```

Next, add a default route for the UE as follows:

```
sudo ip netns exec ue1 ip route add default via 10.45.1.1 dev tun_srsue
```

Check the routing table of ue1:

```
sudo ip netns exec ue1 route -n
```

The output should be as follows:

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
```

(continues on next page)

(continued from previous page)

```
0.0.0.0         10.45.1.1       0.0.0.0         UG    0      0        0 tun_
↪srsue
10.45.1.0       0.0.0.0         255.255.255.0   U     0      0        0 tun_
↪srsue
```

Now ping the UE:

```
ping -i 0.1 10.45.1.2
```

In addition, *iperf* tool can be used to generate traffic at higher data rates than ping. For example, to send UL traffic from UE, one needs to run the following command:

```
sudo ip netns exec ue1 iperf -c 10.45.1.1 -u -b 10M -i 1 -t 60
```

26.6.6 xApp

We use an example `xapp_oran_moni` xApp from the FlexRIC framework. The application connects to NearRT-RIC and uses E2SM_KPM service module to subscribe for measurement data using Report Service Style 1. The metric names are listed in the config file that has to be passed to the xApp:

- `xapp_mon_e2sm_kpm.conf`

Specifically, with the provided config file, the xApp subscribes for two metrics, namely `DRB.UEThpDl` and `DRB.UEThpUl`.

Start the xApp with the following command:

```
./flexric/build/examples/xApp/c/monitor/xapp_oran_moni -c ./xapp_mon_e2sm_kpm.
↪conf
```

If xApp connects successfully to the NearRT-RIC, the following (or similar) should be displayed on the xApp console:

```
Setting the config -c file to /home/pgawlowicz/workspace/srsran_project_docs/.
↪/docs/source/tutorials/source/flexric/source/.config/xapp_mon_e2sm_kpm.conf
[LibConf]: loading service models from SM_DIR: /usr/local/lib/flexric/
[LibConf]: reading configuration for xApp
[LibConf]: NearRT_RIC IP: 127.0.0.1
[LibConf]: E42_Port Port: 36422
[LibConf]: Sub_ORAN_SM Name: KPM, Time: 1000
[LibConf]: format 1, RAN type ngran_gNB, actions = DRB.UEThpDl DRB.UEThpUl
[LibConf]: xApp_DB enable: 0
[LibConf]: xApp_DB user: (null), pass: (null)
[xAap]: Initializing ...
[xApp]: nearRT-RIC IP Address = 127.0.0.1, PORT = 36422
[E2-AGENT]: Opening plugin from path = /usr/local/lib/flexric/librc_sm.so
[E2-AGENT]: Opening plugin from path = /usr/local/lib/flexric/libmac_sm.so
[E2-AGENT]: Opening plugin from path = /usr/local/lib/flexric/libgtp_sm.so
[E2-AGENT]: Opening plugin from path = /usr/local/lib/flexric/libtc_sm.so
[E2-AGENT]: Opening plugin from path = /usr/local/lib/flexric/libslice_sm.so
[E2-AGENT]: Opening plugin from path = /usr/local/lib/flexric/librlc_sm.so
```

(continues on next page)

(continued from previous page)

```
[E2-AGENT]: Opening plugin from path = /usr/local/lib/flexric/libkpm_sm.so
[E2-AGENT]: Opening plugin from path = /usr/local/lib/flexric/libpdcp_sm.so
[NEAR-RIC]: Loading SM ID = 3 with def = ORAN-E2SM-RC
[NEAR-RIC]: Loading SM ID = 142 with def = MAC_STATS_V0
[NEAR-RIC]: Loading SM ID = 148 with def = GTP_STATS_V0
[NEAR-RIC]: Loading SM ID = 146 with def = TC_STATS_V0
[NEAR-RIC]: Loading SM ID = 145 with def = SLICE_STATS_V0
[NEAR-RIC]: Loading SM ID = 143 with def = RLC_STATS_V0
[NEAR-RIC]: Loading SM ID = 2 with def = ORAN-E2SM-KPM
[NEAR-RIC]: Loading SM ID = 144 with def = PDCP_STATS_V0
[xApp]: DB_ENABLE = FALSE
[xApp]: do not initial database
[xApp]: E42 SETUP-REQUEST sent
adding event fd = 5 ev-> 5
[xApp]: E42 SETUP-RESPONSE received
[xApp]: xApp ID = 7
Registered E2 Nodes = 1
Pending event size before remove = 1
Registered node 0 ran func id = 2
 [xApp]: reporting period = 1000 [ms]
```

The following (or similar) will be displayed on the NearRT-RIC console:

```
[iApp]: E42 SETUP-REQUEST received
[iApp]: E42 SETUP-RESPONSE sent
```

