OCUDU Tutorials

Generated 2026-09-05 from 58fa6ca

Tutorials

Step-by-step guides from your first gNB to advanced multi-component deployments. Each tutorial has a single goal: follow it in order and you will have a working system by the end.

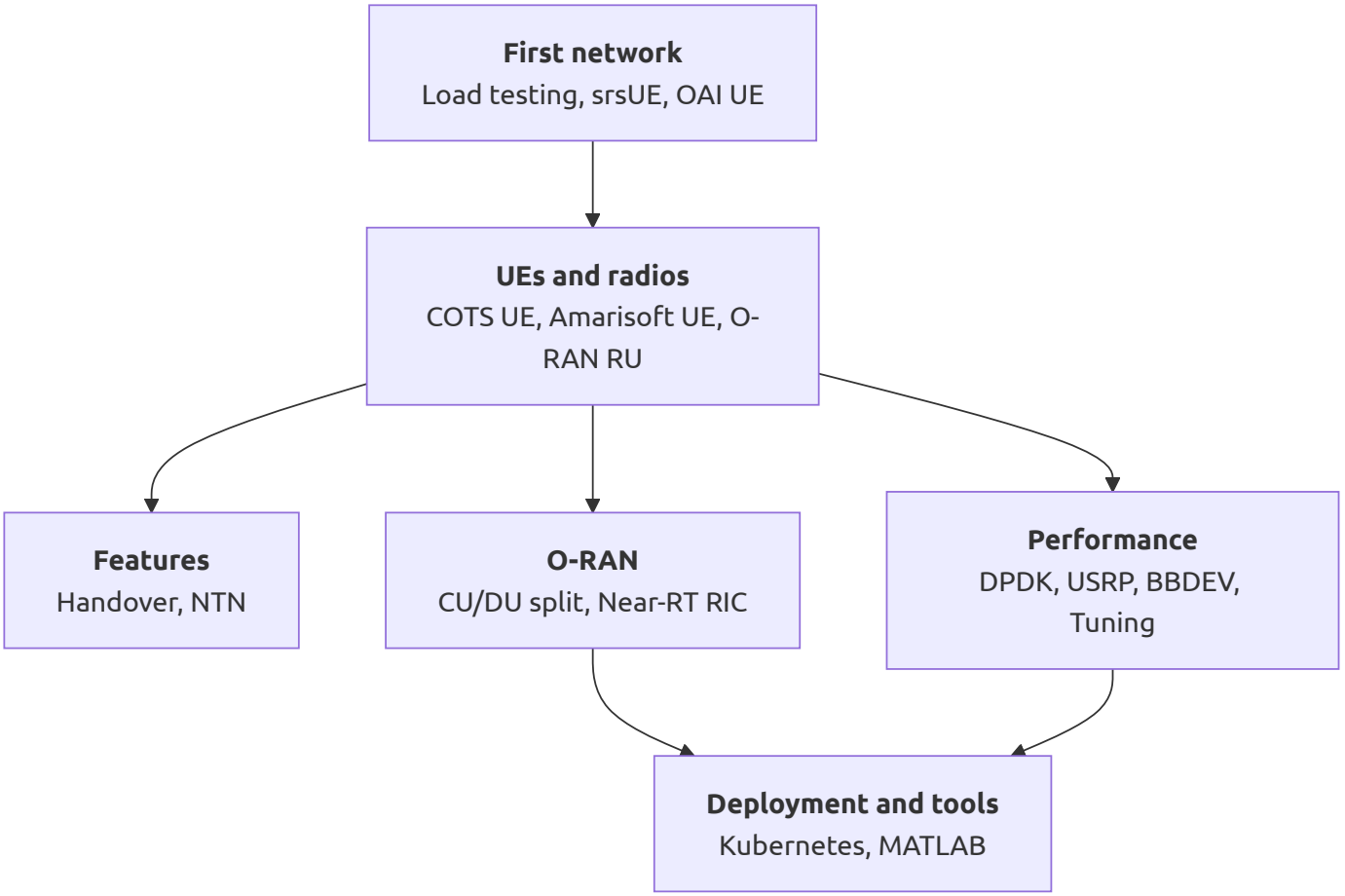
The groups below run roughly from least to most infrastructure. Pick the group that matches what you want to do.

> **NEW TO OCUDU? FOLLOW THIS PATH**
>
> 1. Load testing: validate your build, no radio hardware needed.
> 2. Using srsUE: add a software UE over ZeroMQ.
> 3. Connecting a COTS UE: move to a real device with a USRP.

> **COMING FROM SRSRAN PROJECT?**
>
> See the Migration Guide for instructions on porting your existing srsRAN Project modifications to OCUDU.

How the tutorials fit together

Start with a first network, then branch into whichever direction fits your goal.



First network

No RF hardware required. Validate your build, then bring up a complete software network over ZeroMQ.

Load testing

Verify your OCUDU installation and explore the configuration without radio hardware or a physical UE.

Using srsUE

Build a complete open-source split 8 5G network using srsUE as the UE and Open5GS as the core.

Using the OAI UE

Build an end-to-end open-source 5G TDD network using the OpenAirInterface UE over ZeroMQ, with an OCUDU gNB and Open5GS core.

UEs and radios

Connect a real device, a UE simulator, or an O-RAN radio unit. Requires a USRP RF front-end (or an O-RAN RU for the radio-unit tutorial).

Connecting a COTS UE

Connect a commercial 5G device to OCUDU using a test SIM and a USRP RF front-end.

Connecting an Amarisoft UE

Connect an Amarisoft UE simulator to OCUDU for multi-UE testing scenarios.

Connecting an O-RAN RU

Connect an O-RAN-compliant radio unit to OCUDU over the split 7.2 fronthaul interface.

O-RAN

Split the stack across O-RAN interfaces and integrate a Near-RT RIC. The CU/DU split runs on any supported RF or test setup; the RIC tutorial requires a Near-RT RIC.

Splitting CU and DU

Deploy OCUDU with the CU and DU running as separate processes, connected over the F1 interface.

Integrating a Near-RT RIC

Use the E2 interface to integrate OCUDU with a Near-RT RIC and deploy an xApp.

Features

Enable and test specific gNB features. Handover needs a COTS UE and a USRP; NTN needs an Amarisoft UE.

Testing handover

Configure and test intra-gNB handover between two OCUDU cells with a COTS UE.

Enabling NTN

Enable NTN mode in OCUDU for satellite deployments, covering GEO and LEO scenarios with SIB19 ephemeris and timing support.

Performance

Tune throughput and latency with kernel-bypass I/O, hardware offload, and host tuning. Some tutorials need specific hardware: a DPDK-capable NIC, a USRP, or an Intel accelerator.

Configuring DPDK

Configure DPDK kernel-bypass packet I/O for high-throughput Open Fronthaul connectivity with OCUDU.

Configuring DPDK with USRP

Configure DPDK kernel-bypass packet I/O for use with a USRP RF front-end and OCUDU.

Accelerating with BBDEV

Offload LDPC encoding and decoding to an Intel ACC100 or vRAN Boost (ACC200/VRB1) accelerator via DPDK BBDEV.

Tuning performance

Tune CPU isolation, IRQ affinity, and kernel settings on a Linux host for real-time OCUDU performance.

Deployment and tools

Containerised deployment and supporting tooling. The Kubernetes tutorial needs a cluster.

Running on Kubernetes

Deploy OCUDU as Kubernetes pods in a split 7.2 configuration, with containerised CU, DU, and fronthaul components.

Integrating MATLAB

Integrate MATLAB with OCUDU for signal processing, analysis, and algorithm prototyping.

Load testing OCUDU without radio hardware

Overview

In this tutorial, we will guide you through the process of load testing the OCUDU gNB(-DU) application without the need for specific hardware, via parameters available within the configuration file.

The `ru_dummy` option in the gNB configuration allows users to emulate the RU and focus exclusively on testing the DU, by excluding the OFH layers. The RU Emulator (`ru_emu`) can be used to simulate traffic sent and received by an O-RU in a split 7.2 configuration, while also allowing you to monitor key performance indicators (KPIs).

For scenarios without a core network, the `no_core` flag in the gNB configuration allows the gNB(-DU) to operate without a physical core network.

The `testmode` feature in the gNB application was developed to evaluate the performance of the OCUDU gNB(-DU) on specific hardware. This function emulates single or multiple UEs, simulating traffic across layers such as MAC, PHY, and OFH. By replicating traffic from the CU and RU to the DU, `testmode` provides an effective way to assess and optimize system performance. Integrated directly into the gNB application, `testmode` requires no additional builds beyond the gNB.

Detailed explanations of these scenarios are provided in the following sections. For detailed information on specific configuration parameters and usage of these modes, refer to the OCUDU Configuration Reference.

Using `ru_dummy`

In case you want to do a load test without a physical or emulated RU you can use the `ru_dummy` config option in the gNB config. This will exclude the OFH layer from the test because no data is sent over the network. To use the `ru_dummy` parameter add the following section to your gNB config file:

```
ru_dummy:
dl_processing_delay: 1
time_scaling: 1
```

Using `ru_emu`

The RU Emulator is an extra tool that needs to be built separately from the gNB application. To build the RU Emulator use the following commands from inside OCUDU repository:

```
mkdir build && cd build
cmake ..
cd apps/examples/ofh
make -j $(nproc)
```

In case you want to run the RU Emulator using DPDK use the following cmake command instead:

```
cmake -ENABLE_DPDK=ON ..
```

> **INFO**
>
> Building with DPDK is not a requirement for using `ru_emu`

Create a new configuration file called `emu.yaml` with the following content:

```
log:
filename: /tmp/ru_emu.log
level: warning
ru_emu:
cells:
- bandwidth: 100                          # Bandwidth of the cell
network_interface: ens2f1                # Use BDF instead of interface name for DPDK
ru_mac_addr: 00:33:22:33:00:11           # MAC address of the RU
du_mac_addr: 00:33:22:33:00:66           # MAC address of the DU
enable_promiscuous: false                # Promiscuous mode flag
vlan_tag: 33                             # VLAN tag
dl_port_id: [0, 1, 2, 3]                 # Port IDs for downlink
ul_port_id: [0, 1, 2, 3]                 # Port IDs for uplink
prach_port_id: [4, 5]                    # Port IDs for PRACH
compr_method_ul: "bfp"                   # Compression method for uplink
compr_bitwidth_ul: 9                     # Compression bitwidth for uplink
t2a_max_cp_dl: 470                       # T2a maximum value for downlink Control-Plane
t2a_min_cp_dl: 350                       # T2a minimum value for downlink Control-Plane
t2a_max_cp_ul: 200                       # T2a maximum value for uplink Control-Plane
t2a_min_cp_ul: 90                        # T2a minimum value for uplink Control-Plane
t2a_max_up: 345                          # T2a maximum value for User-Plane
t2a_min_up: 70                           # T2a minimum value for User-Plane
# dpdk:
#   eal_args: "--lcores (0-1)@(0-15)"
```

Adjust the above parameters to match your configuration. If you want to use DPDK, provide the Bus Device Function (BDF) of the NIC in the `network_interface` field. The BDF can be found using the `dpdk-devbind.py -s` command. Also, uncomment the `dpdk` section and provide the correct `eal_args` arguments.

Use the following command to start the RU Emulator:

```
sudo ./ru_emulator -c emu.yaml
```

You should see the following output:

```
Running. Waiting for incoming packets...
|   TIME   | ID  |  RX_TOTAL   | RX_ON_TIME  |  RX_EARLY   |   RX_LATE   |   RX_SEQ_ERR    |
RX_ON_TIME_C  |  RX_EARLY_C   |   RX_LATE_C   |  RX_SEQ_ERR_C   | RX_ON_TIME_C_U |  RX_EARLY_C_
RX_LATE_C_U   | RX_SEQ_ERR_C_U  | RX_SEQ_ERR_PRACH | _CORRUPT  | RX_ERR_DROP |  TX_TOTAL   |
| 15:26:48 |  0  |      0      |      0      |      0      |      0      |     0/0/0/0     |
|       0       |       0       |     0/0/0/0     |       0       |       0       |       0
0       |       0/0       |           |      0      |      0      |
| 15:26:49 |  0  |      0      |      0      |      0      |      0      |     0/0/0/0     |
|       0       |       0       |     0/0/0/0     |       0       |       0       |       0
0       |       0/0       |           |      0      |      0      |
| 15:26:50 |  0  |      0      |      0      |      0      |      0      |     0/0/0/0     |
|       0       |       0       |     0/0/0/0     |       0       |       0       |       0
0       |       0/0       |           |      0      |      0      |
| 15:26:51 |  0  |      0      |      0      |      0      |      0      |     0/0/0/0     |
|       0       |       0       |     0/0/0/0     |       0       |       0       |       0
0       |       0/0       |           |      0      |      0      |
| 15:26:52 |  0  |      0      |      0      |      0      |      0      |     0/0/0/0     |
|       0       |       0       |     0/0/0/0     |       0       |       0       |       0
0       |       0/0       |           |      0      |      0      |
| 15:26:53 |  0  |      0      |      0      |      0      |      0      |     0/0/0/0     |
|       0       |       0       |     0/0/0/0     |       0       |       0       |       0
0       |       0/0       |           |      0      |      0      |
| 15:26:54 |  0  |      0      |      0      |      0      |      0      |     0/0/0/0     |
|       0       |       0       |     0/0/0/0     |       0       |       0       |       0
0       |       0/0       |           |      0      |      0      |
| 15:26:55 |  0  |      0      |      0      |      0      |      0      |     0/0/0/0     |
|       0       |       0       |     0/0/0/0     |       0       |       0       |       0
0       |       0/0       |           |      0      |      0      |
| 15:26:56 |  0  |      0      |      0      |      0      |      0      |     0/0/0/0     |
|       0       |       0       |     0/0/0/0     |       0       |       0       |       0
0       |       0/0       |           |      0      |      0      |
| 15:26:57 |  0  |      0      |      0      |      0      |      0      |     0/0/0/0     |
|       0       |       0       |     0/0/0/0     |       0       |       0       |       0
0       |       0/0       |           |      0      |      0      |
| 15:26:58 |  0  |      0      |      0      |      0      |      0      |     0/0/0/0     |
|       0       |       0       |     0/0/0/0     |       0       |       0       |       0
0       |       0/0       |           |      0      |      0      |
| 15:26:59 |  0  |      0      |      0      |      0      |      0      |     0/0/0/0     |
|       0       |       0       |     0/0/0/0     |       0       |       0       |       0
0       |       0/0       |           |      0      |      0      |
| 15:27:00 |  0  |      0      |      0      |      0      |      0      |     0/0/0/0     |
|       0       |       0       |     0/0/0/0     |       0       |       0       |       0
0       |       0/0       |           |      0      |      0      |
| 15:27:01 |  0  |      0      |      0      |      0      |      0      |     0/0/0/0     |
|       0       |       0       |     0/0/0/0     |       0       |       0       |       0
0       |       0/0       |     0     |      0      |      0      |
> ```

The above KPIs are indicating the KPIs of the RU Emulator. The RU Emulator is now running and w
or incoming packets. Once an RU is connected you should see the `*_ON_TIME_*` counters
increase. The RU is operating properly if you do not see any late, early or err packets.

```

```
## Using `no_core`

In case you want to emulate the core network in cases where no 5G core is available, you can us
`no_core` flag in the gNB config. To use the `no_core` flag add the following section to your g
config:

```yaml
cu_cp:
amf:
no_core: false
```

Using `testmode`

Once an RU and a core network are in place you can start using `testmode`. A sample configuration file can also be found in `ocudu/configs` within OCUDU source files:

```
test_mode:
test_ue:
rnti: 0x44
ri: 1 # Set to 2 or 4 for 2 layer or 4 layer MIMO operation
cqi: 15
nof_ues: 1
pusch_active: true
pdsch_active: true
```

This config will emulate a single UE with `RNTI` = 0x44, `CQI` is set to 15 and `RI` to 1.

Configuration files can be concatenated when running the gNB(-DU), which means users can easily test various configurations without having to modify their base configuration. For this example, the configuration above will be concatenated with the example configuration `gnb_ru_ran550_tdd_n78_100mhz_4x2.yml` which can be found in `ocudu/configs`. This will allow the RU to be tested without a physical UE being connected. This ability extends to other frontends such as USRPs, and also ZMQ.

To run the described scenario, the following command can be used:

```
sudo ./apps/gnb/gnb -c gnb_ru_ran550_tdd_n78_100mhz_4x2.yml -c testmode.yml
```

You should then see the following output:

```
--== OCUDU gNB (commit f41c1db4c3) ==--
Cell pci=1, bw=100 MHz, 4T2R, dl_arfcn=637212 (n78), dl_freq=3558.18 MHz,
dl_ssb_arfcn=634464, ul_freq=3558.18 MHz
Initializing the Open Fronthaul Interface for sector#0: ul_compr=[BFP,9], dl_compr=[BFP,9],
prach_compr=[BFP,9], prach_cp_enabled=false, downlink_broadcast=false
```
```

```
==== gNB started ===
Type <h> to view help
|--------------------DL---------------------|-------------------------UL--------------------
----------
pci rnti | cqi   ri  mcs  brate   ok  nok  (%)  dl_bs | pusch  rsrp  mcs  brate   ok  nok
(%)    bsr     ta  phr
1 0x44 |  15 1.0   28   1.0G 1539    0   0%    10M |  99.9 -99.9   28    75M  400    0   0%
81.5M      0  n/a
1 0x44 |  15 1.0   28   1.0G 1546    0   0%    10M |  99.9 -99.9   28    75M  400    0   0%
81.5M      0  n/a
1 0x44 |  15 1.0   28   1.0G 1541    0   0%    10M |  99.9 -99.9   28    75M  400    0   0%
81.5M      0  n/a
1 0x44 |  15 1.0   28   1.0G 1547    0   0%    10M |  99.9 -99.9   28    75M  399    0   0%
81.5M      0  n/a
1 0x44 |  15 1.0   28   1.0G 1543    0   0%    10M |  99.9 -99.9   28    76M  401    0   0%
81.5M      0  n/a
1 0x44 |  15 1.0   28   1.0G 1542    0   0%    10M |  99.9 -99.9   28    75M  400    0   0%
81.5M      0  n/a
1 0x44 |  15 1.0   28   1.0G 1549    0   0%    10M |  99.9 -99.9   28    75M  400    0   0%
81.5M      0  n/a
1 0x44 |  15 1.0   28   1.0G 1542    0   0%    10M |  99.9 -99.9   28    75M  400    0   0%
81.5M      0  n/a
1 0x44 |  15 1.0   28   1.0G 1546    0   0%    10M |  99.9 -99.9   28    75M  399    0   0%
81.5M      0  n/a
1 0x44 |  15 1.0   28   1.0G 1546    0   0%    10M |  99.9 -99.9   28    75M  400    0   0%
81.5M      0  n/a
1 0x44 |  15 1.0   28   1.0G 1548    0   0%    10M |  99.9 -99.9   28    76M  401    0   0%
81.5M      0  n/a
```

Verifying the output

Before connecting a real UE, confirm that the testmode output looks healthy. **Do not skip this step.** A passing testmode run is the only reliable way to confirm that the gNB and RU are communicating correctly.

Check the following in the console trace:

1. **No KOs in either direction.** The `nok` column and the `(%)` column should both read `0` for both DL and UL. Any non-zero value indicates a problem with the RU-gNB link.

2. **Bitrate matches expectations.** The `brate` field should show full-rate throughput for the configured bandwidth and MIMO configuration. Compare against the example output above for your specific setup.

3. **No warnings in the logs.** Search the log file for warning-level messages:

   ```
   grep "\[W\]" <logfile>
   ```

   If warnings are present, check the Troubleshooting guide and resolve them before proceeding.

4. **RU health is good.** Check the RU's own monitoring interface for power and DPD status. Refer to your RU's documentation for details.

5. **No sustained late or early packets at the RU.** Check the RU KPI output and confirm there are no sustained late or early packet counts. Occasional single occurrences are normal; a sustained stream is not. See RU late or early packets for guidance.

6. **RF channel is clean (optional).** If a spectrum analyzer is available, verify that the signal being produced by the RU looks correct for the configured bandwidth and frequency.

Once all of the above checks pass, you can proceed to connect a UE.

For more information about the test mode please refer to the OCUDU Configuration Reference.

Next steps

- Using srsUE — add a software UE over ZeroMQ, no radio hardware required.

Building a 5G network with srsUE

> **WARNING**
>
> 5G extensions in srsUE are no longer in active development, although it does receive small maintenance updates. See here for further details. srsUE is intended to be used for proof-of-concept and initial testing to allow users to test OCUDU without the need for expensive hardware. It is not intended for scenarios which require a deployment ready solution.

Overview

OCUDU 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 tutorial shows how to create an end-to-end fully open-source 5G network with srsUE, OCUDU gNodeB and Open5GS 5G core network.

Various use-cases are shown here, including both over-the-air and ZeroMQ based setups, and multi-UE emulation.

Hardware and Software Overview

For this tutorial, the following hardware and software are used:

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

> **INFO**
>
> 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.

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)
- Handover is not supported

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 OCUDU:

- GitHub
- Quickstart Guide

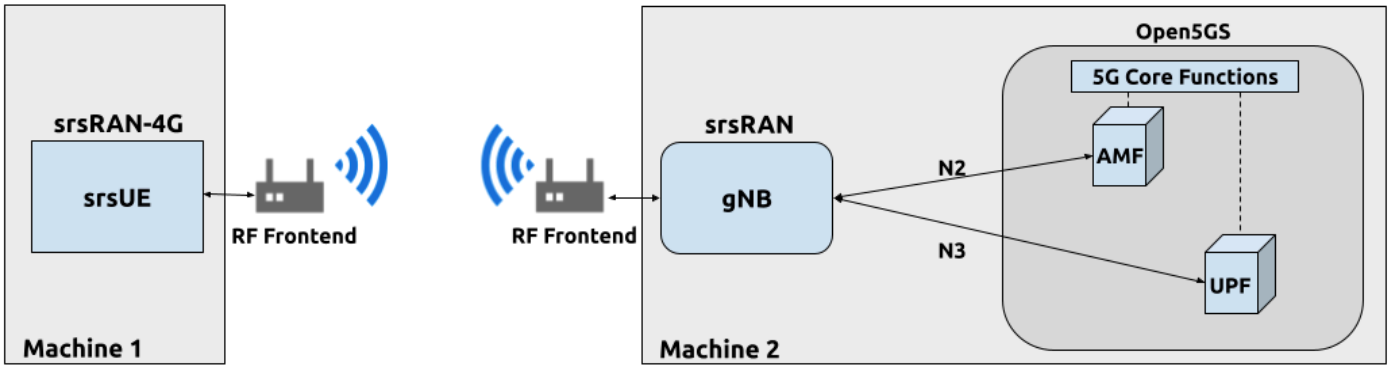
For the purpose of this tutorial, we will use a dockerized Open5GS version provided in OCUDU at `ocudu/docker`, as shown here.

ZeroMQ

For a guide on installing ZMQ and building OCUDU correctly, see here.

Over-the-air Setup

The following diagram presents the setup architecture:



Configuration

You can find a configuration file for this example in the `configs` folder of the OCUDU source files:

- 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:

```
cu_cp:
amf:
addr: 10.53.1.2                 # The address or hostname of the AMF. Check Open5GS config -
> amf -> ngap -> addr
port: 38412
bind_addr: 10.53.1.1            # A local IP that the gNB binds to for traffic from the AMF.
supported_tracking_areas:
- tac: 7
plmn_list:
- plmn: "00101"
tai_slice_support_list:
- sst: 1
inactivity_timer: 7200            # Sets the UE/PDU Session/DRB inactivity timer to 7200
seconds. Supported: [1 - 7200].
```

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                   # Use external reference clock with USRP B210.
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 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 # Use external reference clock with USRP B210.
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 = internet
apn_protocol = ipv4
```

Running the Network

The following order should be used when running the network:

1. 5GC
2. gNB

3. UE

Open5GS Core

OCUDU 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 ./ocudu/docker
docker compose up --build 5gc
```

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

gNB

We run gNB directly from the build folder, i.e., `./ocudu/build/apps/gnb/`, (the config file is also located there) with the following command:

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

The console output should be similar to:

```
--== OCUDU 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=1
```

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 (
Open5GS | 04/17 10:00:43.567: [amf] INFO: gNB-N2 accepted[10.53.1.1] in master_sm module (../sr
Open5GS | 04/17 10:00:43.567: [amf] INFO: [Added] Number of gNBs is now 1 (../src/amf/context.c
Open5GS | 04/17 10:00:43.567: [amf] INFO: gNB-N2[10.53.1.1] max_num_of_ostreams : 30 (../src/am
```

srsUE

Finally, we start srsUE. This is also done directly from within the build folder (i.e., `/srsRAN_4G/build/srsue/`), 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_al
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.

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
-----------------------------------------------------------
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%
```

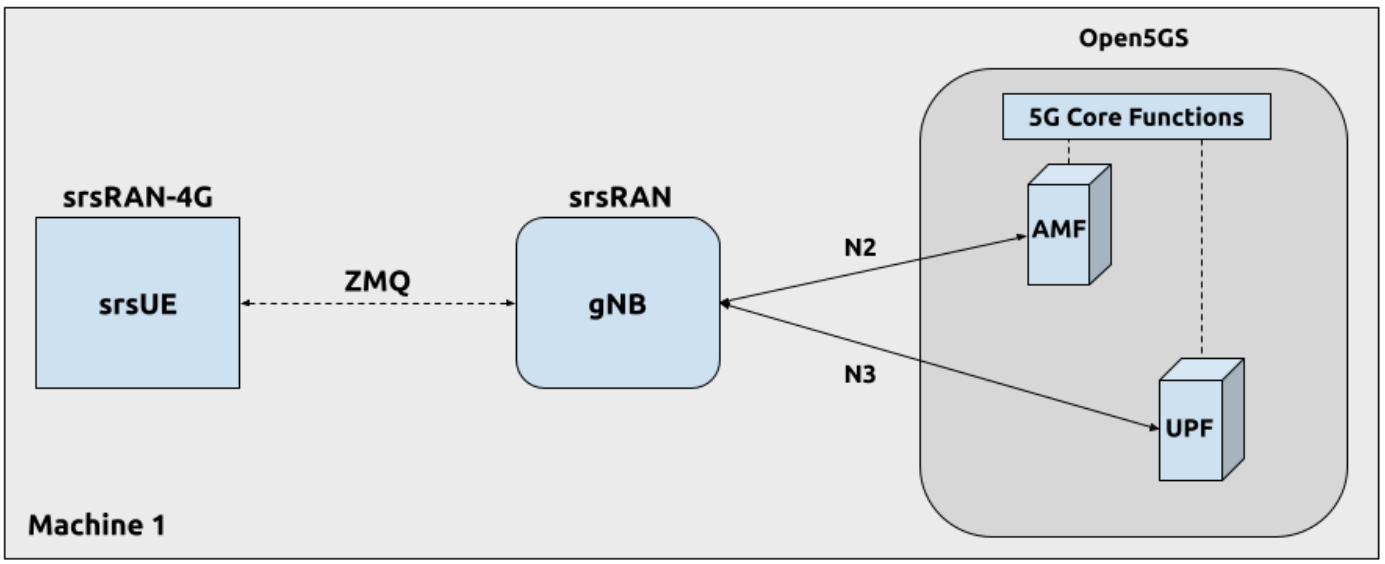
To read more about the UE console trace metrics, see the UE User Manual.

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
```

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:



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=11.52e6
srate: 11.52
tx_gain: 0
rx_gain: 0
```

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 = 11.52e6
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=11.52e6
```

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
```

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.

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 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
```

Multi-UE Emulation

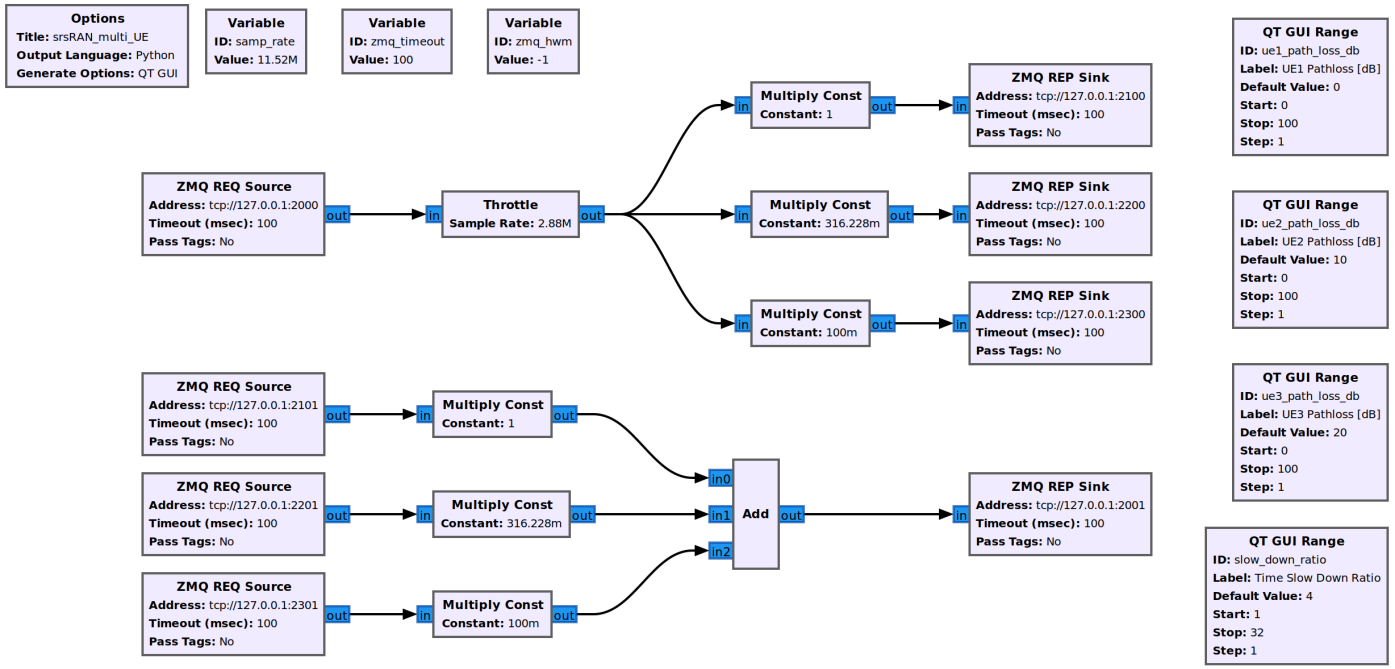
> **INFO**
>
> The Multi-UE scenario outlined here is **not** intended to be an optimized, performant, or scalable solution. It is a simple example to demonstrate how to connect multiple UEs to a single gNB using ZMQ and srsUE. Extending your own set-up beyond what is described here may not perform as expected. Users that require a more performant solution should consider using different solutions, such as AmariUE.

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:

Ports Used in the GNU-Radio flow graph

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

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: `ocudu/docker/open5gs/`

Then, edit the `ocudu/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,63bfa50ee6523365ff14c1f45f8
+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: 64 # Sets number of available PRACH preambles
+ nof_ssb_per_ro: 1 # Sets the number of SSBs per RACH occasion.
+ nof_cb_preambles_per_ssb: 64 # Sets the number of contention based preambles per SSB.
```

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

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 ./ocudu/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 ./ocudu/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.

