openairinterface5g

Техническая документация

Данная документация содержит полное техническое описание проекта openairinterface5g, включая руководства пользователя, API reference, архитектурные решения и историю изменений. Документация сгенерирована автоматически на основе исходного кода проекта.

Сгенерировано: 2025-06-11 10:03

Оглавление

Основные страницы

C/C++ project: ORAN_FHI7.2_Tutorial

C/C++ project: LDPC_T2_OFFLOAD_SETUP

C/C++ project: Aerial_FAPI_Split_Tutorial

C/C++ project: mac-usage

C/C++ project: rrc-usage

C/C++ project: E1-design

C/C++ project: F1-design

C/C++ project: doc/README.md File Reference

C/C++ project: FEATURE_SET

RELEASE NOTES:

v2.2.0 -> November 2024.

General 5G improvements (both gNB and UE):

- Make standalone mode (SA) the default (see `RUNMODEM.md`)
- Experimental support for FR2 operation
- Support for GEO NTN and simulation of GEO satellite channel in RFsimulator (see `RUNMODEM.md`)
- Support 2-step RA
- Add optional LTTng logger in logging module (see `lttng_logs.md`)
- Support for YAML-based config files (alongside libconfig) (see `gnb.sa.band78.106prb.rfsim.yaml` and `nrue.uicc.yaml`)
- Add new L1 scope based on Dear ImGui (see `readme.md`)
- Allow cross-compilation on ARM (via SIMDE SIMD emulation library)
- Allow to build and run with clang
- Support/check for Linux capabilities, allow to run without sudo (see `tuning_and_security.md`)
- OAI does not modify CPU frequency and networking stack (`tuning_and_security.md`)
- Bugfixes in the entire stack (e.g. #547, #663, #674, #687, #712, #736, #739, #741, #756, #762, #773, ...)

5G gNB:

- Support for FR2 interoperability with COTS UE (no beam switching supported yet)
- Add 4-layer DL MIMO (experimental)
- Add gNB Neighbour configuration and Mobility over F1 interface (see `handover-tutorial.md`)
- Enhance O-RAN FHI 7.2: (see `ORAN_FHI7.2_Tutorial.md`)
    - Support different bandwidths (40/60/80/100MHz) and antenna configs (1x1 up to 4x4) for Benetel/VVDN/LITEON RUs

    - Add support for multi-RU support (single-cell/distributed antenna)
- Support AMD T2 Telco card look-aside L1 accelerator (see `LDPC_T2_OFFLOAD_SETUP.md`)
- Support Nvidia Aerial/ARC in-line L1 accelerator (see `Aerial_FAPI_Split_Tutorial.md`)
- Various fixes for multi-UE operation: by default support of up to 16 UEs concurrently
- Documentation for
    - 5G MAC (see `mac-usage.md`)
    - 5G RRC (see `rrc-usage.md`)
    - E1 (see `E1-design.md`)
    - F1 (see `F1-design.md`)

5G UE:

- Basic interoperability with COTS gNB (Nokia gNB)
- Implement PHR reporting
- Implement RRC re-establishment
- Implement PUCCH/PUSCH/SRS power control
- Implement UCI on PUSCH and aperiodic CSI reporting
- Support of cell search (within the selected UE bandwidth) (see `RUNMODEM.md`)
- Enhance connection control: implement timers, resync
- A lot of internal cleanup

This release also includes many fixes and documentation updates. See `doc/README.md` in the repository for an overview of documentation.

v2.1.0 -> February 2024.

This release improves existing 5G support and adds various new features.

- 5G gNB
    - Add support for O-RAN 7.2 fronthaul interface (tested with 3 O-RUs: Benetel, LITEON, VVDN)
    - Add support for 2-layer UL MIMO
    - FDD interoperability with COTS UE

    - Compiles on ARM (through SIMDE)
    - Introduce E2 agent and basic support for E2SM-KPM and E2SM-RC
    - Add support for E1AP
    - Add support for multiple DUs and CU-UPs at one CU-CP
    - FR2 SA with OAI UE
    - Improve computational efficiency
- 5G UE
    - Cleanup in MAC and RRC towards support of 3rd-party gNB
    - FR2 SA with OAI gNB
    - Improve computational efficiency

Overall the stability is improved for the same resource usage.

There is basic FR2 support between OAI gNB and OAI nrUE. COTS UE interoperability is under testing.

This release also includes many fixes and documentation updates.

v2.0.0 -> August 2023.

This release adds support for 5G and maintains previous features:

- 5G SA in gNB
    - PHY, MAC, RLC, PDCP, SDAP, RRC layers
    - 2x2 MIMO and 256-QAM for UL/DL
    - 15 and 30 kHz subcarrier spacings; 10-100 MHz bandwidths
    - Up to 800Mbps throughput or 5ms latency
    - F1, basic E1, 5G FAPI (SCF 222.10.02), split 8 split options
    - Handling of up to 16 UEs
    - RRC procedures for connection setup, multiple PDU sessions, reestablishment
- 5G SA in UE
    - PHY, MAC, RLC, PDCP, SDAP, RRC layers
    - 2x2 MIMO and 256-QAM for UL/DL
    - 15 and 30 kHz subcarrier spacings; 10-100 MHz bandwidths
    - Custom FAPI-like MAC/PHY interface

    - RRC procedures for connection setup and cell measurement
- Basic 5G NSA in gNB
    - X2 sgNB Addition Request between OAI eNB and gNB
- 4G eNB and UE
    - Bugfixes in fairRR scheduler (eNB)
    - Non-standard F1 midhaul removed (eNB)
    - FlexRAN removed (eNB)
    - Sync fixes (UE)
- LTE-M supported
- Support for AW2S devices, RFsimulator channel emulation support

For more information on supported features, please refer to the **feature set**.

v1.2.1 -> February 2020.

- Bug fix for mutex lock for wake-up signal

v1.2.0 -> January 2020.

This version adds the following implemented features:

- LTE-M : eNB support for Mode A repetitions
    - PUSCH CE - 8 Repetitions
- Improved CDRX implementation for monolithic eNB
- Experimental eMBMS support (now also on eNB side)
- Experimental MCE - Multicast Coordination Entity
- Bug fixes

This version also has an improved code quality:

- Better Test Coverage in Continuous Integration:
    - Initial framework to do long-run testing at R2LAB

v1.1.1 -> November 2019.

- Bug fix in the TDD Fair Round-Robin scheduler

v1.1.0 -> July 2019.

This version adds the following implemented features:

- Experimental support of LTE-M
  - Single LTE-M UE attachment, legacy-LTE UE attachment is disabled
- X2 interface and handover (also X2-U interface)
  - In FDD and TDD
- CU/DU split (F1 interface)
  - Tested only in FDD
- CDRX
  - Tested only in FDD
- Experimental eMBMS support (only on UE side)
- Experimental multi-RRU support
  - Tested only in TDD

This version has an improved code quality:

- Simplification of the Build System
  - A single build includes all full-stack simulators, S1/noS1 modes and one HW platform (such as USRP, BladeRF, ...)
- TUN interface is now used as default for the data plane
  - for UE, eNB-noS1 and UE-noS1
- Code Cleanup
- Better Static Code Analysis:
  - Limited number of errors in cppcheck
  - Important Decrease on high Impact errors in CoverityScan
- Better Test Coverage in Continuous Integration:
  - TM2, CDRX, IF4.5, F1
  - OAI UE is tested in S1 and noS1 modes with USRP board
  - Multi-RRU TDD mode
  - X2 Handover in FDD mode

v1.0.3 -> June 2019.

- Bug fix for LimeSuite v19.04.0 API

v1.0.2 -> February 2019.

- Full OAI support for 3.13.1 UHD

v1.0.1 -> February 2019.

- Bug fix for the UE L1 simulator.

v1.0.0 -> January 2019.

This version first implements the architectural split described in the following picture.

Block Diagram

- Only FAPI, nFAPI and IF4.5 interfaces are implemented.
- Repository tree structure prepares future integrations of features such as LTE-M, nbIOT or 5G-NR.
- Preliminary X2 support has been implemented.
- S1-flex has been introduced.
- New tools: config library, telnet server, ...
- A lot of bugfixes and a proper automated Continuous Integration process validates contributions.

Old Releases:

- v0.6.1 -> Mostly bugfixes. This is the last version without NFAPI.
- v0.6 -> RRH functionality, UE greatly improved, better TDD support, a lot of bugs fixed.
    - WARNING: oaisim in PHY abstraction mode does not work, you need to use v0.5.2 for that.
- v0.5.2 -> Last version with old code for oaisim (abstraction mode works)
- v0.5.1 -> Merge of bugfix-137-uplink-fixes. It includes stablity fixes for eNB
- v0.5 -> Merge of enhancement-10-harmony-lts. It includes fixes for Ubuntu 16.04 support

- v0.4 -> Merge of feature-131-new-license. It closes issue#131 and changes the license to OAI Public License V1.0
- v0.3 -> Last stable commit on develop branch before the merge of feature-131-new-license. This is the last commit with GPL License
- v0.2 -> Merge of enhancement-10-harmony to include NGFI RRH + New Interface for RF/BBU
- v0.1 -> Last stable commit on develop branch before enhancement-10-harmony

RUNMODEM

Running OAI 5G Softmodems

This document explains some options for running 5G executables.

After you have **built the softmodem executables** you can set your default directory to the build directory `cmake_targets/ran_build/build/` and start testing some use cases. Below, the description of the different OAI functionalities should help you choose the OAI configuration that suits your need.

[[TOC]]

Simulators

RFsimulator

The RFsimulator is an OAI device replacing the radio heads (for example the USRP device). It allows connecting the oai UE (LTE or 5G) and respectively the oai eNodeB or gNodeB through a network interface carrying the time-domain samples, getting rid of over the air unpredictable perturbations. This is the ideal tool to check signal processing algorithms and protocols implementation. The RFsimulator has some preliminary support for channel modeling.