Next, the xApp sends the `RIC Subscription Request` message and upon successful subscription, it will periodically receive `RIC Indication messages` with the recent measurements of the requested metrics. The following (or similar) should be displayed on the xApp console:

```
Generated of req_id = 1
E42_RIC_SUBSCRIPTION_REQUEST 31
adding event fd = 5 ev-> 6
[xApp]: RIC SUBSCRIPTION REQUEST sent
[xApp]: SUBSCRIPTION RESPONSE received
Pending event size before remove = 1
[xApp]: Successfully SUBSCRIBED to ran function = 2
      1, KPM v2 ind_msg latency > 943897800 s (minimum time unit is in␣
↪second) from E2-node type 2 ID 411
meas record INTEGER_MEAS_VALUE value 28
meas record INTEGER_MEAS_VALUE value 8312
      2, KPM v2 ind_msg latency > 927120585 s (minimum time unit is in␣
↪second) from E2-node type 2 ID 411
meas record INTEGER_MEAS_VALUE value 4
meas record INTEGER_MEAS_VALUE value 11544
      3, KPM v2 ind_msg latency > 910343370 s (minimum time unit is in␣
↪second) from E2-node type 2 ID 411
meas record INTEGER_MEAS_VALUE value 4
meas record INTEGER_MEAS_VALUE value 11411
      4, KPM v2 ind_msg latency > 893566155 s (minimum time unit is in␣
```

(continues on next page)

(continued from previous page)

```
↪second) from E2-node type 2 ID 411
meas record INTEGER_MEAS_VALUE value 4
meas record INTEGER_MEAS_VALUE value 11746
...
```

Note that the metrics' names are not shown in this xApp, but their order should be the same as the order of metric listed in the *xapp_mon_e2sm_kpm.conf* config file (i.e., "DRB.UEThpDl" and "DRB.UEThpUl").

The xApp can be stopped with *CTRL+C* signal. In such case, the following (or similar) should be displayed on the xApp console:

```
^Csignal 2 received !
CTRL+C detect
Remove handle number = 1
E42 RIC_SUBSCRIPTION_DELETE_REQUEST  sdr->ric_id.ran_func_id 2  sdr->ric_id.
↪ric_req_id 1
[xApp]: E42 SUBSCRIPTION-DELETE sent
adding event fd = 5 ev-> 8
      9, KPM v2 ind_msg latency > 809680080 s (minimum time unit is in␣
↪second) from E2-node type 2 ID 411
meas record INTEGER_MEAS_VALUE value 0
meas record INTEGER_MEAS_VALUE value 0
[xApp]: E42 SUBSCRIPTION DELETE RESPONSE received
Pending event size before remove = 1
[xApp]: Successfully received SUBSCRIPTION-DELETE-RESPONSE
Closing the agent socket: Socket operation on non-socket
[xApp]: Sucessfully stopped
Test xApp run SUCCESSFULLY
```

The following (or similar) will be displayed on the NearRT-RIC console:

```
[iApp]: SUBSCRIPTION-REQUEST xapp_ric_id->ric_id.ran_func_id 2
[E2AP] SUBSCRIPTION REQUEST generated
[NEAR-RIC]: nb_id 411 port = 1715
[NEAR-RIC]: nb_id 411 port = 1715
[NEAR-RIC]: SUBSCRIPTION DELETE REQUEST tx

[iApp]: RIC_SUBSCRIPTION_DELETE_REQUEST sent
[iApp]: RIC_SUBSCRIPTION_DELETE_RESPONSE sent
```

26.7 E2AP packet analyzer

26.7.1 Enable E2AP PCAP

You can enable E2AP PCAPs by following *this guide*.