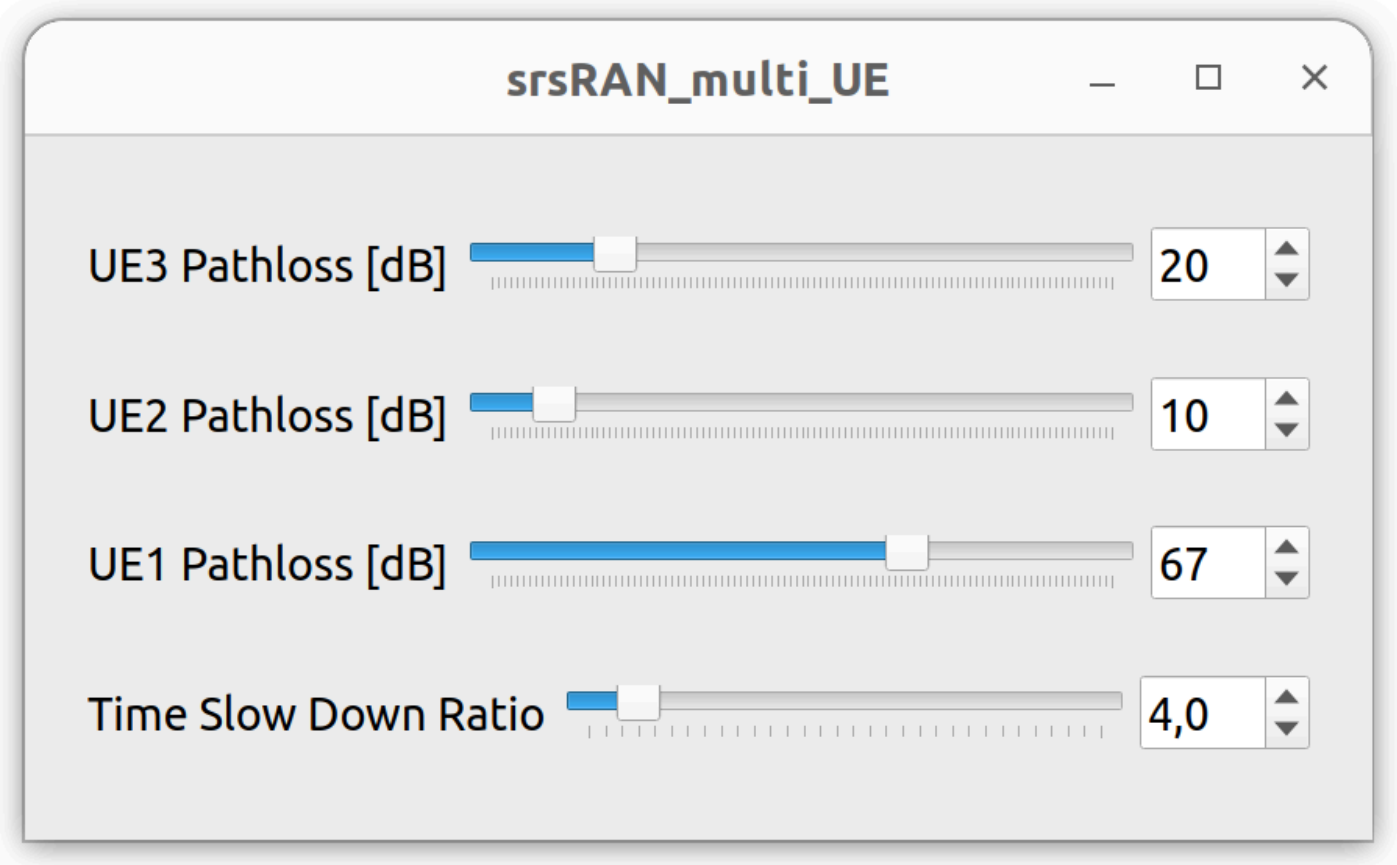
Now, you can start traffic (e.g., ping/iperf) to/from each UE using the commands described in the previous section.

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%
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.

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.

Troubleshooting

Performance Issues

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

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.

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.

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 now 2 (../src/amf/c
open5gs_5gc | 02/02 09:06:13.742: [amf] INFO: RAN_UE_NGAP_ID[2] AMF_UE_NGAP_ID[3] TAC[7] CellI
open5gs_5gc | 02/02 09:06:13.742: [amf] INFO: [suci-0-001-01-0000-0-0-0123456791] Unknown UE by
open5gs_5gc | 02/02 09:06:13.742: [amf] INFO: [Added] Number of AMF-UEs is now 2 (../src/amf/c
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/
open5gs_5gc | 02/02 09:06:13.743: [dbi] INFO: [imsi-001010123456791] Cannot find IMSI in DB (..
```

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:9999/ 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
```

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.

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
```

Run ZMQ-based setup on two host machines

If you want to run gnb and srsUE on two host machines, you need to adapt the IP addresses used for the ZMQ connection as follows:

1. gNB config:

```
ru_sdr:
device_driver: zmq                # The RF driver name.
device_args:
tx_port=tcp://0.0.0.0:2000,rx_port=tcp://<UE_PC_IP_ADDRESS>:2001,base_srate=23.04e6
...
```

1. srsUE config:

```
[rf]
...
device_name = zmq
```

```
device_args = tx_port=tcp://0.0.0.0:2001,rx_port=tcp://<GNB_PC_IP_ADDRESS>:2000,base_srate=2
3.04e6
```

Multiple gNBs connected to a single Docker-based Open5gs

To connect the 2nd gNB to the same Open5gs running in docker, one needs to add a second IP address to the bridge interface connecting to the Open5gs docker container.

1. Get the name of the Open5gs docker bridge:

```
ip -o addr show | grep "10.53.1.1"
```

1. Add a second IP address to the bridge:

```
sudo ip addr add 10.53.1.3/24 dev BRIDGE_NAME
```

1. Use the IP:10.53.1.3 as `amf.bind_addr` in the 2nd gNB config.
2. Remember to use different ZMQ ports for the gNB-UE pairs. Also, the UE has to have different IMSI, that need to be added to the Open5gs database.

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_fr
ame_size=8000,clock=external
...
[expert]
lte_sample_rates = true
```

Limitations

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.

Next steps

- Using the OAI UE — try a second open-source software UE.
- Connecting a COTS UE — connect a real 5G device over a USRP.

Building a 5G network with the OpenAirInterface UE

> **WARNING**
>
> Improving the interoperability of the OpenAirInterface (OAI) UE is an on-going effort at the Duranta project. This tutorial is based on the OAI `2026.w25` release and is intended to be used for proof-of-concept and initial testing to allow users to test OCUDU with an open source UE that is in active development. Please reach out to the Duranta community for general feedback and technical support for the OAI UE.

Overview

OCUDU is a 5G CU/DU solution and does not include a UE application. The Duranta OpenAirInterface project includes an open source 5G UE (OAI UE). Unlike the srsRAN 4G project's prototype 5G UE (srsUE), the OAI UE is in active development and is TDD capable. However, the OAI UE prior to `2026.w25` had interoperability issues with OCUDU. Without support for UCI on PUSCH, in particular for multiplexing CSI on PUSCH, the OAI UE could not operate with the OCUDU gNB in UL with CSI RS enabled. Likewise, with CSI RS disabled, DL operation required fixed MCS.

Historically, the OCUDU and Duranta projects had incompatible virtual radio interfaces, ZeroMQ (ZMQ) and RFSim respectively. The OAI `2026.w17` release introduced a compatible ZMQ radio. While not strictly required for interoperability, this feature enables testing with no hardware cost.

This tutorial shows how to create an end-to-end fully open-source 5G TDD network with OAI UE, OCUDU gNodeB and Open5GS 5G core network.

The ZMQ-based virtual radio use case is shown here. Various use cases such as over-the-air hardware setup and multi-UE emulation will be added later.

Hardware and Software Overview

For this tutorial, the following hardware and software are used:

- PC with Ubuntu 24.04.3 LTS
- OCUDU
- Duranta OpenAirInterface (2026.w25 or later)
- Two Ettus Research USRP B210s (connected over USB3)
- Open5GS 5G Core
- ZeroMQ

> **INFO**
>
> 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.

Duranta OpenAirInterface

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

ZeroMQ

Building and running OAI with ZMQ radio is also documented in the OAI ZMQ README.

Open5GS

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

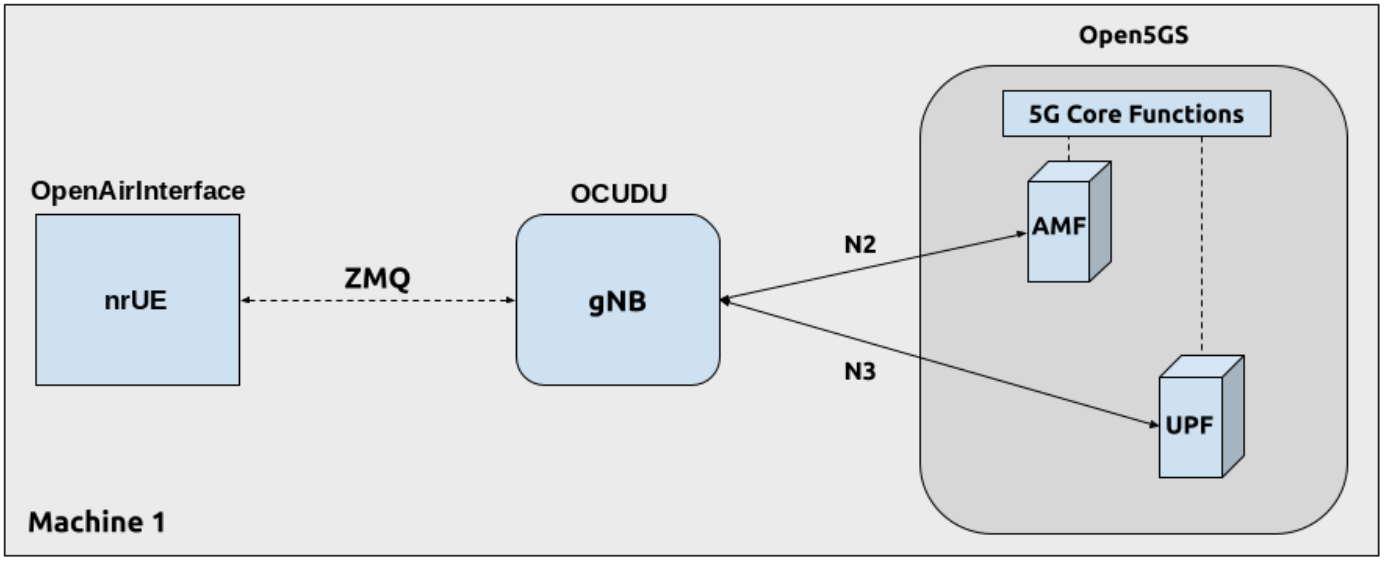
Open5GS is a C-language open-source implementation for 5G Core and EPC. The following links will provide you with the information needed to download and set-up Open5GS so that it is ready to use with OCUDU:

- GitHub
- Quickstart Guide

For the purpose of this tutorial, we will use a dockerized Open5GS version provided in OCUDU at `ocudu/docker`.

ZeroMQ-based Setup

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



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://127.0.0.1:4556,rx_port=tcp://127.0.0.1:4557
srate: 23.04
tx_gain: -12 # backoff from full scale, required by OAI fixed point FFT
rx_gain: 0
```

Note: OCUDU TX is designed to drive DAC hardware with a signal with peak at full-scale. PHY generates this signal with a **12 dB crest factor**. However, with the ZMQ interface, this signal arrives as-is at the OAI RX. OAI uses fixed point FFT, so it requires an input signal with peak well below full scale or there will be saturation in the internal FFT calculation. With radio hardware and channel in the loop, the input signal to the OAI FFT will most certainly have peak well below full scale.

The following cell configuration should be matched by the OAI UE configuration.

```
cell_cfg:
dl_arfcn: 632628
band: 78
```

```
channel_bandwidth_MHz: 20
common_scs: 30
```

TDD is enabled by default since the frequency band is **n78**. The following TDD configuration is provided as an example for setting the TDD pattern:

```
cell_cfg:
...
tdd_ul_dl_cfg:
dl_ul_tx_period: 5
nof_dl_slots: 3
nof_dl_symbols: 10
nof_ul_slots: 1
nof_ul_symbols: 2
```

CSI RS is enabled by default. However, when CSI RS is enabled, the number of CSI REs must greater than 0, and up to 8.

```
cell_cfg:
...
csi:
csi_rs_enabled: true
pucch:
nof_cell_csi_res: 1
```

OAI UE

The OAI UE UICC fields should match the subscriber database of the 5G Core Network

```
uicc0 = {
imsi = "001010123456780";
key = "00112233445566778899aabbccddeeff";
opc= "63bfa50ee6523365ff14c1f45f88737d";
pdu_sessions = ({ dnn = "internet";
nssai_sst = 1;
});
}
```

Then, the ZMQ radio driver is configured as follows:

```
device = {
name = "oai_zmqdevif";
};

zmq = (
{
tx_channels = ( "tcp://127.0.0.1:4557" );
```

```
rx_channels = ( "tcp://127.0.0.1:4556" );
}
);
```

In addition, match the PRB/bandwith, numerology/SCS, frequency band, and ARFCN/carrier frequency configuration of the gNB:

```
r = 51;
numerology = 1;
band = 78;
C = 3489420000;
```

Enable carrier scanning to find the SSB offset automatically. Optionally, enable 3/4 FFT sample rate with the `E` flag to match the gNB. Finally, the UE capabilities file is required for interoperability with any third party gNB. Some example UE capabilities file provided in-tree that work with OCUDU gNB.

```
ue-scan-carrier = 1;
E = 1;
uecap_file = "../../../targets/PROJECTS/GENERIC-NR-5GC/CONF/uecap_ports1.xml";
```

Running the Network

Once the config files are updated, the network can be set up on a single host machine.

Open5GS Core

OCUDU 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 ./ocudu/docker
docker compose up --build 5gc
```

Note that we have already configured Open5GS to operate correctly with OCUDU. Moreover, the UE database is populated with the credentials used by our OAI UE.

OCUDU gNB

We run gNB directly from the build folder, i.e., `./ocudu/build/apps/gnb/`, (the config file is also located there) with the following command:

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

The console output should be similar to:

```
--== OCUDU gNB (commit 76a15775c9) ==--

Lower PHY in executor sequential baseband mode.
Available radio types: uhd, zmq and realtime_loopback.
Cell pci=1, bw=20 MHz, 1T1R, dl_arfcn=632628 (n78), dl_freq=3489.42 MHz, dl_ssb_arfcn=632256, u

N2: Connection to AMF on 10.53.1.2:38412 completed
==== gNB started ===
Type <h> to view help
```

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_5gc | 06/25 07:22:50.219: [amf] INFO: gNB-N2 accepted[10.53.1.1]:35496 in ng-path modul
open5gs_5gc | 06/25 07:22:50.219: [amf] INFO: gNB-N2 accepted[10.53.1.1] in master_sm module (
open5gs_5gc | 06/25 07:22:50.222: [amf] INFO: [Added] Number of gNBs is now 1 (../src/amf/conte
open5gs_5gc | 06/25 07:22:50.222: [amf] INFO: gNB-N2[10.53.1.1] max_num_of_ostreams : 30 (../sr
```

OAI UE

Finally, we start OAI UE. This is also done directly from within the build folder i.e., `./openairinterface5g/cmake_targets/ran_build/build/`), with the config file in the same location:

```
sudo ./nr-uesoftmodem -O oaiue_zmq.conf
```

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

```
[NAS] Received PDU Session Establishment Accept, UE IPv4: 10.45.1.2
[SDAP] UE 0 PDU session 1: cached QFI 1
[SDAP] UE 0 PDU session 1: bringing TUN oaitun_ue1 up
[UTIL] threadCreate() for ue_tun_read_0_p1: creating thread with affinity ffffffff, priority 1
[OIP] TUN Interface oaitun_ue1 successfully configured, IPv4 10.45.1.2, IPv6 (null)
[NR_MAC] [157.7] Received TA_COMMAND 30 TAGID 0 CC_id 0
Entering ITTI signals handler
TYPE <CTRL-C> TO TERMINATE
[NR_MAC] UE 0 RNTI 4601 stats sfn: 256.4, cumulated bad DCI 0
DL harq: 19/0
UL harq: 9/0 avg code rate 0.9, avg bit/symbol 5.0, avg per TB: (nb RBs 7.0, nb symbols 14.0)
[NR_MAC] UE 0 RNTI 4601 stats sfn: 384.4, cumulated bad DCI 0
DL harq: 19/0
UL harq: 9/0 avg code rate 0.9, avg bit/symbol 5.0, avg per TB: (nb RBs 7.0, nb symbols 14.0)
```

It is clear that the connection has been made successfully once the UE has been assigned an IP, this is seen in `Received PDU Session Establishment Accept, UE IPv4: 10.45.1.2`. The TUN Interface is then

confirmed with the `TUN Interface oaitun_ue1 successfully configured, IPv4 10.45.1.2, IPv6 (null)` message.

Testing the Network

Routing Configuration

The OAI UE application configures the TUN interface and IP routing. Verify TUN interface:

```
ip addr show oaitun_ue1
```

It should show:

```
371: oaitun_ue1: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group de
link/none
inet 10.45.1.2/24 scope global oaitun_ue1
valid_lft forever preferred_lft forever
inet6 fe80::c0de:8b89:4908:d1ac/64 scope link stable-privacy
valid_lft forever preferred_lft forever
```

Check the route from the UE IP (10.45.1.2) to the 5G Core container IP (10.53.1.2)

```
ip route get 10.53.1.2 from 10.45.1.2
```

It should show the policy-based routing rule configured in the OAI UE source code:

```
10.53.1.2 from 10.45.1.2 dev oaitun_ue1 table 9999 uid 1000
cache
```

Ping

Uplink

To test the connection in the uplink direction, run ping from the host OS:

```
ping -I 10.45.1.2 10.53.1.2 -c 3
```

Downlink

Run the downlink ping from inside the 5G Core container:

```
```

```
docker exec -it open5gs_5gc ping 10.45.1.2 -c 3
```

Ping Output

Example **ping** output:

```
PING 10.45.1.2 (10.45.1.2) 56(84) bytes of data.
64 bytes from 10.45.1.2: icmp_seq=1 ttl=64 time=34.0 ms
64 bytes from 10.45.1.2: icmp_seq=2 ttl=64 time=41.6 ms
64 bytes from 10.45.1.2: icmp_seq=3 ttl=64 time=39.9 ms

--- 10.45.1.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 34.000/38.490/41.586/3.250 ms
```

iPerf3

Network-side (Server)

Start the iPerf3 server inside the 5G Core container:

```
docker exec -it open5gs_5gc 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 (Client)

With the network and the iPerf3 server up and running, the client can be run on the host OS by binding to the UE TUN interface IP:

```
# UL
iperf3 -c 10.53.1.2 -B 10.45.1.2 -t 10 -i 1
# DL
iperf3 -c 10.53.1.2 -B 10.45.1.2 -t 10 -i 1 -R
```

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 gNB.

Iperf3 Output

Example **server** iPerf3 output:

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

```
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 10.45.1.2, port 50253
[ 5] local 10.53.1.2 port 5201 connected to 10.45.1.2 port 38587
[ ID] Interval Transfer Bitrate
[ 5] 0.00-1.07 sec 1.88 MBytes 14.7 Mbits/sec
[ 5] 1.07-2.14 sec 1.62 MBytes 12.8 Mbits/sec
[ 5] 2.14-3.13 sec 1.62 MBytes 13.8 Mbits/sec
[ 5] 3.13-4.12 sec 1.75 MBytes 14.9 Mbits/sec
[ 5] 4.12-5.23 sec 1.25 MBytes 9.36 Mbits/sec
[ 5] 5.23-6.30 sec 896 KBytes 6.89 Mbits/sec
[ 5] 6.30-7.27 sec 768 KBytes 6.48 Mbits/sec
[ 5] 7.27-8.16 sec 896 KBytes 8.23 Mbits/sec
[ 5] 8.16-9.25 sec 1.00 MBytes 7.71 Mbits/sec
[ 5] 9.25-10.23 sec 896 KBytes 7.51 Mbits/sec
[ 5] 10.23-10.77 sec 512 KBytes 7.72 Mbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval Transfer Bitrate
[ 5] 0.00-10.77 sec 13.0 MBytes 10.1 Mbits/sec receiver
```

Example **client** iPerf3 output:

```
# iperf3 -c 10.53.1.2 -B 10.45.1.2 -t 10 -i 1
Connecting to host 10.53.1.2, port 5201
[ 5] local 10.45.1.2 port 38587 connected to 10.53.1.2 port 5201
[ ID] Interval Transfer Bitrate Retr Cwnd
[ 5] 0.00-1.00 sec 2.62 MBytes 22.0 Mbits/sec 0 164 KBytes
[ 5] 1.00-2.00 sec 1.62 MBytes 13.6 Mbits/sec 0 243 KBytes
[ 5] 2.00-3.00 sec 2.38 MBytes 19.9 Mbits/sec 0 325 KBytes
[ 5] 3.00-4.00 sec 2.25 MBytes 18.9 Mbits/sec 0 416 KBytes
[ 5] 4.00-5.00 sec 1.00 MBytes 8.39 Mbits/sec 0 481 KBytes
[ 5] 5.00-6.00 sec 1.00 MBytes 8.39 Mbits/sec 0 523 KBytes
[ 5] 6.00-7.00 sec 1.00 MBytes 8.39 Mbits/sec 0 563 KBytes
[ 5] 7.00-8.00 sec 1.25 MBytes 10.5 Mbits/sec 0 609 KBytes
[ 5] 8.00-9.00 sec 1.25 MBytes 10.5 Mbits/sec 0 655 KBytes
[ 5] 9.00-10.00 sec 1.25 MBytes 10.5 Mbits/sec 0 703 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval Transfer Bitrate Retr
[ 5] 0.00-10.00 sec 15.6 MBytes 13.1 Mbits/sec 0 sender
[ 5] 0.00-10.77 sec 13.0 MBytes 10.1 Mbits/sec receiver
```

OCUDU gNB Console Traces

During iPerf3 UL test:

```
|--------------------DL---------------------|-------------------------------UL-----------------
pci rnti | cqi ri mcs brate ok nok (%) dl_bs | pusch rsrp ri mcs brate ok nok (%) bsr ta phr
1 4601 | 15 1.0 27 1.2M 777 0 0% 7 | 41.1 -35.2 1 27 16.4M 400 0 0% 700k 239n 38
1 4601 | 15 1.0 27 1.2M 767 0 0% 0 | 41.1 -35.2 1 27 16.4M 400 0 0% 700k 239n 38
1 4601 | 15 1.0 27 1.2M 768 0 0% 0 | 41.1 -35.2 1 27 16.4M 400 0 0% 700k 239n 38
1 4601 | 15 1.0 27 1.3M 787 0 0% 11 | 41.1 -35.2 1 27 16.3M 400 0 0% 700k 239n 38
1 4601 | 15 1.0 27 194k 114 0 0% 0 | 41.2 -35.2 1 27 1.95M 51 0 0% 0 238n 38
```

During iPerf3 DL test:

```
|--------------------DL---------------------|-------------------------------UL-----------------
pci rnti | cqi ri mcs brate ok nok (%) dl_bs | pusch rsrp ri mcs brate ok nok (%) bsr ta phr
1 4601 | 15 1.0 27 66M 1600 0 0% 1.24M | 42.0 -35.2 1 27 1.25M 170 0 0% 384 243n 38
1 4601 | 15 1.0 27 66M 1600 0 0% 1.61M | 42.1 -35.2 1 27 1.27M 176 0 0% 1.45k 243n 38
1 4601 | 15 1.0 27 66M 1600 0 0% 2.03M | 42.1 -35.2 1 27 1.29M 178 0 0% 384 243n 38
1 4601 | 15 1.0 27 66M 1600 0 0% 2.42M | 42.1 -35.2 1 27 1.27M 173 0 0% 1.45k 243n 38
1 4601 | 15 1.0 27 66M 1600 0 0% 2.56M | 42.0 -35.2 1 27 1.30M 176 0 0% 1.04k 243n 38
1 4601 | 15 1.0 27 66M 1600 0 0% 2.56M | 42.1 -35.2 1 27 1.13M 160 0 0% 535 242n 38
1 4601 | 15 1.0 27 54M 1321 0 0% 0 | 42.1 -35.2 1 27 1.87M 215 0 0% 0 245n 38
```

Troubleshooting

Performance Issues

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

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.

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 OAI UE console output will look similar to this:

```
[NR_MAC] [UE 0][130.5][RAPROC] 4-Step RA procedure succeeded. CBRA: Contention Resolution is su
[NR_RRC] [UE0][RAPROC] Logical Channel DL-CCCH (SRB0), Received NR_RRCSetup
[RLC] Added srb 1 to UE 0
[NR_RRC] State = NR_RRC_CONNECTED
[NAS] Generate Initial NAS Message: Registration Request
[NAS] [UE 0] Received NR_NAS_CONN_ESTABLISH_IND: asCause 0
[NR_RRC] [UE 0][RAPROC] Logical Channel UL-DCCH (SRB1), Generating RRCSetupComplete (bytes33)
[MAC] [UE 0] Applying CellGroupConfig from gNodeB
[NR_MAC] NR_SRI_PUSCH_PowerControl not implemented, power control will not work as intended
[NR_RRC] [UE 0] Received RRC Release (gNB 0)
[NAS] [UE 0] Received NAS_DOWNLINK_DATA_IND type FGS_REGISTRATION_REJECT with length 4
[NAS] Received Registration reject message too short
[RLC] Released RLC entity: ue_id=0, lc_id=1
```

```
[NR_RRC] RRC moved into IDLE state
[NAS] [UE 0] Received NR_NAS_CONN_RELEASE_IND: cause OTHER
```

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 | 07/06 22:09:49.507: [amf] INFO: InitialUEMessage (../src/amf/ngap-handler.c:437)
open5gs_5gc | 07/06 22:09:49.507: [amf] INFO: [Added] Number of gNB-UEs is now 1 (../src/amf/c
open5gs_5gc | 07/06 22:09:49.507: [amf] INFO: RAN_UE_NGAP_ID[0] AMF_UE_NGAP_ID[3] TAC[7] CellI
open5gs_5gc | 07/06 22:09:49.507: [amf] INFO: [suci-0-001-01-0000-0-0-0123456786] Unknown UE by
open5gs_5gc | 07/06 22:09:49.507: [amf] INFO: [Added] Number of AMF-UEs is now 1 (../src/amf/c
open5gs_5gc | 07/06 22:09:49.507: [gmm] INFO: Registration request (../src/amf/gmm-sm.c:1339)

...

open5gs_5gc | 07/06 22:09:49.509: [dbi] INFO: [imsi-001010123456786] Cannot find IMSI in DB (..
```

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.

The dockerized Open5GS Core provided in OCUDU is populated from the `SUBSCRIBER_DB` entry in `ocudu/docker/open5gs/open5gs.env`. View this file to confirm the subscriber credentials:

```
cat ocudu/docker/open5gs/open5gs.env
```

The `SUBSCRIBER_DB` line lists the subscriber as comma-separated fields, beginning with the IMSI, key, and OPc:

```
SUBSCRIBER_DB=001010123456780,00112233445566778899aabbccddeeff,opc,63bfa50ee6523365ff14c1f45f88
```

Confirm that the IMSI (`001010123456780`), key, and OPc match the `imsi`, `key`, and `opc` values in the `uicc0` section of your `oaiue_zmq.conf`. Correct the UE's config file if needed.

In addition to the credentials, the DNN requested by the UE in its PDU session must match the APN provisioned for the subscriber. The subscriber database is populated by the `ocudu/docker/open5gs/add_users.py` script, whose default APN name is `internet`:

```
def add_user(
imsi,
key="00112233445566778899aabbccddeeff",
op=None,
opc="63bfa50ee6523365ff14c1f45f88737d",
amf="9001",
apn="internet",
qci="9",
```

```
ip_alloc="",
):
```

Confirm that the `dnn` in the `pdu_sessions` block of your `oaiue_zmq.conf` is also set to `internet`. If the UE's DNN and the provisioned APN do not match, the UE will fail to establish a PDU session and will not be assigned an IP address.

Unlike a missing subscriber, a DNN mismatch allows registration to complet. In this case the core log shows a `Registration complete` message followed by a `DNN ... Not Supported` warning and a `UE Context Release`:

```
open5gs_5gc | 07/06 22:14:17.552: [gmm] INFO: [imsi-001010123456780] Registration complete (../

...

open5gs_5gc | 07/06 22:14:17.552: [gmm] WARNING: [imsi-001010123456780] Ue requested DNN "foo"
open5gs_5gc | 07/06 22:14:19.356: [amf] INFO: UE Context Release [Action:2] (../src/amf/ngap-ha
```

UE IP Forwarding

To ensure that UE traffic is routed correctly to the internet, IP forwarding and source NAT must be configured. This differs from the srsUE tutorial, where the srsUE runs in its own network namespace (`ue1`) and forwarding is applied on the host machine. The OAI UE instead creates the `oaitun_ue1` interface in the host's default namespace, and its traffic reaches the internet through the Open5GS core container, so forwarding and NAT are configured **inside that container**.

First, enable IP forwarding inside the core container so its kernel forwards packets between the internal UE tunnel (`ogstun`) and its outbound interface:

```
docker exec open5gs_5gc sysctl -w net.ipv4.ip_forward=1
```

Then add a source NAT rule to the container's `nat` table. For packets from the UE subnet (`10.45.0.0/16`) that leave on any interface other than the internal `ogstun` tunnel, `MASQUERADE` rewrites the source address to that outbound interface's address. The internet then sees the traffic as coming from the container, so return packets can be routed back and translated to the UE:

```
docker exec open5gs_5gc iptables -t nat -A POSTROUTING -s 10.45.0.0/16 ! -o ogstun -j
MASQUERADE
```

To check that this has been configured correctly, ping an external address such as the Google DNS from the UE interface:

```
ping -I oaitun_ue1 8.8.8.8
```

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

Limitations

The current OAI UE implementation has the following feature limitation:

- With CSI RS enabled on the OCUDU gNB, Tracking Reference Signal (TRS) is not handled by the OAI UE

Next steps

- Connecting a COTS UE — connect a real 5G device over a USRP.

Connecting a commercial 5G device to OCUDU

> **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.

Overview

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

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

- PC with Ubuntu 22.04 or later
- OCUDU CU/DU
- RF-frontend compatible with OCUDU, e.g. a USRP
- 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)

> **INFO**
>
> Optionally you can also use of an external clock source such as an Octoclock or GPSDO like the Leo Bodnar GPS reference clock.

Setup Considerations

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 OCUDU:

- GitHub
- Quickstart Guide

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 OCUDU.

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.

Configuration

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

- gnb_rf_b200_tdd_n78_20mhz.yml

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

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

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

1. gNB
2. ISIM
3. Core
4. COTS UE

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 `internal`. If you are using an external clock source (such as the Leo Bodnar GPSDO) set this 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.

```
cu_cp:
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.
supported_tracking_areas: # Configure the TA associated with the CU-CP
- tac: 7
plmn_list:
- plmn: "90170"
tai_slice_support_list:
- sst: 1

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
- clock: # Specify the clock source used by the RF.
+ clock: internal # Set to external only if using Leo Bodnar GPSDO as 10 MHz reference.
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
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.
```

> **NOTE**
>
> OCUDU supports all NR bands. However, not all BWs are supported in all bands, to confirm that your configuration is correct, you should reference *Table 5.3.5-1* in TS 38.104.

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 MCC, 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.

Adding user-controlled PLMNs

If you want to add user-controlled PLMNs to the USIM, you can do this by extending the EF_PLMNwACT EF. The following commands will add the PLMN `90170` to the USIM. First verify the current PLMN list which should include your home PLMN, e.g. `00101`. Then with edit_binary_decoded you can add the new PLMN.