It is planned to enhance this simulator with the following functionalities:

- Support for multiple eNodeB's or gNodeB's for hand-over tests

This is an easy use-case to setup and test, as no specific hardware is required. The **rfsimulator page** contains the detailed documentation.

L2 nFAPI Simulator

This simulator connects an eNodeB and UEs through an nFAPI interface, short-cutting the L1 layer. The objective of this simulator is to allow multi UEs simulation, with a large number of UEs (ideally up to 255).

As for the RFsimulator, no specific hardware is required. The **L2 nfapi**simulator page" contains the detailed documentation. @section autotoc_md649 Running with a true radio head OAI supports different radio heads, the following are tested in the CI: 1. Monolithic eNodeB where the whole signal processing is performed in a single process 2. IF4P5 mode, where frequency

domain samples are carried over ethernet, from the RRU which implement part of L1(FFT,IFFT,part of PRACH), to a RAU 3. Monolithic gNodeB: see next section, or the @ref "/tmp/tmpqbhrntcs/doc/NR_SA_Tutorial_COTS_UE.md" "standalone tutorial" @section autotoc_md650 5G NR @subsection autotoc_md651 NSA setup with COTS UE This setup requires an EPC, an OAI eNB and gNB, and a COTS Phone. A dedicated page describe the setup can be found here. The <tt>--nsa</tt> flag must be used to run gNB in non-standalone mode. @subsubsection autotoc_md652 Launch eNB @icode{bash} sudo ./lte-softmodem -O ../../../targets/PROJECTS/GENERIC-LTE-EPC/CONF/enb.band7.tm1.50prb.usrpb210.conf @endicode @subsubsection autotoc_md653 Launch gNB @icode{bash} sudo ./nr-softmodem -O ../../../targets/PROJECTS/GENERIC-LTE-EPC/CONF/gnb.band78.tm1.106PRB.usrpn300.conf --nsa @endicode You should see the X2 messages in Wireshark and at the eNB. @subsection autotoc_md654 SA setup with OAI NR-UE The standalone mode is the default mode. Before tag <tt>2024.w45</tt>, the default mode was NSA. Thus, in the past, to run either the gNB or the UE in standalone mode, it was necessary to provide the <tt>--sa</tt> flag in the command line. This is not necessary anymore; if in doubt, you can provide <tt>--sa</tt> on the command line. The gNB will ignore this option if not relevant; the UE might complain that it does not know this option, which means that standalone is already active. The default (SA) mode does the following: - At the gNB: * The RRC encodes SIB1 according to the configuration file and transmits it through NR-BCCH-DL-SCH. - At the UE: * Decode SIB1 and starts the 5G NR Initial Access Procedure for SA: 1) 5G-NR RRC Connection Setup 2) NAS Authentication and Security 3) 5G-NR AS Security Procedure 4) 5G-NR RRC Reconfiguration 5) Start Downlink and Uplink Data Transfer Command line parameters for UE in standalone mode: - <tt>-C</tt> : downlink carrier frequency in Hz (default value 0) - <tt>--CO</tt> : uplink frequency offset for FDD in Hz (default value 0) - <tt>--numerology</tt> : numerology index (default value 1) - <tt>-r</tt> : bandwidth in terms of RBs (default value 106) - <tt>--band</tt> : NR band number (default value 78) - <tt>--ssb</tt> : SSB start subcarrier (default value 516) To simplify the configuration for the user testing OAI UE with OAI gNB, the latter prints the following LOG that guides the user to correctly set some of the UE command line parameters: @icode{shell} [PHY] Command line parameters for OAI UE: -C 3319680000 -r 106 --numerology 1 --ssb 516 @endicode You can run this, using USRPs, on two separate machines: @icode{shell} sudo ./nr-softmodem -O ../../../targets/PROJECTS/GENERIC-NR-5GC/CONF/gnb.sa.band78.fr1.106PRB.usrpb210.conf --gNBs.[0].min_rxtxtime 6 sudo ./nr-uesoftmodem -r 106 --numerology 1 --band 78 -C 3619200000 --ssb 516 @endicode With the RFsimulator (on the same machine), just add the option <tt>--rfsim</tt> to both gNB and NR UE command lines. UE capabilities can be passed according to the @ref "UE-Capabilities" "UE Capabilities" section. A detailed tutorial is provided at this page @ref "/tmp/tmpqbhrntcs/doc/NR_SA_Tutorial_OAI_nrUE.md" "**NR_SA_Tutorial_OAI_nrUE.md**". @subsection autotoc_md655 Optional NR-UE command line options Here are some useful command line options for the NR UE: <table class="markdownTable"> <tr class="markdownTableHead"> <th class="markdownTableHeadNone"> Parameter

Description

`--ue-scan-carrier`

Scan for cells in current bandwidth. This option can be used if the SSB position of the gNB is unknown. If multiple cells are detected, the UE will try to connect to the first cell. By default, this option is disabled and the UE attempts to only decode SSB given by `--ssb`.

`--ue-fo-compensation`

Enables the initial frequency offset compensation at the UE. Useful when running over the air and/or without an external clock/time source.

`--cont-fo-comp`

Enables the continuous frequency offset (FO) estimation and compensation. Parameter value `1` specifies that the main FO contribution comes from the local oscillator's (LO) accuracy. Parameter value `2` specifies that the main FO contribution comes from Doppler shift.

`--initial-fo`

Sets the known initial frequency offset. Useful especially with large Doppler frequency, e.g. LEO satellite.

`--freq-sync-P`

Sets the coefficient for the Proportional part of the PI-controller for the continuous frequency offset compensation. Default value 0.01.

`--freq-sync-I`

Sets the coefficient for the Integrating part of the PI-controller for the continuous frequency offset compensation. Default value 0.001.

`--ntn-initial-time-drift`

Sets the initial NTN DL time drift (feeder link and service link), given in µs/s.

`--autonomous-ta`

Enables the autonomous TA update, based on DL drift (useful if main contribution to DL drift is movement, e.g. LEO satellite).

`--time-sync-P`

Sets the coefficient for the Proportional part of the PI-controller for the time synchronization. Default value 0.5.

`--time-sync-I`

Sets the coefficient for the Integrating part of the PI-controller for the time synchronization. Default value 0.0.

`--usrp-args`

Equivalent to the `sdr_addrs` field in the gNB config file. Used to identify the USRP and set some basic parameters (like the clock source).

`--clock-source`

Sets the clock source (internal or external).

`--time-source`

Sets the time source (internal or external).

You can view all available options by typing:

```
./nr-uesoftmodem --help
```

Common gNB and NR UE command line options

Three-quarter sampling

The command line option `-E` can be used to enable three-quarter sampling for split 8 sample rate. Required for certain radios (e.g., 40MHz with B210). If used on the gNB, it is a good idea to use for the UE as well (and vice versa).

UE Capabilities

The `--uecap_file` option can be used to pass the UE Capabilities input file (path location + filename), e.g.

```
--uecap_file ../../../targets/PROJECTS/GENERIC-NR-5GC/CONF/
uecap_ports1.xml
```

for 1 layer or e.g.

```
--uecap_file ../../../targets/PROJECTS/GENERIC-NR-5GC/CONF/
uecap_ports2.xml
```

for 2 layers.

This option is available for the following combinations of operation modes and gNB/nrUE softmodems:

Mode	Executable	Description
SA	nr-uesoftmodem	Send UE capabilities from the UE to the gNB via RRC
phy-test	nr-softmodem	Mimic the reception of UE capabilities by the gNB
do-ra	nr-softmodem	Mimic the reception of UE capabilities by the gNB

e.g.

```
sudo ./nr-uesoftmodem -r 106 --numerology 1 --band 78 -C 3319680000 --ue-nb-ant-tx 2 --ue-
        nb-ant-rx 2 --uecap_file /opt/oai-nr-ue/etc/uecap.xml
```

How to run a NTN configuration

NTN channel

A 5G NR NTN configuration only works in a non-terrestrial setup. Therefore either SDR boards and a dedicated NTN channel emulator are required, or RFsimulator has to be configured to simulate a NTN channel.

As shown on the **rfsimulator page**, RFsimulator provides different possibilities. E.g. to perform a simple simulation of a satellite in geostationary orbit (GEO), these parameters should be added to both gNB and UE command lines:

```
--rfsimulator.prop_delay 238.74
```

For simulation of a satellite in low earth orbit (LEO), two channel models have been added to rfsimulator:

- `SAT_LEO_TRANS` : transparent LEO satellite with gNB on ground
- `SAT_LEO_REGEN` : regenerative LEO satellite with gNB on board

Both channel models simulate the delay and Doppler for a circular orbit at 600 km height according to the Matlab function dopplerShiftCircularOrbit. An example configuration to simulate a transparent LEO satellite with rfsimulator would be:

```
channelmod = {
  max_chan=10;
  modellist="modellist_rfsimu_1";
  modellist_rfsimu_1 = (
    {
      model_name     = "rfsimu_channel_enB0"
      type           = "SAT_LEO_TRANS";
      noise_power_dB = -100;
    },
```

```
    {
      model_name     = "rfsimu_channel_ue0"
      type           = "SAT_LEO_TRANS";
      noise_power_dB = -100;
    }
  );
};
```

This configuration is also provided in the file `targets/PROJECTS/GENERIC-NR-5GC/CONF/channelmod_rfsimu_LEO_satellite.conf`.

Additionally, rfsimulator has to be configured to apply the channel model. This can be done by either providing this line in the conf file in section `rfsimulator`:

```
options = ("chanmod");
```

Or by providing this the the command line parameters:

```
--rfsimulator.options chanmod
```

gNB

The main parameters to cope with the large NTN propagation delay are cellSpecificKoffset, ta-Common, ta-CommonDrift and the ephemeris data (satellite position and velocity vectors).

The parameter `cellSpecificKoffset_r17` is the scheduling offset used for the timing relationships that are modified for NTN (see TS 38.213). The unit of the field Koffset is number of slots for a given subcarrier spacing of 15 kHz.

The parameter `ta-Common-r17` is used to provide the propagation delay between the reference point (at the gNB) and the satellite. The granularity of ta-Common is 4.072 × 10^(-3) µs. Values are given in unit of corresponding granularity.