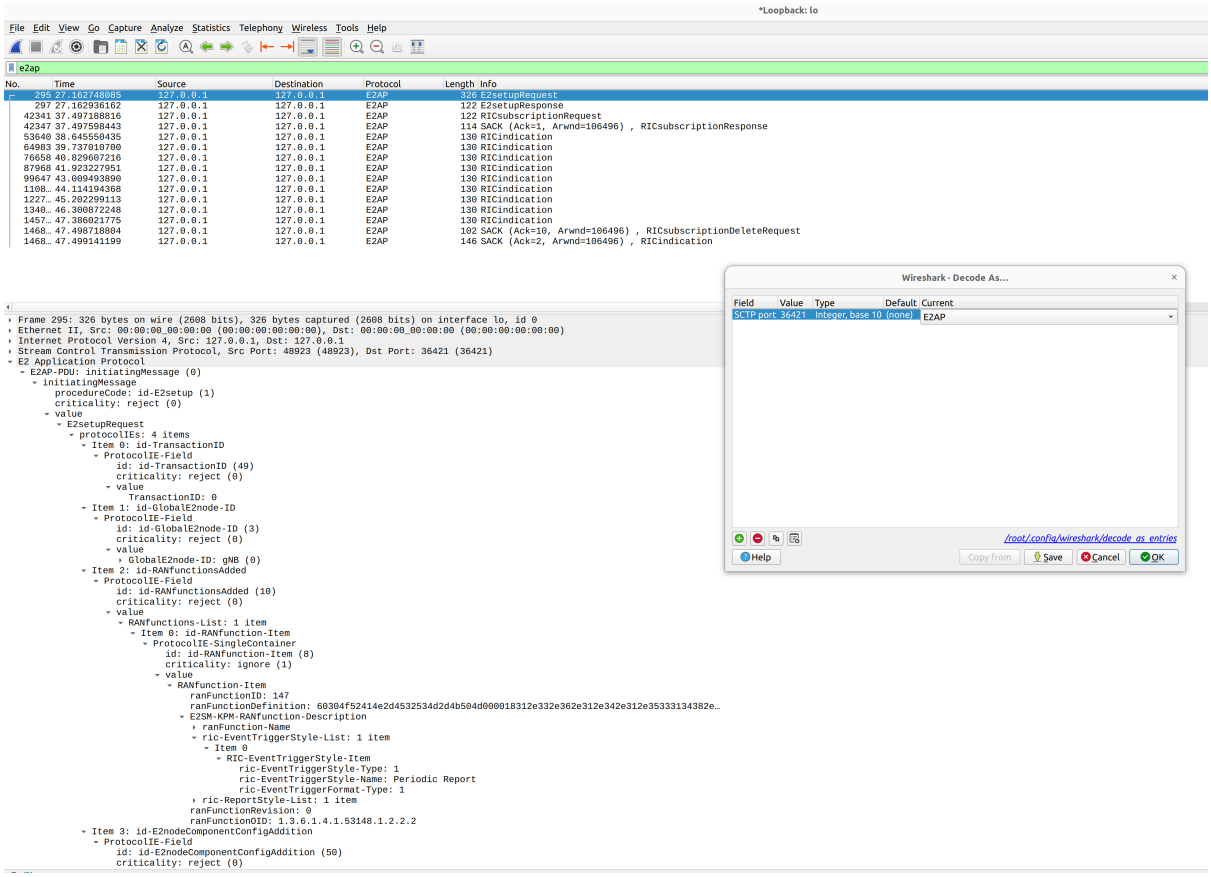
26.7.2 Live capture

Wireshark can be used to collect E2AP packets exchanged between E2 agent (located in srsRAN gNB) and NearRT-RIC at runtime. This requires the following steps to be executed:

1. Start sniffing on the loopback interface.
2. Set filter to *sctp.port == 36421*.
3. Right-click on any packet -> Decode As.. -> set Current to E2AP
4. Now filter can be set to *e2ap* to show only E2AP messages.

Note that at least Wireshark version 4.0.7 is needed to correctly decode and display E2AP packets (i.e., earlier Wireshark versions do not support E2APv3 protocol and as a result will display information about the Malformed Packets).

The figure below shows an example trace of E2AP packets.



26.8 Troubleshooting

26.8.1 PCAP

E2AP dissector is still under development in Wireshark. Therefore, some fields are not decoded correctly in Wireshark version 4.0.7. Currently, the best option is to compile Wireshark from the source code. The screenshots presented in this tutorial were obtained with Wireshark version 4.1.0 (v4.1.0rc0-3390-g4f4a54e6d3f9).

26.8.2 Core Network not running

If the dockerized version of Open5Gs fails to run it may be due to the ports set in *docker-compose.yml* are already in use on your PC. For example, you may see an error like the following:

```
ERROR: for bdfcb7644f79_open5gs_5gc  Cannot start service 5gc: driver failed␣
↪programming external connectivity on endpoint open5gs_5gc␣
↪(2919e37332feb0a3001c44985b7e3d310ae82b7adb0e2cb1d9c214ed29ff39fa): Error␣
↪starting userland proxy: listen tcp4 0.0.0.0:3000: bind: address already in␣
↪use

ERROR: for 5gc  Cannot start service 5gc: driver failed programming external␣
↪connectivity on endpoint open5gs_5gc␣
↪(2919e37332feb0a3001c44985b7e3d310ae82b7adb0e2cb1d9c214ed29ff39fa): Error␣
↪starting userland proxy: listen tcp4 0.0.0.0:3000: bind: address already in␣
↪use
ERROR: Encountered errors while bringing up the project
```

In this case, the docker-compose file can be modified so that a different host port is used as `3000` is already in use. To do this, line 40 of the *docker-compose.yml* file can be update to use `3001` as the host port:

```
  services:
    5gc:
      container_name: open5gs_5gc
      build:
        context: open5gs
        target: open5gs
        args:
          OS_VERSION: "22.04"
          OPEN5GS_VERSION: "v2.6.1"
      environment:
        MONGODB_IP: ${MONGODB_IP:-127.0.0.1}
        SUBSCRIBER_DB: ${SUBSCRIBER_DB:-001010123456780,
↪00112233445566778899aabbccddeeff,opc,63bfa50ee6523365ff14c1f45f88737d,8000,
↪9,10.45.1.2}
        OPEN5GS_IP: ${OPEN5GS_IP:-10.53.1.2}
        UE_IP_BASE: ${UE_IP_BASE:-10.45.0}
        DEBUG: ${DEBUG:-false}
      privileged: true
      ports:
```

(continues on next page)

(continued from previous page)

```
-       - "3000:3000/tcp"
+       - "3001:3000/tcp"
      # Uncomment port to use the 5gc from outside the docker network
        #- "38412:38412/sctp"
      command: 5gc -c open5gs-5gc.yml
      healthcheck:
        test: [ "CMD-SHELL", "nc -z 127.0.0.20 7777" ]
        interval: 3s
        timeout: 1s
        retries: 60
      networks:
        ran:
          ipv4_address: ${OPEN5GS_IP:-10.53.1.2}
```

26.8.3 UE issues

If the UE cannot connect to the network, ensure that the correct `cell_cfg` parameters are set in the gNB.