```
pySIM-shell (MF)> select MF
pySIM-shell (MF)> select ADF.USIM
pySIM-shell (MF/ADF.USIM)> select EF.PLMNwAcT
pySIM-shell (00:MF/ADF.USIM/EF.PLMNwAcT)> read_binary_decoded
[
{
"mcc": "001",
"mnc": "01",
"act": [
"E-UTRAN NB-S1",
"E-UTRAN WB-S1",
```

```
"EC-GSM-IoT",
"GSM",
"GSM COMPACT",
"NG-RAN",
"UTRAN",
"cdma2000 1xRTT",
"cdma2000 HRPD"
]
}
..
pySIM-shell (00:MF/ADF.USIM/EF.PLMNwAcT)> edit_binary_decoded
{
"mcc": "901",
"mnc": "70",
"act": [
"E-UTRAN NB-S1",
"E-UTRAN WB-S1",
"EC-GSM-IoT",
"GSM",
"GSM COMPACT",
"NG-RAN",
"UTRAN",
"cdma2000 1xRTT",
"cdma2000 HRPD"
]
}
```

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:
- - 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
```

nrf.yml

In the NRF configuration, the following modifications need to be made:

- Set the PLMN to match that of the gNB and AMF configuration files
- Check that the address of the AMF matches that of the gNB configuration file

```
logger:
file:
path: /var/log/open5gs/nrf.log
# level: info # fatal|error|warn|info(default)|debug|trace

global:
max:
ue: 1024 # The number of UE can be increased depending on memory size.
# peer: 64

nrf:
serving: # 5G roaming requires PLMN in NRF
- plmn_id:
- mcc: 999
- mnc: 70
+ mcc: 901
+ mnc: 70
sbi:
server:
- addr: 127.0.0.10
port: 7777
```

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.

COTS UE

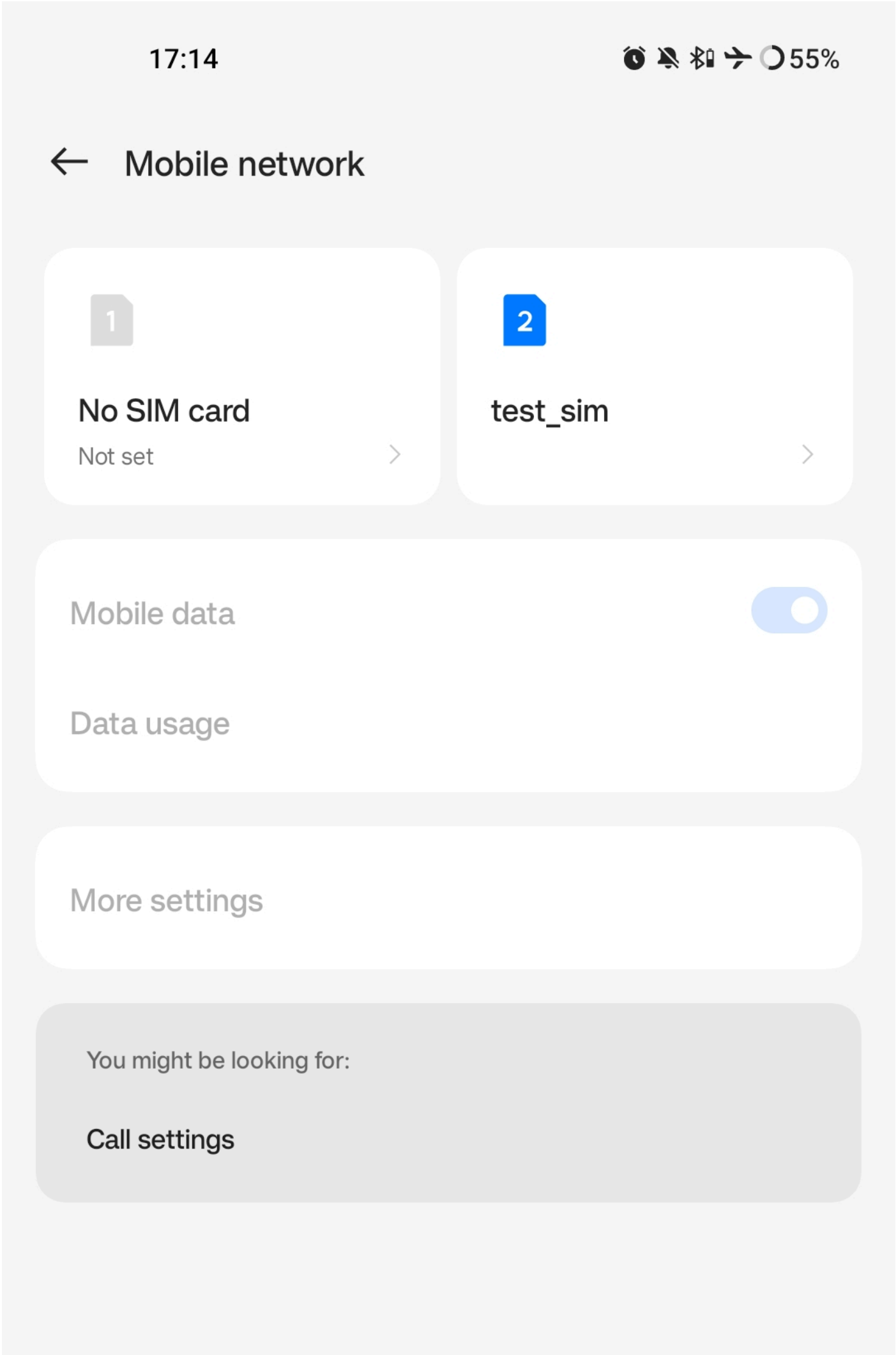
To configure the OnePlus 8t to connect to the network the following steps must be taken:

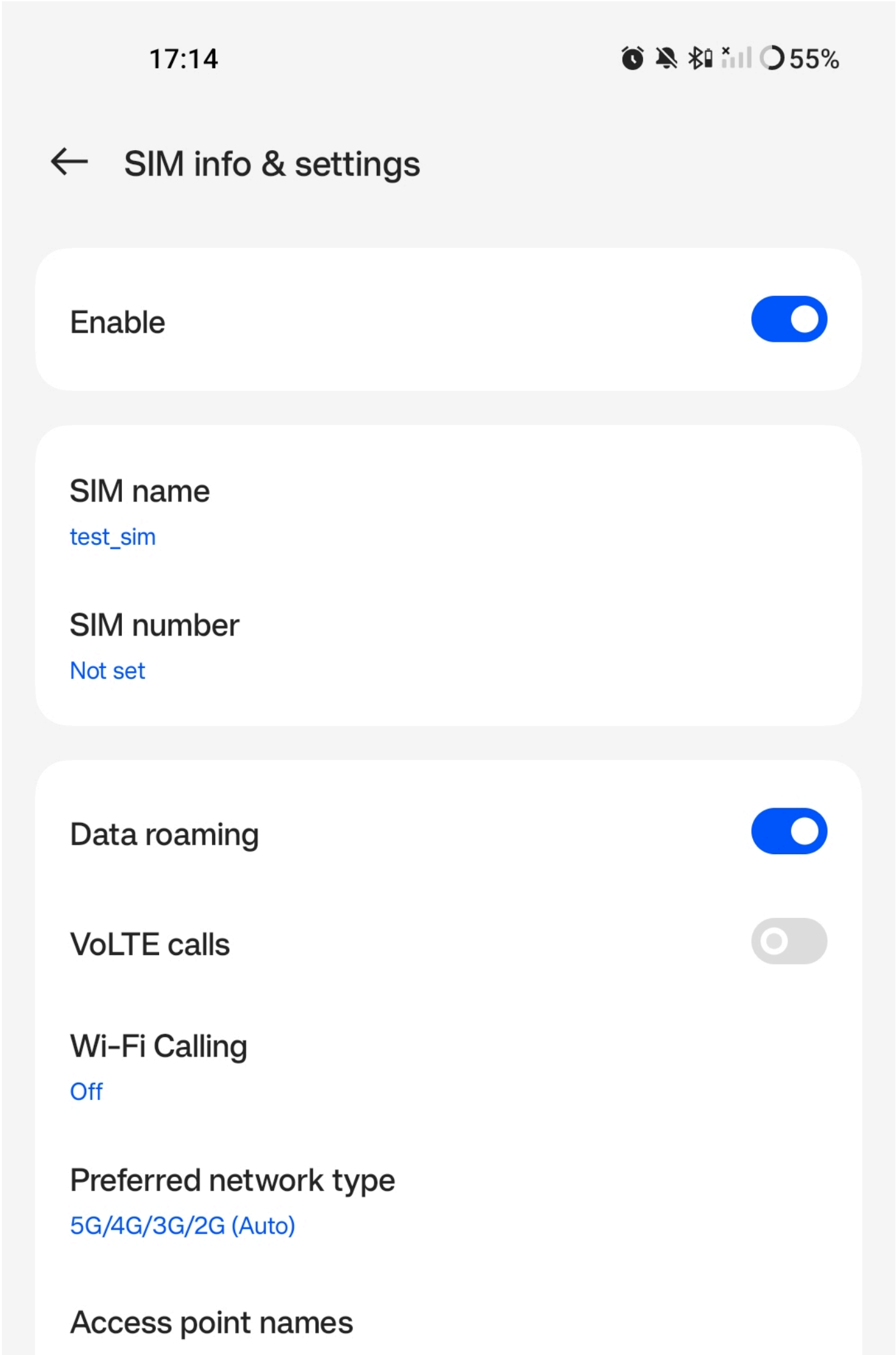
1. Enable the ISIM
2. Enable 5G SA Mode
3. Enable data roaming
4. Disable VoLTE and/or VoNR
5. Configure the APN
6. Force NR only

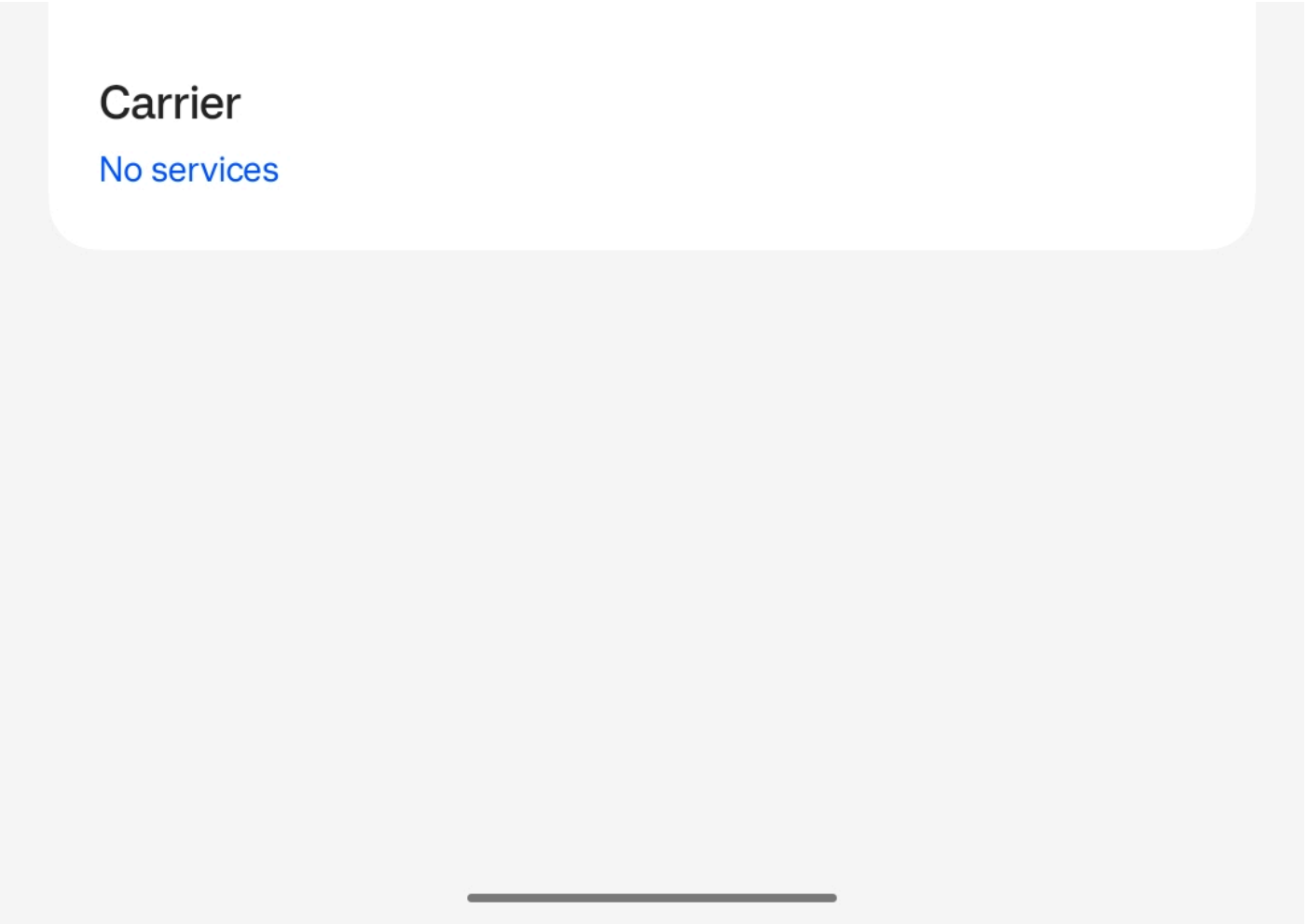
Enable ISIM, 5G and data roaming

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.


17:14
55%
Mobile network
1
No SIM card
Not set
2
test_sim
Mobile data
Data usage
More settings
You might be looking for:
Call settings


17:14
55%
SIM info & settings
Enable
SIM name
test_sim
SIM number
Not set
Data roaming
VoLTE calls
Wi-Fi Calling
Off
Preferred network type
5G/4G/3G/2G (Auto)
Access point names



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

Edit access point

Name
srsapn

APN
srsapn

Proxy
Not set

Port
Not set

Username
Not set

Password
Not set

Server
Not set

MMSC

Not set

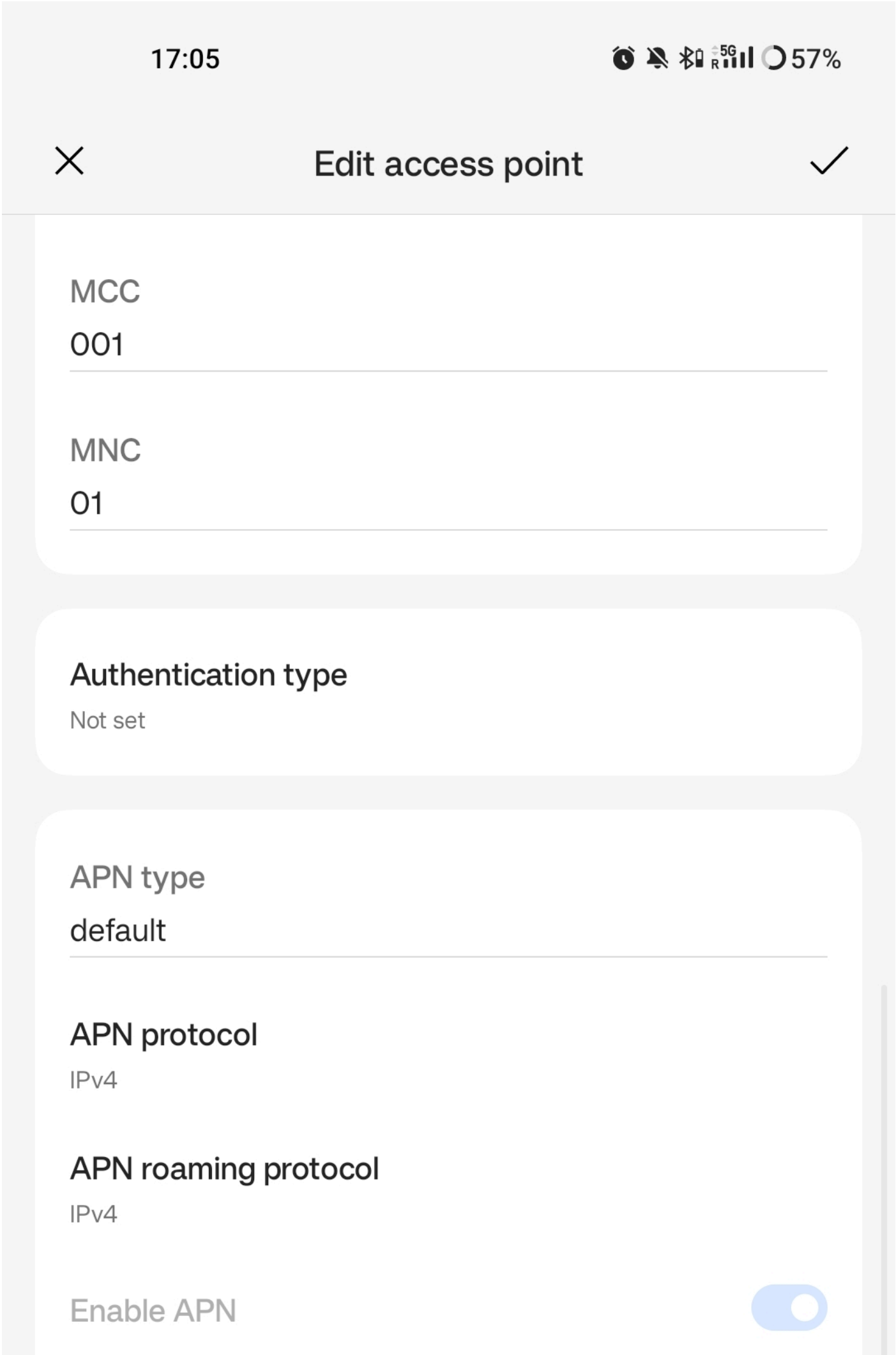
MMS proxy

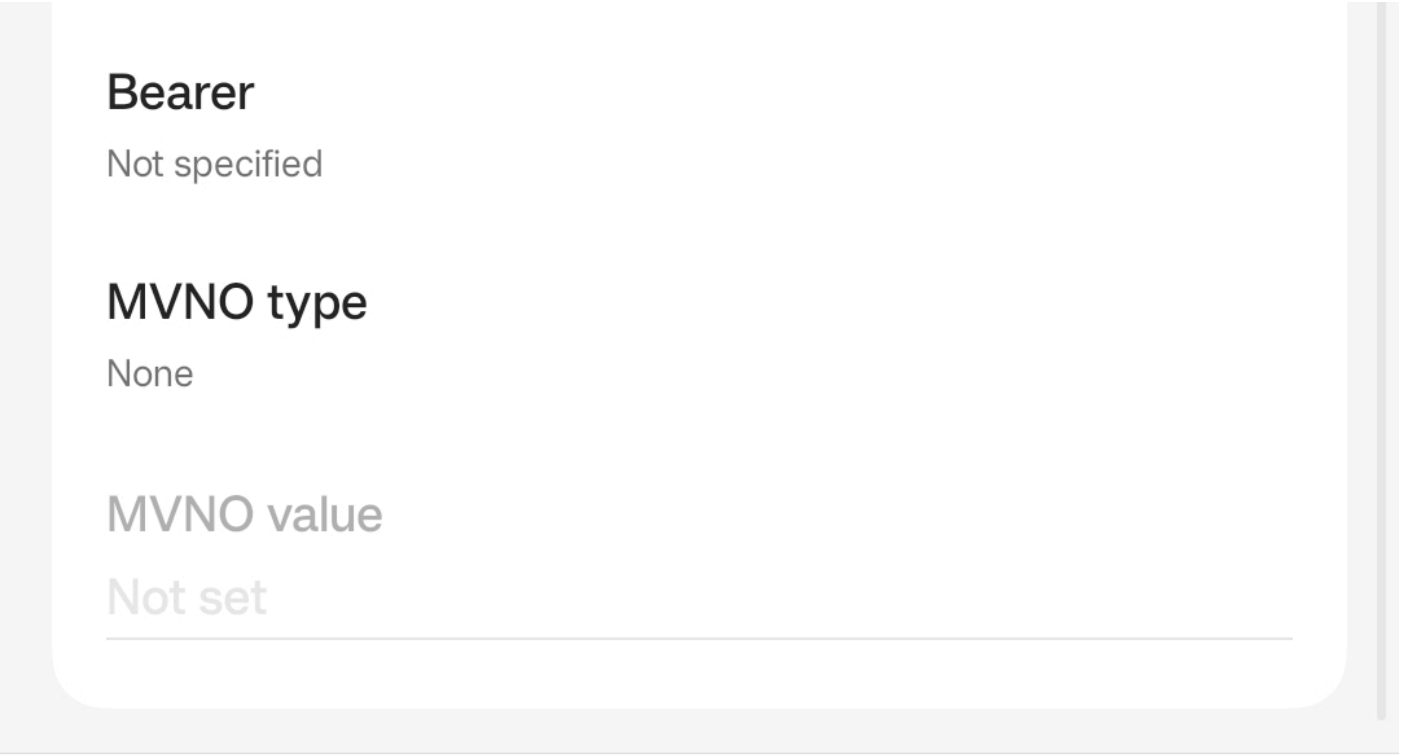
Not set

MMS port

Not set

Delete APN


17:05
57%
Edit access point
MCC
001
MNC
01
Authentication type
Not set
APN type
default
APN protocol
IPv4
APN roaming protocol
IPv4
Enable APN



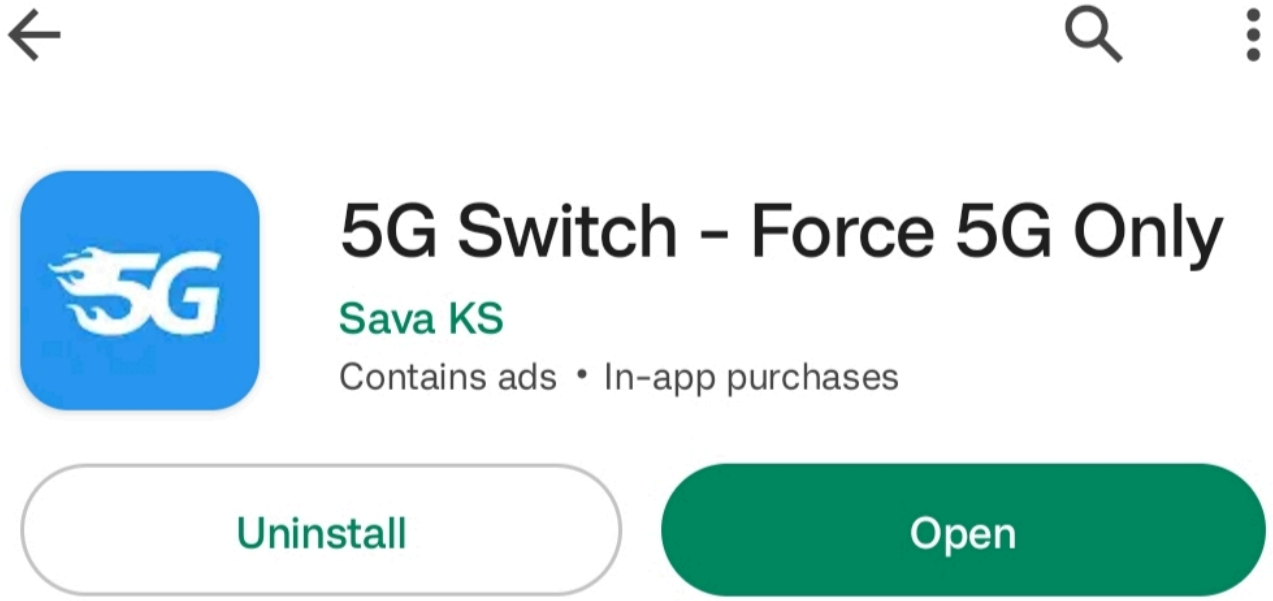
The above images show the APN configuration used in this example. The key points to note are the following:

- 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.

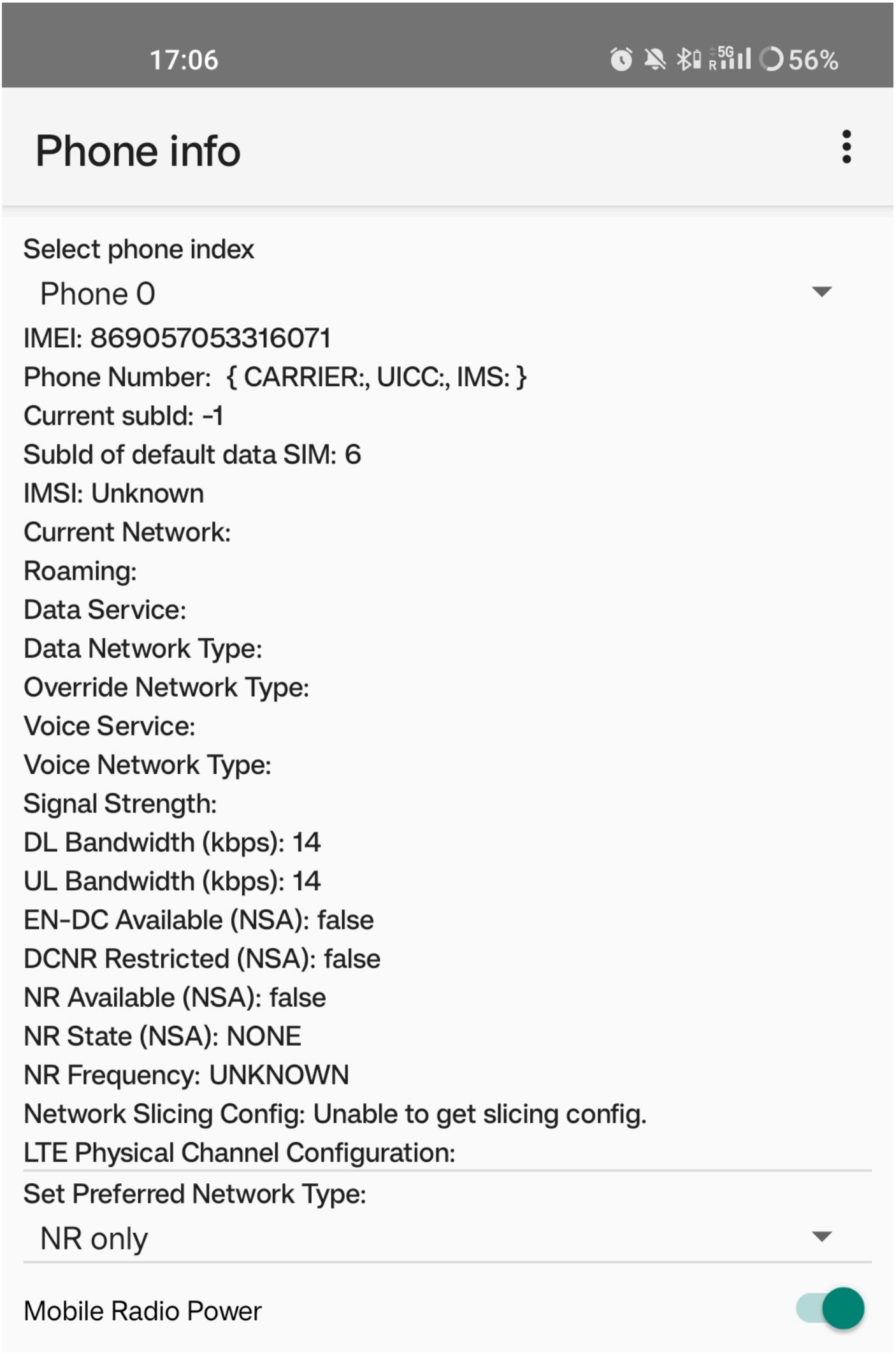
Force NR

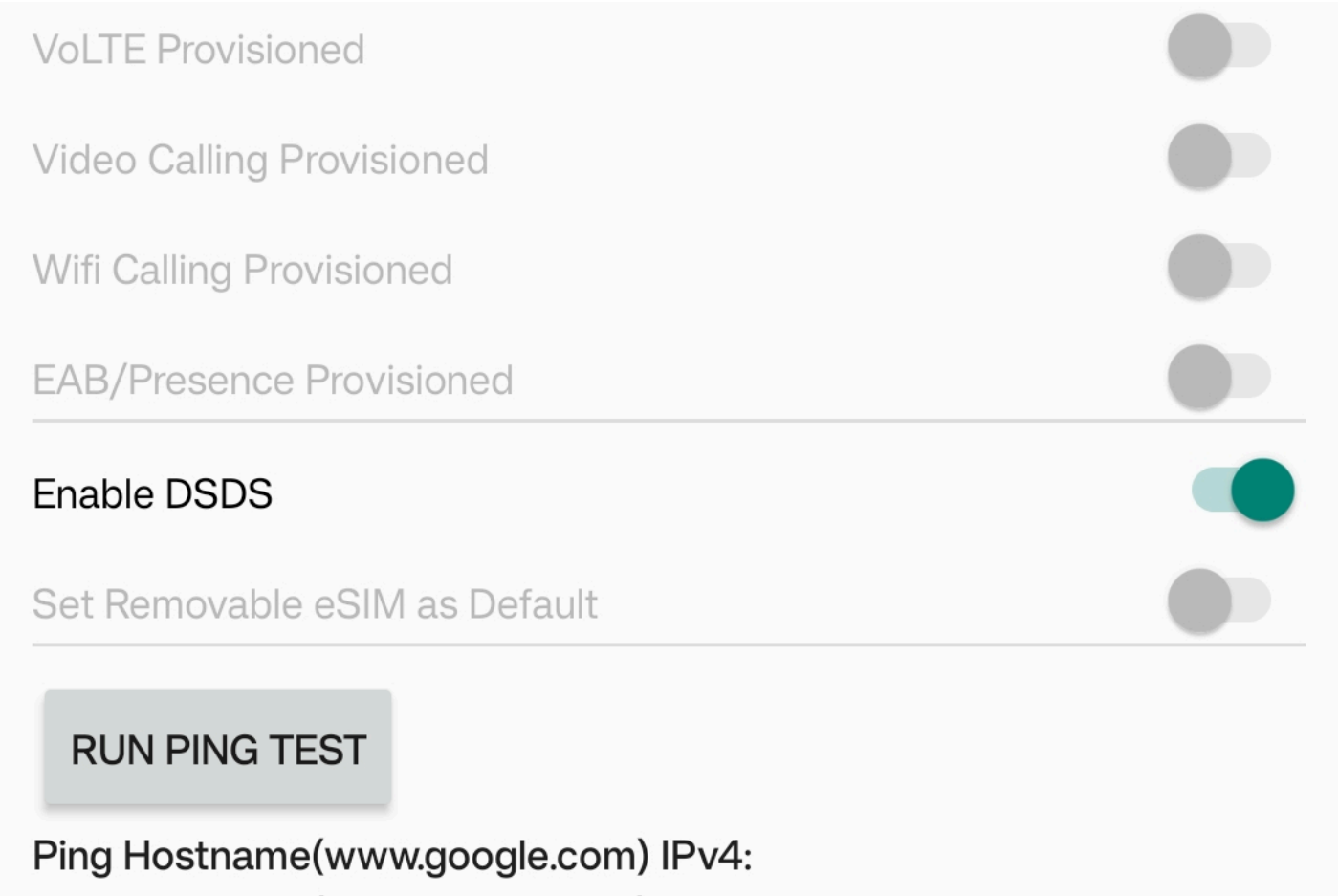
The application `5G Switch - Force 5G Only` can be used to force your device to only see 5G NR networks. This works with devices that are not rooted, and was used as part of this setup to ensure the device could see and attach to the network. Although it was not a requirement to get the phone to connect 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:


17:06
56%
Phone info
Select phone index
Phone 0
IMEI: 869057053316071
Phone Number: { CARRIER:, UICC:, IMS: }
Current subId: -1
SubId of default data SIM: 6
IMSI: Unknown
Current Network:
Roaming:
Data Service:
Data Network Type:
Override Network Type:
Voice Service:
Voice Network Type:
Signal Strength:
DL Bandwidth (kbps): 14
UL Bandwidth (kbps): 14
EN-DC Available (NSA): false
DCNR Restricted (NSA): false
NR Available (NSA): false
NR State (NSA): NONE
NR Frequency: UNKNOWN
Network Slicing Config: Unable to get slicing config.
LTE Physical Channel Configuration:
Set Preferred Network Type:
NR only
Mobile Radio Power



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.