The parameter `ta-CommonDrift-r17` indicates the drift rate of the common TA. The granularity of ta-CommonDrift is 0.2 × 10^(-3) µs/s. Values are given in unit of corresponding granularity.

The satellite position and velocity vartors are provided using the following parameters:

`positionX-r17`, `positionY-r17`, `positionZ-r17`: X, Y, Z coordinate of satellite position state vector in ECEF. Unit is meter. Step of 1.3 m. Actual value = field value * 1.3.

`velocityVX-r17`, `velocityVY-r17`, `velocityVZ-r17`: X, Y, Z coordinate of satellite velocity state vector in ECEF. Unit is meter/second. Step of 0.06 m/s. Actual value = field value * 0.06.

These parameters can be provided to the gNB in the conf file in the section `servingCellConfigCommon`:

```
...
# GEO satellite
      cellSpecificKoffset_r17                                 = 478;
      ta-Common-r17                                           = 58629666; # 238.74 ms
      positionX-r17                                           = 0;
      positionY-r17                                           = 0;
      positionZ-r17                                           = 32433846;
      velocityVX-r17                                          = 0;
      velocityVY-r17                                          = 0;
      velocityVZ-r17                                          = 0;
# LEO satellite
#      cellSpecificKoffset_r17                                 = 40;
#      ta-Common-r17                                           = 4634000; # 18.87 ms
#      ta-CommonDrift-r17                                      = -230000; # -46 µs/s
#      positionX-r17                                           = 0;
#      positionY-r17                                           = -2166908; #
        -2816980.4 m
#      positionZ-r17                                           =  4910784; #
        6384019.2 m
#      velocityVX-r17                                          = 0;
#      velocityVY-r17                                          = 115246; # 6914.76 m/s
#      velocityVZ-r17                                          =  50853; # 3051.18 m/s
...
```

Besides this, some timers, e.g. `sr_ProhibitTimer_v1700`, `t300`, `t301` and `t319`, in the conf file section `gNBs.[0].TIMERS` might need to be extended for GEO satellites.

```
...
    TIMERS :
    {
      sr_ProhibitTimer       = 0;
      sr_TransMax            = 64;
      sr_ProhibitTimer_v1700 = 512;
      t300                   = 2000;
      t301                   = 2000;
      t319                   = 2000;
    };
...
```

To improve the achievable UL and DL throughput in conditions with large RTT (esp. GEO satellites), there is a feature defined in REL17 to disable HARQ feedback. This allows to reuse HARQ processes immediately, but it breaks compatibility with UEs not supporting this REL17 feature. To enable this feature, the `disable_harq` flag has to be added to the gNB conf file in the section `gNBs.[0]`

```
...
    sib1_tda     = 5;
    min_rxtxtime = 6;
    disable_harq = 1; // <--

    servingCellConfigCommon = (
    {
...
```

The settings for a transparent GEO satellite scenario are already provided in the file `ci-scripts/conf_files/gnb.sa.band254.u0.25prb.rfsim.ntn.conf`. Using this conf file, an example gNB command for FDD, 5 MHz BW, 15 kHz SCS, transparent GEO satellite 5G NR NTN is this:

```
cd cmake_targets
sudo ./ran_build/build/nr-softmodem -O ../ci-scripts/conf_files/
        gnb.sa.band254.u0.25prb.rfsim.ntn.conf --rfsim
```

To configure NTN gNB with 32 HARQ processes in downlink and uplink, add these settings in conf files under section `gNBs.[0]`

```
...
    num_dlharq = 32;
    num_ulharq = 32;
...
```

To simulate a LEO satellite channel model with rfsimulator in UL (DL is simulated at the UE side) either the `channelmod` section as shown before has to be added to the gNB conf file, or a channelmod conf file has to be included like this:

```
@include "channelmod_rfsimu_LEO_satellite.conf"
```

The settings for a transparent LEO satellite scenario are already provided in the file `ci-scripts/conf_files/gnb.sa.band254.u0.25prb.rfsim.ntn-leo.conf`. Using this conf file, an example gNB command for FDD, 5 MHz BW, 15 kHz SCS, trasparent LEO satellite 5G NR NTN is this:

```
cd cmake_targets
sudo ./ran_build/build/nr-softmodem -O ../ci-scripts/conf_files/
        gnb.sa.band254.u0.25prb.rfsim.ntn-leo.conf --rfsim
```

NR UE

At UE side, only few parameters have to be provided, as the UE receives most relevant parameters via SIB19 from the gNB. But to calculate the UE specific TA, the UE position

has to be provided in the `ue.conf` file. Also the LEO channel model has to be configured, e.g. by using an `@include` statement, just like on the gNB side:

```
...
position0 = {
    x = 0.0;
    y = 0.0;
    z = 6377900.0;
}

@include "channelmod_rfsimu_LEO_satellite.conf"
```

So an example NR UE command for FDD, 5MHz BW, 15 kHz SCS, transparent GEO satellite 5G NR NTN is this:

```
cd cmake_targets
sudo ./ran_build/build/nr-uesoftmodem -O ../targets/PROJECTS/GENERIC-NR-5GC/CONF/ue.conf --
        band 254 -C 2488400000 --CO -873500000 -r 25 --numerology 0 --ssb 60 --rfsim --
        rfsimulator.prop_delay 238.74
```

For LEO satellite scenarios, the parameter `--ntn-initial-time-drift` must be provided via command line, as the UE needs this value to compensate for the time drift during initial sync, before SIB19 was received. This parameter provides the drift rate of the complete DL timing (incl. feeder link and service link) in µs/s. Also, to perform an autonomous TA update based on the DL drift, the boolean parameter `--autonomous-ta` should be added in case of a LEO satellite scenario.

For LEO satellite scenario we assume the LO to be very accurate and the main FO contribution comes from Doppler shift. Therefore, we use the command line parameter `--cont-fo-comp 2` to continuously compensate the DL Doppler and pre-compensate the UL Doppler. The initial Doppler frequency offset must be provided via command line with the parameter `--initial-fo`.

For other information on optional NR UE command line options, please refer here.

So an example NR UE command for FDD, 5MHz BW, 15 kHz SCS, transparent LEO satellite 5G NR NTN is this:

```
cd cmake_targets
sudo ./ran_build/build/nr-uesoftmodem -O ../targets/PROJECTS/GENERIC-NR-5GC/CONF/ue.conf --
        band 254 -C 2488400000 --CO -873500000 -r 25 --numerology 0 --ssb 60 --rfsim --
        rfsimulator.prop_delay 20 --rfsimulator.options chanmod --time-sync-I 0.1 --ntn-
        initial-time-drift -46 --autonomous-ta --initial-fo 57340 --cont-fo-comp 2
```

Specific OAI modes

phy-test setup with OAI UE

The OAI UE can also be used in front of a OAI gNB without the support of eNB or EPC and circumventing random access. In this case both gNB and eNB need to be run with the `--phy-test` flag. At the gNB this flag does the following

- it reads the RRC configuration from the configuration file
- it encodes the RRCConfiguration and the RBconfig message and stores them in the binary files `rbconfig.raw` and `reconfig.raw` in the current directory
- the MAC uses a pre-configured allocation of PDSCH and PUSCH with randomly generated payload instead of the standard scheduler. The options `-m`, `-l`, `-t`, `-M`, `-T`, `-D`, and `-U` can be used to configure this scheduler. See `./nr-softmodem -h` for more information.

At the UE, the `--phy-test` flag will read the binary files `rbconfig.raw` and `reconfig.raw` from the current directory and process them. If you wish to provide a different path for these files, please use the options `--reconfig-file` and `--rbconfig-file`.

```
sudo ./nr-softmodem -O ../../../targets/PROJECTS/GENERIC-LTE-EPC/CONF/
       gnb.band78.tm1.106PRB.usrpn300.conf --phy-test
```

```
sudo ./nr-uesoftmodem --phy-test [--reconfig-file ../../../ci-scripts/rrc-files/reconfig.raw
       --rbconfig-file ../../../ci-scripts/rrc-files/rbconfig.raw]
```

In summary:

- If you are running on the same machine and launched the 2 executables (`nr-softmodem` and `nr-uesoftmodem`) from the same directory, nothing has to be done.
- If you launched the 2 executables from 2 different folders, just point to the location where you launched the `nr-softmodem`:
    - `sudo ./nr-uesoftmodem --rfsim --phy-test --reconfig-file /the/path/where/you/launched/nr-softmodem/reconfig-file --rbconfig-file /the/path/where/you/launched/nr-softmodem/rbconfig-file --rfsimulator.serveraddr <TARGET_GNB_INTERFACE_ADDRESS>`

- If you are not running on the same machine, you need to **COPY** the two raw files
  - `scp usera@machineA:/the/path/where/you/launched/nr-softmodem/r*config.raw userb@machineB:/the/path/where/you/will/launch/nr-uesoftmodem/`
  - Obviously this operation should be done before launching the `nr-uesoftmodem` executable.

In phy-test mode it is possible to mimic the reception of UE Capabilities at gNB through the command line parameter `--uecap_file`. Refer to the UE Capabilities section for more details.

noS1 setup with OAI UE

Instead of randomly generated payload, in the phy-test mode we can also inject/receive user-plane traffic over a TUN interface. This is the so-called noS1 mode.

The noS1 mode is applicable to both gNB/UE, and enabled by passing `--noS1` as an option. The gNB/UE will open a TUN interface which the interface names and IP addresses `oaitun_enb1`/10.0.1.1, and `oaitun_ue1`/10.0.1.2, respectively. You can then use these interfaces to send traffic, e.g.,

```
iperf -sui1 -B 10.0.1.2
```

to open an iperf server on the UE side, and

```
iperf -uc 10.0.1.2 -B 10.0.1.1 -i1 -t10 -b1M
```

to send data from the gNB down to the UE.

Note that this does not work if both interfaces are on the same host. We recommend to use two different hosts, or at least network namespaces, to route traffic through the gNB/UE tunnel.