If the UE is connecting, but there is no PDU session being established you should check the following:

- The APN configuration is the same across both the UE and Core
- You are using the latest version of srsUE
- IP Forwarding for the core has been enabled, you can do this by following this guide.
- IP Forwarding for the UE has been enabled, see the following section

26.8.4 UE IP Forwarding

To ensure that the UE traffic is sent correctly to the internet the correct IP forwarding must be enabled. IP Forwarding should be enabled on the **host machine**, i.e. the one running the Open5GS docker container. This can be done with the following command:

```
sudo sysctl -w net.ipv4.ip_forward=1
sudo iptables -t nat -A POSTROUTING -o <IFNAME> -j MASQUERADE
```

Where <IFNAME> is the name of the interface connected to the internet.

To check that this has been configured correctly run the following command:

```
sudo ip netns exec ue1 ping -i 1 8.8.8.8
```

If the UE can ping the Google DNS, then the internet can be successfully accessed.

26.8.5 2nd Open5GS instance (installed manually)

The routing entries on the host PC for IPs: *10.45.0.0* and *10.53.1.0* should use the same interface, e.g.:

```
route -n

Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.0.1     0.0.0.0         UG    100    0        0 eno1
10.45.0.0       10.53.1.2       255.255.0.0     UG    0      0        0 br-
↪dfa5521eb807
10.53.1.0       0.0.0.0         255.255.255.0   U     0      0        0 br-
↪dfa5521eb807
...
```

However, if a second instance of Open5GS (that was installed manually) is running on the host PC, the route to *10.45.0.0* goes to *ogstun* interface. For this reason, a UE cannot access the Internet, as the host will send packets to the manually installed Open5GS version. To solve this routing issue, you can disable (or even remove) the manually installed Open5GS – please check sections 6 and/or 7 of the Open5GS tutorial. In addition, you might need to disable the *ogstun* interface with the following command:

```
sudo ifconfig ogstun 0.0.0.0 down
```

26.8.6 RIC running on a different machine

If you are running your RIC on a different machine, you will need to correctly configure the E2 `bind_addr` parameter in the gNB config file. This is shown in the example config, with the line commented out. If you are running the RIC on a separate machine simply uncomment this option.

CHAPTER

TWENTYSEVEN

MATLAB TESTING TOOLS

27.1 Overview

This tutorial explains the main features of srsRAN-matlab, a MATLAB-based project for testing srsRAN Project. More specifically, this tutorial will show how to generate a new set of test vectors for the srsRAN Project tests, how to analyze the uplink IQ samples recorded by srsRAN gNB, and how to run end-to-end, link-level simulations for testing PHY components of srsRAN Project. This will be done across three independent sections.

srsRAN-matlab offers three main tools: the test vector generators, the uplink analyzers and the link-level simulators.

27.1.1 Test Vector Generation

Test vectors are mainly used to test, develop and debug the PHY components of the srsRAN Project. This tutorial will show how to generate the set of vectors used for unit testing inside the srsRAN repository. As well as outlining how to generate a new set of random vectors for broadening the extent of the tests.

27.1.2 Signal Analyzers

The signal analyzers are useful for testing the uplink chain of the gNB. Specifically, they provide visual hints about the signal quality in the uplink slots.

27.1.3 Simulators

The simulators can be used to estimate the performance of the PHY uplink channels under different configurations and channel conditions provided by MATLAB's 3GPP-compliant models.

27.2 Set-Up Considerations

For this application note, the following hardware and software are used:

- A PC with Ubuntu 22.04.3 LTS
- srsRAN-matlab
- srsRAN Project
- MathWorks MATLAB (R2022b or R2023a) with the 5G Toolbox

Note: Running the srsRAN-matlab testing suite requires a working copy of MATLAB and its 5G Toolbox

27.3 Installation

Assuming that srsRAN Project and MATLAB have both been downloaded and installed, the next step is to download srsRAN-matlab.

This can be done with the following command:

```
git clone https://github.com/srsran/srsRAN_matlab.git
```

Note: This tutorial assumes that srsRAN Project is installed in the users come directory.