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

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 `ocudu/build/apps/gnb` and use the following command:

```
sudo ./gnb -c gnb_rf_b200_tdd_n78_20mhz.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:

```
--== OCUDU 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.
[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)
```

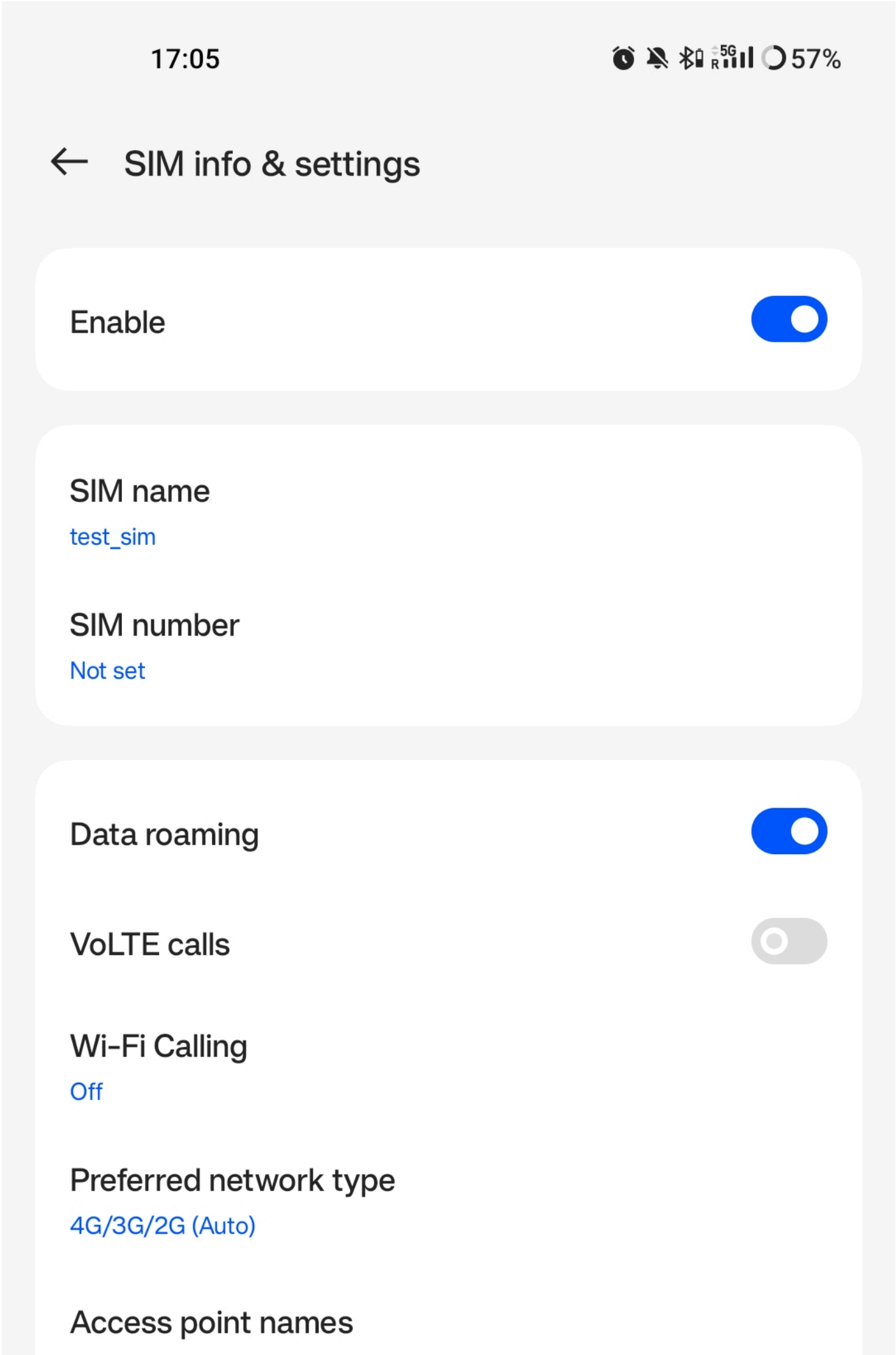
Connecting to the Network

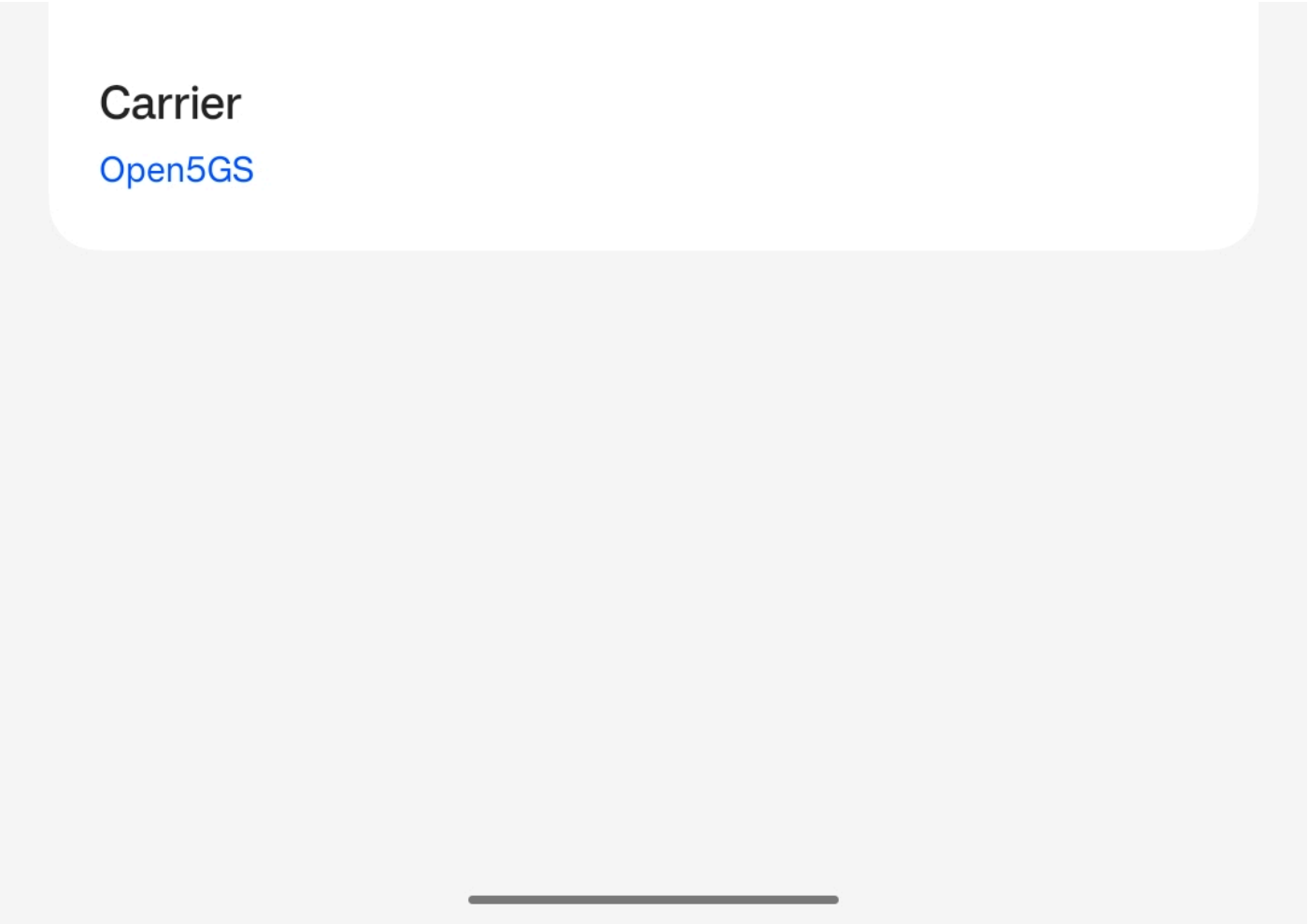
The COTS UE can now search for the network. To do this, navigate to *Mobile Network > SIM 1 > Carrier* and search for the network.

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:


17:05
57%
SIM info & settings
Enable
SIM name
test_sim
SIM number
Not set
Data roaming
VoLTE calls
Wi-Fi Calling
Off
Preferred network type
4G/3G/2G (Auto)
Access point names



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)
```

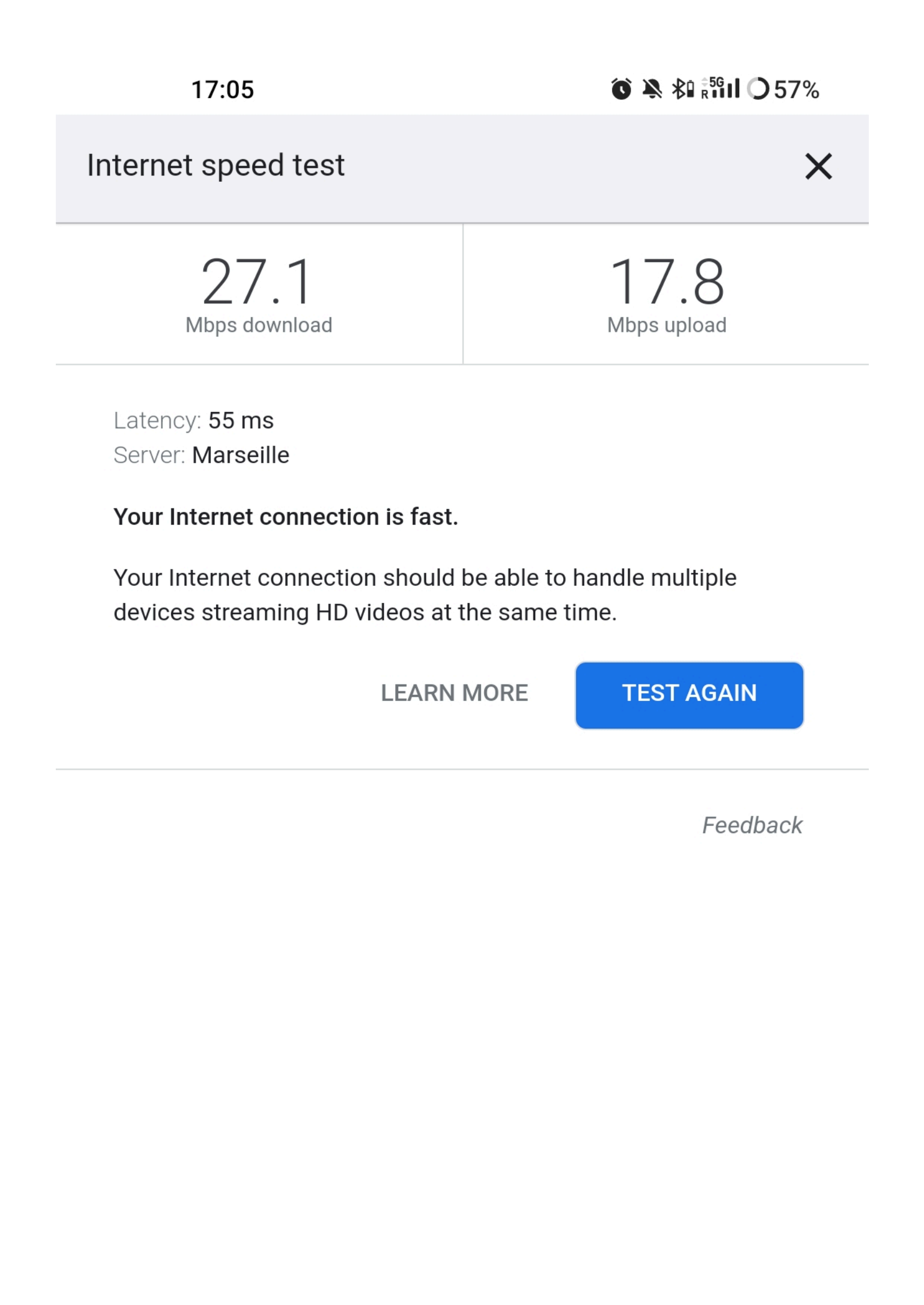
The gNB trace should show the following:

```
------------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
```

Traffic and Testing

Speed Test

Running a speedtest directly from google gives the following results:


17:05
57%
Internet speed test
27.1
Mbps download
17.8
Mbps upload
Latency: 55 ms
Server: Marseille
Your Internet connection is fast.
Your Internet connection should be able to handle multiple devices streaming HD videos at the same time.
LEARN MORE
TEST AGAIN
Feedback

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%   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
```

Video Test

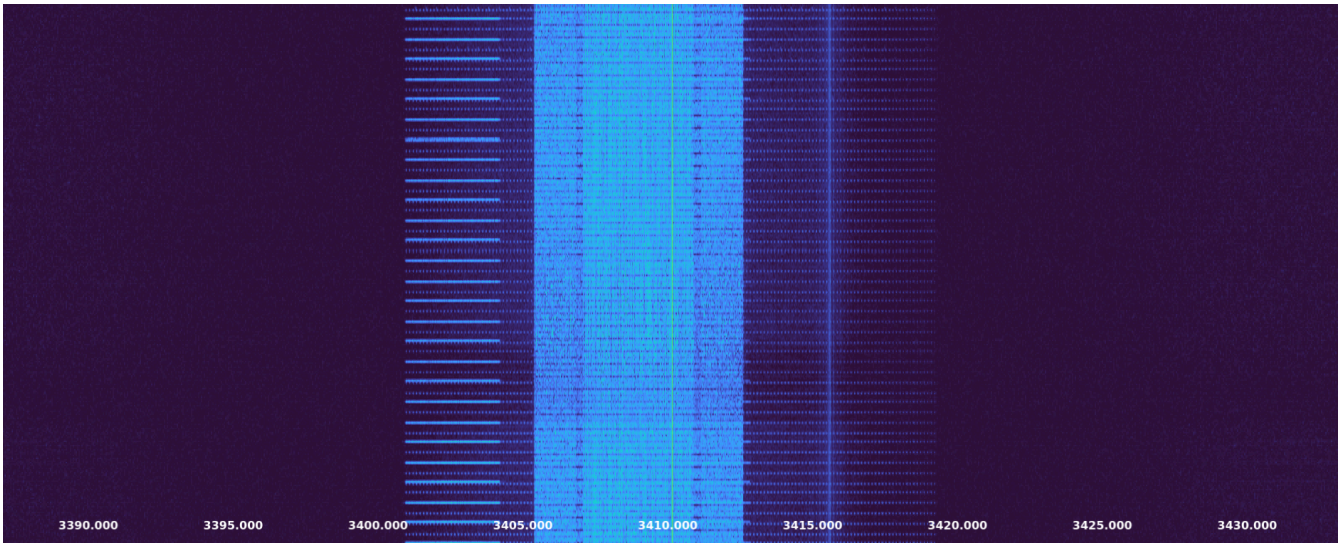
The following shows an example trace output seen while streaming video from the internet:

```
------------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
```

Troubleshooting

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



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.

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

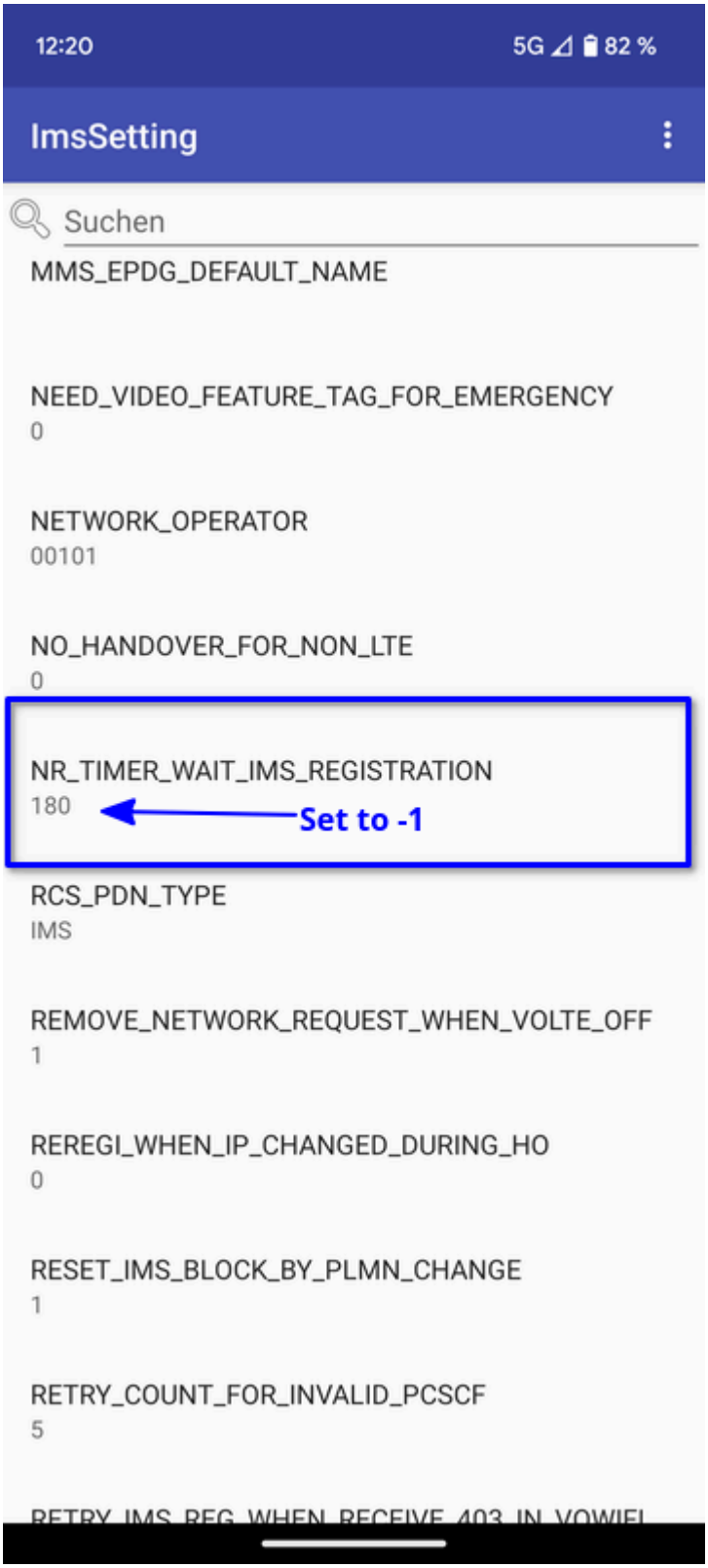
UE Disconnects after a few Minutes

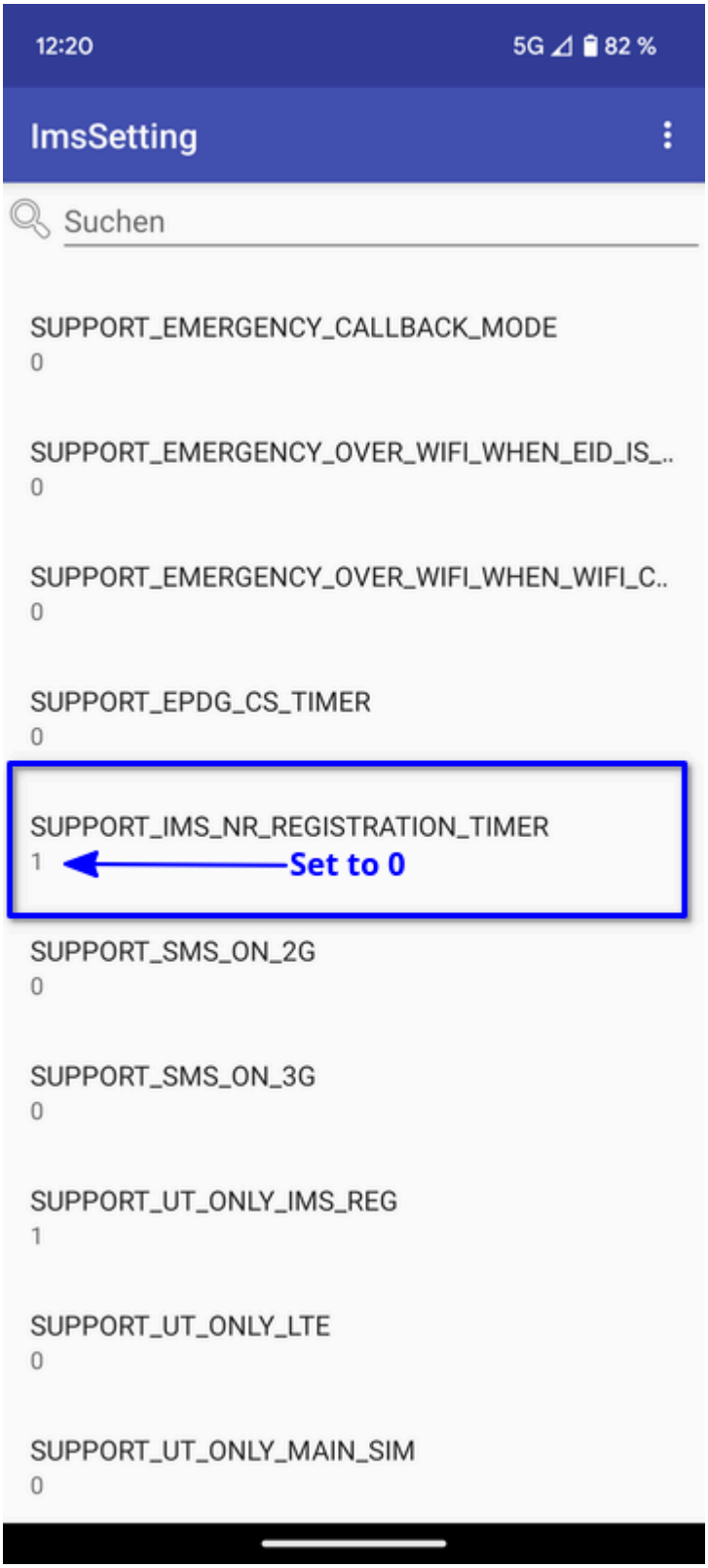
Some Android smartphones silently drop the network connection if IMS is not configured within a couple of minutes after attach. We confirmed this behavior with Google Pixel 6 having a timeout of 180s (3 minutes), but it may apply to other devices and vendors as well.

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:


12:20
5G 82 %
ImsSetting
Suchen
MMS_EPDG_DEFAULT_NAME
NEED_VIDEO_FEATURE_TAG_FOR_EMERGENCY
0
NETWORK_OPERATOR
00101
NO_HANDOVER_FOR_NON_LTE
0
NR_TIMER_WAIT_IMS_REGISTRATION
180
Set to -1
RCS_PDN_TYPE
IMS
REMOVE_NETWORK_REQUEST_WHEN_VOLTE_OFF
1
REREGI_WHEN_IP_CHANGED_DURING_HO
0
RESET_IMS_BLOCK_BY_PLMN_CHANGE
1
RETRY_COUNT_FOR_INVALID_PCSCF
5



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.

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.

Next steps

- Connecting an Amarisoft UE — scale up to multi-UE testing with a UE simulator.
- Testing handover — move a UE between two cells.
- Connecting an O-RAN RU — connect an O-RAN 7.2 radio unit.

Connecting an Amarisoft UE to OCUDU

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 OCUDU. 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, OCUDU 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 OCUDU gNB to AmariUE using virtual radios (ZMQ), and using physical radios (USRP B210).

Hardware and Software Overview

For this tutorial, the following hardware and software are used:

- PC with Ubuntu 22.04 or later
- OCUDU CU/DU
- 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.

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.

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.

Installation

ZeroMQ

To install ZMQ, and build OCUDU such that it is recognized, see the installation guide.

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`.

ZeroMQ driver for Amarisoft UE

> **NOTE**
>
> These steps should only be completed **after** compiling OCUDU as mentioned above, as they require the build files of OCUDU and of the Amarisoft UHD RF frontend driver.

Interfacing the Amarisoft UE with OCUDU over ZMQ requires a custom TRX driver implemented by SRS, which can be found in OCUDU source files in `ocudu/utils/trx_ocudu`, as shwon here.

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 `ocudu/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_ocudu_test
ctest -R trx_ocudu_test
```

Make sure CMake finds the file `trx_driver.h` in the specified folder. CMake should print the following:

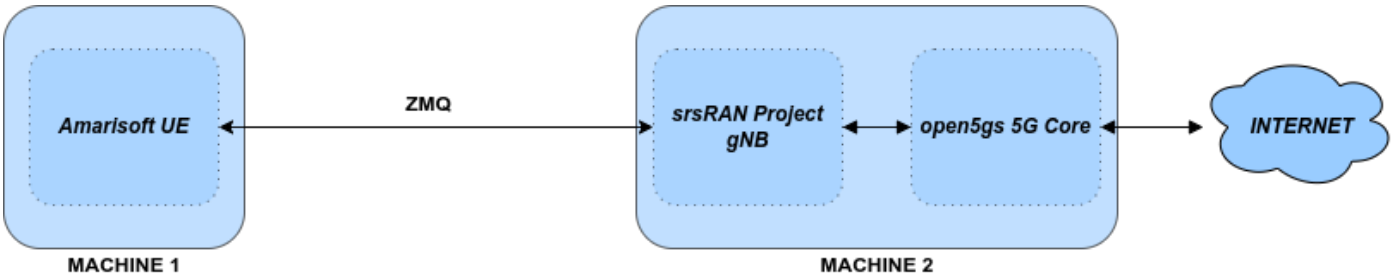
```
-- Found trx_driver.h in TRX_DRIVER_DIR=/home/user/amarisoft/2021-03-15/trx_uhd-linux-2021-
03-15/trx_driver.h
```

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 ocudu/build/utils/trx_ocudu/libtrx_ocudu.so trx_ocudu.so
```

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:



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.

OCUDU

Modify the `amf` within the `cu_cp` section with the IP of AMF and IP to which gNB needs to bind in order to connect to AMF. You will also need to define the TA list for the CU-cp:

```
cu_cp:
amf:
addr: 172.22.0.10                 # The address or hostname of the AMF.
bind_addr: 172.22.0.1             # A local IP that the gNB binds to for traffic from the
AMF.
supported_tracking_areas:         # Configure the TA associated with the CU-CP
- tac: 7
plmn_list:
- plmn: "00101"
tai_slice_support_list:
- sst: 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 `rf_driver` section of Amarisoft UE configuration with IPs from which UE sends and receives radio samples via ZMQ driver:

```
rf_driver: {
name: "ocudu",
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.

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.

OCUDU

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.

--== OCUDU gNB (commit 2c67c80) ==--

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.

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
[ 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

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.

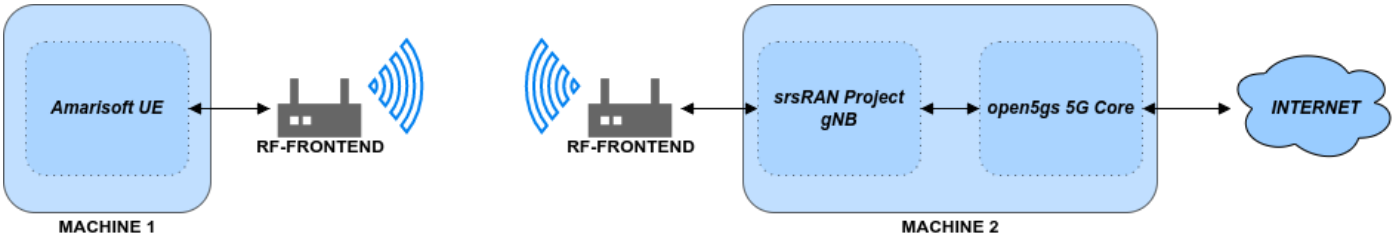
```
sim_events: [
{
event: "power_on",
start_time: 0,
},
]
```

> **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.

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:



Configuration

You can find a configuration file for this example in the `configs` folder of the OCUDU 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.

OCUDU

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 `rf_driver` section of Amarisoft UE configuration to send and receive radio samples via UHD driver:

```
rf_driver: {
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 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 */
```

And the tx timing offset:

```
{
...
cell_groups: [{
tx_time_offset: -150,
...
}],
...
}
```

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
```

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.

OCUDU

Let's now launch the gNB from its build directory.

```
sudo ./apps/gnb/gnb -c gnb_rf_b210_tdd_n78_20mhz.yml
```

The console output should be similar as follows:

```
Available radio types: uhd and zmq.

--== OCUDU 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...
[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
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
```

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
[ 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

Troubleshooting

ZMQ setup

Amarisoft UE in ZMQ based setup may not connect to OCUDU 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.

Reference clock for over-the-air

If you encounter issues with the UE 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.

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.

Next steps

- Testing handover — move a UE between two cells.

Connecting an O-RAN 7.2 radio unit

> **INFO**
>
> This tutorial covers the hardware-agnostic setup and configuration steps for connecting an O-RU via split 7.2. Hardware-specific details, including sample configuration files, for individual RU models can be found in the Radio Units section under integrations.

Overview

OCUDU 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 OCUDU CU/DU via split 7.2.

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 OCUDU through the Open Fronthaul (OFH) Library. This library is a from-scratch implementation with zero third-party dependencies that is optimized for performance and portability. It has been designed to minimize the integration and configuration burden associated with using OCUDU with 3rd-party O-RUs.

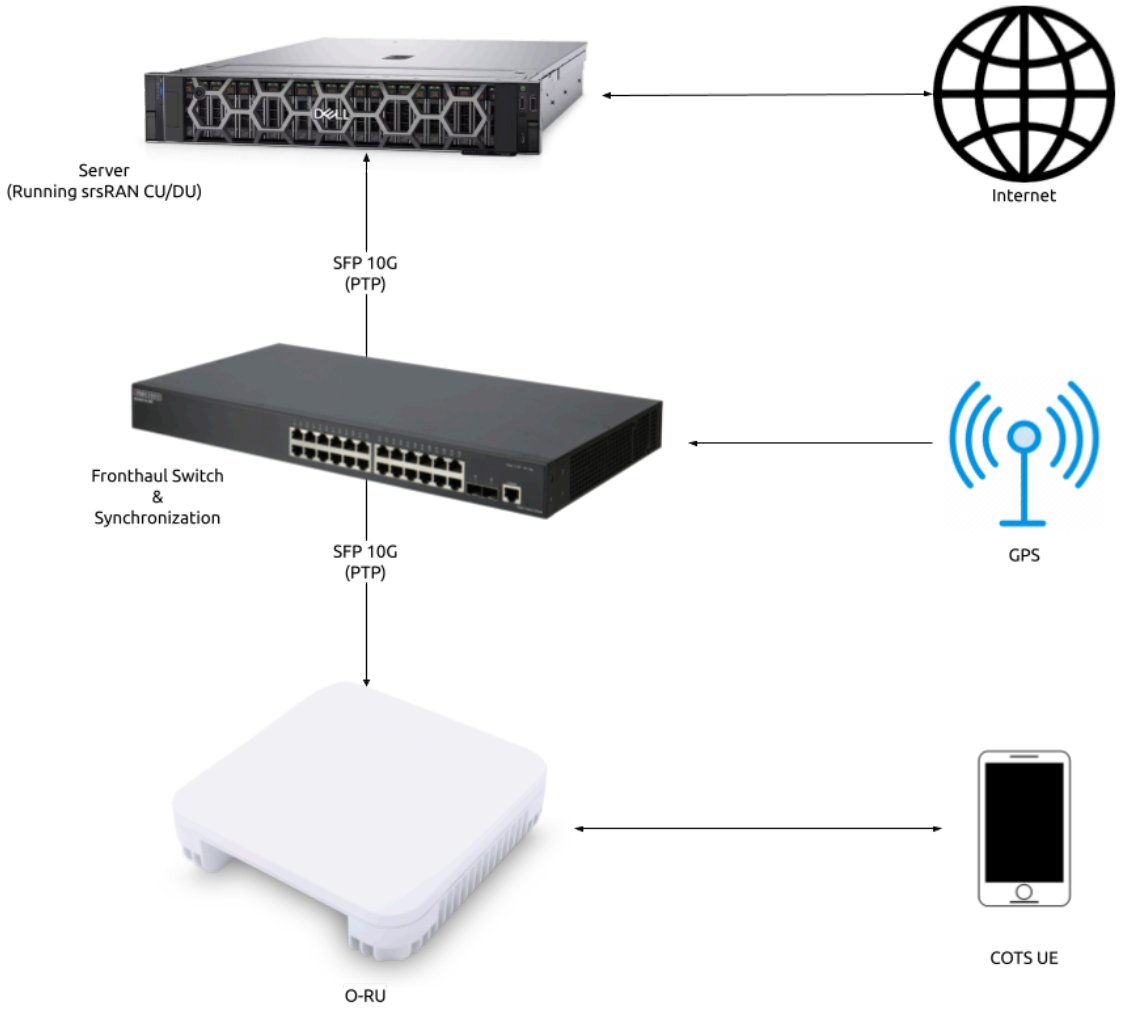
Setup Considerations

This tutorial uses the following hardware:

- Server (Running OCUDU CU/DU)
  - CPU: AMD Ryzen 7 5700G
  - MEM: 64GB
  - NIC: Intel Corporation E810 NIC
  - OS: Ubuntu 22.04 (5.15.0-1037-realtime)
- Fronthaul switch & synchronization
- RU
- COTS UE (OnePlus 9)

With the following software:

- OCUDU
- Open5GS 5G Core



CU/DU

The CU/DU is provided by this project. 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.