This option is only really helpful for phy-test/do-ra (see below) modes, in which the UE does not connect to a core network. If the UE connects to a core network, it receives an IP address for which it automatically opens a network interface.

do-ra setup with OAI

The do-ra flag is used to ran the NR Random Access procedures in contention-free mode. Currently OAI implements the RACH process from Msg1 to Msg3.

In order to run the RA, the `--do-ra` flag is needed for both the gNB and the UE.

In do-ra mode it is possible to mimic the reception of UE Capabilities at gNB through the command line parameter `--uecap_file`. Refer to the UE Capabilities section for more details.

To run using the RFsimulator:

```
sudo ./nr-softmodem -O ../../../targets/PROJECTS/GENERIC-LTE-EPC/CONF/
        gnb.band78.tm1.106PRB.usrpn300.conf --do-ra --rfsim
sudo ./nr-uesoftmodem --do-ra --rfsim --rfsimulator.serveraddr 127.0.0.1
```

Using USRPs:

```
sudo ./nr-softmodem -O ../../../targets/PROJECTS/GENERIC-LTE-EPC/CONF/
        gnb.band78.tm1.106PRB.usrpn300.conf --do-ra
```

On a separate machine:

```
sudo ./nr-uesoftmodem --do-ra
```

Run OAI with SDAP & Custom DRBs

To run OAI gNB with SDAP, simply include `--gNBs.[0].enable_sdap 1` to the binary's arguments.

The DRB creation is dependent on the 5QI. If the 5QI corresponds to a GBR Flow it assigns a dedicated data radio bearer. The Non-GBR flows use a shared data radio bearer.

To hardcode the DRBs for testing purposes, simply add `--gNBs.[0].drbs x` to the binary's arguements, where `x` is the number of DRBs, along with SDAP. The hardcoded DRBs will be treated like GBR Flows. Due to code limitations at this point the max. number of DRBs is 4.

IF setup with OAI

OAI is also compatible with Intermediate Frequency (IF) equipment. This allows to use RF front-end that with arbitrary frequencies bands that do not comply with the standardised 3GPP NR bands.

To configure the IF frequencies it is necessary to use two command-line options at UE side:

- `if_freq`, downlink frequency in Hz
- `if_freq_off`, uplink frequency offset in Hz

Accordingly, the following parameters must be configured in the RUs section of the gNB configuration file:

- `if_freq`
- `if_offset`

Run OAI with custom DL/UL arbitrary frequencies

The following example uses DL frequency 2169.080 MHz and UL frequency offset -400 MHz, with a configuration file for band 66 (FDD) at gNB side.

On two separate machines with USRPs, run:

```
sudo ./nr-softmodem -O ../../../targets/PROJECTS/GENERIC-LTE-EPC/CONF/
        gnb.band66.tm1.106PRB.usrpx300.conf
sudo ./nr-uesoftmodem --if_freq 2169080000 --if_freq_off -400000000
```

5G gNB MIMO configuration

In order to enable DL-MIMO in OAI 5G softmodem, the prerequisite is to have `do_CSIRS = 1` in the configuration file. This allows the gNB to schedule CSI reference signal and to acquire from the UE CSI measurements to be able to schedule DLSCH with MIMO.

The following step is to set the number of PDSCH logical antenna ports. These needs to be larger or equal to the maximum number of MIMO layers requested (for 2-layer MIMO it is necessary to have at least two logical antenna ports).

This image shows an example of gNB 5G MIMO logical antenna port configuration. It has to be noted that logical antenna ports might not directly correspond to physical antenna ports and each logical antenna port might consist of a sub-array of antennas.

In 5G the basic element is a dual-polarized antenna, therefore the minimal DL MIMO setup with two logical antenna ports would consist of two cross-polarized antenna elements. In a single panel configuration, as the one in the picture, this element can be repeated vertically and/or horizontally to form an equi-spaced 1D or 2D array. The values N1 and N2 represent the number of antenna ports in the two dimensions and the supported configurations are specified in Section 5.2.2.2.1 of TS 38.214.

The DL logical antenna port configuration can be selected through configuration file. `pdsch_AntennaPorts_N1` can be used to set N1 parameter, `pdsch_AntennaPorts_N2` to set N2 and `pdsch_AntennaPorts_XP` to set the cross-polarization configuration (1 for single pol, 2 for cross-pol). To be noted that if XP is 1 but N1 and/or N2 are larger

than 1, this would result in a non-standard configuration and the PMI selected would be the identity matrix regardless of CSI report. The default value for each of these parameters is 1. The total number of PDSCH logical antenna ports is the multiplication of those 3 parameters.

Finally the number of TX physical antenna in the RU part of the configuration file, `nb_tx`, should be equal or larger than the total number of PDSCH logical antenna ports.

It is possible to limit the number supported DL MIMO layers via RRC configuration, e.g. to a value lower than the number of logical antenna ports configured, by using the configuration file parameter `maxMIMO_layers`.

Example of configuration file with parameters for 2-layer MIMO

OAI gNB LTTng Tracing Setup Guide

Overview

This guide will walk you through setting up tracing for an OpenAirInterface (OAI) gNB (gNodeB) using LTTng (Linux Trace Toolkit Next Generation) and Babeltrace.

What is LTTng and Why Use It?

LTTng, or Linux Trace Toolkit Next Generation, is a powerful logging framework designed for Linux systems. It provides low-overhead tracing capabilities, allowing developers to monitor and analyze system behavior in real-time without significant performance impact. LTTng offers several advantages:

- **Low Overhead**: LTTng introduces minimal overhead to the system, making it suitable for use in production environments without affecting system performance.
- **Customizable**: LTTng allows users to define custom tracepoints in their applications, providing fine-grained control over what events to trace and collect.
- **Scalability**: It can scale to large distributed systems, making it suitable for tracing complex software stacks across multiple nodes.

Prerequisites

- Ubuntu system

Note: only LTTng 2.3.8 is supported.

Building OAI gNB

1. **Clean and Build OAI gNB with LTTng:**

   1.1 Install Dependencies

   ```
   ./build_oai --ninja -I --clean --enable-LTTNG
   ```

   1.2 Build gNB and nrUE

   ```
   ./build_oai --ninja --gNB --nrUE -w SIMU --enable-LTTNG
   ```

Setting up LTTng

1. **Start LTTng Session and Relay:**

```
sudo lttng-sessiond -d
sudo lttng-relayd -d
```

2. **Create Live LTTng Session:**

```
sudo lttng create my-session --live --set-url=net://127.0.0.1
```

3. **Enable gNB Trace Events:**

```
sudo lttng enable-event --userspace OAI:gNB
```

4. **Start LTTng Tracing:**

```
sudo lttng start
```

Running the gNB

1. **Run gNB:**

```
./$binary_path -O $configuration_file PARALLEL_SINGLE_THREAD --rfsimulator.serveraddr
        server --rfsim -E
```

Verifying Tracepoints

1. **List Active Tracepoints:**

```
sudo lttng list -u
```

Possible Output:

```
UST events:
-------------

PID: 1154722 - Name: /home/firecell/Desktop/FirecellRepos/firecellrd-oai5g-ran/
        cmake_targets/ran_build/build/nr-softmodem
      OAI:gNB (loglevel: TRACE_DEBUG_FUNCTION (12)) (type: tracepoint)
```

Analyzing Traces

1. **Install Babeltrace:**

```
sudo apt-get install babeltrace
```

2. **Capture Trace Logs Live:**
   - To learn the full path of the trace session:

     ```
     babeltrace --input-format=lttng-live net://localhost
     ```

   - Trace logs using the full path:

     ```
     bash babeltrace --input-format=lttng-live net://localhost/host/
     firecell-XPS-15-9530/my-session
     ```

   Possible Output:

   ``` [19:35:32.181608002] (+2.664882127) firecell-XPS-15-9530 OAI:gNB: { cpu_id = 10 }, { MODNAME = "OAI-NR_MAC info", EVENTID = -1, SFN = -1, SLOT = -1, FUNCTION = "gNB_dlsch_ulsch_scheduler", LINE = 246, MSG = "Frame.Slot 0.0\n" }

```
3. **Capture Trace Logs Offline:**

    - Create an offline trace session with a specified output directory:
```

   bash sudo lttng create offline_session –output=/home/trace_offline/ ```

   - Enable gNB trace events:

     ```
     sudo lttng enable-event --userspace OAI:gNB
     ```

   - Start capturing trace logs:

     `bash sudo lttng start`

   - Stop the trace capture:

     `bash sudo lttng stop`

   - Destroy the trace session:

     `bash sudo lttng destroy`

   - Use Babeltrace to analyze the captured trace logs:

     `bash sudo babeltrace /home/trace_offline/`
     ```

openair1/PHY/TOOLS/readme.md File Reference

tuning_and_security

This document outlines some information specific to system tuning for running all executables (`nr-softmodem`, `lte-softmodem`, `nr-uesoftmodem`, `lte-uesoftmodem`, `nr-cuup`, called softmodems for short in the following). Also, it explains which Linux capabilities are required to run, and how to run without sudo.

Table of Contents

Performance Tuning

Please also refer to the advanced configuration in the tutorial, which groups many tips and tricks.

CPU

OAI used to try to set the minimum CPU-DMA latency to 2 us by writing to `/dev/cpu_dma_latency`. However, it is unclear if this has a significant effect. Further, in containerized workloads, it might not be possible to set this at all. We assume the user of OAI sets this value before starting OAI. See the Linux kernel documentation for more information.

OAI used to try to set the minimum frequency of cores running OAI to the maximum frequency. We assume that the user sets CPU frequency policies accordingly; in fact, tutorials generally suggest to either set a performance CPU governor, or disable any sleep states at all. Hence, setting these low-level parameters seems useless, and we assume the user of OAI disables sleep states before starting OAI.

To disable all sleep states, simply run the following:

```
sudo cpupower idle-set -D0
```

Sometimes, dependencies might not be installed, in which case you should install what `cpupower` asks you to install. Note that this is not persistent.

See the Linux kernel documentation for more information.

You can disable hyper-threading in the BIOS.

You can disable KPTI Protections for Spectre/Meltdown for more performance. **This is a security risk.** Add `mitigations=off nosmt` in your grub config and update grub.

Ethernet-based Radios

For ethernet-based radios, such as AW2S, some USRPs, and 7.2 radios, increase the ringbuffers to a maximum. Read on interface `<fronthaul-interface-name>` using option `-g`, then set it with `-G`:

```
ethtool -g <fronthaul-interface-name>
ethtool -G <fronthaul-interface-name> rx <maximum-rx-value> tx <maixmum-tx-value>
```

Also, you can increase the kernel's default and maximum read and write socket buffer sizes to a high values, e.g., 134217728:

```
sudo sysctl -n -e -q -w net.core.rmem_default=134217728
sudo sysctl -n -e -q -w net.core.rmem_max=134217728
sudo sysctl -n -e -q -w net.core.wmem_default=134217728
sudo sysctl -n -e -q -w net.core.wmem_max=134217728
```

Capabilities

Historically, all softmodems are executed as `root`, typically using `sudo`. This remains a possibility, but we do not recommend it due to security considerations. Rather, consider giving specific capabilities as outlined below. Read `capabilities(7)` (`man 7 capabilities`) for more information on each of the below capabilities.

Note that we tested this using 5G executables; 4G should work, but have not been tested as extensively. If in doubt, run eNB/lteUE using `sudo`. The below comments on capabilities apply in general as well; however, 4G executable might not warn about missing capabilities or just fail.

Refer to any of the docker-compose files under `ci-scripts/` to see how to give capabilities in docker. If you run from source, you can use `setcap` to mark a process to run with specific capabilities. For instance, you can add the "general capabilities" as described further below on the files like this:

```
sudo setcap cap_sys_nice+ep ./nr-softmodem
sudo setcap cap_ipc_lock+ep ./nr-softmodem
sudo setcap cap_sys_nice+ep ./nr-uesoftmodem
sudo setcap cap_ipc_lock+ep ./nr-uesoftmodem
sudo setcap cap_net_admin+ep ./nr-uesoftmodem
```

To make only temporary changes to capabilities, use `capsh`. It needs to be started with all capabilities, so needs `sudo`. To drop all capabilities, issue:

```
sudo -E capsh --drop="cap_sys_nice,cap_chown,cap_dac_read_search,cap_fowner,cap_fsetid,\
cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_linux_immutable,cap_net_bind_service,\
cap_net_broadcast,cap_net_admin,cap_net_raw,cap_ipc_lock,cap_ipc_owner,cap_sys_module,\
cap_sys_rawio,cap_sys_chroot,cap_sys_ptrace,cap_sys_pacct,cap_sys_admin,cap_sys_boot,\
cap_sys_resource,cap_sys_time,cap_sys_tty_config,cap_mknod,cap_lease,cap_audit_write,\
cap_audit_control,cap_setfcap,cap_mac_override,cap_mac_admin,cap_syslog,cap_wake_alarm,\
cap_block_suspend,cap_audit_read,cap_perfmon,cap_bpf,cap_checkpoint_restore" --print -- \
  -c "/absolute/path/to/nr-softmodem -O /absolute/path/to/config.conf"
```

(For readability, the command has been separated onto multiple lines through `\`). To run with `SYS_NICE`, remove the first capability (`cap_sys_nice`) from the list of dropped capabilities.

General capabilities

- `SYS_NICE`: required by all softmodems to assign a higher priority to threads to increase the likelihood that the Linux scheduler schedules a thread for CPU time. Also, in some configurations, CPU pinning is employed to ensure consistent performance, also known as setting a CPU affinity.

  This capability is necessary by default when running any softmodem, for setting real-time requirements on processing threads. To allow to run without these requirements, the softmodems check if `SYS_NICE` is available, and skips any thread priority and affinity settings if the capability is not available. This allows to run any softmodem without root privileges in RFsim; you can see this by either observing a corresponding warning at the beginning of the execution, or the fact that no affinity/default priority is set for new threads.

- `IPC_LOCK`: OAI tries to lock all its virtual address space into RAM, preventing that memory from being paged to the swap area. If this capability is not present, a warning is printed.
- `NET_ADMIN`: Required at the UE to open the interface `oaitun_ue1` or eNB/gNB in noS1 mode. 5G executables will throw an error if they cannot create or modify an interface, but will continue running. It is therefore possible to run without this capability, but you cannot inject any traffic into the system. 4G executables might need this requirement, and possibly fail.

Capabilities with DPDK

Additionally to the "general capabilities" above, you need these capabilities when running with 7.2 fronthaul, which uses the xran library with a dependency on DPDK:

- `IPC_LOCK` (becomes mandatory with DPDK)
- `SYS_RESOURCE`
- `NET_RAW`
- `SYS_ADMIN`: This is required by DPDK when using IOVA PA (Physical Address) mode to read `/proc/self/pagemaps`. However, if DPDK EAL is configured to use IOVA VA (Virtual Address) mode, this capability is no longer required.

Capabilities with UHD

You don't need any additional capabilities for UHD beyond the "general capabilities" for performance outlined above. Make sure that the USB device(s) are readable and writable by the current user (e.g., `uhd_usrp_probe` can be executed by a non-root user).

Capabilities with AW2S

You don't need any additional capabilities for AW2S beyond the "general capabilities" for performance outlined above.

Other radios

Other radios have not been tested. If they do not work without additional capabilities beyond the "general capabilities", please file a bug report.

handover-tutorial

This tutorial explains how to perform handovers. For the moment, only F1 handovers are supported.

[[TOC]]

Considered setup for F1 handover

We consider one CU and two DUs, connected over F1. The UE is initially connected over the radio interface ("Uu") to DU0. Via movement to a new position ("new pos"), it will trigger an event such that the CU triggers a handover of the UE from DU0 to DU1. Alternatively, a manual trigger can do the same.

F1 Handover setup

Steps to run F1 handover with OAI UE

Measurement reporting and processing of RRC Reconfiguration for Mobility are not completed at the UE. Nevertheless, it is possible to make simple handover tests without any radio setup, on a single PC, with the OAI UE, in RFsimulator.

Build with telnet support

Since the UE does not support any measurement reporting, it cannot trigger a handover on its own; it has to be triggered manually through telnet. Thus, build both gNB and UE as well as activate the build of telnet to that purpose:

```
./build_oai --ninja --nrUE --gNB --build-lib telnetsrv
```

Run the setup

This tutorial assumes you have a core network running; **refer to the**corresponding tutorial" if this is not the case yet. We will use the TDD configuration files in the repository for the CU as well for DU0 and <a href="../targets/PROJECTS/GENERIC-NR-5GC/CONF/gnb-du.sa.band78.106prb.rfsim.pci1.conf" >DU1. Note how the DUs differ in their DU ID (for identification at the CU), nr cellid (global

identification), physical cell ID (identification through UE), frequency (limitation at OAI UE), and IP address.

Make sure that using RFsimulator and the CU and each DU, you can achieve a full connection of the UE (independently, i.e., running one DU a time). Once this is done, follow below steps to trigger a handover:

Start the CU including telnet support:

```
sudo ./nr-softmodem -O ../../../targets/PROJECTS/GENERIC-NR-5GC/CONF/gnb-cu.sa.f1.conf --tel
```

Start DU0:

```
sudo ./nr-softmodem --rfsim -O ../../../targets/PROJECTS/GENERIC-NR-5GC/CONF/gnb-du.sa.band7
```

Start the UE, and let it connect completely:

```
sudo ./nr-uesoftmodem -C 3450720000 -r 106 --numerology 1 --ssb 516 -O <config>  --rfsim --r
```

Note how the RFsimulator roles have been switched, and RFsim server is at the UE side; this is important. Replace `<config>` with the UE configuration matching your core. If you followed the CN and oaiUE tutorials, you can remove `-O <config>` and replace it with `--uicc0.imsi 001010000000001`.

Once the UE is connected, start DU1:

```
sudo ./nr-softmodem --rfsim -O ../../../targets/PROJECTS/GENERIC-NR-5GC/CONF/gnb-du.sa.band7
```

Once DU1 is online, you can trigger a handover by issuing this command

```
echo ci trigger_f1_ho | nc 127.0.0.1 9090 && echo
```

You should see how the UE switches from one DU to another. See additional information further below.

A number of remarks:

1. It is important that you start DU0, UE, DU1 in order, and having UE connect to DU0 before starting DU1. This is because we don't employ any channel emulation, and the UE could not decode the SIB1 of DU0 to connect.

1. The RFsimulator roles are switched. Typically, the gNB RFsim acts as the server and the UE as a client. However, RFsim is limited to one server with multiple clients.

Since the UE should be able to connect to both DUs, it has to act as the server, and both DUs are a client.

1. If you see errors `could not open a socket` and/or `Could not start the RF device`, this means that RFsim could not be started. Handover will not work; please refer to the preceding point to fix this (i.e., run the UE as the RFsim server).

1. In some cases, if the RFsim server is at the UE, the whole system can block; in this case, stop UE and all DUs and restart (the CU can keep running).

Additional information to the manual HO trigger

You can trigger the handover manually by logging in through telnet:

```
telnet 127.0.0.1 9090
```

and then manually typing `ci trigger_f1_ho`.

The command using `nc` (`netcat`) above triggers handovers directly from bash. It does the same as logging in through telnet otherwise, and is a shorthand.

The full command is