Once it has been downloaded, the working directory for srsRAN-matlab should be added to MATLAB's search path. This can be done from the MATLAB console with the following command:

```
cd ~/srsRAN_matlab
addpath .
```

To verify you have added srsRAN-matlab successfully to MATLAB's search path, run the following command (again from the MATLAB console):

```
runtests('unitTests', Tag='matlab code')
```

If successful, the following output should be shown at the end of the console output:

```
ans =

  1x94 TestResult array with properties:

    Name
    Passed
    Failed
    Incomplete
    Duration
    Details

Totals:
   94 Passed, 0 Failed, 0 Incomplete.
   42.2176 seconds testing time.
```

Test Vectors

Analyzers

Simulators

The PHY components of srsRAN Project are tested by feeding each component with vectors of input data and comparing the resulting output with their expected values. In srsRAN Project, the test vectors for a PHY component usually consist of a number of binary files with input and output data, and a single shared header file that describes the test set-up and the content of the binary files. The binary files are packed in a single tarball. For example, the test vectors of the channel estimator are provided by the files `port_channel_estimator_test_data.h` and `port_channel_estimator_test_data.tar.gz` in `~/srsRAN_Project/tests/unittests/phy/upper/signal_processors`.

The files `srs<ComponentName>Unittest.m` in the main directory of `srsRAN-matlab` provide the classes for generating such PHY input-output test vectors. This is done by leveraging MATLAB 5G Toolbox. These classes inherit from the MATLAB `matlab.unittest.TestCase` class, meaning all of the tools within MATLAB's unit testing framework can be used with them. To facilitate the generation of test vectors, a simplified interface is provided with srsRAN-matlab.

To generate the test vectors for all PHY components the following code needs to be run from the MATLAB console:

```
runSRSRANUnittest('all', 'testvector')
```

This will generate a `.h` and `.tar.gz` file for each of the PHY components and place them in the folder `~/srsRAN_matlab/testvector_outputs`..

The test vectors for a single PHY component can also be generated. This is done by replacing `all` with the name of the desired component, as per its declaration in `~/srsRAN_Project/include/srsran/`. For example, the test vectors for the channel estimator, whose interface is declared in `~/srsRAN_Project/include/srsran/phy/upper/signal_processors/port_channel_estimator.h`, can be generated with the following command:

```
runSRSRANUnittest('port_channel_estimator', 'testvector')
```

Once the test vectors have been generated, the pairs of `.h` and `tar.gz` files in the `testvector_outputs` folder can be transferred to the srsRAN Project folder with the MATLAB command:

```
srsTest.copySRStestvectors('testvector_outputs', '~/srsRAN_Project/')
```

This command will automatically copy all test vectors to the proper subdirectory inside `~/srsRAN_Project/tests/unittests/phy`.

By default, executing `runSRSRANUnittest` will reproduce the same test vectors as the ones provided with the srsRAN Project repository. To generate a random set of vectors, we simply need to add the `RandomShuffle` option. This can be done with the following command:

```
runSRSRANUnittest('all', 'testvector', RandomShuffle=true)
```

srsRAN-matlab provides some tools to analyze the signal received by the srsRAN gNB and help debugging the uplink channels. These can be found in `apps/analyzers`. In this tutorial, we will focus on the analyzer for PUSCH transmissions; for the other analyzers, which are very similar, please follow the instruction in their help text.

The following shows some of the other analyzer options:

```
% The Resource Grid analyzers only plots the energy map of a slot.
>> help srsResourceGridAnalyzer
```

(continues on next page)

(continued from previous page)

```
% For analyzing PUCCH transmissions.
>> help srsPUCCHAnalyzer

% For analyzing PRACH transmissions.
>> help srsPRACHAnalyzer
```

To use the PUSCH analyzer, the gNB needs to be configured to collect IQ samples. This can be done with by adding the following to the gNB configuration file:

```
log:
  filename: /tmp/gnb.log                          # save the log to a␣
↪specified file
  phy_level: debug                                # debug log level for PHY␣
↪layer set to debug
  phy_rx_symbols_filename: /tmp/iq.bin            # save IQ samples to a␣
↪specified file
```

The gNB can then be run as normal. The IQ samples will then be generated. This can be done with the following command:

```
sudo ./gnb -c config.yml
```

Note: The generated IQ samples will occupy a large amount of disk space. It is recommended to not run the gNB with this configuration for too long.

After running the gNB, open the `gnb.log` and locate a PUSCH transmission to analyze. The following example shows the PUSCH transmission that will be analyzed in this tutorial:

```
2023-10-08T19:14:54.738749 [Upper PHY] [I] [  690.17] RX_SYMBOL: sector=0␣
↪offset=79705 size=8568
2023-10-08T19:14:54.738854 [UL-PHY1 ] [D] [  690.17] PUSCH: rnti=0x4601 h_
↪id=0 prb=[3, 6) symb=[0, 14) mod=QPSK rv=0 tbs=11 crc=OK iter=1.0 sinr=20.
↪1dB t=182.0us
   rnti=0x4601
   h_id=0
   bwp=[0, 51)
   prb=[3, 6)
   symb=[0, 14)
   oack=0
   ocsi1=0
   part2=entries=[]
   alpha=0.0
   betas=[0.0, 0.0, 0.0]
   mod=QPSK
   tcr=0.1171875
   rv=0
   bg=2
   new_data=true
   n_id=1
```