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 in the Radio Units section under integrations, along with details on using them with OCUDU.

For all set-ups the RU should be connected to the fronthaul switch via SFP+ fiber cable through the main fronthaul interface.

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 OCUDU:

- GitHub
- Quickstart Guide

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.

> **INFO**
>
> 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.

Configuration

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 if it is available in the Switches and Timing section under integrations.

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

PTP is used to synchronize the DU with the fronthaul switch. The PTP process should be running on the DU and the PTP sync should be checked to ensure it is synchronized correctly. ptp4l and phc2sys are the two main components of the PTP synchronization with linuxptp. To avoid any issues with the PTP, it is recommended to disable the Network Time Protocol (NTP) on the DU. Use the following command to disable NTP on Ubuntu 22.04:

```
sudo timedatectl set-ntp false
```

We have created an example configuration file for G.8275.1 Multicast PTP profile for LinuxPTP version 4. It can be downloaded `here`.

Install the LinuxPTP v4 using the following command:

```
git clone http://git.code.sf.net/p/linuxptp/code linuxptp
cd linuxptp/
git checkout v4.2
make
sudo make install
```

Run ptp4l using the following command:

```
sudo 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:

```
sudo 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 in the range of -100 to 100.

In both of the above commands `enp1s0f0` is the network interface on our DU that gets the PTP sync.

> **WARNING**
>
> Both `ptp4l` and `phc2sys` must be kept running at all times while OCUDU is active. If either process stops or loses lock, OFH packet drops or late/early packet errors will occur. See the Troubleshooting guide for help diagnosing these issues.

Enabling PTP with DPDK

If you are using PTP and OFH with DPDK on the same interface, you will need to use virtual functions (VFs) to ensure that the PTP sync is correctly passed to the relevant components.

First, identify the network port you want to use for PTP and OFH. You can use the following command to list the available network ports:

```
dpdk-devbind.py -s
```

You will see the following output or similar:

```
Network devices using using kernel driver
===========================================
0000:51:00.0 'Ethernet Controller E810-C for SFP 1593' if=enp81s0f0 drv=ice unused=vfio-pci
```

```
*Active*
0000:51:00.1 'Ethernet Controller E810-C for SFP 1593' if=enp81s0f1 drv=ice unused=vfio-pci
*Active*
0000:51:00.2 'Ethernet Controller E810-C for SFP 1593' if=enp81s0f2 drv=ice unused=vfio-pci
0000:51:00.3 'Ethernet Controller E810-C for SFP 1593' if=enp81s0f3 drv=ice unused=vfio-pci
```

In the above output, `enp81s0f0` is the network port we want to use for PTP and OFH.

Next, you will need to create the VF, you can do this by running the following commands. Remember to replace `<INTERFACE NAME>` with the name of your network interface (e.g., `enp81s0f0`):

```
# Remove old VFs
echo 0 | /sys/class/net/<INTERFACE NAME>/device/sriov_numvfs

# Create new VF
echo 1 | /sys/class/net/<INTERFACE NAME>/device/sriov_numvfs

# Set the MAC address for the VF
sudo ip link set <INTERFACE NAME> vf 0 mac 00:33:22:33:00:11 spoofchk off
```

Now, you can verify that the VF as been created by running the following command:

```
dpdk-devbind.py -s
```

You should see the following output or similar:

```
Network devices using using kernel driver
==========================================
0000:51:00.0 'Ethernet Controller E810-C for SFP 1593' if=enp81s0f0 drv=ice unused=vfio-pci
*Active*
0000:51:01.0 'Ethernet Adaptive Virtual Function 1889' drv=iavf unused=vfio-pci
0000:51:00.1 'Ethernet Controller E810-C for SFP 1593' if=enp81s0f1 drv=ice unused=vfio-pci
*Active*
0000:51:00.2 'Ethernet Controller E810-C for SFP 1593' if=enp81s0f2 drv=ice unused=vfio-pci
0000:51:00.3 'Ethernet Controller E810-C for SFP 1593' if=enp81s0f3 drv=ice unused=vfio-pci
```

In the above example, we can see that the VF has been created and is using the `iavf` driver in the line *Adaptive Virtual Function*. Note, this is **only** for the Intel E810 NIC, other NICs may have different drivers.

Next, you will need to bind the VF to the `vfio-pci` driver, you can do this by running the following command. Before running this command, make sure that `vfio-pci` is enabled, see the DPDK tutorial for more information on how to enable it.

```
sudo dpdk-devbind.py --bind=vfio-pci 0000:51:01.0
```

Finally, verify the binding by running the following command:

```
dpdk-devbind.py -s
```

You should see the following output or similar:

```
Network devices using DPDK-compatible driver
============================================
0000:51:01.0 'Ethernet Adaptive Virtual Function 1889' drv=vfio-pci unused=iavf

Network devices using kernel driver
===================================
0000:51:00.0 'Ethernet Controller E810-C for SFP 1593' if=enp81s0f0 drv=ice unused=vfio-pci
*Active*
0000:51:00.1 'Ethernet Controller E810-C for SFP 1593' if=enp81s0f1 drv=ice unused=vfio-pci
*Active*
0000:51:00.2 'Ethernet Controller E810-C for SFP 1593' if=enp81s0f2 drv=ice unused=vfio-pci
0000:51:00.3 'Ethernet Controller E810-C for SFP 1593' if=enp81s0f3 drv=ice unused=vfio-pci
```

OCUDU configuration

Sample configuration files for the CU/DU can be found in the `configs` folder of the OCUDU source files. 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 Radio Units section under Integrations for more information on configuring the CU/DU.

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 if it is available in the Radio Units section under Integrations.

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.

Initializing the Network

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.

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 *ocudu/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:

```
--== OCUDU 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.
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

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.

COTS UE

For full details on configuring and connecting a COTS UE to OCUDU 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.

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 OCUDU 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%   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
```

Integration Guide

Hardware-specific guides for O-RUs and switches tested with OCUDU in an O-RAN split 7.2 compliant network can be found in the Integrations section. All of the hardware items listed there have been tested in-house.

Next steps

- Configuring DPDK — add kernel-bypass fronthaul I/O for higher throughput.
- Running on Kubernetes — deploy the components as Kubernetes pods.

Deploying a CU/DU split over F1

Overview

This tutorial outlines the steps required to configure and run the O-DU and O-CU applications to create an E2E O-RAN compliant network with a CU-DU split. Two setups are shown: an over-the-air setup using a USRP as the RF frontend, and a ZeroMQ-based setup for testing without physical radio hardware.

The over-the-air setup uses a USRP, resulting in a Split 8 configuration. To implement a Split 7.2x configuration, use this guide in conjunction with the RU Guide.

Hardware and Software Overview

For this tutorial, the following hardware and software are used:

- PC with, e.g. Ubuntu 24.04 LTS
- OCUDU
- srsRAN 4G (23.11 or later)
- Ettus Research B210 USRP (connected over USB 3.0)
- Open5GS 5G Core (running bare metal)
- COTS UE (Xiaomi 12 5G)
- ZeroMQ

OCUDU

If you have not already done so, install the latest version of OCUDU and all of its dependencies. This is outlined in the Installation Guide.

B210

This example uses a USRP B210, it must be connected to the PC via USB 3.0. The use of an external clock is not compulsory, but for setups where the connection is unstable or the UE struggles to connect it is recommended.

Open5GS

For this example, Open5GS is running bare metal on the host machine.

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

OCUDU:

- GitHub
- Quickstart Guide

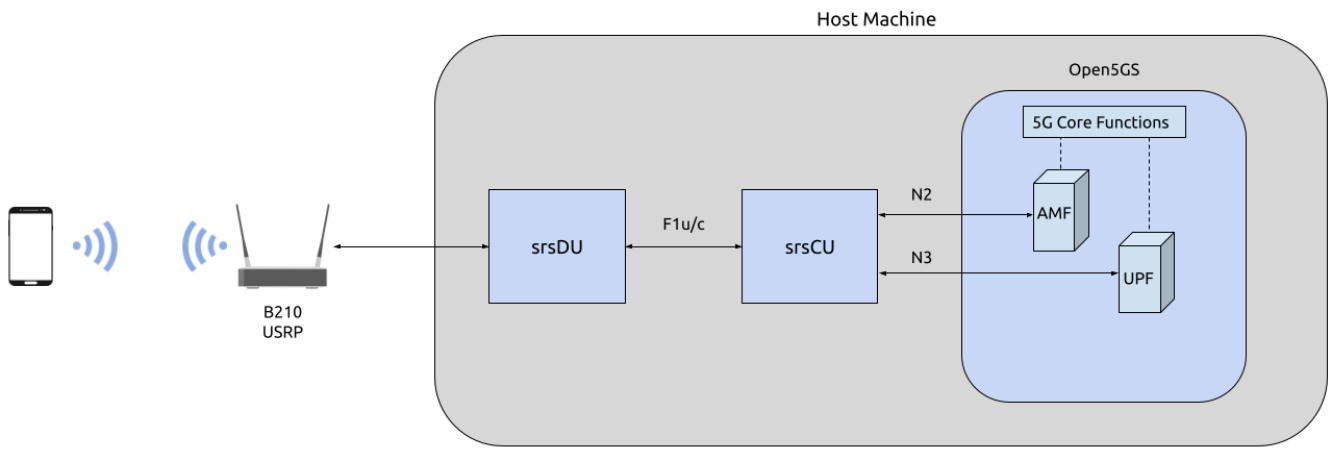
COTS UE

A 5G SA capable COTS UE is used for this tutorial, specifically the Xiaomi 12 5G. A detailed list of COTS UEs that have been tested with OCUDU can be found here.

For more information on connecting a COTS UEs to OCUDU, see the full tutorial.

Setup

This tutorial covers two setups: an over-the-air setup using a USRP, and a ZeroMQ-based setup using srsUE as a simulated UE. Select the tab below that matches your setup, then follow that tab through Configuration, Running the Network, and Connecting to the Network.



Configuration

For the CU-DU split two configuration files are needed, one for CU and one for DU. These configuration files are explained in detail here.

Core

As previously stated, Open5GS is running bare metal for this example. No configuration changes are needed, simply register the credentials of the UE being used if you haven't done so already. The Quickstart Guide linked above outlines how to configure the core.

OCUDU CU

To configure the CU, the `amf`, `f1ap` and `f1u` fields must be configured correctly in both the `cu_cp` and `cu_up`. The following configuration file shows the minimum requirements to configure srsCU:

```
cu_cp:
amf:
addr: 127.0.1.100                   # The address or hostname of the AMF.
bind_addr: 127.0.1.1                # A local IP that the gNB binds to for traffic
from the AMF.
supported_tracking_areas:           # Configure the TA associated with the CU-CP
- tac: 7
plmn_list:
- plmn: "00101"
tai_slice_support_list:
- sst: 1
f1ap:
bind_addr: 127.0.10.1               # Configure the F1AP bind address, this will
enable the CU-cp to connect to the DU

cu_up:
f1u:
socket:                             # Define UDP/IP socket(s) for F1-U interface.
-                                   # Socket 1
bind_addr: 127.0.10.1                # Sets the address that the F1-U socket will
bind to.
```

The `amf` parameters are specific to the local configuration of the core. If you are running Open5GS via the docker scripts provided with OCUDU, your configuration will be different. The same is true if you have made any other local changes to how Open5GS has been configured.

OCUDU DU

To configure the DU, the `f1ap` and `f1u` parameters must be configured, as well as the `ru_sdr` and `cell_cfg` parameters. As with srsCU, the following are the minimum requirements to configure srsDU:

```
f1ap:
cu_cp_addr: 127.0.10.1                  # The address of CU-CP
bind_addr: 127.0.1.2

f1u:
socket:
-
bind_addr: 127.0.1.2

ru_sdr:
device_driver: uhd
device_args: type=b200,num_recv_frames=64,num_send_frames=64
srate: 23.04
otw_format: sc12
tx_gain: 80
rx_gain: 40

cell_cfg:
dl_arfcn: 650000
band: 78
channel_bandwidth_MHz: 20
common_scs: 30
plmn: "00101"
```

```
tac: 7
pci: 1
```

In this example, the DU is configured to work with a USRP B210, and to create a 20 MHz cell. The specifics of the RU being used and the desired cell can be changed as needed. The `f1ap` configuration must remain constant with the associated configuration in the CU.

Running the Network

The following running order must be followed to correctly initialize the network:

1. Open5GS
2. OCUDU CU
3. OCUDU DU

Core

If the Open5GS documentation has been followed correctly, then the core should already be running as a service in the background. If not, then start the core according to the steps in the Open5GS docs.

OCUDU CU

First, navigate to the CU application folder. This can be done with the following command:

```
cd ~/ocudu/build/apps/ocu
```

To run the CU the following command can be used (assuming the configuration file is also located in the same folder):

```
sudo ./ocu -c cu.yml
```

If the CU is running correctly, you should see the following in the console:

```
--== OCUDU CU (commit 2be82d8ea3) ==--

N2: Connection to AMF on 127.0.1.100:38412 completed
F1-C: Listening for new connections on 127.0.10.1:38472...
==== CU started ===
Type <h> to view help
```

OCUDU DU

The DU is run in the same way as the CU.

First, navigate to the correct folder:

```
cd ~/ocudu/build/apps/du
```

The DU can be run with the following command (assuming the configuration file is also located in the same folder):

```
sudo ./odu -c du.yml
```

If the DU is running correctly, you will see the following in the console:

```
--== OCUDU DU (commit 2be82d8ea3) ==--

Lower PHY in quad executor mode.
Available radio types: uhd.
[INFO] [UHD] linux; GNU C++ version 11.4.0; Boost_107400; DPDK_23.11; UHD_4.8.0.0-64-
g0dede88c
[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: B200mini
[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.
[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, 1T1R, dl_arfcn=650000 (n78), dl_freq=3750 MHz,
dl_ssb_arfcn=649632, ul_freq=3750 MHz

F1-C: Connection to CU-CP on 127.0.10.1:38472 completed
==== DU started ===
Type <h> to view help
```

Connecting to the Network

Connecting the COTS UE to the network follows the same steps outlined in this tutorial.

Console Outputs

The CU console will not display any further automatic outputs once the UE is connected; however, the usual trace and outputs associated with the “vanilla” gNB output can be seen in the DU console.

Typing `t` on the DU console will result in something similar to the following output once the UE has connected:

```
|-------------------DL---------------------|-------------------------UL--------------
---------------
pci rnti | cqi   ri  mcs  brate   ok  nok  (%)  dl_bs | pusch  rsrp  mcs  brate   ok
nok  (%)    bsr     ta  phr
1 4601 |  15 1.0   21   9.2k   11    1   8%      0 |  24.2   ovl   26    33k    8    0
0%      0   -81n    0
1 4601 |  15 1.0   27   429k   84    0   0%      0 |  31.6 -11.5   28   221k   25    0
0%      0      0    7
1 4601 |  15 1.0   27   686k  119    0   0%      0 |  32.7 -12.4   28   236k   44    0
0%      0   -56n   17
1 4601 |  15 1.0   27   664k  110    0   0%      0 |  32.1 -12.8   28   353k   46    0
0%     10   -32n   16
1 4601 |  15 1.0   27   517k   64    0   0%      0 |  33.6 -12.3   28   124k   29    0
0%    198   -40n   17
1 4601 |  15 1.0   27    60k   36    0   0%      0 |  33.0 -11.8   28   127k   21    0
0%      0   -24n   17
```

Troubleshooting

CU and DU not connecting

If the DU fails to establish an F1AP connection to the CU, the DU console will not print `F1-C: Connection to CU-CP on ... completed`. The DU will also not proceed past cell configuration. Check the following:

- The `f1ap.cu_cp_addr` (OTA) or `f1ap.addrs` (ZMQ) value in the DU config must exactly match the `f1ap.bind_addr` (OTA) or `f1ap.bind_addrs` (ZMQ) value in the CU config.
- Start the CU before the DU. The DU retries the F1AP connection. It does not proceed with startup until it connects to the CU.
- Confirm that no firewall rule or other process on the host is blocking the configured F1AP port. Check with `sudo netstat -tulnp | grep <port>`, for example.
- If the CU and DU run on separate hosts, confirm the bind and target addresses reflect their actual reachable IP addresses. Do not reuse the loopback addresses from the single-host examples above.

DU and srsUE not exchanging samples (ZeroMQ setup)

If srsUE cannot detect the cell, or the DU repeatedly logs errors related to samples or timeouts, suspect a ZMQ port mismatch:

- Confirm the DU's `tx_port` matches srsUE's `rx_port`. Confirm the DU's `rx_port` matches srsUE's `tx_port`, as described under Running the Network in the ZeroMQ tab above. Forgetting to swap the ports between the DU and srsUE configs is the most common cause of this issue.

- Confirm `base_srate` in the DU config matches the corresponding sample rate in the srsUE config; a mismatch produces synchronization errors.
- Make sure only one instance of the DU or srsUE is running. A leftover process can hold a ZMQ port open. This blocks a new instance from binding to it.

UE does not connect to the core

If the UE reaches the DU without registering with the core, check the following:

- Confirm the UE's subscriber credentials exist in the Open5GS subscriber database.
- Confirm the `plmn` and `tac` values in the DU config match the tracking area configured in the CU-CP and in Open5GS.

See the Troubleshooting guide for further steps at the core network level.

Next steps

- Connecting an O-RAN RU — connect an O-RAN 7.2 radio unit.
- Integrating a Near-RT RIC — add the E2 interface and deploy an xApp.
- Running on Kubernetes — deploy the components as Kubernetes pods.

Integrating a Near-RT RIC and deploying an xApp

Overview

This tutorial shows how to use the E2 interface exposed by OCUDU and showcases its interoperability with third-party RIC frameworks. For this purpose, we use O-RAN Alliance compliant NearRT-RICs and xApps provided in ORAN SC RIC and FlexRIC projects.

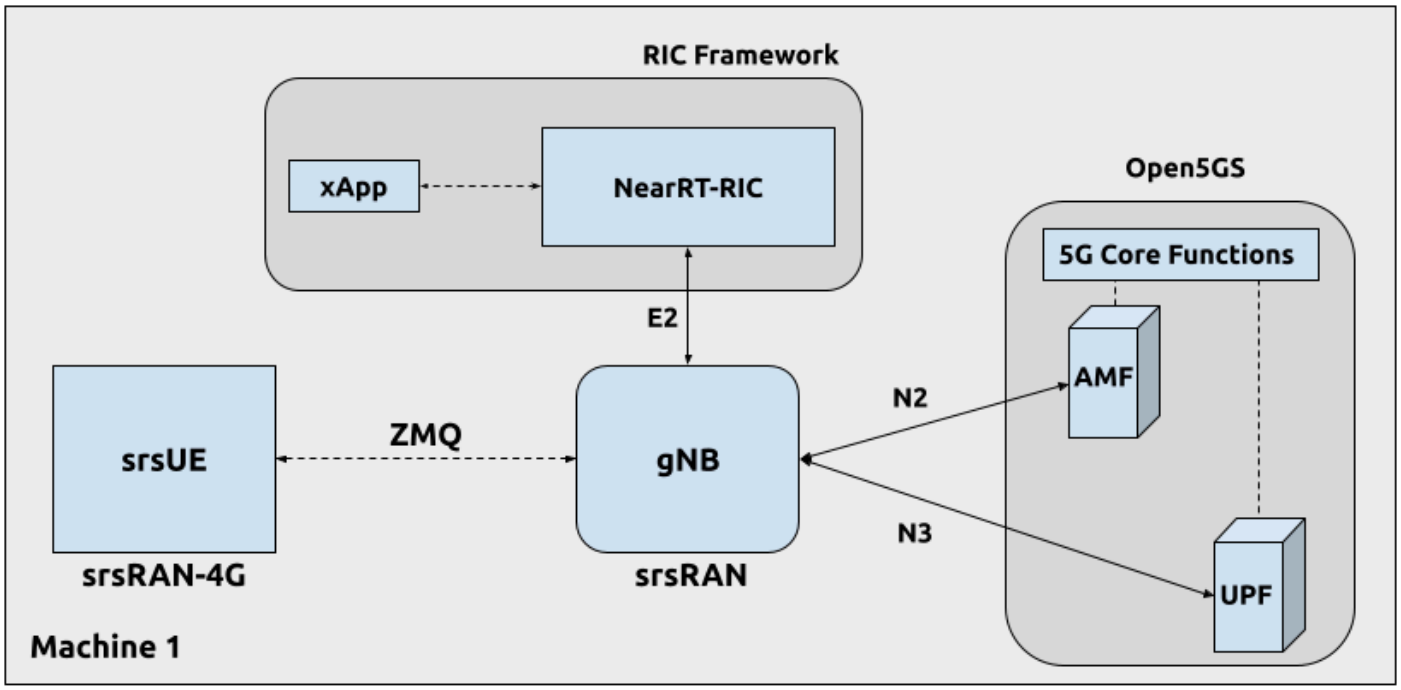
Our E2 interface implementation is based on the following O-RAN technical specifications:

- 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

> **INFO**
>
> To consult other O-RAN ALLIANCE Specifications, please click here.

Setup Overview

The following diagram presents the setup architecture used in this tutorial:



Hardware and Software Overview

For this tutorial, the following hardware and software are used:

- PC with Ubuntu 24.04.1 LTS
- OCUDU
- srsRAN UE (srsRAN 4G 23.04 or later)
- ZeroMQ
- ORAN SC RIC **or** FlexRIC
- Open5GS 5G Core
- Wireshark (Version 4.0.7 or later)

Limitations

The E2 interface implementation supports E2SM-KPM and E2SM-RC service models with the following limitations:

- E2SM-RC service model:
  - Only Control Service Style 2 is supported, shown here
- E2SM-KPM service model:
  - All Report Service Styles (1–5 and 255) are supported, as shown here
  - Minimum monitoring period is 1 s
  - 27 metrics are currently exposed; see the Supported E2 Metrics for the full list

---

Installation

For the purpose of presenting the usage of E2 interface exposed by OCUDU, we use the NearRT-RIC and an example KPM monitoring xApps from the ORAN SC RIC and FlexRIC frameworks.

> **INFO**
>
> Depending on which RIC you are using, you should select the appropriate instructions from the tabs below. The tabs are grouped throughout the tutorial, so you only need to select the appropriate instructions in this section.

O-RAN Software Community (SC) Near-Real-time RIC is a reference platform that aligns with the architecture and specifications created in the O-RAN Alliance working groups.

The ORAN SC RIC platform is an advanced software framework with a functionality that includes platform health monitoring, alarms, etc. Since it relies on Kubernetes and Helm for deployment, reliability, and scalability, its vanilla installation is quite complex, involves many steps, and requires a high level of knowledge and expertise in those frameworks.

In this tutorial, we use a minimal version of the O-RAN SC near-RT RIC (`i-release`), that can be easily deployed as a multi-container application using a single Docker command, eliminating the necessity for Kubernetes or Helm. Specifically, we will deploy ORAN SC RIC using Docker and configuration files provided in this repository.

> **INFO**
>
> It is recommended to download and install Docker Engine, instead of Docker Desktop. To learn more on why, click here.

The ORAN SC RIC deployment is performed as follows:

```
git clone https://github.com/srsran/oran-sc-ric
cd ./oran-sc-ric
docker compose up
```

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 OCUDU:

- GitHub
- Quickstart Guide

For the purpose of this tutorial, we will use a dockerized Open5GS version provided in OCUDU at `ocudu/docker`,as shown here.

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 OCUDU and srsRAN 4G (assuming you have already installed all the required dependencies).

> **INFO**
>
> If you have already built and installed srsRAN 4G and OCUDU prior to installing ZMQ and other dependencies you will have to re-build both to ensure the ZMQ drivers have been recognized correctly.

OCUDU

Install additional OCUDU dependencies by running:

```
sudo apt install libgtest-dev libyaml-cpp-dev libmbedtls-dev
```

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

```
git clone https://gitlab.com/ocudu/ocudu.git
cd ocudu
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
...
```

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.

Configuration

Here, we use ZMQ-based setup, and hence the configuration files are based on those introduced in Using srsUE tutorial.

The following config files were modified to use ZMQ-based RF driver and enable E2 interface in OCUDU 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 Using srsUE tutorial.

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.

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
e2sm_rc_enabled: true             # Enable RC 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_du_filename: /tmp/gnb_du_e2ap.pcap          # Path where the DU E2AP PCAP is stored.
e2ap_cu_cp_filename: /tmp/gnb_cu_cp_e2ap.pcap    # Path where the CU-CP E2AP PCAP is stored.
e2ap_cu_up_filename: /tmp/gnb_cu_up_e2ap.pcap    # Path where the CU-UP E2AP PCAP is stored.
```

Enable Enable RLC and DU scheduler metrics reporting that will feed E2SM_KPM service model with measurements data:

```
metrics:
layers:
enable_rlc: true                # Enable RLC metrics
enable_sched: true              # Enable DU scheduler metrics
periodicity:
du_report_period: 1000          # Set DU statistics report period to 1s
```

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

Open5GS Core

OCUDU 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 ./ocudu/docker
docker compose up 5gc
```

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

NearRT-RIC

Before starting the ORAN SC RIC platform as a multi-container application, run the following command from the `oran-sc-ric` directory:

```
cd ./oran-sc-ric
nano docker-compose.yml
```

Edit the docker-compose.yml and delete the # character from the 76th and 77th line of the file, from this:

```
#ports:
#  - "36421:36421/sctp"
```

to this:

```
ports:
- "36421:36421/sctp"
```

so the OCUDU gNB can access the ORAN SC RIC outside the docker network.

Then start the ORAN SC RIC platform as a multi-container application, run the following command from the `oran-sc-ric` directory:

```
cd ./oran-sc-ric
docker compose up
```

Docker should download and build (when running for the first time) seven containers. When all containers are successfully deployed, the following message should be displayed on the NearRT-RIC console output:

```
ric_submgr | RMR is ready now ...
```

gNB

We run gNB directly from the build folder, i.e., `./ocudu/build/apps/gnb/`, (the config file is also located there) with the following command:

```
# when running with ORAN SC RIC
sudo ./gnb -c gnb_zmq.yaml e2 --addr="10.0.2.10" --bind_addr="10.0.2.1"

# when running with FlexRIC
sudo ./gnb -c gnb_zmq.yaml e2 --addr="127.0.0.1" --bind_addr="127.0.0.1"
```

The gNB console output should be similar to:

```
--== OCUDU 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=1

==== 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:

```
ric_rtmgr_sim | 2024/04/02 11:07:39 POST /ric/v1/handles/associate-ran-to-e2t body: [{"E2T
```

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 (i.e., `/srsRAN_4G/build/srsue/`), 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.
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.

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
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
```

xApps

To start the provided example `kpm_mon_xapp.py`, run the following command from the `oran-sc-ric` directory:

```
docker compose exec python_xapp_runner ./kpm_mon_xapp.py --
metrics=DRB.UEThpDl,DRB.UEThpUl --kpm_report_style=5
```

The xApp allows subscribing with all E2SM-KPM Report Styles (i.e., 1-5) and to set the metric names. With the above parameters, the xApp should subscribe to DRB.UEThpUl and DRB.UEThpUl measurements, and display the content of received RIC_INDICATION messages. The xApp console output should be similar to:

```
RIC Indication Received from gnb_001_001_00019b for Subscription ID: 5, KPM Report Style:
E2SM_KPM RIC Indication Content:
-ColletStartTime: 2024-04-02 13:24:56
-Measurements Data:
--UE_id: 0
---granulPeriod: 1000
---Metric: DRB.UEThpDl, Value: [7]
---Metric: DRB.UEThpUl, Value: [7]
```

On start, the xApp sends a subscription request to the NearRT-RIC, therefore the following (or similar) should be displayed on the NearRT-RIC console:

```
ric_rtmgr_sim | 2024/04/02 13:35:33 POST /ric/v1/handles/xapp-subscription-handle body: {"
```

On exit, the xApp sends a subscription delete request to the NearRT-RIC and the following (or similar) should be displayed on the NearRT-RIC console:

```
ric_rtmgr_sim | 2024/04/02 13:35:40 DELETE /ric/v1/handles/xapp-subscription-handle body:
```

E2AP packet analyzer

Enable E2AP PCAP

You can enable E2AP PCAPs by following this guide.

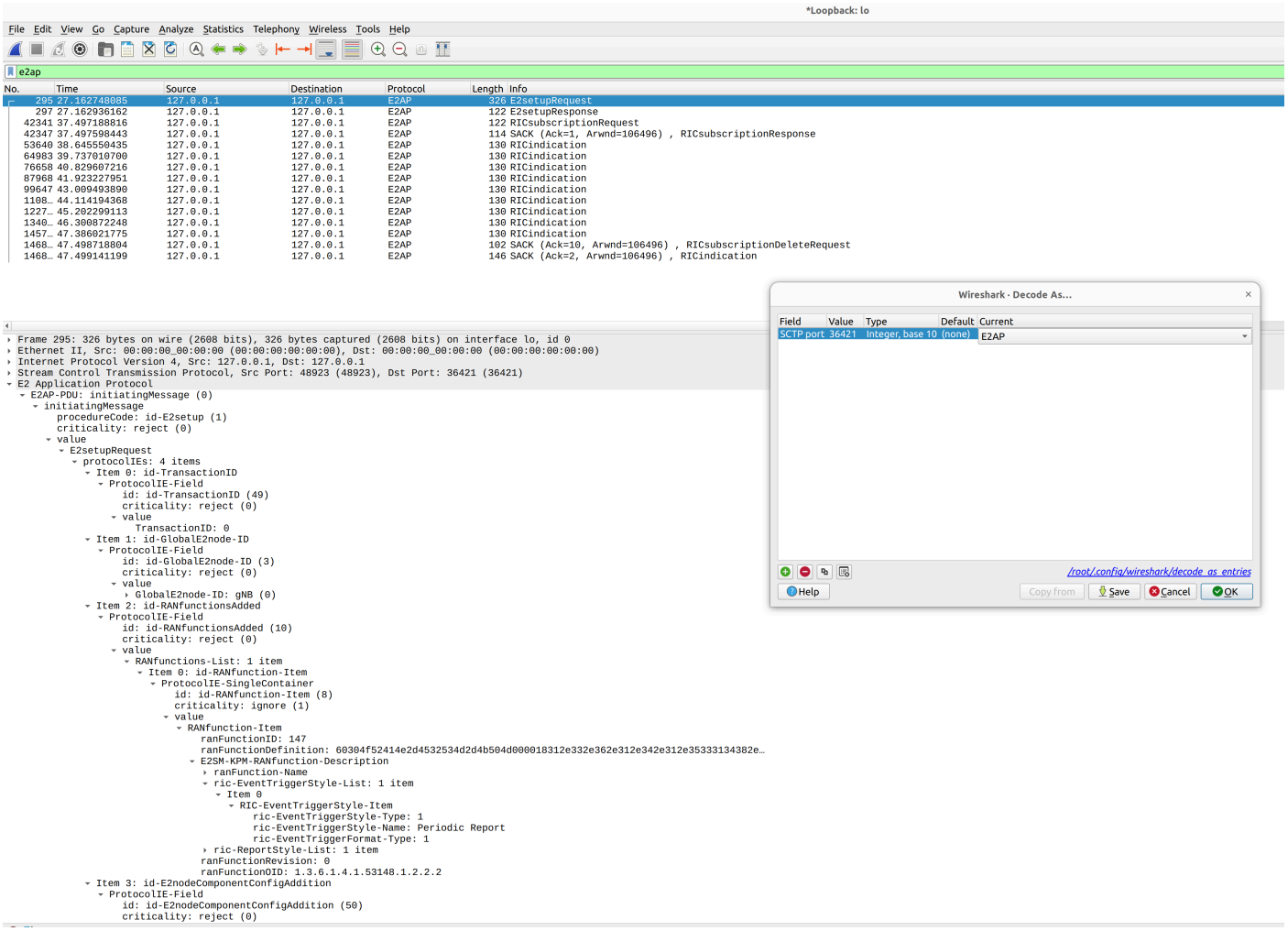
Live capture

Wireshark can be used to collect E2AP packets exchanged between E2 agent (located in OCUDU 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.