```
ci trigger_f1_ho [cu-ue-id]
```

`cu-ue-id` is optional. If only one UE context is present in the RRC, it will trigger the handover for this UE in a round-robin fashion across all DUs. If there is only one DU, the handover request will be rejected. Similarly, if there are multiple UE contexts present at RRC, the handover will be rejected, and you have to manually type the CU UE ID. You can see a list of all UEs in the file `nrRRC_stats.log` that is printed periodically in the working directory of the CU.

Steps to run F1 handover with COTS UE

You can do handover across DUs with a COTS UE. Note that these DUs should be separated by at least multiple meters to ensure that the UE will receive different signal strengths when moving between cells.

We only support intra-frequency handovers yet. We have verified with USRPs only, although other radios should work as well.

For UEs, we verified Quectel modules and iPhones. Note, though, that not all phones might work; for instance, we did not achieve handovers with a OnePlus Nord, yet.

Steps

First, make sure that you can run both DUs with the CU independently. Use the same radio hardware for both radios to ensure that both cells can be received equally good by the UE.

In order to enable handovers (triggered by the UE), you have to configure the neighbour relation of the DUs at the CU. To do so, proceed as follows:

1. To simplify filling the right values in the neighbour configuration, you can rely on the information the CU has about both DUs. Start the CU and both DUs. Navigate to the directory from which you started the CU, and print RRC statistics:

   ```
   cat nrRRC_stats.log
   ```

1. Fill in the `neighbour-config.conf` configuration file as shown below, and `@include` it in the CU file.

1. Start the CU and both DUs.

1. Bring the phone close to one cell, and leave flight mode. It should connect to the DU to which it is closer.

1. Move the UE towards the other DU; it should trigger an "A3 event" (Neighbour Becomes Better than Serving), and the CU will trigger the handover to the other DU.

The output on the terminal should be the same as with RFsim. If no handover is triggered:

- Make sure that both DUs use the same hardware.
- Make sure that the UE sees both cells. For instance, you can switch to flight mode, go closer to the other DU, and switch off flight mode – the UE should connect to that second UE.
- We did not manage handover with every phone yet – make sure you use one of the list provided above.

You can also force a handover through telnet as described above. (In fact, the decision about a handover is always at the network-side, the UE only "assists" through measurements telling the CU that one DU is stronger than others. Hence, "forcing" a handover just means that you manually trigger the handover, instead of waiting for UE measurement report.)

Example neighbour configuration

Below is an example neighbour configuration. It is based on this DU information gathered from `nrRRC_stats.log` at the CU:

```
[1] DU ID 3585 (gNB-in-docker) assoc_id 4161: nrCellID 11111111, PCI 1, SSB ARFCN 643296
    TDD: band 78 ARFCN 642024 SCS 30 (kHz) PRB 106
[2] DU ID 3584 (gNB-in-docker) assoc_id 4163: nrCellID 12345678, PCI 0, SSB ARFCN 643296
    TDD: band 78 ARFCN 642024 SCS 30 (kHz) PRB 106
```

Note how both DUs have one cell on the same frequency and the same radio configuration. From this, fill the neighbour list as shown below.

Concretely, the first cell is `12345678` (on DU `[2]`), and it has `11111111` (on DU `[1]`) as its neighbour; hence in the first block, you fill `physical_cellId` and other values for DU `[1]`, and vice versa.

The below configuration further enables periodic measurements, A2 event ("Serving becomes worse than threshold"), and A3 events ("Neighbour Becomes Better than Serving"). The A2 event can be disabled by setting `enable = 0`. A3 events cannot be disabled as of now. Further, the A3 events can be made specific to cells; `cell_id = -1` means "any cell".

```
neighbour_list = (
  {
    nr_cellid = 12345678;
    neighbour_cell_configuration = (
      {
        gNB_ID = 0xe01;
        nr_cellid = 11111111;
        physical_cellId = 1;
        absoluteFrequencySSB = 643296;
        subcarrierSpacing = 1; #30 KHz
        plmn = { mcc = 001; mnc = 01; mnc_length = 2};
        tracking_area_code = 1;
      }
    )
  },
  {
    nr_cellid = 11111111;
    neighbour_cell_configuration = (
      {
        gNB_ID = 0xe00;
        nr_cellid = 12345678;
        physical_cellId = 0;
        absoluteFrequencySSB = 643296;
        subcarrierSpacing = 1; #30 KHz
        plmn = { mcc = 001; mnc = 01; mnc_length = 2};
        tracking_area_code = 1;
      }
    )
  }
 );
```

```
nr_measurement_configuration = {
  Periodical = {
    enable = 1;
    includeBeamMeasurements = 1;
    maxNrofRS_IndexesToReport = 4;
  };

  A2 = {
    enable = 1;
    threshold = 60;
    timeToTrigger = 1;
  };

  A3 = ({
    cell_id = -1; #Default
    offset = 10;
    hysteresis = 0;
    timeToTrigger = 1
  })
};
```

Handovers triggers and NTN

Typically, in terrestrial networks, channel measurements as well as criteria such as load in base stations, is used to determine when and where to handover a UE.

NTN

Doppler spreading and time selectivity of the channel are already a challenge for conventional terrestrial networks. However, in the context of non-terrestrial networks (NTN), and low-earth orbit (LEO) systems, the satellites can have speeds up 7.56 km/s, which is much faster than 0.14 km/s of a high-speed train in terrestrial networks. Also, the delays in LEO are more varied and longer, and the path loss is larger, because the communication distances is up to 10 times longer than in terrestrial networks. Moreover, downlink in LEO presents a high interference from adjacent satellite beams, and all these features contribute to reduced received signal strength variation in these networks compared to terrestrial networks. Typically, the criteria used in algorithms presented in conventional terrestrial networks to trigger a handover from a gNB to another one is based on signal strength measurements (cf., A3 event above). However, the reduced received signal strength variation in NTNs make these algorithms inefficient for LEO systems. Therefore, algorithms with criteria for handover triggering that address the specifics of LEO systems are crucial for an efficient handover processing that ensures a robust communication with low dropping probability. Some criteria for handovers in NTN are as follows:

- **Measurement-based triggering**: This method is based on signal strength measurement, and as stated above, it may not be efficient. The triggering thresholds and which measurement events to use as triggers, as reference signal

received power (RSRP), reference signal received quality (RSRQ), or received signal strength indicator (RSSI), should be configured. This method relies on UE estimates and established channel estimation techniques, however it would require neighbouring cell lists which can be hard because the fast-moving of satellites leads to a fast cell coverage deviation.

- **Location-based triggering**: This method is based on UE and satellite location, which can be applied jointly (or not) with another trigger as the measurement-based trigger. For instance, for a deterministic satellite movement, it is possible to predict the configure triggering condition, and the initial association of the UE can be performed based on the distance with the nearest satellite, because we can know the location of the UE and NTN satellite to compute the distance.
- **Elevation angles of source and target cells based triggering**: This method is similar to the previous one, but it is based on the largest elevation angle.
- **Time/timer-based triggering**: This method uses triggering conditions based on UTC time or a timer-based solution, which can also be applied jointly (or not) with another trigger as the measurement-based trigger. The timer-based handover trigger considers the deterministic satellite movement to predict the time duration for which the satellite's footprint covers a certain zone.
- **Timing advance value-based triggering**: This method uses the timing advance value (independently or jointly with another trigger) to trigger a handover to the target cell. It is appropriate to overcome the Random-Access preamble reception issue, where the UE needs to pre-compensate the instant which sends the preamble. However, UEs with GNSS support are required to perform this method.

Simple location/time-based trigger

A location-based handover trigger, somewhat aligned with 3GPP Rel.17, taking advantage of deterministic satellite movement, can be implemented whereby it is assumed that the locations of the UE and the NTN satellite are known. From a practical point of view, this means that we know how long it takes to perform the handover, and therefore we only need to implement a timer.

It is possible to modify the source code to trigger a time-based handover, or combine with other methods. If all what is required is a trigger every 15 seconds, you can also resort to the telnet-based handover trigger above, and run in a terminal:

```
while true; do
  echo ci trigger_f1_ho | nc -N 127.0.0.1 9090 && echo
  sleep 15
done
```

ORAN_FHI7.2_Tutorial

OAI 7.2 Fronthaul Interface 5G SA Tutorial

Table of Contents

[[TOC]]

Prerequisites

The hardware on which we have tried this tutorial:

Hardware (CPU,RAM)	Operating System (kernel)	NIC (Vendor,Driver,Firmware)
Intel(R) Xeon(R) Gold 6354 36-Core, 128GB	RHEL 9.2 (5.14.0-284.18.1.rt14.303.el9_2.x86_64)	Intel X710, i40e, 9.20 0x8000d95e 22.0.9
Intel(R) Xeon(R) Gold 6354 36-Core, 128GB	Ubuntu 22.04.3 LTS (5.15.0-1033-realtime)	Intel X710, i40e, 9.00 0x8000cfeb 21.5.9
AMD EPYC 9374F 32-Core Processor, 128GB	Ubuntu 22.04.2 LTS (5.15.0-1038-realtime)	Intel E810 ,ice, 4.00 0x8001184e 1.3236.0

NOTE:

- These are not minimum hardware requirements. This is the configuration of our servers.
- The NIC card should support hardware PTP time stamping.
- If you are using Intel servers then use only Ice Lake or newer generations. In case of AMD use only 4th generation, Genoa or newer.
- If you try on any other server apart from the above listed, then choose a desktop/server with clock speed higher than 3.0 GHz and `avx512` capabilities.
- This tutorial gives few instructions for Arm targets, but DU execution on Arm systems is yet not functional.

This feature is intended to enable experiments and future improvements on Arm systems.

NICs we have tested so far:

Vendor	Firmware Version
Intel X710	9.20 0x8000d95e 22.0.9
Intel E810-XXV	4.00 0x8001184e 1.3236.0
E810-C	4.20 0x8001784e 22.0.9
Intel XXV710	6.02 0x80003888

Note:

- With AMD servers/desktop machines with PCIe 5.0 we have only used E810 cards.
- If you are using Mellanox NIC, please be aware that DPDK can't bind the NIC as vfio-pci. Instead it must be bind with mlx driver.

PTP enabled switches and grandmaster clock we have in are lab:

Vendor	Software Version
CISCO C93180YC-FX3	10.2(4)
Fibrolan Falcon-RX/812/G	8.0.25.4
Qulsar Qg2 (Grandmaster)	12.1.27

S-Plane synchronization is mandatory. S-plane support is done via `ptp4l` and `phc2sys`. Make sure your version matches.

Software	Software Version
`ptp4l`	3.1.1
`phc2sys`	3.1.1

We have only verified LLS-C3 configuration in our lab, i.e. using an external grandmaster, a switch as a boundary clock, and the gNB/DU and RU. We haven't tested any RU without S-plane. We tested the category A radio units listed below.

Vendor	Software Version
VVDN LPRU	03-v3.0.5
LiteON RU	01.00.08/02.00.03/02.00.10

|Benetel 650 |RAN650-1v1.0.4-dda1bf5|RAN650-1v1.2.2-2fa04bc| |Benetel 550 | RAN550-1v1.0.4-605a25a|RAN550-1v1.2.2-2fa04bc| |Foxconn RPQN |v3.1.15q.551_rc10 |

Tested libxran releases:

Vendor
`oran_e_maintenance_release_v1.0`
`oran_f_release_v1.0`

Note: The libxran driver of OAI identifies the above E release version as "5.1.0" (E is fifth letter, then 1.0), and the above F release as "6.1.0".

Configure your server

1. Disable Hyperthreading (HT) in your BIOS. In all our servers HT is always disabled.
2. We recommend you to start with a fresh installation of OS (either RHEL or Ubuntu). You have to install realtime kernel on your OS (Operating System). Based on your OS you can search how to install realtime kernel.
3. Install realtime kernel for your OS
4. Change the boot commands based on the below section. They can be performed either via `tuned` or via manually building the kernel

CPU allocation

This section is important to read, regardless of the operating system you are using.

Your server could be:

- One NUMA node (See one NUMA node example): all the processors are sharing a single memory system.
- Two NUMA nodes (see two NUMA nodes example): processors are grouped in 2 memory systems.
    - Usually the even (ie `0,2,4,...`) CPUs are on the 1st socket
    - And the odd (ie (`1,3,5,...`) CPUs are on the 2nd socket

DPDK, OAI and kernel threads require to be properly allocated to extract maximum real-time performance for your use case.

1. **NOTE**: Currently the default OAI 7.2 configuration file requires isolated **CPUs 0,2,4** for DPDK/libXRAN, **CPU 6** for `ru_thread`, **CPU 8** for `L1_rx_thread` and **CPU 10** for `L1_tx_thread`. It is preferrable to have all these threads on the same socket.

2. Allocating CPUs to the OAI nr-softmodem is done using the `--thread-pool` option. Allocating 4 CPUs is the minimal configuration but we recommend to allocate at least **8** CPUs. And they can be on a different socket as the DPDK threads.
3. And to avoid kernel preempting these allocated CPUs, it is better to force the kernel to use un-allocated CPUs.

Let summarize for example on a `32-CPU` single NUMA node system, regardless of the number of sockets:

Applicative Threads	Allocated CPUs
XRAN DPDK usage	0,2,4
OAI `ru_thread`	6
OAI `L1_rx_thread`	8
OAI `L1_tx_thread`	10
OAI `nr-softmodem`	1,3,5,7,9,11,13,15
kernel	16-31

In below example we have shown the output of `/proc/cmdline` for two different servers, each of them have different number of NUMA nodes. **Be careful in isolating the CPUs in your environment.** Apart from CPU allocation there are additional parameters which are important to be present in your boot command.

Modifying the `linux` command line usually requires to edit string `GRUB_CMDLINE_LINUX` in `/etc/default/grub`, run a `grub` command and reboot the server.

- Set parameters `isolcpus`, `nohz_full` and `rcu_nocbs` with the list of CPUs to isolate for XRAN.
- Set parameter `kthread_cpus` with the list of CPUs to isolate for kernel.

Set the `tuned` profile to `realtime`. If the `tuned-adm` command is not installed then you have to install it. When choosing this profile you have to mention the isolated cpus in `/etc/tuned/realtime-variables.conf`. By default this profile adds

```
skew_tick=1 isolcpus=managed_irq,domain,<cpu-you-choose>
intel_pstate=disable nosoftlockup
```

in the boot command. **Make sure you don't add them while changing `/etc/default/grub`.**

```
tuned-adm profile realtime
```

Checkout anyway the examples below.

One NUMA Node

Below is the output of `/proc/cmdline` of a single NUMA node server,

```
NUMA:
  NUMA node(s):          1
  NUMA node0 CPU(s):     0-31
```

```
isolcpus=0-15 nohz_full=0-15 rcu_nocbs=0-15 kthread_cpus=16-31 rcu_nocb_poll nosoftlockup
        default_hugepagesz=1GB hugepagesz=1G hugepages=20 amd_iommu=on iommu=pt
        mitigations=off skew_tick=1 selinux=0 enforcing=0 tsc=reliable nmi_watchdog=0
        softlockup_panic=0 audit=0 vt.handoff=7
```

Example taken for AMD EPYC 9374F 32-Core Processor

Two NUMA Nodes

Below is the output of `/proc/cmdline` of a two NUMA node server,

```
NUMA:
  NUMA node(s):          2
  NUMA node0 CPU(s):     0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34
  NUMA node1 CPU(s):     1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35
```

```
mitigations=off usbcore.autosuspend=-1 intel_iommu=on intel_iommu=pt selinux=0 enforcing=0
        nmi_watchdog=0 softlockup_panic=0 audit=0 skew_tick=1 isolcpus=managed_irq,domain,
        0,2,4,6,8,10,12,14,16 nohz_full=0,2,4,6,8,10,12,14,16
        rcu_nocbs=0,2,4,6,8,10,12,14,16 rcu_nocb_poll intel_pstate=disable nosoftlockup
        cgroup_disable=memory mce=off hugepagesz=1G hugepages=40 hugepagesz=2M hugepages=0
        default_hugepagesz=1G isolcpus=managed_irq,domain,0,2,4,6,8,10,12,14
        kthread_cpus=18-35 intel_pstate=disable nosoftlockup tsc=reliable
```

Example taken for Intel(R) Xeon(R) Gold 6354 CPU @ 3.00GHz

Common

Configure your servers to maximum performance mode either via OS or in BIOS. If you want to disable CPU sleep state via OS then use the below command:

```
# to disable
sudo cpupower idle-set -D 0
#to enable
sudo cpupower idle-set -E
```

The above information we have gathered either from O-RAN documents or via our own experiments. In case you would like to read the O-RAN documents then here are the links:

1. O-RAN-SC O-DU Setup Configuration
2. O-RAN Cloud Platform Reference Designs 2.0,O-RAN.WG6.CLOUD-REF-v02.00,February 2021

PTP configuration

Note: You may run OAI with O-RAN 7.2 Fronthaul without a RU attached (e.g. for benchmarking). In such case, you can skip PTP configuration and go to DPDK section.

1. You can install `linuxptp` rpm or debian package. It will install ptp4l and phc2sys.

```
#RHEL
sudo dnf install linuxptp -y
#Ubuntu
sudo apt install linuxptp -y
```

Once installed you can use this configuration file for ptp4l (`/etc/ptp4l.conf`). Here the clock domain is 24 so you can adjust it according to your PTP GM clock domain

```
[global]
domainNumber            24
slaveOnly               1
time_stamping           hardware
tx_timestamp_timeout    1
logging_level           6
summary_interval        0
#priority1               127

[your_PTP_ENABLED_NIC]
network_transport       L2
hybrid_e2e              0
```

You need to increase `tx_timestamp_timeout` to 50 or 100 for Intel E-810. You will see that in the logs of ptp.

Create the configuration file for ptp4l (`/etc/sysconfig/ptp4l`)

```
OPTIONS="-f /etc/ptp4l.conf"
```

Create the configuration file for phc2sys (`/etc/sysconfig/phc2sys`)

```
OPTIONS="-a -r -r -n 24"
```

The service of ptp4l (`/usr/lib/systemd/system/ptp4l.service`) should be configured as below:

```
[Unit]
Description=Precision Time Protocol (PTP) service
After=network-online.target
Wants=network-online.target

[Service]
Type=simple
EnvironmentFile=-/etc/sysconfig/ptp4l
ExecStart=/usr/sbin/ptp4l $OPTIONS

[Install]
WantedBy=multi-user.target
```

and service of phc2sys (`/usr/lib/systemd/system/phc2sys.service`) should be configured as below:

```
[Unit]
Description=Synchronize system clock or PTP hardware clock (PHC)
After=ntpdate.service ptp4l.service

[Service]
Type=simple
EnvironmentFile=-/etc/sysconfig/phc2sys
ExecStart=/usr/sbin/phc2sys $OPTIONS

[Install]
WantedBy=multi-user.target
```

Debugging PTP issues

You can see these steps in case your ptp logs have erorrs or `rms` reported in `ptp4l` logs is more than 100ms. Beware that PTP issues may show up only when running OAI and XRAN. If you are using the `ptp4l` service, have a look back in time in the journal: `journalctl -u ptp4l.service -S <hours>:<minutes>:<seconds>`

1. Make sure that you have `skew_tick=1` in `/proc/cmdline`
2. For Intel E-810 cards set `tx_timestamp_timeout` to 50 or 100 if there are errors in ptp4l logs
3. Other time sources than PTP, such as NTP or chrony timesources, should be disabled. Make sure they are enabled as further below.
4. Make sure you set `kthread_cpus=<cpu_list>` in `/proc/cmdline` .
5. If `rms` or `delay` in `ptp4l` or `offset` in `phc2sys` logs remain high then you can try pinning the `ptp4l` and `phc2sys` processes to an isolated CPU.

```
#to check there is NTP enabled or not
timedatectl | grep NTP
#to disable
timedatectl set-ntp false
```

DPDK (Data Plane Development Kit)

Download DPDK version 20.11.9.