(continues on next page)

(continued from previous page)

```
  dmrs_mask=00100001000100
  n_scr_id=1
  n_scid=false
  n_cdm_g_wd=2
  dmrs_type=1
  lbrm=3168bytes
  slot=690.17
  cp=normal
  nof_layers=1
  ports=0
  dc_position=306
  crc=OK
  iter=1.0
  max_iter=1
  min_iter=1
  nof_cb=1
  sinr_ch_est=26.9dB
  sinr_eq=23.9dB
  sinr_evm[sel]=20.1dB
  evm=0.06
  epre=+22.7dB
  rsrp=+22.7dB
  t_align=-0.2us
```

Once the transmission has been located and selected, its description can be used to populate configuration options in the srsRAN-matlab analyzer.

From the MATLAB console, run the following command:

```
cd apps/analyzers
[carrier, pusch, extra] = srsParseLogs
```

You will then see the following output:

```
Copy the relevant section of the logs to the system clipboard (typically␣
↪select and Ctrl+C), then switch back to MATLAB and press any key.
Parsing the following log section:
```

You should then copy the selected PUSCH transmission details from the log file, and paste it directly into the MATLAB console. The output should look like the following:

```
2023-10-08T19:14:54.738854 [UL-PHY1 ] [D] [  690.17] PUSCH: rnti=0x4601 h_
↪id=0 prb=[3, 6) symb=[0, 14) mod=QPSK rv=0 tbs=11 crc=OK iter=1.0 sinr=20.
↪1dB t=182.0us
  rnti=0x4601
  h_id=0
  bwp=[0, 51)
  prb=[3, 6)
  symb=[0, 14)
  oack=0
  ocsi1=0
```

(continues on next page)

(continued from previous page)

```
  part2=entries=[]
  alpha=0.0
  betas=[0.0, 0.0, 0.0]
  mod=QPSK
  tcr=0.1171875
  rv=0
  bg=2
  new_data=true
  n_id=1
  dmrs_mask=00100001000100
  n_scr_id=1
  n_scid=false
  n_cdm_g_wd=2
  dmrs_type=1
  lbrm=3168bytes
  slot=690.17
  cp=normal
  nof_layers=1
  ports=0
  dc_position=306
  crc=OK
  iter=1.0
  max_iter=1
  min_iter=1
  nof_cb=1
  sinr_ch_est=26.9dB
  sinr_eq=23.9dB
  sinr_evm[sel]=20.1dB
  evm=0.06
  epre=+22.7dB
  rsrp=+22.7dB
  t_align=-0.2us
```

The function will show the log for confirmation and ask for the sub-carrier spacing and the number of RBs in the resource grid:

```
Do you want to continue? [Y]/N y
Subcarrier spacing in kHz: 30
Grid size as a number of RBs: 51
```

Finally, `srsParseLogs` returns an nrCarrierConfig object, *carrier*, an nrPUSCHConfig object, *pusch*, and the *extra* structure with additional information about the PUSCH transport block. It should look like the following:

```
carrier =

  nrCarrierConfig with properties:

              NCellID: 1
    SubcarrierSpacing: 30
```

(continues on next page)

(continued from previous page)

```
         CyclicPrefix: 'normal'
            NSizeGrid: 51
           NStartGrid: 0
                NSlot: 17
               NFrame: 690

   Read-only properties:
       SymbolsPerSlot: 14
     SlotsPerSubframe: 2
        SlotsPerFrame: 20

pusch =

  nrPUSCHConfig with properties:

              NSizeBWP: 51
             NStartBWP: 0
            Modulation: 'QPSK'
             NumLayers: 1
           MappingType: 'A'
      SymbolAllocation: [0 14]
                PRBSet: [3 4 5]
    TransformPrecoding: 0
    TransmissionScheme: 'nonCodebook'
       NumAntennaPorts: 1
                  TPMI: 0
      FrequencyHopping: 'neither'
     SecondHopStartPRB: 1
         BetaOffsetACK: 20
        BetaOffsetCSI1: 6.2500
        BetaOffsetCSI2: 6.2500
            UCIScaling: 1
                   NID: 1
                  RNTI: 17921
                NRAPID: []
                  DMRS: [1x1 nrPUSCHDMRSConfig]
            EnablePTRS: 0
                  PTRS: [1x1 nrPUSCHPTRSConfig]

extra =

  struct with fields:

                      RV: 0
          TargetCodeRate: 0.1172
    TransportBlockLength: 88
              dcPosition: 306
```

The final step is to run the PUSCH analyzer, providing as inputs the objects just created by `srsParseLogs`, the path to the IQ record, the offset and the length of the slot (both expressed as a

number of IQ samples). Both the offset and the length of the slot can be found in the log file, on a line like the following one

```
2023-10-08T19:14:54.738749 [Upper PHY] [I] [  690.17] RX_SYMBOL: sector=0␣
↪offset=79705 size=8568
```