Troubleshooting

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).

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 externa

ERROR: for 5gc Cannot start service 5gc: driver failed programming external connectivity on enc
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,63bfa50ee6
OPEN5GS_IP: ${OPEN5GS_IP:-10.53.1.2}
UE_IP_BASE: ${UE_IP_BASE:-10.45.0}
DEBUG: ${DEBUG:-false}
privileged: true
ports:
- - "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}
```

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

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.

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
```

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.

Next steps

- Splitting CU and DU — run the CU and DU as separate processes over F1.

Testing intra-gNB handover

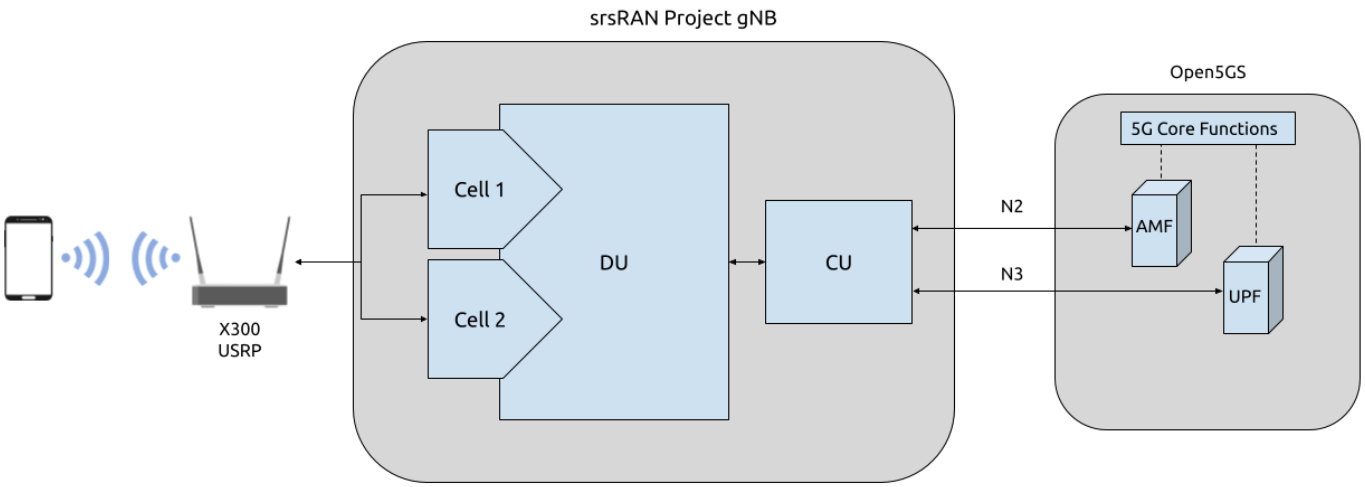
Overview

OCUDU supports all major handover flavors. In this tutorial however the focus is on intra Intra-gNB handover, i.e. handover between cells connected to the same CU-CP. This allows the UE to move between two cells, handover can be triggered manually via the command line or automatically by physically moving the UE.

In practice this is enabled by creating two cells within a DU within the OCUDU gNB binary. If you're using a USRP, this requires one has two independent RF chains. Each cell will be given an associated RF chain, which the UE then being able move between these.

The feature is of course also supported by using the 7.2 split, but is outside the scope of this tutorial.

The following diagram presents the setup architecture:



Hardware and Software Overview

For this tutorial, the following hardware and software are used:

- PC with, e.g. Ubuntu 24.04 LTS
- OCUDU
- Ettus Research X310 USRP (connected over 10 GigE)
- Open5GS 5G Core (running in docker)
- COTS UE (Fairphone 5)

OCUDU

If you have not already done so, install the latest version of OCUDU and all of its dependencies. This is outlined in the Installation Guide.

X310 USRP

Handover requires the USRP being used to have a dual channel RF-frontend with independent RF chains. As a result, this tutorial uses an X310 as the RF-frontend.

Not all dual channel USRPs have independent RF chains, this means that, for example, a B200-series USRP is not suitable for this use case.

Open5GS

For the purpose of this tutorial, we will use a dockerized Open5GS version provided in OCUDU at `ocudu/docker` as the 5G Core, as shown here.

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 OCUDU:

- GitHub
- Quickstart Guide

COTS UE

A 5G SA capable COTS UE is used for this tutorial, specifically the Fairphone 5. A detailed list of COTS UEs that have been tested with OCUD can be found here.

For more information on connecting a COTS UEs to OCUDU, see the full tutorial.

Configuration

To configure OCUDU to enable handover between cells, the following steps must be taken:

- Add the two cells to the `cells` list
- Update the `cu_cp` `mobility` options

A sample configuration file can be downloaded here: handover_x310.yml

A detailed breakdown of the changes to the configuration is as follows.

A `cells` list is created containing two cells. Each is given a different Physical Cell ID, and PRACH root sequence index.

```
cells:
-
# Cell 1
pci: 1                              # Set the Physcial Cell ID
prach:
prach_root_sequence_index: 0        # Set the PRACH root sequence index
-
# Cell 2
pci: 2                              # Set the Physcial Cell ID
prach:
prach_root_sequence_index: 64       # Set the PRACH root sequence index
```

The cells and mobility information must also be added to the `cu_cp`, under the `mobility` options. Firstly the `cells` list needs to be updated with each of the cells, the `report_configs` must also be updated with the necessary information for each of the cells and the conditions for triggering handover. As usual, the `amf` is also configured within the `cu_cp`.

```
cu_cp:
amf:
addr: 10.12.1.105                                   # The address or hostname of the
AMF.
bind_addr: 10.12.1.148                              # A local IP that the gNB binds
to for traffic from the AMF.
supported_tracking_areas:                           # Configure the TA associated
with the CU-CP
- tac: 7                                            # Tracking area code
plmn_list:                                        # List of PLMNs associated with
the tracking area
- plmn: "00101"                                   # PLMN ID
tai_slice_support_list:                          # List of slices supported by the
tracking area
- sst: 1                                         # Slice/Service Type
mobility:
trigger_handover_from_measurements: true            # Set the CU-CP to trigger
handover when neighbor cell measurements arrive
cells:                                              # List of cells available for
handover known to the cu-cp
- nr_cell_id: 0x19B0                                # Cell ID for cell 1
periodic_report_cfg_id: 1                         # Set the periodic report
configuration ID for this cell
ncells:                                           # Neighbor cell(s) available for
handover
- nr_cell_id: 0x19B1                              # Cell ID of neighbor cell
available for handover
report_configs: [ 2 ]                            # Report configurations to configure
for this neighbor cell
- nr_cell_id: 0x19B1                                # Cell ID for cell 2
periodic_report_cfg_id: 1                         # Set the periodic report
configuration ID for this cell
ncells:                                           # Neighbor cell(s) available for
handover
- nr_cell_id: 0x19B0                              # Cell ID of neighbor cell
available for handover
report_configs: [ 2 ]                            # Report configurations to configure
for this neighbor cell
report_configs:                                     # Sets the report configuration
```

```
for triggering handover
- report_cfg_id: 1                                    # Report config ID 1
report_type: periodical                              # Sets the report type as
periodical
report_interval_ms: 480                              # Sets to report every 480ms
- report_cfg_id: 2                                    # Report config ID 2
report_type: event_triggered                         # Sets the report type as event
triggered
event_triggered_report_type: a3                      # Sets the event triggered report
type as A3
meas_trigger_quantity: rsrp                          # Sets the measurement trigger
quantity as rsrp
meas_trigger_quantity_offset_db: 3                   # Sets the measurement trigger
quantity offset to 3dB
hysteresis_db: 0                                     # Sets the hysteresis to 0dB
time_to_trigger_ms: 100                              # Sets the time to trigger to
100ms
```

A3 events are defined as events when intra-frequency handover should be triggered. In the above configuration the conditions under which such an event should be triggered are set as when the neighbor cell's RSRP measurement is 1.5 dB better than serving cell. The hysteresis is set to 0, which means that handover is triggered when an offset exactly 1.5 dB is met. Once these conditions are met, the UE will handover between the cells. This is true for handover in each direction.

Connecting the COTS UE

Connecting the COTS UE to the network follows the same steps outline in this tutorial.

Triggering Handover

Handover can be triggered in two ways:

- Adjusting the Tx gain from the console while the gNB is running
- Physically moving the UE from left to right away from the primary cell and towards the secondary cell.

From Console

The Tx gain of each cell can be manually controlled during run time from the console.

While the gNB is running, you can dynamically adjust the Tx gain using the following command:

```
tx_gain <port_id> <gain_dB>
```

> **INFO**
>
> The Rx gain can also be set the same way, using `rx_gain <port_id> <gain_dB>`. To trigger handover it is only necessary to modify the Tx gain.

In this example, the gain of the cell with PCI 1 will be changed (this corresponds to the output of RF0 on the USRP), while the gain of the cell with PCI 2 will be fixed (this corresponds to the output of RF1 on the USRP). Decreasing the gain to 10 dB, will cause the UE to move to the PCI 2, increasing the gain back to 30 dB will cause the UE to move back to the cell with PCI 1.

The gain can be adjusted using the following command:

```
tx_gain 0 10
```

Handover will then be triggered, this can be seen in the console as the PCI of the serving cell changes:

```
|--------------------DL---------------------|-------------------------UL--------------------
----------
pci rnti | cqi  ri  mcs  brate   ok  nok  (%)  dl_bs | pusch  rsrp  mcs  brate   ok  nok
(%)    bsr    ta  phr
1 4601 |  15   1   28   2.4M  178    0   0%      0 |  27.4 -40.4   28   214k   50    0   0%
0   0us   28
1 4601 |  15   1   28   2.5M  180    0   0%      0 |  27.4 -40.6   28   214k   50    0   0%
0   0us   28

tx_gain 0 10
Tx gain set to 10.0 dB for port 0.

2 5601 |  15   1   26   1.2M   81    2   2%      0 |  31.2 -35.7   27   100k   24    0   0%
0   0us   27
2 5601 |  15   1   27   2.4M  175    0   0%   1.5k |  30.8 -37.2   28   214k   50    0   0%
0   0us   27
2 5601 |  15   1   28   2.4M  177    0   0%      0 |  30.0 -37.4   28   214k   50    0   0%
0   0us   27
2 5601 |  15   1   28   2.4M  178    0   0%      0 |  30.7 -37.4   28   214k   50    0   0%
0   0us   27
```

To get the UE to move back to the original cell the following command can be used:

```
tx_gain 0 30
```

With the following output confirming the handover back to the original cell:

```
|--------------------DL---------------------|-------------------------UL--------------------
----------
pci rnti | cqi  ri  mcs  brate   ok  nok  (%)  dl_bs | pusch  rsrp  mcs  brate   ok  nok
(%)    bsr    ta  phr
```

```
2 5601 |  15   1   28   2.5M  185    0   0%      0 |  29.6 -37.9   28   214k   50    0   0%
0   0us   27
2 5601 |  15   1   28   2.5M  179    0   0%      0 |  28.6 -39.1   28   215k   50    0   0%
0   0us   27
2 5601 |  15   1   28   2.4M  182    0   0%      0 |  30.4 -37.9   28   214k   50    0   0%
0   0us   27
2 5601 |  15   1   28   2.5M  185    0   0%   1.5k |  30.6 -37.4   28   214k   50    0   0%
0   0us   27
2 5601 |  15   1   28   2.5M  178    0   0%      0 |  30.3 -36.3   28   214k   50    0   0%
0   0us   27

tx_gain 0 30
Tx gain set to 30.0 dB for port 0.

|--------------------DL---------------------|-------------------------UL--------------------
----------
1 4602 |  15   1   26   217k   15    2  11%      0 |  10.9 -54.0   24   4.5k    2    3  60%
0   0us   38
1 4602 |  15   1   27   2.5M  174    0   0%      0 |  16.3 -50.0   23   208k   49   18  26%
0   0us   38
1 4602 |  15   1   27   2.4M  172    3   1%      0 |  15.0 -51.3   18   214k   50    0   0%
0   0us   38
1 4602 |  15   1   27   2.4M  173    0   0%   1.5k |  15.1 -51.3   19   214k   50    0   0%
0   0us   38
1 4602 |  15   1   28   2.5M  183    0   0%      3 |  14.5 -51.7   18   214k   50    0   0%
0   0us   38
```

This can then be repeated as desired to trigger handover between the cells when ever it is required based on the needs of your use case.

Physically

The following image shows the UE moving up and down along the Y-axis, this causes the UE to handover between cells as the signal strength of the serving cell decreases as the UE moves away from it. The inverse happens to the neighbor cell, the signal strength increase as the UE moves into its serving area. This triggers an A3 event and causes the UE to handover between cells once the conditions defined in the configuration file are met. In this case, once the RSRP of the neighbor cell has a 1.5 dB offset to the current serving cell.

Physically, this translates to moving the UE left-to-right between the antennas of the USRP, where PCI 1 corresponds to the cell associated with RF0 and PCI 2 corresponds to the cell associated with RF1.

As handover is triggered, it can be seen in the console as the PCI of the serving cell changing:

```
|--------------------DL---------------------|-------------------------UL--------------------
----------
pci rnti | cqi  ri  mcs  brate   ok  nok  (%)  dl_bs | pusch  rsrp  mcs  brate   ok  nok
(%)    bsr    ta  phr
1 4602 |  15   1   28    44M 1400    0   0%  6.14M |  32.5 -16.3   28   214k   50    0   0%
0   0us   25
1 4602 |  15   1   28    44M 1400    0   0%  6.14M |  31.9 -16.2   28   214k   50    0   0%
0   0us   25
1 4602 |  15   1   28    44M 1371   29   2%  6.14M |  31.8 -15.6   28   214k   50    0   0%
```

```
0   0us   18
2 5603 |  15   1   24    30M 1178   26   2%  6.15M |  33.0 -11.8   27   179k   43    0   0%
0   0us   23
2 5603 |  15   1   26    41M 1400    0   0%  6.14M |  33.1 -11.9   28   214k   50    0   0%
0   0us   23
2 5603 |  15   1   28    44M 1400    0   0%  6.14M |  32.6 -12.0   28   214k   50    0   0%
0   0us   23
```

Moving the UE back across to the area covered by Cell 1 (PCI 1), the following can be seen in the console output:

```
|--------------------DL---------------------|-------------------------UL----------------------
----------
pci rnti | cqi  ri  mcs  brate   ok  nok  (%)  dl_bs | pusch  rsrp  mcs  brate   ok  nok
(%)    bsr    ta  phr
2 5603 |  15   1   28    44M 1400    0   0%  6.14M |  32.8  -9.0   28   214k   50    0   0%
0   0us   20
2 5603 |  15   1   28    44M 1400    0   0%  6.14M |  33.3  -9.5   28   214k   50    0   0%
0   0us   20
2 5603 |  15   1   28    44M 1400    0   0%  6.14M |  32.5  -8.9   28   214k   50    0   0%
0   0us   19
1 4604 |  15   1   22   2.7M  128   41  24%  2.21M |  29.4  -7.9   25    18k    5    0   0%
0   0us   17
1 4604 |  15   1   21    29M 1400    0   0%  6.15M |  34.6 -14.6   28   215k   51    0   0%
0   0us   24
1 4604 |  15   1   24    36M 1400    0   0%  6.14M |  34.5 -14.5   28   214k   50    0   0%
0   0us   24
```

On some occasions, the UE may “ping-pong” between cells, if it sits in the area where the cell-coverage overlaps, this behavior is normal and can be rectified by moving the UE further into the area covered by either cell. This may also be mitigated by configuring a hysteresis value.

Grafana Output

Using the Grafana GUI the handover process, and it’s impact on certain metrics, can be clearly observed. The following image shows an example scenario where a UE is moving between two cells. In this scenario the UE is being physically moved between the cells to trigger handover.

In the above image the traffic is being generated between the UE and gNB using iPerf.

Next steps

- Tuning performance — tune the host for real-time performance.

Enabling 5G NTN for satellite deployments

Overview

This tutorial demonstrates how to configure and run a 5G NR SA Non-Terrestrial Network (NTN) using OCUDU and an Amarisoft UE. NTN is a network deployment where communication between the gNB and UE is relayed via non-terrestrial components, such as satellites.

Deploying NTN introduces unique challenges due to the different link dynamics and characteristics compared to traditional terrestrial networks. To address these challenges, 3GPP introduced several enhancements and features in the 5G NR specifications (Release 17). Key features and enhancements include:

- **Frequency bands and spectrum allocation:** new frequency bands suitable for satellite communication have been standardized to ensure high-capacity links.
- **Timing adjustments and delay management:** mechanisms have been introduced to handle the long and variable propagation delays associated with satellite communication.
- **Doppler shift compensation:** the relative movement of satellites, which can reach speeds of up to 27,000 km/h, introduces non-negligible Doppler shifts that must be compensated for to maintain reliable communication links.

In 3GPP NTN, the UE is responsible for managing long propagation delays and compensating for Doppler frequency shifts. To facilitate this, the gNB broadcasts a new SIB19 block containing NTN-related information, including the current position of the satellite. The UE uses this information to compute the current delay and Doppler frequency and adjust its transmission accordingly.

For more information, you can read the following documents:

- 3GPP TS 38.331 version 17.1.0 Release 17
- 3GPP TS 38.300 version 17.1.0 Release 17

OCUDU supports GEO, MEO, and LEO NTN scenarios. This tutorial covers GEO and LEO.

Choosing a path

Both paths use the same Amarisoft UE and Open5GS core, and the steps are almost identical. They differ only in how the satellite channel is emulated: an external GNU Radio emulator, or the Amarisoft UE's built-in NTN channel emulator.

- **External GNU Radio emulator (GEO):** use this for the simplest end-to-end GEO test. An external GNU Radio flowgraph applies a fixed link delay over ZMQ, with static ephemeris. GEO only.

- **Built-in emulator (GEO/LEO):** use this for realistic GEO or LEO passes. The Amarisoft UE's built-in NTN emulator applies the link delay and Doppler internally from a real satellite TLE, and the gNB ephemeris is refreshed live during a LEO pass.

	External GNU Radio emulator (GEO)	Built-in emulator (GEO/LEO)
Best for	Simplest GEO smoke test	Realistic GEO and LEO passes
Orbits	GEO	GEO and LEO
Channel effects	Fixed delay	Variable delay and Doppler
Ephemeris source	Static, hand-written	Generated from a real TLE
Live SIB19 updates	Not used	Yes (needed for LEO)
Extra tooling	GNU Radio Companion	Amarisoft licence for the built-in emulator; Python 3 for the optional helper scripts

HOW TO READ THIS TUTORIAL

Most of this tutorial is common to both paths. Wherever the paths differ, the section has an **External GNU Radio emulator** and a **Built-in emulator** tab. Picking a tab applies your choice across the whole page, so you only choose once.

Key concepts

The following parameters appear in the gNB NTN configuration. Each one is documented in full in the Configuration Reference.

- **Ephemeris:** the satellite position and velocity. OCUDU can broadcast the ephemeris either as an ECEF state vector (`ephemeris_info_ecef`) or as orbital parameters (`ephemeris_orbital`).
- **Epoch time:** the reference time to which the ephemeris and timing parameters apply.
- **Cell-specific k-offset:** a scheduling offset (`cell_specific_koffset`) that accounts for the round-trip time of the NTN link.
- **Timing advance (TA):** NTN separates the common TA (the delay to a reference point) from the UE-specific TA. The TA scheduler is tuned so that a new TA command is not issued before the previous one has been applied by the UE.
- **Feeder link and service link:** the service link connects the UE to the satellite, while the feeder link connects the satellite to the ground gateway. When the gNB is on the ground rather than on the satellite, feeder link Doppler compensation applies.

- **GEO compared with LEO:** a GEO link behaves as a large, effectively constant delay. A LEO satellite moves quickly across the sky, so its delay and Doppler change continuously during a pass and the ephemeris broadcast in SIB19 has to be refreshed while the UE is connected.

Step 1: Prerequisites and installation

Hardware and software

For this tutorial, the following hardware and software are used:

- PC with Ubuntu 22.04.1 LTS
- OCUDU (26.04 or later) built with ZeroMQ support
- Amarisoft UE with NTN support (2023-12-15 or later)
- Open5GS 5G Core
- Docker and Docker Compose, to run the Open5GS core
- ZeroMQ

The Amarisoft UE simulator (AmariUE) is a commercial solution for functional and performance testing of 5G networks; as a compliant LTE, NB-IOT, and NR UE it can simulate multiple UEs concurrently.

Open5GS is an open source 5G Core. This tutorial runs it in Docker using the Compose file in the OCUDU source tree at `ocudu/docker` (from the clone in the build step below). The image is already configured for OCUDU, and its subscriber database is pre-populated with the Amarisoft UE test SIM, so it needs no manual setup. It exposes the AMF at `10.53.1.2`, which is the address the gNB connects to in its `cu_cp.amf` section.

Install ZeroMQ

On Ubuntu, the ZeroMQ development libraries can be installed with:

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

Build OCUDU

Compile OCUDU with ZeroMQ enabled (assuming the other dependencies are already installed). ZeroMQ is activated by the `-DENABLE_EXPORT=ON -DENABLE_ZEROMQ=ON` flags on the `cmake` command:

```
git clone https://gitlab.com/ocudu/ocudu.git
cd ocudu
mkdir build
cd build
```

```
cmake ../ -DENABLE_EXPORT=ON -DENABLE_ZEROMQ=ON
make -j`nproc`
```

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

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

> **INFO**
>
> If you built OCUDU before installing ZMQ, you will have to re-build it so the ZMQ drivers are recognized correctly.

Install the Amarisoft UE and the ZeroMQ TRX driver

Download and install the Amarisoft UE (this tutorial uses version 2023-12-15 or later).

Interfacing the Amarisoft UE with OCUDU requires a custom TRX driver implemented by SRS, found in the OCUDU source at `ocudu/utils/trx_ocudu`. The Amarisoft UE release folder (`amarisoft.2026-03-13.tar.gz`) contains a `trx_uhd-2026-03-13.tar.gz` file; uncompress both before proceeding.

Compile the driver from `ocudu/build`:

```
cmake ../ -DENABLE_EXPORT=TRUE -DENABLE_ZEROMQ=TRUE -DENABLE_TRX_DRIVER=TRUE -
DTRX_DRIVER_DIR=<PATH TO trx_uhd-2026-03-13>
make trx_ocudu_test
ctest -R trx_ocudu_test
```

Make sure CMake finds `trx_driver.h` in the specified folder:

```
-- Found trx_driver.h in TRX_DRIVER_DIR=/home/user/amarisoft/2026-03-13/trx_uhd-2026-03-
13/trx_driver.h
```

Then create a symbolic link so the UE application can load the driver. From the Amarisoft UE build folder:

```
ln -s ocudu/build/utils/trx_ocudu/libtrx_ocudu.so trx_ocudu.so
```

Install the path-specific tooling

Install GNU Radio Companion, which runs the GEO NTN channel emulator:

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

Step 2: Configuration

The tutorial ships prepared configuration files so you can avoid errors while editing configs manually. The description of any parameter not covered here is available in the Configuration Reference. Details of the ZMQ-based setup are explained in the Amarisoft UE tutorial. The Amarisoft UE also requires the `ue-ifup` script that ships with it, located in the `config` folder of the UE application.

Download the prepared files for your path from the links in the sections below. Each config passed to the gNB with `-c` is read from the current working directory, so keep the files there or pass a full path (for example, `-c ~/configs/ocudu_gnb.yml`). Put the Amarisoft UE config in the Amarisoft UE directory, alongside the `ue-ifup` script.

gNB RF driver

The gNB uses the ZMQ-based RF driver to exchange samples over virtual radios. The endpoints differ between the two paths.

The gNB exchanges samples with the GNU Radio emulator:

```
ru_sdr:
device_driver: zmq
device_args: tx_port=tcp://127.0.0.1:2000,rx_port=tcp://127.0.0.1:2001
srate: 5.76
```

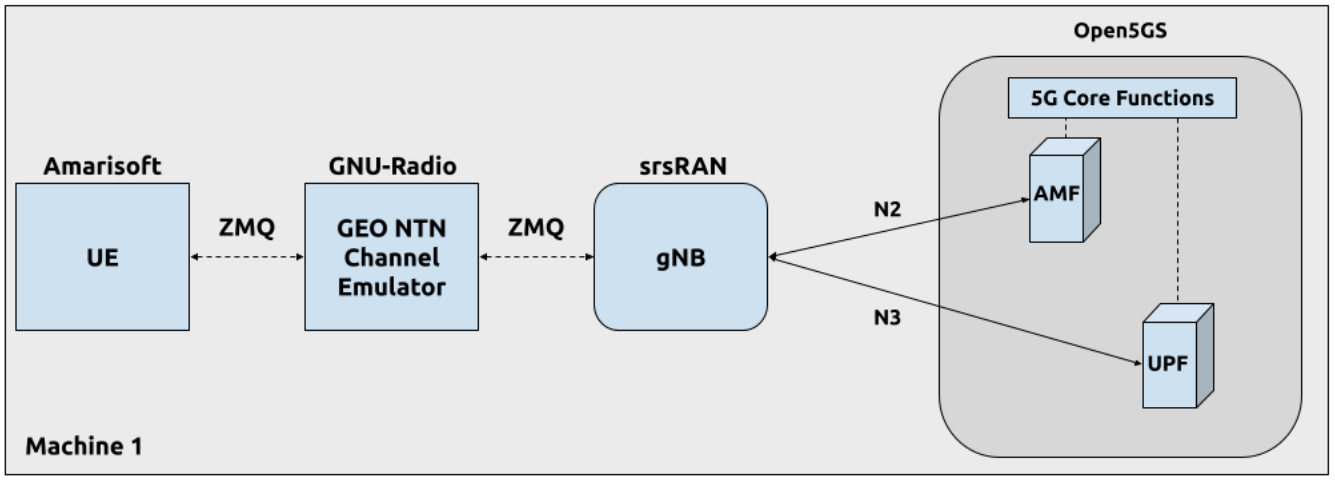
gNB NTN configuration

Enabling NTN features in the gNB requires the following:

- using one of the available bands (here `band: 256`) and ARFCN (DL and SSB)
- disabling Msg3 HARQ retransmissions (`max_msg3_harq_retx: 0`)
- using Preamble Format 1 to improve timing robustness (here `prach_config_index: 31`)
- adapting periods and timers to match the NTN link RTT
- enabling transmission of SIB19

- adding an `ntn` config section with the parameters used to configure the gNB in NTN mode and to fill SIB19

The following diagram shows the components in this setup:



The `cell_cfg` section for the GEO scenario is as follows:

```
cell_cfg:
dl_arfcn: 437000                  # ARFCN of the downlink carrier (center frequency).
band: 256                         # Use NTN band.
channel_bandwidth_MHz: 5          # Bandwidth 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 core
configuration).
pdsch:
nof_harqs: 16                   # Sets the number of Downlink HARQ processes.
max_nof_harq_retxs: 0           # Disable HARQ retransmissions.
prach:
prach_config_index: 31          # Use Preamble Format 1 to improve the timing
robustness.
max_msg3_harq_retx: 0           # Disable Msg3 HARQ retransmissions.
sib:
t300: 2000                      # Extend the RRC Connection Establishment timer in ms.
t301: 2000                      # Extend the RRC Connection Re-establishment timer in
ms.
t311: 3000                      # Extend the RRC Connection Re-establishment procedure
timer in ms.
t319: 2000                      # Extend the RRC Connection Resume timer in ms.
si_window_length: 40            # Set SI Window Length.
si_sched_info:
- si_period: 16               # Set SIB2 period.
sib_mapping: 2              # Enable SIB2.
- si_period: 16               # Set SIB19 period.
sib_mapping: 19             # Enable SIB19.
si_window_position: 2       # Set SIB19 position.
pucch:
sr_period_ms: 80               # Set Scheduling Request period.
```

```
csi:
csi_rs_period: 80               # Set CSI-RS report period.
```

The `ntn` section holds the static satellite ephemeris:

```
cell_cfg:
ntn:
cell_specific_koffset:  240       # Cell-specific k-offset.
ta_common:  0                     # TA common offset.
ephemeris_info_ecef:              # Satellite ephemeris in position and velocity state
vector format.
pos_x:  -28105880
pos_y:  31509747
pos_z:  -1691895
vel_x:  34
vel_y:  9
vel_z:  -385
```

Finally, the Timing Advance (TA) scheduler must account for the large propagation delays in GEO NTN. In particular, avoid measuring and issuing new TA commands before the previous command has been applied by the UE:

```
cell_cfg:
ta:
ta_cmd_offset_threshold: 1                # Threshold above which a Timing Advance
Command is triggered.
ta_measurement_slot_period: 40            # Periodicity, in slots, over which the new
TA command is computed.
ta_measurement_slot_prohibit_period: 250  # Delay, in slots, between issuing a TA_CMD
and restarting measurements.
ta_target: 0                              # Timing advance target in units of TA.
```

The gnb_zmq.yml file contains the basic (generic) gNB config, while the NTN-related parameters are defined in a separate geo_ntn.yml file.

Amarisoft UE configuration

Enabling NTN in the UE requires the following (the exact keys differ between the two paths):

- matching the gNB band and ARFCN (here `band: 256`, `dl_nr_arfcn: 437000`)
- enabling NTN operation, and for the built-in path the channel simulator
- setting the UE ground position, used to compute the link delay and Doppler
- pointing the ZMQ `rf_driver` at the gNB

The UE connects to the GNU Radio channel emulator over ZMQ, so its `rf_driver` section is:

```
rf_driver: {
/* OCUDU zmq RF device */
name: "ocudu",
log_level: "info",
tx_port0:  "tcp://*:2101",
rx_port0:  "tcp://localhost:2100",
},
```

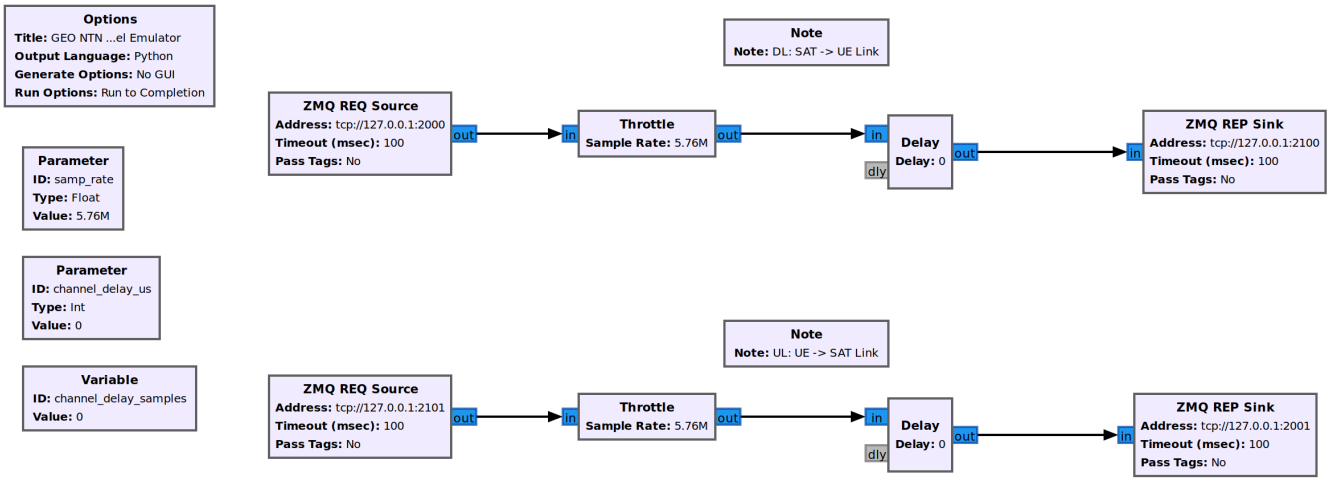
The `cell_groups` section is as follows:

```
cell_groups: [{
group_type: "nr",
multi_ue: false,
cells: [{
rf_port: 0,
bandwidth: 5,
sample_rate: 5.76,
band: 256,
dl_nr_arfcn: 437000,
ssb_nr_arfcn: 437090,
ssb_subcarrier_spacing: 15,
subcarrier_spacing: 15,
n_antenna_dl: 1,
n_antenna_ul: 1,
ntn: true,
ntn_ground_position: {
latitude: -2.2970186,
longitude: 131.7327201,
altitude: 1
},
}],
```

The complete file is provided as ue-nr-ntn-geo.cfg.

Channel emulator

GNU Radio Companion runs the channel emulator, which delays signal samples between the gNB and UE over ZMQ. Download it as a flow-graph or a Python script:



The upper graph handles downlink samples and the lower graph handles uplink samples: each signal is received from the gNB (UE) over a ZMQ socket, delayed by the NTN link delay, and forwarded to the UE (gNB). It introduces only the link delay, which is sufficient to demonstrate NTN operation. Doppler shift, delay variation, and path loss are not modelled: in the GEO scenario the Doppler shift is negligible, the slow delay variation is handled by the gNB using TA commands, and path loss does not affect the NTN protocol.

Step 3: Run the network

Start the components in the order shown for your path, each in its own terminal.

1. Start the Open5GS core (`--build` is only needed the first time):

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

2. Start the GEO NTN channel emulator using the pre-generated script:

```
python3 ./geo_ntn_channel_emulator.py --channel-delay-us=119720
```

The delay value of 119720 us matches the link delay between the GEO satellite (position in `ephemeris_info_ecef`) and the UE (coordinates in `ntn_ground_position`).

3. Start the gNB from the OCUDU source root, with the config files in the working directory:

```
sudo ./build/apps/gnb/gnb -c gnb_zmq.yml -c geo_ntn.yml cell_cfg ntn --epoch_timestamp
$(date -u +"%Y-%m-%dT%H:%M:%S.%3N")
```

4. Start the Amarisoft UE:

```
sudo ./lteue ue-nr-ntn-geo.cfg
```

Verifying the connection

Once the gNB starts, its console shows the cell parameters and an AMF connection attempt:

```
--== OCUDU gNB (commit d9a4b15) ==--

Connecting to AMF on 10.53.1.2:38412
Available radio types: zmq.
Cell pci=1, bw=5 MHz, 1T1R, dl_arfcn=437000 (n256), dl_freq=2185.0 MHz, dl_ssb_arfcn=437090, ul

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

The `Connecting to AMF` message indicates the gNB initiated a connection to the core. On success, the Open5GS console logs a matching `gNB-N2 accepted` entry.

The `ue-ifup` script must be in the same directory as the UE and executable (`chmod +x ./ue-ifup`) so the simulator can create the UE network namespace. Once samples flow, the UE detects the cell (`Cell 0: SIB found`) and starts the attach procedure. Verify the connection with the `ue` command:

```
(ue) ue
# UE_ID CL RNTI RRC_STATE EMM_STATE #ERAB IP_ADDR
NR 0 1 0 4601 running registered 1 10.45.1.2
```

The connection has succeeded once the UE has an IP (here: `10.45.1.2`).

Step 4: Test the network

Routing configuration

Before you can ping the UE, add a route to the UE on the **host machine** (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 with `route -n`. It should contain entries similar to the following (the `Iface` names might differ):

```
```

```
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
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
```

Ping

Test the connection in the uplink direction:

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

Or the downlink direction (take the UE IP from the UE console, as it can change on reconnect):

```
ping 10.45.1.2
```

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=762 ms
64 bytes from 10.45.1.1: icmp_seq=2 ttl=64 time=723 ms

--- 10.45.1.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss
rtt min/avg/max/mdev = 723.0/742.5/762.0/19.5 ms
```

> **NOTE**
>
> The round-trip latency depends on the orbit. A GEO link shows a large, roughly constant delay (several hundred milliseconds), whereas a LEO link shows a lower delay that varies through the pass as the satellite rises, culminates, and sets.

Iperf

Test throughput in the uplink direction:

```
sudo ip netns exec ue1 iperf -c 10.45.1.1 -i 1 -t 100 -u -b25M
```

Or the downlink direction:

```
iperf -c 10.45.1.2 -i 1 -t 1000 -u -b 25M
```

Example gNB console trace when running iperf:

```
|--------------------DL---------------------|-------------------------UL-----------------------
pci rnti | cqi ri mcs brate ok nok (%) dl_bs | pusch rsrp ri mcs brate ok nok (%) bsr ta phr
1 4601 | 15 1.0 27 21M 988 0 0% 3.15M | 51.9 -8.9 1 27 20.1M 988 0 0% 700k -151n 23
1 4601 | 15 1.0 27 21M 988 0 0% 3.83M | 51.9 -8.9 1 27 20.1M 988 0 0% 700k -152n 23
```

Troubleshooting

Running gNB and Amarisoft UE on separate machines

When running the gNB and the Amarisoft UE on two separate host machines (for example, using an Amarisoft CallBox), you need to adapt the IP addresses used as the TX and RX endpoints in the ZMQ-based RF drivers.

If the NTN channel emulator runs on the same PC as the gNB:

1. The `ru_sdr` section in the gNB config stays unchanged (`tx_port=tcp://127.0.0.1:2000,rx_port=tcp://127.0.0.1:2001`).
2. In the GNU Radio channel emulator, change the DL transmit endpoint from `tcp://127.0.0.1:2100` to `tcp://0.0.0.0:2100`, and the UL receive endpoint from `tcp://127.0.0.1:2101` to `tcp://$UE_IP:2101`.
3. In the Amarisoft UE `rf_driver`, set `tx_port0: "tcp://*:2101"` and `rx_port0: "tcp://$GNB_IP:2100"`.

Limitations

- **ZeroMQ-based setup only:** the channel emulator uses ZMQ sockets to transfer signal samples. Running over the air would require a real RF NTN channel emulator.
- **GEO scenario only:** the emulator introduces only a fixed delay. LEO requires an emulator that also simulates delay variation and Doppler; use the Amarisoft path for that.
- **Satellite-based gNB placement:** this scenario assumes the gNB is on the satellite, so there is no feeder link.
- **Disabled HARQ retransmissions:** HARQ retransmissions are disabled. This is a valid option as specified in the NTN standards.

Next steps

- Connecting an Amarisoft UE — scale up to multi-UE testing with a UE simulator.

Configuring DPDK for the Open Fronthaul

> **INFO**
>
> 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.

Overview

This tutorial outlines how to configure DPDK for use with OCUDU for front-haul connectivity.

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 OCUDU, 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.

Further Reading

- DPDK Build Guide
- Linux Drivers
- Using Hugepages in a Linux Environment
- Running Hugepages
- EAL Parameters

Installing DPDK

> **INFO**
>
> 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 the most recent DPDK LTS. At the time of writing the tutorial this has been `25.11.0`. This can be installed with the following commands:

```
wget https://fast.dpdk.org/rel/dpdk-25.11.tar.xz
tar xvf dpdk-25.11.tar.xz dpdk-25.11/
cd dpdk-25.11/
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' 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.

Configuring DPDK

Configure Hugepages

DPDK requires `hugepages` to be configured to run correctly. The `dpdk-hugepages.py` helper script can be used to configure this correctly. We recommend to use 2GB of the 2G hugepages for the CU/DU running a single sector 4x2 100MHz. If you run more sectors, you will need to increase the amount of hugepages.

```
sudo ./dpdk-hugepages.py -p 1G --setup 2G
```

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=2
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-5.15.0-1082-realtime root=/dev/mapper/ubuntu--vg-ubuntu--lv quiet splash
intel_iommu=on iommu=pt hugepagesz=1G hugepages=2 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.

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`.

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 OCUDU 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].

Running OCUDU with DPDK

Once DPDK has been installed and configured you will need to create a clean build of OCUDU to enable the use of DPDK.

If you have not done so already, download the code-base with the following command:

```
git clone https://gitlab.com/ocudu/ocudu.git
```

Then build the code-base, making sure to include the correct flags when running cmake:

```
cd ocudu
mkdir build
cd build
cmake -DENABLE_DPDK=True -DASSERT_LEVEL=MINIMAL ..
make -j $(nproc)
make test -j $(nproc)
```

You can now run OCUDU as normal. If everything is running 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

--== OCUDU 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`.

Next steps

- Configuring DPDK with USRP — apply DPDK to a USRP front-end.

- Accelerating with BBDEV — offload LDPC encode and decode to a hardware accelerator.
- Tuning performance — tune the host for real-time performance.

Configuring DPDK with a USRP front-end

> **INFO**
>
> This tutorial assumes the machine being used has an Intel processor. For AMD processors, some of the commands (specifically regarding IOMMU and specific DPDK drivers) will change. The overall steps will remain mostly the same.

Overview

This tutorial outlines how to configure DPDK for use with OCUDU gNB in a USRP X310.

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 OCUDU, this can enable users to achieve higher throughput and a more stable performance with certain configurations. For instance, usecases that require users to run a 100MHz with 2x2 MIMO configuration can greatly benefit from using DPDK.

Further Reading

- DPDK Build Guide
- Linux Drivers
- Using Hugepages in a Linux Environment
- Running Hugepages
- EAL Parameters

Useful Documentation

- USRP X3x0 Series
- UHD configuration files (Location)
- Using DPDK in UHD
- Getting Started with DPDK and UHD

Dependencies

There are some specific version requirements, since we're doing a UHD+DPDK+OCUDU setup. The Ubuntu version is `24.04.2 LTS (Noble Numbat)`.

UHD+DPDK Dependencies

- UHD 3.x requires DPDK 17.11
- UHD 4.0 and 4.1 require DPDK 18.11
- UHD 4.2 to 4.7 can use any version of DPDK from 18.11 to 21.11
- UHD 4.8 to 4.9 can use any version of DPDK from 18.11 to 24.11

OCUDU+DPDK Dependencies

- DPDK version >= 22.11

Tested dependencies:

- DPDK 22.11 + UHD 4.9.0.0 + OCUDU

Installing DPDK

> **INFO**
>
> 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.

```
sudo apt install build-essential tar wget python3-pip libnuma-dev meson ninja-build python3-pyelftools
```

Following the previously stated dependencies, this tutorial uses version `22.11.0`. It can be installed with the following commands:

```
wget https://fast.dpdk.org/rel/dpdk-22.11.tar.xz
tar -xf dpdk-22.11.tar.xz
cd dpdk-22.11
meson build
```

```
ninja -C build
sudo ninja -C build 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 dpdk-devbind.py -s
```

You should see an output similar to the following:

```
Network devices using DPDK-compatible driver
============================================

Network devices using kernel driver
===================================
0000:02:00.0 'Ethernet Controller X710 for 10GbE SFP+ 1572' if=enp2s0f0np0 drv=i40e
unused=vfio-pci *Active*
0000:02:00.1 'Ethernet Controller X710 for 10GbE SFP+ 1572' if=enp2s0f1np1 drv=i40e
unused=vfio-pci *Active*
0000:57:00.0 'Ethernet Controller I226-V 125c' if=enp87s0 drv=igc unused=vfio-pci *Active*
0000:58:00.0 'Ethernet Controller I226-LM 125b' if=enp88s0 drv=igc unused=vfio-pci
0000:59:00.0 'MT7922 802.11ax PCI Express Wireless Network Adapter 0616' if=wlp89s0
drv=mt7921e unused=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 back to the DPDK documentation and that of your OS maintainers.

Installing UHD

UHD 4.9.0.0 must be compiled specifically with the DPDK driver enabled.

```
git clone https://github.com/EttusResearch/uhd.git
cd uhd/host
git checkout v4.9.0.0
mkdir build
cd build
cmake -DENABLE_DPDK=ON -DENABLE_EXAMPLES=ON -DENABLE_UTILS=ON ../
make -j$(nproc)
sudo make install
sudo ldconfig
```

For this tutorial we use FPGA Image Flavor XG, making both SFP+ Ports 10 Gigabit. To do this, first make sure your images are up to date.

```
uhd_images_downloader
```

And then load the FPGA image:

```
sudo uhd_image_loader --args="type=x300,fpga_image=XG"
```

Configuring DPDK

Configure Hugepages

DPDK requires `hugepages` to be configured to run correctly. The `dpdk-hugepages.py` helper script can be used to configure this correctly. We tested the use of 8GB of the 1G hugepages for the gNB running a 2x2 100MHz. If you run more sectors, you will need to increase the amount of hugepages.

```
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`, an example is:

```
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash intel_iommu=on iommu=pt hugepagesz=1G hugepages=8
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=/boot/vmlinuz-6.8.0-106-generic root=UUID=5bb3c555-c05a-4c69-9d3e-bd79d4cc53e3 ro
quiet splash intel_iommu=on iommu=pt hugepagesz=1G hugepages=8 default_hugepagesz=1G
vt.handoff=7

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.

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
===================================
0000:02:00.0 'Ethernet Controller X710 for 10GbE SFP+ 1572' if=enp2s0f0np0 drv=i40e
unused=vfio-pci
0000:02:00.1 'Ethernet Controller X710 for 10GbE SFP+ 1572' if=enp2s0f1np1 drv=i40e
unused=vfio-pci
0000:57:00.0 'Ethernet Controller I226-V 125c' if=enp87s0 drv=igc unused=vfio-pci *Active*
0000:58:00.0 'Ethernet Controller I226-LM 125b' if=enp88s0 drv=igc unused=vfio-pci
0000:59:00.0 'MT7922 802.11ax PCI Express Wireless Network Adapter 0616' if=wlp89s0
drv=mt7921e unused=vfio-pci
```

The network cards we want to use are `Ethernet Controller X710 for 10GbE SFP+ 1572`, port 0 and 1. Interface name `enp2s0f0np0` and `enp2s0f1np1`, bus ID `0000:02:00.0` and `0000:02:00.1`. 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 enp2s0f0np0 down
sudo ifconfig enp2s0f1np1 down
sudo dpdk-devbind.py --bind vfio-pci 0000:02:00.0 0000:02:00.1
```

Every Tx/Rx channel consumes around 6Gbps, so two 10Gbps interfaces are required for 100MHz 2x2 MIMO.

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
============================================
0000:02:00.0 'Ethernet Controller X710 for 10GbE SFP+ 1572' drv=vfio-pci unused=i40e
0000:02:00.1 'Ethernet Controller X710 for 10GbE SFP+ 1572' drv=vfio-pci unused=i40e
```

If the bind was successful, the output will show `drv=vfio-pci`.

UHD DPDK Config

Following UHD documentation a `uhd.conf` file is used to configure the interfaces UHD will use with DPDK. Furthermore, this is also needed to provide an IP address to the DPDK interface so that it can reach the USRP.

An example configuration for `uhd.conf` is:

```
[use_dpdk=1]
dpdk_mtu=9000
dpdk_corelist=0,4,6
dpdk_num_mbufs=32384
dpdk_num_desc=4096

[dpdk_mac=xx:xx:xx:xx:xx:xx]
dpdk_ipv4=192.168.40.1/24
dpdk_lcore=4

[dpdk_mac=xx:xx:xx:xx:xx:xx]
dpdk_ipv4=192.168.30.1/24
dpdk_lcore=6
```

- `dpdk_num_desc=4096` maximizes the hardware ring buffer for the X710. The `dpdk_num_mbufs` value can be decreased if facing hardware constraints, maintain power of 2 values. Consider a

`dpdk_num_mbufs` value greater than `dpdk_num_desc`.

- `dpdk_corelist`: the used CPUs are arbitrary, just be sure to use even numbers to have full leverage of CPU pairs.

- `dpdk_mac`: MAC address of the interfaces where you want to use DPDK, for this tutorial `enp2s0f0np0` and `enp2s0f1np1`.

Now that DPDK is configured, and before starting the gNB, you should make sure that USRP can be found with DPDK and that benchamrks can run clean (no lates, underflows, or overflows).

Use the `uhd_find_devices` command, you should specifie the ip address of the USRP.

```
sudo uhd_find_devices --
args="use_dpdk=1,type=x300,addr=192.168.30.2,second_addr=192.168.40.2"
```

The ouptut should be similar to this:

```
[INFO] [UHD] linux; GNU C++ version 13.3.0; Boost_108300; DPDK_22.11; UHD_4.9.0.HEAD-0-
g006d7f76
EAL: Detected CPU lcores: 20
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: Ignore mapping IO port bar(1)
EAL: Ignore mapping IO port bar(4)
EAL: Probe PCI driver: net_i40e (8086:1572) device: 0000:02:00.0 (socket -1)
EAL: Ignore mapping IO port bar(1)
EAL: Ignore mapping IO port bar(4)
EAL: Probe PCI driver: net_i40e (8086:1572) device: 0000:02:00.1 (socket -1)
TELEMETRY: No legacy callbacks, legacy socket not created
--------------------------------------------------
-- UHD Device 0
--------------------------------------------------
Device Address:
serial: 347CB97
addr: 192.168.30.2
fpga: XG
name:
product: X310
type: x300
```

INFO

Note once agian that the fpga image `XG` is specified. The default image for USRP X310 uses a 10GbE and a 1GbE SFP+ ports, while the `XG` image uses 10GB on both SFP+ ports.

Now run the benchmark to verify that UHD can sustain the data rate, in `uhd/examples`, with the duration of 60 seconds:

```
sudo ./benchmark_rate --tx_rate=184.32e6 --rx_rate=184.32e6 --tx_channels=0,1 --
rx_channels=0,1 --
args="use_dpdk=1,type=x300,addr=192.168.30.2,second_addr=192.168.40.2,master_clock_rate=184.
32e6,num_recv_frames=8000,num_send_frames=8000" --duration 60
```

A succesful benchmark rate summary should look like this:

```
Benchmark rate summary:
Num received samples: 22192032614
Num dropped samples: 0
Num overruns detected: 0
Num transmitted samples: 22150390440
Num sequence errors (Tx): 0
Num sequence errors (Rx): 0
Num underruns detected: 0
Num late commands: 0
Num timeouts (Tx):        0
Num timeouts (Rx):        0
```

Running OCUDU with DPDK

Once DPDK has been installed and configured you will need to create a clean build of OCUDU to enable the use of DPDK.

If you have not done so already, download the code-base with the following command:

```
git clone https://gitlab.com/ocudu/ocudu.git
```

Then build the code-base, making sure to include the correct flags when running cmake:

```
cd ocudu
mkdir build
cd build
cmake -DENABLE_DPDK=True -DASSERT_LEVEL=MINIMAL ..
```

An example output for the `cmake`, to verify that DPDK module is correctly found:

```
-- Checking for module 'libdpdk>=22.11'
-- Found libdpdk, version 22.11.0
```

```
-- DPDK LIBRARIES: -Wl,--as-needed-L/usr/local/lib/x86_64-linux-gnu-lrte_node-lrte_graph-lrte_p
-- DPDK INCLUDE DIRS: /usr/local/include;/usr/include/x86_64-linux-gnu;/usr/include;/usr/includ
```

Then continue the build:

```
make -j $(nproc)
make test -j $(nproc)
```

gNB configuration file

The configuration of the `ru_sdr` will require the `use_dpdk=1` flag, which will trigger the configuration file `uhd.conf`. Device discovery via DPDK is not currently implemented, so the device args *mgmt_addr*, *addr*, and *second_addr* (if applicable) must all be specified at runtime.

USRP X310 does not possess a *mgmt_addr* so it is not specified, in the case of N310 USRP use the RJ45 port.

```
ru_sdr:
device_driver: uhd
device_args: type=x300,addr=192.168.30.2,second_addr=192.168.40.2,use_dpdk=1
clock: internal
sync: internal
srate: 184.32
tx_gain: 30
rx_gain: 26
```

```
cell_cfg:
dl_arfcn: 660000 # change according to your setup
band: 77 # change according to your setup
channel_bandwidth_MHz: 100
common_scs: 30
plmn: "00101"
tac: 1
pci: 1
nof_antennas_ul: 1
nof_antennas_dl: 2
pdsch:
mcs_table: qam256
pusch:
mcs_table: qam64
```

Before running OCUDU, it is recommended to run gNB in test mode to guarantee the gNB configuration can be sustained with maximum PDSCH and PUSCH load.

Errors and warnings can be checked with the command:

```
```

```
tail -f /tmp/gnb.log | grep -E "\[E|W\]"
```

If everything is running correctly you should see the following console output:

```
--== OCUDU gNB (commit a540ec6878) ==--

Lower PHY in triple executor mode.
Available radio types: uhd and zmq.
[INFO] [UHD] linux; GNU C++ version 13.3.0; Boost_108300; DPDK_22.11; UHD_4.9.0.HEAD-0-
g006d7f76
[INFO] [LOGGING] Fastpath logging disabled at runtime.
Making USRP object with args 'type=x300,addr=192.168.30.2,second_addr=192.168.40.2,use_dpdk=
1,master_clock_rate=184.32e6,send_frame_size=8000,recv_frame_size=8000'
EAL: Detected CPU lcores: 20
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: Ignore mapping IO port bar(1)
EAL: Ignore mapping IO port bar(4)
EAL: Probe PCI driver: net_i40e (8086:1572) device: 0000:02:00.0 (socket -1)
EAL: Ignore mapping IO port bar(1)
EAL: Ignore mapping IO port bar(4)
EAL: Probe PCI driver: net_i40e (8086:1572) device: 0000:02:00.1 (socket -1)
TELEMETRY: No legacy callbacks, legacy socket not created
[INFO] [X300] X300 initialization sequence...
[INFO] [X300] Maximum frame size: 8000 bytes.
[INFO] [X300] Maximum frame size: 8000 bytes.
[INFO] [X300] Radio 1x clock: 184.32 MHz
[INFO] [MULTI_USRP]     1) catch time transition at pps edge
[INFO] [MULTI_USRP]     2) set times next pps (synchronously)
[WARNING] [0/Radio\#1] Attempting to set tick rate to 0. Skipping.
[WARNING] [0/Radio\#0] Attempting to set tick rate to 0. Skipping.
Cell pci=1, bw=100 MHz, 2T1R, dl_arfcn=660000 (n77), dl_freq=3900 MHz, dl_ssb_arfcn=657312,
ul_freq=3900 MHz

N2: Connection to AMF on 10.0.23.62:38412 completed
==== gNB started ===
```

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`.

Misc Links

- Real-time failure in RF
- mbuf alloc

Next steps

- Accelerating with BBDEV — offload LDPC encode and decode to a hardware accelerator.
- Tuning performance — tune the host for real-time performance.

Hardware acceleration with BBDEV

> **INFO**
>
> This tutorial assumes that DPDK is already installed and configured on your system. For details, see the DPDK tutorial.

Overview

LDPC encoding and decoding are the most compute-intensive operations in the 5G NR PHY layer. Offloading them to a dedicated hardware accelerator reduces CPU load significantly, which either frees those cores for other gNB functions or allows the same host to support a higher bandwidth configuration or more concurrent UEs.

DPDK's BBDEV (wireless baseband device) library provides a common interface to hardware accelerators that handle PHY-layer processing. OCUDU integrates with BBDEV-capable devices through its hardware abstraction layer (HAL). This tutorial covers building, configuring, and running OCUDU with Intel's ACC100 and vRAN Boost 1.0 (ACC200/VRB1) accelerators.

This example uses test mode with dummy RU emulation to keep the setup simple. No radio hardware is required.

Device naming

This tutorial covers two Intel accelerator families:

Name	Also known as	Notes
ACC100	Intel vRAN Dedicated Accelerator ACC100	Older device; has dedicated on-board DDR for HARQ buffers
ACC200 / VRB1	Intel vRAN Boost 1.0	Newer device; no dedicated DDR, uses host memory for HARQ

In OCUDU configuration, `acc200` and `vrb1` are aliases and use the same code path as `acc100`. The tutorial uses **ACCx00** to refer to either device where the steps are identical.

Hardware and Software Requirements

- PC with Ubuntu 22.04.2

- OCUDU (built from source)
- DPDK (already installed, see the DPDK tutorial)
- PF-BBDEV Configuration Application
- An ACC100 PCIe accelerator card or ACC200/VRB1 (integrated on 4th Gen Xeon processors)

Installation

PF-BBDEV Configuration Application

The PF-BBDEV Configuration (`pf_bb_config`) application configures the hardware accelerator at the host level after DPDK takes ownership of the device. For more information, see the pf_bb_config documentation.

To install `pf_bb_config`, run:

```
git clone https://github.com/intel/pf-bb-config.git
cd pf-bb-config/
./build.sh
```

OCUDU gNB

BBDEV hardware acceleration requires an OCUDU build with DPDK enabled. Navigate to your OCUDU build directory and rerun cmake with the required flags:

```
cd ocudu/build
cmake ../ -DENABLE_DPDK=ON -DASSERT_LEVEL=MINIMAL
make -j`nproc`
```

> **INFO**
>
> `-DASSERT_LEVEL=MINIMAL` is optional but recommended for production deployments. If you have not set up the build directory yet, follow the Installation Guide first.

Enabling DPDK triggers compilation of hardware-acceleration code for both PUSCH and PDSCH PHY-layer processing. Verify that DPDK was found and that BBDEV hardware-acceleration was enabled by checking the cmake output includes:

```
-- Checking for module 'libdpdk>=22.11'
-- Found libdpdk, version 25.11.0
...
-- BBDEV hardware-acceleration enabled for PDSCH.
-- BBDEV hardware-acceleration enabled for PUSCH.
```

If the first two lines are absent, or if you see `Building without DPDK support as DPDK dependencies could not be resolved`, check the output of the DPDK installation step in the DPDK tutorial.

Test Mode with ACCx00

Configuration

ACCx00

The accelerator must be bound to DPDK's `vfio-pci` driver and configured with `pf_bb_config` before starting OCUDU. The steps below use PCI address `0000:8a:00.0` for the physical function (PF); substitute the address of your device.

Step 1: Bind the PF to vfio-pci

As shown in the DPDK tutorial:

```
cd dpdk/usertools/
sudo ./dpdk-devbind.py -b vfio-pci 0000:8a:00.0
```

Expected output (confirming `drv=vfio-pci`):

```
Baseband devices using DPDK-compatible driver
=============================================
0000:8a:00.0 'Device 0d5c' drv=vfio-pci unused=
```

Step 2: Configure the device and enable SR-IOV

SR-IOV (Single Root I/O Virtualization) allows a single physical device to expose one or more virtual functions (VFs) to the OS. OCUDU communicates with a VF via the DPDK EAL. For a single gNB instance, creating 1 VF is usually sufficient; up to 64 VFs are supported when multiple workloads or instances need to share the device.