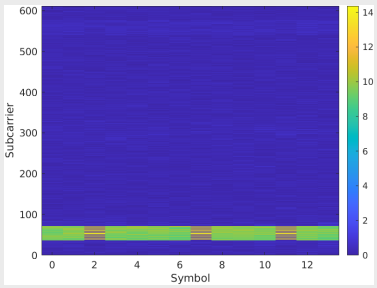
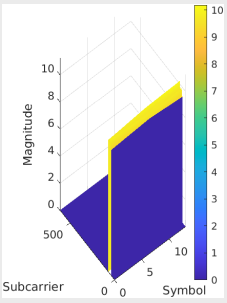
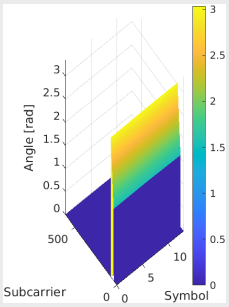
Note: The slot ID (`[ 690.17]` in our example) should be the same as that of the PUSCH log.

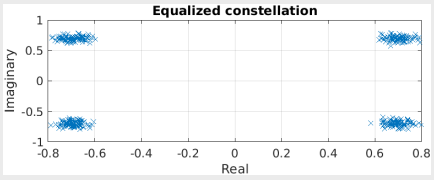
The command to run the PUSCH analyzer from the MATLAB console is:

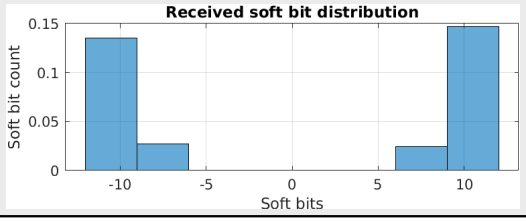
```
srsPUSCHAnalyzer(carrier, pusch, extra, '/tmp/iq.bin', 79705, 8568)
The block CRC is OK.
```

This should then output figures displaying the slot energy distribution, the magnitude of the estimated channel, the phase of the estimated channel, the equalized constellation and the received soft bit distribution.

The following figures show these:

Slot Energy Distribution	Magnitude of the Estimated Channel	Phase of the Estimated Channel
		





This example demonstrates how to test the throughput and BLER performance of the srsRAN gNB's own PUSCH processor using srsRAN-matlab simulators. By leveraging MATLAB's 5G Toolbox we can build a simulation set-up that is as close as possible to the one required by 3GPP conformance tests (see TS38.104 and TS38.141). Although not fully representative of a real-world deployment with RUs and over-the-air transmission, these simulation are useful for obtaining a first estimation of the performance of the system.

Compiling the MEXs

The inclusion of srsRAN Project PHY blocks into a MATLAB simulator is achieved by means of MEX functions, which are small C++ libraries that can be called from MATLAB. Therefore, the first step for running the srsRAN-matlab simulators is to build the MEX executables.

First, we compile srsRAN Project with the ENABLE_EXPORT flag, to export (some of) its libraries for external projects. This can be done from the command line with the following command:

```
cd ~/srsRAN_Project
cmake -B buildExport -DENABLE_EXPORT:BOOL=ON
cmake --build buildExport -j 'nproc'
```

This builds srsRAN Project inside buildExport and generates the file buildExport/srsran.cmake, which provides all the details required to import the necessary srsRAN CMake targets from external projects.

Note: The ENABLE_EXPORT flag implies the generation of position-independent code (with the -fPIC compiler option) - as a result, you may experience reduced performance when running the gNB.

The MEX libraries should now be built for srsRAN-matlab. From the command line, run the following:

```
cd ~/srsRAN_matlab/+srsMEX/source
cmake -B buildMEX -DSRSRAN_BINARY_DIR:PATH="~/srsRAN_Project/buildExport" -
↪DMatlab_ROOT_DIR:PATH="/path/to/MATLAB/R2023a"
cmake --build buildMEX -j 'nproc'
```

To check that the above was run successfully, execute the following command from the main srsRAN-matlab directory:

```
runtests('unitTests', Tag='mex code')
```

This should output the following, or similar:

```
ans =

  1x45 TestResult array with properties:

    Name
    Passed
    Failed
    Incomplete
    Duration
    Details

Totals:
   6 Passed, 0 Failed, 39 Incomplete.
   14.7124 seconds testing time.
```

You can then run:

```
runSRSRANUnittest('all', 'testmex')
```

If successful, the runSRSRANUnittest will generate test vectors, these will be fed into the MEX versions of the srsRAN Project PHY components. An output similar to the following will be shown:

```
Failure Summary:

     Name                                                                      ␣
↪                                                                              ␣
↪                                        Failed  Incomplete  Reason(s)
   ␣
↪==============================================================================
     srsPRACHDetectorUnittest[RandomDefault=true#ext,outputPath=_home_david_
↪Code_MATLAB_srsgnb_matlab_testvector_outputs#ext]/mexTest(DuplexMode=FDD,
↪PreambleFormat=1,UseZCZ=false,nAntennas=1)                X       Filtered␣
↪by assumption.
    ------------------------------------------------------------------------
↪------------------------------------------------------------------------
↪------------------------------------------------------------------------
↪--------
     srsPRACHDetectorUnittest[RandomDefault=true#ext,outputPath=_home_david_
↪Code_MATLAB_srsgnb_matlab_testvector_outputs#ext]/mexTest(DuplexMode=FDD,
↪PreambleFormat=1,UseZCZ=false,nAntennas=2)                X       Filtered␣
↪by assumption.
    ------------------------------------------------------------------------
↪------------------------------------------------------------------------
↪------------------------------------------------------------------------
↪--------

...

     srsPRACHDetectorUnittest[RandomDefault=true#ext,outputPath=_home_david_
↪Code_MATLAB_srsgnb_matlab_testvector_outputs#ext]/mexTest(DuplexMode=TDD,
↪PreambleFormat=A1,UseZCZ=true,nAntennas=2)                X       Filtered␣
↪by assumption.
    ------------------------------------------------------------------------
↪------------------------------------------------------------------------
↪------------------------------------------------------------------------
↪--------
     srsPRACHDetectorUnittest[RandomDefault=true#ext,outputPath=_home_david_
↪Code_MATLAB_srsgnb_matlab_testvector_outputs#ext]/mexTest(DuplexMode=TDD,
↪PreambleFormat=A1,UseZCZ=true,nAntennas=4)                X       Filtered␣
↪by assumption.
```

Where only `Incomplete` tests should show. If a test shows as `Failed` an error has occurred.

Running the PUSCH Simulator

In the MATLAB console, from the main srsRAN-matlab directory, a simulator object can be created as follows:

```
cd apps/simulators/PUSCHBLER
sim = PUSCHBLER
```

This should give the following output:

```
sim =

  PUSCHBLER with properties:

   Configuration
                          NCellID: 1
                             RNTI: 1
                SubcarrierSpacing: 15
                     CyclicPrefix: 'Normal'
                        NSizeGrid: 52
                           PRBSet: [0 1 2 3 4 5 6 7 8 9 10 ... ]
                 SymbolAllocation: [0 14]
                      MappingType: 'A'
            DMRSConfigurationType: 1
                       DMRSLength: 1
           DMRSAdditionalPosition: 1
                DMRSTypeAPosition: 2
                         MCSTable: 'qam64'
                         MCSIndex: 0
                          NRxAnts: 1
                          NTxAnts: 1
                        NumLayers: 1
                     DelayProfile: 'AWGN'
          PerfectChannelEstimator: true
                       EnableHARQ: false
                ImplementationType: 'matlab'
                   QuickSimulation: true
     DisplaySimulationInformation: false
                DisplayDiagnostics: false
```

The simulation set-up can now be modified as desired by the user. In particular, the `ImplementationType` should be changed to `srs`. Doing so allows the PHY components of srsRAN Project to be used (via the MEX libraries above) instead of those from the MATLAB 5G Toolbox.

This can be done with the following command:

```
sim.ImplementationType = 'srs'
```

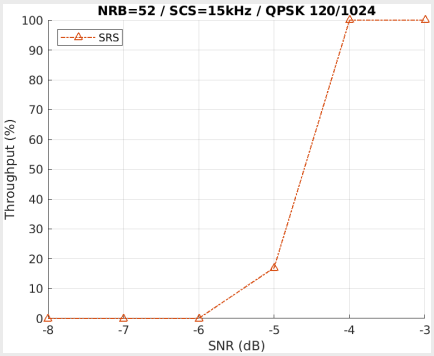
A simulation can then be run to evaluate the throughput and BLER of the PUSCH transmission. This can be done by running `sim([SNR Range], [# Frames])`. An example simulation may look like the following:

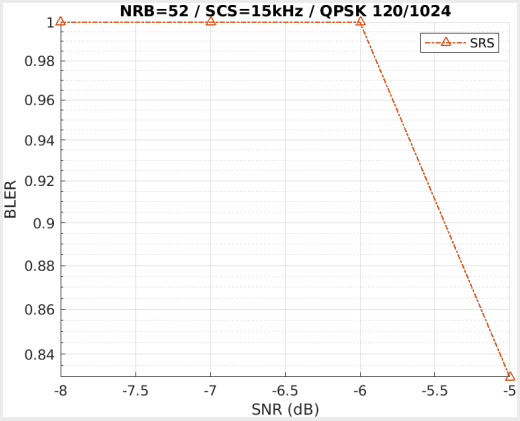
```
sim(-8:-3, 10)
```

The resulting throughput and BLER estimations can then be plot with the following command:

```
sim.plot()
```

This will give the following output:


NRB=52 / SCS=15kHz / QPSK 120/1024
SRS
Throughput (%)
SNR (dB)
100
90
80
70
60
50
40
30
20
10
0
-8
-7
-6
-5
-4
-3


NRB=52 / SCS=15kHz / QPSK 120/1024
SRS
BLER
SNR (dB)
1
0.98
0.96
0.94
0.92
0.9
0.88
0.86
0.84
-8
-7.5
-7
-6.5
-6
-5.5
-5