```
cd pf-bb-config/
# Replace the UUID with a value of your choice — it must match the --vfio-vf-token in
eal_args
sudo ./pf_bb_config ACC100 -v 00112233-4455-6677-8899-aabbccddeeff -c
acc100/acc100_config_1vf_5g.cfg
echo 1 | sudo tee /sys/module/vfio_pci/parameters/enable_sriov
echo 1 | sudo tee /sys/bus/pci/devices/0000\:8a\:00.0/sriov_numvfs
```

`acc100_config_1vf_5g.cfg` and `vrb1_config_vf_5g.cfg` are predefined configuration files included with the `pf_bb_config` installation.

The `-v` flag assigns a VFIO token (a UUID) to the VF. This token is a shared secret between `pf_bb_config` and the DPDK EAL: the same value must appear as `--vfio-vf-token` in `eal_args`. The value `00112233-4455-6677-8899-aabbccddeeff` used throughout this tutorial is an example; you can replace it with any valid UUID.

It is also possible to use the ACCx00 with the `igb_uio` driver (see the DPDK tutorial for build and usage details). In that case, the `pf_bb_config` command must point to a physical function-based (`_pf`) configuration and does not include the `-v` VF token flag.

OCUDU gNB

The following configuration file provides a simple example for using the ACCx00 in test mode (1 cell, 4T4R, 100 MHz, single UE). Save it as `gnb_accx00_testmode.yml`:

```
# SPDX-FileCopyrightText: Copyright (C) 2021-2026 Software Radio Systems Limited
# SPDX-License-Identifier: BSD-3-Clause-Open-MPI

cu_cp:
amf:
no_core: true

log:
filename: /tmp/gnb.log
all_level: warning

cu_up:
warn_on_drop: false

buffer_pool:
nof_segments: 1048576

cell_cfg:
dl_arfcn: 381500
band: 39
channel_bandwidth_MHz: 100
common_scs: 30
plmn: "00101"
tac: 7
pci: 1
nof_antennas_dl: 4
nof_antennas_ul: 4
pdsch:
min_ue_mcs: 27
mcs_table: qam256
pusch:
min_ue_mcs: 27
mcs_table: qam256
rv_sequence: 0

cells:
- pci: 1

ru_dummy:

test_mode:
```

```
  test_ue:
  rnti: 0x1234
  pdsch_active: true
  pusch_active: true
  ri: 4
  nof_ues: 1

  expert_phy:
  max_request_headroom_slots: 3
  max_proc_delay: 4
  pusch_dec_max_iterations: 2

  hal:
  eal_args: "--lcores (0-1)@(3-29,33-59) --vfio-vf-token=00112233-4455-6677-8899-
  aabbccddeeff -a 0000:8b:00.1 --log-level=error"
  bbdev_hwacc:
  hwacc_type: acc100
  id: 0
  pdsch_enc:
  nof_hwacc: 32
  cb_mode: false
  dedicated_queue: true
  pusch_dec:
  nof_hwacc: 32
  force_local_harq: false
  dedicated_queue: true
  harq_context_size: 5184
```

All BBDEV parameters live in the `hal` section. The following extract explains each parameter:

```
hal:
bbdev_hwacc:
hwacc_type: acc100        # Accepted values: acc100, acc200, vrb1 (acc200/vrb1 are aliases)
id: 0                     # BBDEV device index; 0 when only one accelerator is installed
pdsch_enc:
nof_hwacc: 32           # Number of hardware-accelerated LDPC encoder functions to reserve
cb_mode: false          # false = encode full TB in one operation; true = encode each CB
separately
dedicated_queue: true   # true = dedicated queue per function (default); false = shared
queue
pusch_dec:
nof_hwacc: 32           # Number of hardware-accelerated LDPC decoder functions to reserve
force_local_harq: false # false = use ACC100 on-board DDR for HARQ (ACC200 always uses host
memory)
dedicated_queue: true   # true = dedicated queue per function (default); false = shared
queue
harq_context_size: 5184 # HARQ context repository size; sized here for max 162 CBs/TB × 32
UEs
eal_args: "--lcores (0-1)@(3-29,33-59) --vfio-vf-token=00112233-4455-6677-8899-aabbccddeeff
-a 0000:8b:00.1 --log-level=error"
```

Key configuration parameters:

`hwacc_type` — selects the BBDEV accelerator. Accepted values are `acc100`, `acc200`, and `vrb1`. `acc200` and `vrb1` are aliases and use the same code path as `acc100`.

`id` — BBDEV device index (typically `0` when only one accelerator is installed). Use `rte_bbdev_count` / `rte_bbdev_is_valid` from the BBDEV API to enumerate installed devices.

`pdsch_enc` — hardware-accelerated PDSCH encoding:

- `nof_hwacc` — number of hardware-accelerated LDPC encoder functions to reserve. Set to `0` to disable hardware acceleration and fall back to software. The maximum value is 64. This value is per cell.
- `cb_mode` — when `false` (default), the entire TB is encoded in a single operation (lower overhead). When `true`, each code block (CB) is encoded separately. CB mode is forced automatically if the TB or rate-matched CB size exceeds the maximum `mbuf` size (64 000 bytes).
- `dedicated_queue` — when `true` (default), each accelerated function instance gets its own hardware queue, allocated upfront. When `false`, a queue is reserved dynamically per operation, which is more flexible if the number of function instances exceeds available queues, but adds per-operation overhead.

`pusch_dec` — hardware-accelerated PUSCH decoding:

- `nof_hwacc` — number of hardware-accelerated LDPC decoder functions to reserve. Set to `0` to disable hardware acceleration and fall back to software. The maximum value is 64. This value is per cell.
- `force_local_harq` — when `false` (default), HARQ uses the ACC100's embedded DDR. Set to `true` to use host memory instead (strongly discouraged for ACC100; significant performance penalty). The ACC200 has no dedicated DDR, so host memory is always used regardless of this setting.
- `dedicated_queue` — same as for `pdsch_enc` above.
- `harq_context_size` — size of the HARQ context repository for tracking CB offsets in DDR memory. Size to the maximum expected value (max CBs per TB × max UEs). The default in the example (5184 = 162 CBs × 32 UEs) is appropriate for a 100 MHz 32-UE configuration.

`eal_args` — arguments forwarded to the DPDK EAL:

- `-a <vf_address>` — the PCI address of the **VF** created by SR-IOV, not the PF. After enabling SR-IOV, confirm the VF address with `sudo dpdk-devbind.py -s` and substitute it here.
- `--vfio-vf-token` — the UUID assigned during `pf_bb_config`. Must match the `-v` argument used there.
- `--lcores (0-1)@(3-29,33-59)` — maps EAL threads 0–1 to cores 3–29 and 33–59. Adjust for your host's core count.
- Use `--vfio-intr=msi` to request VFIO MSI interrupts (the only type supported by ACC100). If Open Fronthaul with DPDK is also enabled, MSI interrupt configuration will not complete; in that case, use `--log-level=error` to suppress undesired warning messages when closing the gNB application.

> **INFO**
>
> When `upper_phy` parameters from `expert_execution` are in use, the number of DL/UL threads (including PUSCH decoder threads) is bounded by the corresponding `nof_hwacc` values in the `hal`

section.

Running the gNB with the ACCx00

```
sudo ./apps/gnb/gnb -c gnb_accx00_testmode.yml
```

Expected console output:

```
--== OCUDU gNB (commit 57abe27c28) ==--

Warning: the configured maximum PDSCH concurrency (0) is overridden by the number of PDSCH
encoder hardware accelerated functions (32)
Warning: the configured maximum PUSCH and SRS concurrency (4) is overridden by the number of
PUSCH decoder hardware accelerated functions (32)

Cell pci=1, bw=100 MHz, 4T4R, dl_arfcn=381500 (n39), dl_freq=1907.5 MHz,
dl_ssb_arfcn=373490, ul_freq=1907.5 MHz

==== gNB started ===
Type <h> to view help

|--------------------DL---------------------|-------------------------------UL--------------
---------------
pci rnti | cqi  ri  mcs  brate   ok  nok  (%)  dl_bs | pusch  rsrp  ri  mcs  brate   ok  nok
(%)    bsr     ta  phr
1 1234 |  15 4.0   27   1.2G 1400    0   0%    10M |  99.9 -99.9   1   27   154M  600    0
0%  81.9M      0  n/a
```

The two `Warning` lines are expected. They indicate that OCUDU has overridden the software concurrency limits (derived from `expert_execution.threads`) with the number of hardware-accelerated function instances configured in `nof_hwacc`. This is correct behaviour: when BBDEV acceleration is active, the hardware function count determines the maximum concurrent encode/decode operations, not the software thread count.

Teardown

To release the accelerator and restore it to kernel control:

Step 1: Stop the gNB (Ctrl+C or send SIGTERM).

Step 2: Remove the VFs:

```
echo 0 | sudo tee /sys/bus/pci/devices/0000\:8a\:00.0/sriov_numvfs
```

Step 3: Unbind the PF from vfio-pci:

```
sudo dpdk-devbind.py --unbind 0000:8a:00.0
```

After unbinding, the device will appear in the "Other devices" section of `dpdk-devbind.py -s` with no driver attached. This is the normal state for an ACC100/ACC200 on a system without the vendor kernel driver installed. The device is now available for re-binding in a future session.

Next steps

- Tuning performance — tune the host for real-time performance.

Tuning a host for real-time performance

> **INFO**
>
> This tutorial does not cover machine-specific configuration steps (e.g., BIOS settings) or the installation and fine-tuning of real-time Linux kernels.

Overview

This tutorial provides a step-by-step guide to tuning a Linux-based machine for maximizing the performance of OCUDU. While optimal performance of OCUDU may require additional machine-specific fine-tuning, the recommendations outlined here are designed to meet the needs of most users.

TuneD

The approach to performance tuning outlined here utilizes TuneD, a system tuning service for Linux that uses `udev` to monitor connected devices and dynamically adjust system settings according to user-definable profiles. TuneD provides both static and adaptive tuning capabilities, allowing users to optimize the performance of the system for specific workloads or environments. For that, the user-definable profiles, amongst others, can include `GRUB` command-line arguments, CPU tuning parameters and sysf-exposed kernel parameters (e.g. `kobject`).

Installation

The TuneD tool can be installed with the following commands:

```
sudo apt install tuned
sudo ln -s /boot/grub/grub.cfg /etc/grub2.cfg
```

Once installed, edit `/etc/grub.d/00_tuned` and add the following line at the end of the file:

```
echo "export tuned_params"
```

Tuning profile

Custom profiles can be created to optimize the system performance according to user-specific requirements. It is advised to base those on pre-existing real-time profiles (e.g.,

`/usr/lib/tuned/realtime/tuned.conf`). As part of this tutorial you can download a custom OCUDU tuning profile, aimed at optimizing the gNB performance:

- Tuning profile
- Tuning script

To use the tuning profile we recommend creating a new folder inside `/usr/lib/tuned/` and copying both files above to it (e.g., `/usr/lib/tuned/ocudu/`). We also recommend users open the tuning profile and inspect its contents, as extensive in-line comments have been added to explain the selected choices. Once the files are copied, please make sure that the script is executable:

```
sudo chmod +x /usr/lib/tuned/ocudu/startup.sh
```

Before activating the OCUDU tuning profile you need to make sure that (in case it is installed) the `power-profiles-daemon` is disabled:

```
sudo systemctl stop power-profiles-daemon.service
sudo systemctl disable power-profiles-daemon.service
```

The OCUDU tuning profile can be activated with the following command:

```
sudo tuned-adm profile ocudu
```

The current active tuning profile can be checked at `/etc/tuned/active_profile` or with the following command:

```
tuned-adm active
```

We recommend configuring the TuneD service to automatically start at system startup. You can do this with the following command:

```
sudo systemctl enable tuned.service
```

Finally, the machine must be rebooted in order for the modified `GRUB` arguments to be applied.

Further Reading

- Performance Tuning Guide - TuneD.
- Real-time group scheduing.

Next steps

- Running on Kubernetes — deploy the components as Kubernetes pods.

Deploying OCUDU on Kubernetes

Overview

This tutorial outlines the steps required to deploy the OCUDU gNB for a split 7.2 architecture using Kubernetes. This approach is well-suited for scenarios that require network deployment and management over extended periods, with high service availability and enhanced fault tolerance.

In short, Kubernetes can be described as follows:

> When managing test networks, Kubernetes plays a pivotal role in streamlining and simplifying operations. It provides a unified platform for orchestrating network functions, optimizing resource usage, and automating scaling based on demand. Its built-in fault tolerance ensures consistent service availability, while its support for continuous deployment accelerates innovation. Kubernetes also simplifies deployment across varied environments, promoting adaptability. In essence, it offers a cohesive and scalable framework for efficient network function management.

Note, however, that such deployments may not be ideal for research and development-focused use cases that require fine-tuning of configuration files or source code modification. For iterative development and testing, "bare metal" setups are often more appropriate.

Further Reading

We recommend that users unfamiliar with Kubernetes or Helm begin by reviewing the following resources:

- Getting started with Kubernetes
- Getting started with Helm

Setup Considerations

This tutorial will cover the following topics:

- Set up Kubernetes/K3s nodes
  - Install a real-time kernel
  - Optimize system performance using TuneD
  - Install DPDK
- Set up PTP synchronization
  - LLS-C1 and LLS-C3
- Deploy the core network (Open5GS)

- Deploy the gNB
  - Connect to internal core networks and SMOs within the cluster
  - Assign DPDK devices without using the SR-IOV plugin
- Run load testing
  - cyclictest
  - OCUDU RU Emulator
- Visualize KPIs using Grafana

This tutorial uses a single-node cluster running Ubuntu 24.04. Kubernetes version 1.24 or newer is required. A basic understanding of Kubernetes and Helm is assumed.

CU/DU

The CU/DU is provided by OCUDU. The Open Fronthaul (OFH) Library offers the required interface between the DU and the RU.

RU

For this tutorial, you can use any of the RUs supported by OCUDU. For more information on supported O-RUs, see the Radio Units section under integrations.

5G Core

This tutorial uses the Open5GS 5G Core.

Open5GS is an open-source implementation of the 5G Core and EPC, written in C. The following links provide guidance on downloading and setting up Open5GS to work with OCUDU:

- Open5GS GitHub
- Open5GS Quickstart Guide

Clocking & Synchronization

The split 7.2 interface requires tight synchronization between the DU and RU. O-RAN WG4 has defined various synchronization mechanisms for use with Open Fronthaul, outlined in *O-RAN.WG4.CUS.0-R003-v11.00*, Section 11.

This tutorial explains how to configure LLS-C1 and LLS-C3 setups.

- **LLS-C1**: The DU acts as the PTP grandmaster, and the RU is a PTP client.
- **LLS-C3**: Both the DU and RU are PTP clients, synchronized to a common grandmaster.

The PTP grandmaster is typically a GPS or Rubidium clock, while the PTP client is usually a network interface card (NIC) with PTP support.

Set up a K8s/K3s Bare Metal Cluster

1. Deploy a Kubernetes Cluster

The installation of Kubernetes varies across distributions and the tools used for deployment. Depending on your requirements and environment, you can choose the tool that best suits your needs. In this tutorial, we deploy a single-node K3s cluster on Ubuntu 24.04 Server. K3s is a lightweight Kubernetes distribution that is easy to install and manage. It is designed for resource-constrained environments and edge computing, making it a great choice for bare metal Kubernetes deployments.

Popular tools for deploying Kubernetes include:

- Kubespray
- kubeadm
- K3s
- Rancher

The installation of K3s is very straightforward and can be completed with a single command. The following command installs K3s on your server:

```
curl -sfL https://get.k3s.io | sh -
```

For more information, refer to the official K3s documentation.

2. Install Realtime Kernel

The real-time kernel in Ubuntu 24.04 LTS, built on the PREEMPT_RT patch, ensures low-latency and deterministic performance for time-sensitive operations. By prioritizing critical processes and ensuring predictable response times, it is ideal for telco applications. This release also improves support for Raspberry Pi hardware, enabling optimized real-time computing across diverse applications.

To install the real-time kernel on Ubuntu 24.04, you must obtain a free Canonical Pro subscription. Register on the Canonical website and create an account. After that, use your Pro token and the following commands to install the kernel:

```
sudo pro attach <your-token>
sudo pro enable realtime-kernel
```

Reboot the system after the installation is complete. For more information, refer to the Ubuntu documentation.

3. Install TuneD

For performance tuning using TuneD, refer to the tuning tutorial in our documentation.

4. Install DPDK

For DPDK installation instructions, refer to the DPDK guide.

Set Up PTP Synchronization

PTP synchronization can be established using tools like ptp4l, ts2phc, and phc2sys. These tools can be deployed using the OCUDU linuxptp Helm chart. As a first step, install the OCUDU Helm repository:

```
helm repo add ocudu https://gitlab.com/ocudu/ocudu_elements/ocudu_helm/
```

Depending on your setup, PTP components can be deployed in different configurations. The most common ones are **LLS-C1** and **LLS-C3**, which can use either unicast or multicast transmission.

- In the **LLS-C1** configuration, the DU server drives PTP synchronization, and the RU acts as a client. The RU receives PTP messages from the DU.
- In the **LLS-C3** configuration, both the DU and RU are clients receiving PTP messages from a common PTP grandmaster.

In this tutorial, we demonstrate how to deploy both LLS-C1 and LLS-C3 configurations using the G.8275.1 multicast profile of linuxptp. For more information, refer to the official linuxptp documentation.

The configuration is set in the values.yaml file of the Helm chart.

LLS-C1 example configuration:

```
config:
dataset_comparison: "G.8275.x"
G.8275.defaultDS.localPriority: "128"
maxStepsRemoved: "255"
logAnnounceInterval: "-3"
logSyncInterval: "-4"
logMinDelayReqInterval: "-4"
serverOnly: "1"
clientOnly: "0"
G.8275.portDS.localPriority: "128"
ptp_dst_mac: "01:80:C2:00:00:0E"
network_transport: "L2"
domainNumber: "24"
```

LLS-C3 example configuration:

```
config:
dataset_comparison: "G.8275.x"
G.8275.defaultDS.localPriority: "128"
maxStepsRemoved: "255"
```

```
logAnnounceInterval: "-3"
logSyncInterval: "-4"
logMinDelayReqInterval: "-4"
serverOnly: "0"
clientOnly: "1"
G.8275.portDS.localPriority: "128"
ptp_dst_mac: "01:80:C2:00:00:0E"
network_transport: "L2"
domainNumber: "24"
```

For additional configuration options, refer to the linuxptp Helm chart README. An example values.yaml can be found in the repo as well.

To deploy the PTP components, use the following command:

```
helm install ptp4l ocudu/linuxptp -f values.yaml
```

If the server is under heavy load and PTP performance degrades, you can assign the linuxptp Pod an exclusive CPU core by editing the resources section of the values.yaml file. This ensures the linuxptp Pod is isolated from other workloads:

```
resources:
requests:
cpu: "1"
memory: "512Mi"
limits:
cpu: "1"
memory: "512Mi"
```

Set Up Core Network: Open5GS

Open5GS is an open-source implementation of the 5G Core and EPC, written in C. The following links provide the necessary information to download and set up Open5GS for use with OCUDU:

- Open5GS GitHub
- Open5GS Quickstart Guide

First, install a PersistentVolume (PV) and a PersistentVolumeClaim (PVC) for MongoDB.

- Example PV and PVC for Open5GS

Apply the PV and PVC manifest:

```
kubectl apply -f open5gs-pv-pvc.yaml
```

The PV is configured using hostPath. Ensure that the path exists and has the proper file access rights on the host system. The default path is `/mnt/data/vol`. If needed, create it and set the file access rights using:

```
mkdir -p /mnt/data/vol
chown -R 1001:1001 /mnt/data/vol
```

Next, prepare the values.yaml file and set the required RAN parameters. You can use the following as a starting point:

- Example values.yaml for Open5GS

Deploy Open5GS using Helm. This example assumes your values.yaml references the previously created PVC:

```
helm install open5gs oci://registry-1.docker.io/gradiant/open5gs --version 2.2.5 -f 5gSA-values.yaml -n open5gs --create-namespace
```

You should see the following output:

```
Pulled: registry-1.docker.io/gradiant/open5gs:2.2.0
Digest: sha256:99d49ab6bb2d4a5c78be31dd2c3a99a0780de79bd22d0bfa9df734ca2705940a
NAME: open5gs
LAST DEPLOYED: Mon Dec 9 11:09:17 2024
NAMESPACE: open5gs
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Wait for all Pods to be in the Running state. Check with:

```
kubectl get pods -n open5gs
```

Once the components are running, you can edit subscribers via the Open5GS WebUI. To do this, forward port 9999 of the open5gs-webui service to your local machine:

```
kubectl port-forward svc/open5gs-webui 9999:9999 -n open5gs
```

Expected output:

```
Forwarding from 127.0.0.1:9999 -> 9999
Forwarding from [::1]:9999 -> 9999
```

Leave the shell open and access the WebUI by visiting http://localhost:9999 in your browser. (Default credentials: **admin** / **1423**). Once you're done editing subscribers, you can close the shell.

Set Up gNB

To deploy the gNB, edit the values.yaml file and set the desired RAN parameters. An example values.yaml for the OCUDU Helm Chart can be found here.

If you haven't already added the OCUDU Helm repository, add it using:

```
helm repo add ocudu https://gitlab.com/ocudu/ocudu_elements/ocudu_helm/
```

In the following, we explain how to set up different scenarios using the OCUDU Helm Chart.

1. Connecting to Internal Core Networks Within the Cluster

When all components run within the same Kubernetes cluster, you can use DNS hostnames instead of a LoadBalancer. For example, if the Open5GS core network is deployed in the same cluster, use the AMF service's hostname to connect to it.

To determine the cluster domain, run:

```
kubectl run -it --image=ubuntu --restart=Never shell -- sh -c 'apt-get update > /dev/null &&
apt-get install -y dnsutils > /dev/null && nslookup kubernetes.default | grep Name | sed
"s/Name:\skubernetes.default//"'
```

Example output:

```
If you dont see a command prompt, try pressing enter.
debconf: delaying package configuration, since apt-utils is not installed

.svc.kubernetes.local
```

In this case, the cluster domain is svc.kubernetes.local. To construct a service hostname:

```
<service-name>.<namespace>.svc.<cluster-domain>
```

To list all available services:

```
kubectl get services -A
```

Example output:

```
NAMESPACE NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S)                  AGE
default kubernetes ClusterIP 10.96.0.1        <none>        443/TCP                  10d
default open5gs-amf-ngap ClusterIP 10.111.110.41    <none>        38412/SCTP
16h
[...]
```

Here, the AMF service name is open5gs-amf-ngap and the namespace is default. Therefore, the hostname is:

`open5gs-amf-ngap.default.svc.kubernetes.local`

Use this hostname in the amf section of the gNB configuration in values.yaml.

For more information, refer to the official Kubernetes DNS documentation.

2. Assign DPDK Devices Without the SR-IOV Plugin

To assign PFs or VFs directly to the container without using the SR-IOV plugin, you must grant the Pod full access to the host system. In values.yaml, set the following:

Enable access to the host network:

```
network:
hostNetwork: true
```

Enable privileged mode and set required capabilities:

```
securityContext:
capabilities:
add: ["SYS_NICE", "NET_ADMIN"]
privileged: true
```

With this setup, the gNB Pod has full access to the hosts network stack. This enables the Pod to access both external and internal Kubernetes network resources.

Load Testing

In the following, we present two methods to test the maximum load on the system.

1. OCUDU RU Emulator

The OCUDU RU Emulator is a tool that emulates a Radio Unit (RU). It prints KPIs such as early and late packets, which are useful for debugging network issues and evaluating how much load a deployment can handle. You can quickly deploy the RU Emulator using the dedicated Helm chart.

Before deploying the RU Emulator, you must obtain the RU and DU MAC addresses, along with the bandwidth, VLAN tag, and compression method. These parameters are **mandatory**:

- ru_mac_addr: MAC address of the interface used for Open Fronthaul (OFH) traffic.
- du_mac_addr: MAC address of the DU interface used for OFH traffic.

Example configuration:

```
ru_emu:
cells:
- bandwidth: 100
network_interface: enp4s0f0
ru_mac_addr: 50:7c:6f:45:44:33
du_mac_addr: 00:11:22:33:44:00
vlan_tag: 6
ul_port_id: [0]
compr_method_ul: "bfp"
compr_bitwidth_ul: 9
```

Use the following configuration inside the values.yaml file to enable the RU Emulator:

```
securityContext:
capabilities:
add: ["SYS_NICE", "NET_ADMIN"]
privileged: true
```

Also make sure to explicitly disable SR-IOV by setting:

```
sriovConfig:
enabled: false
```

Ensure that the network_interface and du_mac_addr values are set correctly for your deployment.

2. Assess Maximum Latency Using cyclictest

cyclictest is a tool used to measure application latency on real-time systems. To assess the maximum latency on your system, you can deploy cyclictest using the OCUDU rt-test Helm chart.

The rt-test Helm chart is designed to run two real-time test tools: cyclictest and stress-ng. While cyclictest measures system latency and jitter, stress-ng generates high CPU and memory load to simulate system

stress. Running both together allows you to evaluate how well the system maintains real-time performance under load.

To configure your test scenario, update the config section of the values.yaml file. For example:

```
config:
rt_tests.yml: |-
stress-ng: "--matrix 0 -t 12h"
cyclictest: "-m -p95 -d0 -a 1-15 -t 16 -h400 -D 12h"
```

This configuration runs stress-ng for 12 hours using the –matrix workload, and launches cyclictest pinned to CPU cores 1–15 across 16 threads with high priority, running for 12 hours. For more information on cyclictest and stress-ng, refer to the cyclictest and stress-ng documentation.

To deploy rt-test, use the following Helm command:

```
helm install rt-tests ocudu/rt-tests -n ocudu --create-namespace
```

Visualizing KPIs via Grafana

To visualize gNB KPIs, we provide a Grafana dashboard designed to work with the metrics server included in the OCUDU Helm repository. The metrics server collects and parses gNB metrics, stores them in an InfluxDB database, and the Grafana dashboard then displays them.

If you haven't already added the OCUDU Helm repository, add it now:

```
helm repo add ocudu https://gitlab.com/ocudu/ocudu_elements/ocudu_helm/
```

The Grafana dashboard comes with a pre-configured values.yaml file. The only field that must be adjusted is the **cluster domain**, which is required to resolve service hostnames.

To determine your cluster domain, run:

```
kubectl run -it --image=ubuntu --restart=Never shell -- sh -c 'apt-get update > /dev/null &&
apt-get install -y dnsutils > /dev/null && nslookup kubernetes.default | grep Name | sed
"s/Name:\skubernetes.default//"'
```

This command launches a temporary container and runs a DNS query against the kubernetes.default service. Expected output:

```

```

```
If you dont see a command prompt, try pressing enter.
debconf: delaying package configuration, since apt-utils is not installed

.svc.kubernetes.local
```

In this case, the cluster domain is svc.kubernetes.local. Adjust the values.yaml file by replacing the default domain (.svc.cluster.local) with the string returned by the above command.

Download the default values.yaml file using:

```
wget https://gitlab.com/ocudu/ocudu_elements/ocudu_helm/-/raw/main/charts/grafana-ocudu/values.yaml?ref_type=heads
```

We use Telegraf to collect metrics from the running OCUDU instance. Make sure that the `WS_URL` points to the right WebSocket URL and port that is configured in the OCUDU Helm.

Once updated, delete the temporary container:

```
kubectl delete pod shell
```

Now deploy the Grafana dashboard:

```
helm install ocudu-grafana ocudu/grafana-deployment -f values.yaml -n ocudu --create-namespace
```

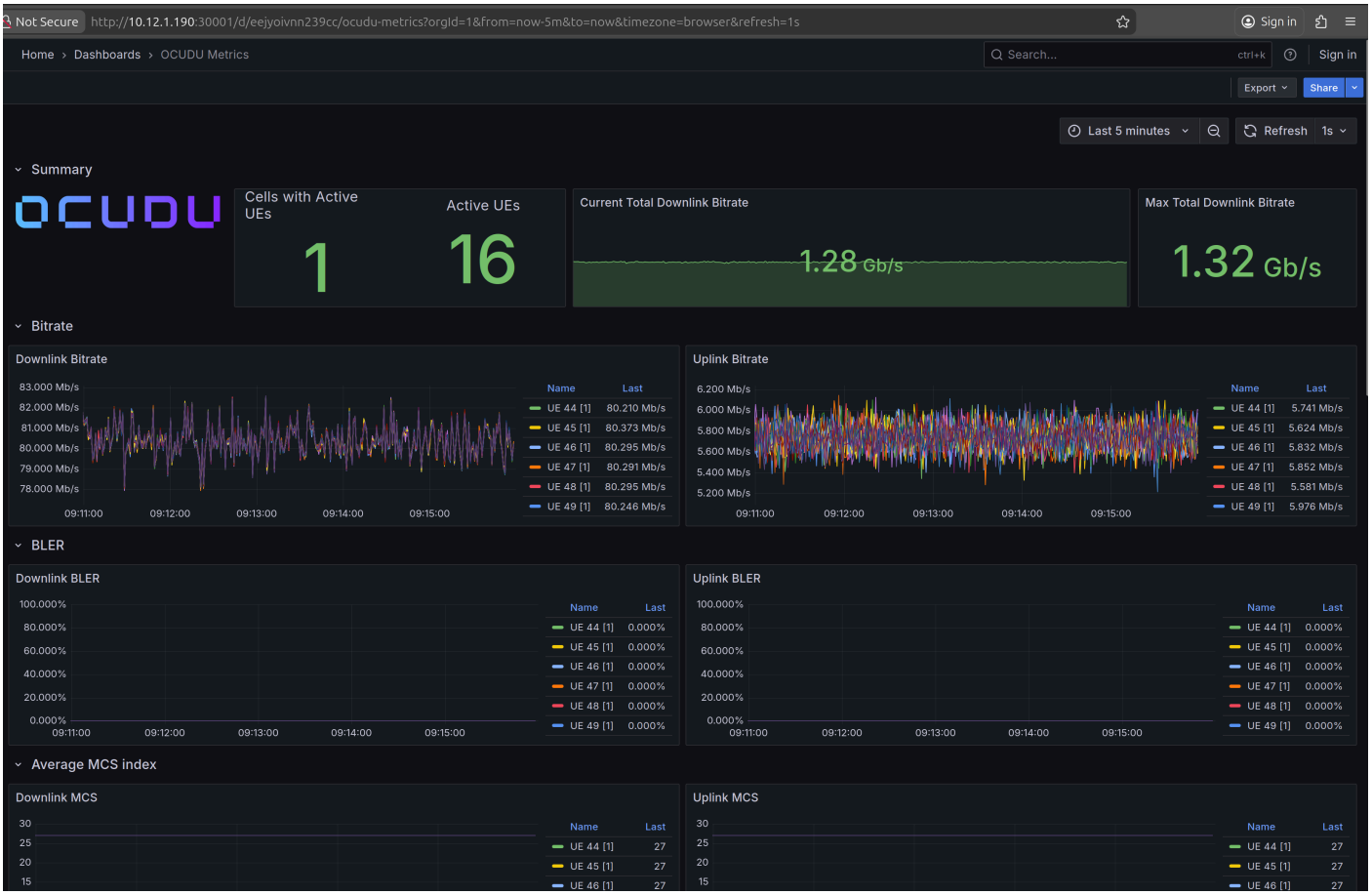
After all components are running, the gNB application can start sending metrics to the metrics server.

To access the Grafana dashboard, forward the service port to your local machine:

```
export POD_NAME=$(kubectl get pods --namespace ocudu -l "app.kubernetes.io/name=grafana,app.kubernetes.io/instance=ocudu-grafana" -o jsonpath="{.items[0].metadata.name}")
kubectl --namespace ocudu port-forward $POD_NAME 3000
```

Open your browser and go to: http://localhost:3000

An example of the Grafana dashboard is shown below:



Clean Up Deployments

To clean up all deployments, use the following commands:

To delete the OCUDU deployment:

```
helm uninstall ocudu-gnb -n ocudu
```

To delete the linuxptp deployment:

```
helm uninstall linuxptp -n ocudu
```

To delete the Open5GS deployment:

```
helm uninstall open5gs -n open5gs
```

To delete the Grafana deployment:

```
helm uninstall grafana -n ocudu
```

Next steps

- Splitting CU and DU — run the CU and DU as separate processes over F1.

Integrating MATLAB with OCUDU

Overview

This tutorial explains the main features of OCUDU MATLAB, a MATLAB-based project for testing OCUDU. More specifically, this tutorial will show how to generate a new set of test vectors for the OCUDU tests, how to analyze the uplink IQ samples recorded by the OCUDU gNB, and how to run end-to-end, link-level simulations for testing PHY components of OCUDU. This will be done across three independent sections.

OCUDU MATLAB offers three main tools: the test vector generators, the uplink analyzers and the link-level simulators.

Test Vector Generation

Test vectors are mainly used to test, develop and debug the PHY components of OCUDU. This tutorial will show how to generate the set of vectors used for unit testing inside the OCUDU repository.

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.

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.

Set-Up Considerations

For this tutorial, the following hardware and software are used:

- A PC with Ubuntu 24.04 LTS
- OCUDU MATLAB
- OCUDU
- MathWorks MATLAB (R2024b or R2025b) with the 5G Toolbox

> **INFO**
>
> Running the OCUDU MATLAB testing suite requires a working and licensed copy of MATLAB and its 5G Toolbox.

Installation

Assuming that OCUDU and MATLAB have both been downloaded and installed, the next step is to download OCUDU MATLAB.

This can be done with the following command:

```
git clone https://gitlab.com/ocudu/ocudu_elements/ocudu-matlab
```

> **INFO**
>
> This tutorial assumes that OCUDU is installed in the users home directory.

Once it has been downloaded, the working directory for OCUDU MATLAB should be added to MATLAB's search path. This can be done from the MATLAB console with the following command:

```
cd ~/ocudu-matlab
addpath .
```

To verify you have added OCUDU MATLAB successfully to MATLAB's search path, run the following command (again from the MATLAB console):

```
runtests('tests/smoke', Tag='matlab code')
```

If successful, the final section of the output should look similar to:

```
ans =

1x131 TestResult array with properties:

Name
Passed
Failed
Incomplete
Duration
Details

Totals:
131 Passed, 0 Failed, 0 Incomplete.
242.2176 seconds testing time.
```

The PHY components of OCUDU can be tested by feeding each component with vectors of input data and comparing the resulting output with their expected values. In this section of the tutorial

we will learn how to generate these test vectors with OCUDU MATLAB and add the corresponding tests to the OCUDU main code.

Test vector generation

The files `ocudu<ComponentName>Unittest.m` in the main directory of `OCUDU 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 OCUDU MATLAB.

To generate the test vectors for all PHY components the following code needs to be run from the MATLAB console:

```
runOCUDUunittest('all', 'testvector')
```

This will generate a `.h` file, with the vector descriptions, and a `.tar.gz` file, with the actual test vectors, for each of the PHY components and place them in the folder `~/ocudu-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 `~/ocudu/include/ocudu/`. For example, the test vectors for the channel estimator, whose interface is declared in `~/ocudu/include/ocudu/phy/upper/signal_processors/port_channel_estimator.h`, can be generated with the following command:

```
runOCUDUunittest('port_channel_estimator', 'testvector')
```

Once the test vectors have been generated, the pairs of `.h` and `tar.gz` files in the `testvector_outputs` folder should be transferred to the `ocuduVectorTests` folder with the MATLAB command:

```
ocuduTest.copyOCUDUtestvectors('testvector_outputs', 'ocuduVectorTests')
```

By default, executing `runOCUDUunittest` will always generate the same test vectors. To generate a random set of vectors, we simply add the `RandomShuffle` option:

```
runOCUDUunittest('all', 'testvector', RandomShuffle=true)
```

Build and run the vector tests

The folder `ocuduVectorTests` contains C++ code that, together with the test vectors generated above, can be imported into OCUDU as a plugin:

```
cd ~/ocudu
mkdir plugins
cd plugins
ln -s ~/ocudu-matlab/ocuduVectorTests ocudu_vectortests
cd ..
```

Next, make sure that plugins are imported in OCUDU when building the project:

```
cmake -B buildplugins -DENABLE_PLUGINS:BOOL=ON
```

CMake will notify the inclusion of the vector tests with the message

```
-- Adding plugin: plugins/ocudu_vectortests
```

Finally, compile and run the tests with

```
cmake --build buildplugins -j $(nproc) --target all_vector_tests
ctest --test-dir buildplugins -j $(nproc) -L vectortest
```

Next steps

- Load testing — validate your build without radio hardware.