```
# on debian
sudo apt install wget xz-utils libnuma-dev
# on Fedora/RHEL
sudo dnf install wget xz numactl-devel
cd
wget http://fast.dpdk.org/rel/dpdk-20.11.9.tar.xz
```

DPDK Compilation and Installation

```
# Installing meson : it should pull ninja-build and compiler packages
# on debian
sudo apt install meson
# on Fedora/RHEL
sudo dnf install meson
tar xvf dpdk-20.11.9.tar.xz && cd dpdk-stable-20.11.9

meson build
ninja -C build
sudo ninja install -C build
```

Verify the installation is complete

Check if the LD cache contains the DPDK Shared Objects after update:

```
sudo ldconfig -v | grep rte_
    librte_fib.so.0.200.2 -> librte_fib.so.0.200.2
    librte_telemetry.so.0.200.2 -> librte_telemetry.so.0.200.2
    librte_compressdev.so.0.200.2 -> librte_compressdev.so.0.200.2
    librte_gro.so.20.0 -> librte_gro.so.20.0.2
    librte_mempool_dpaa.so.20.0 -> librte_mempool_dpaa.so.20.0.2
    librte_distributor.so.20.0 -> librte_distributor.so.20.0.2
    librte_rawdev_dpaa2_cmdif.so.20.0 -> librte_rawdev_dpaa2_cmdif.so.20.0.2
    librte_mempool.so.20.0 -> librte_mempool.so.20.0.2
    librte_pmd_octeontx2_crypto.so.20.0 -> librte_pmd_octeontx2_crypto.so.20.0.2
    librte_common_cpt.so.20.0 -> librte_common_cpt.so.20.0.2
....
```

You may not have the `/usr/local/lib`, `/usr/local/lib64`, or custom DPDK installation paths in the `LD_LIBRARY_PATH`. In this case, add it as below; if you installed into a custom

```
sudo echo "/usr/local/lib" > /etc/ld.so.conf.d/local-lib.conf
```

```
sudo echo "/usr/local/lib64" >> /etc/ld.so.conf.d/local-lib.conf
sudo ldconfig
sudo ldconfig -v | grep rte_
```

Check if the PKG-CONFIG tool discovers the libraries:

```
pkg-config --libs libdpdk --static
```

► Possible output

```
-lrte_node -lrte_graph -lrte_bpf -lrte_flow_classify -lrte_pipeline -lrte_table -lrte_port -
        lrte_fib -lrte_ipsec -lrte_vhost -lrte_stack -lrte_security -lrte_sched -
        lrte_reorder -lrte_rib -lrte_rawdev -lrte_pdump -lrte_power -lrte_member -lrte_lpm -
        lrte_latencystats -lrte_kni -lrte_jobstats -lrte_ip_frag -lrte_gso -lrte_gro -
        lrte_eventdev -lrte_efd -lrte_distributor -lrte_cryptodev -lrte_compressdev -
        lrte_cfgfile -lrte_bitratestats -lrte_bbdev -lrte_acl -lrte_timer -lrte_hash -
        lrte_metrics -lrte_cmdline -lrte_pci -lrte_ethdev -lrte_meter -lrte_net -lrte_mbuf -
        lrte_mempool -lrte_rcu -lrte_ring -lrte_eal -lrte_telemetry -lrte_kvargs -Wl,--
        whole-archive -lrte_common_cpt -lrte_common_dpaax -lrte_common_iavf -
        lrte_common_octeontx -lrte_common_octeontx2 -lrte_bus_dpaa -lrte_bus_fslmc -
        lrte_bus_ifpga -lrte_bus_pci -lrte_bus_vdev -lrte_bus_vmbus -lrte_mempool_bucket -
        lrte_mempool_dpaa -lrte_mempool_dpaa2 -lrte_mempool_octeontx -lrte_mempool_octeontx2
        -lrte_mempool_ring -lrte_mempool_stack -lrte_pmd_af_packet -lrte_pmd_ark -
        lrte_pmd_atlantic -lrte_pmd_avp -lrte_pmd_axgbe -lrte_pmd_bond -lrte_pmd_bnxt -
        lrte_pmd_cxgbe -lrte_pmd_dpaa -lrte_pmd_dpaa2 -lrte_pmd_e1000 -lrte_pmd_ena -
        lrte_pmd_enetc -lrte_pmd_enic -lrte_pmd_failsafe -lrte_pmd_fm10k -lrte_pmd_i40e -
        lrte_pmd_hinic -lrte_pmd_hns3 -lrte_pmd_iavf -lrte_pmd_ice -lrte_pmd_igc -
        lrte_pmd_ixgbe -lrte_pmd_kni -lrte_pmd_liquidio -lrte_pmd_memif -lrte_pmd_netvsc -
        lrte_pmd_nfp -lrte_pmd_null -lrte_pmd_octeontx -lrte_pmd_octeontx2 -lrte_pmd_pfe -
        lrte_pmd_qede -lrte_pmd_ring -lrte_pmd_sfc -lrte_pmd_softnic -lrte_pmd_tap -
        lrte_pmd_thunderx -lrte_pmd_vdev_netvsc -lrte_pmd_vhost -lrte_pmd_virtio -
        lrte_pmd_vmxnet3 -lrte_rawdev_dpaa2_cmdif -lrte_rawdev_dpaa2_qdma -lrte_rawdev_ioat
        -lrte_rawdev_ntb -lrte_rawdev_octeontx2_dma -lrte_rawdev_octeontx2_ep -
        lrte_rawdev_skeleton -lrte_pmd_caam_jr -lrte_pmd_dpaa_sec -lrte_pmd_dpaa2_sec -
        lrte_pmd_nitrox -lrte_pmd_null_crypto -lrte_pmd_octeontx_crypto -
        lrte_pmd_octeontx2_crypto -lrte_pmd_crypto_scheduler -lrte_pmd_virtio_crypto -
        lrte_pmd_octeontx_compress -lrte_pmd_qat -lrte_pmd_ifc -lrte_pmd_dpaa_event -
        lrte_pmd_dpaa2_event -lrte_pmd_octeontx2_event -lrte_pmd_opdl_event -
        lrte_pmd_skeleton_event -lrte_pmd_sw_event -lrte_pmd_dsw_event -
        lrte_pmd_octeontx_event -lrte_pmd_bbdev_null -lrte_pmd_bbdev_turbo_sw -
        lrte_pmd_bbdev_fpga_lte_fec -lrte_pmd_bbdev_fpga_5gnr_fec -Wl,--no-whole-archive -
        lrte_node -lrte_graph -lrte_bpf -lrte_flow_classify -lrte_pipeline -lrte_table -
        lrte_port -lrte_fib -lrte_ipsec -lrte_vhost -lrte_stack -lrte_security -lrte_sched -
        lrte_reorder -lrte_rib -lrte_rawdev -lrte_pdump -lrte_power -lrte_member -lrte_lpm -
        lrte_latencystats -lrte_kni -lrte_jobstats -lrte_ip_frag -lrte_gso -lrte_gro -
        lrte_eventdev -lrte_efd -lrte_distributor -lrte_cryptodev -lrte_compressdev -
        lrte_cfgfile -lrte_bitratestats -lrte_bbdev -lrte_acl -lrte_timer -lrte_hash -
        lrte_metrics -lrte_cmdline -lrte_pci -lrte_ethdev -lrte_meter -lrte_net -lrte_mbuf -
        lrte_mempool -lrte_rcu -lrte_ring -lrte_eal -lrte_telemetry -lrte_kvargs -Wl,-
        Bdynamic -pthread -lm -ldl
```

If DPDK was installed into `/usr/local/lib`, `/usr/local/lib64`, or another custom path, you have to point to the right directory with `PKG_CONFIG_PATH`, for instance:

```
export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/usr/local/lib64/pkgconfig/
```

```
pkg-config --libs libdpdk --static
```

If you want to de-install this version of DPDK

Go back to the version folder you used to build and install

```
cd ~/dpdk-stable-20.11.9
sudo ninja deinstall -C build
```

Build OAI-FHI gNB

Clone OAI code base in a suitable repository, here we are cloning in `~/openairinterface5g` directory,

```
git clone https://gitlab.eurecom.fr/oai/openairinterface5g.git ~/openairinterface5g
cd ~/openairinterface5g/
```

Build ORAN Fronthaul Interface Library

Download ORAN FHI DU library, checkout the correct version, and apply the correct patch (available in `oai_folder/cmake_targets/tools/oran_fhi_integration_patches`).

E release

```
git clone https://gerrit.o-ran-sc.org/r/o-du/phy.git ~/phy
cd ~/phy
git checkout oran_e_maintenance_release_v1.0
git apply ~/openairinterface5g/cmake_targets/tools/oran_fhi_integration_patches/E/
        oaioran_E.patch
```

F release

```
git clone https://gerrit.o-ran-sc.org/r/o-du/phy.git ~/phy
cd ~/phy
git checkout oran_f_release_v1.0
git apply ~/openairinterface5g/cmake_targets/tools/oran_fhi_integration_patches/F/
        oaioran_F.patch
```

Compile the fronthaul interface library by calling `make` and the option `XRAN_LIB_SO=1` to have it build a shared object. Note that we provide two environment variables `RTE_SDK` for the path to the source tree of DPDK, and `XRAN_DIR` to set the path to the fronthaul library.
For building for a Arm target, set as well the environment variable `TARGET=armv8`. DU

execution on Arm systems is yet not functional. This feature is intended to enable experiments and future improvements on Arm systems.

Note: you need at least gcc-11 and g++-11.

```
cd ~/phy/fhi_lib/lib
make clean
RTE_SDK=~/dpdk-stable-20.11.9/ XRAN_DIR=~/phy/fhi_lib make XRAN_LIB_SO=1 # E release
WIRELESS_SDK_TOOLCHAIN=gcc RTE_SDK=~/dpdk-stable-20.11.9/ XRAN_DIR=~/phy/fhi_lib make
        XRAN_LIB_SO=1 # F release
...
[AR] build/libxran.so
./build/libxran.so
```

The shared library object `~/phy/fhi_lib/lib/build/libxran.so` must be present before proceeding.

For Arm targets only: Install the Arm RAN Acceleration library

DU execution on Arm systems is yet not functional. This feature is intended to enable experiments and future improvements on Arm systems.

Clone, configure and build ArmRAL:

Note: Use option `-DCMAKE_INSTALL_PREFIX=<install-dir>` of cmake to set the installation directory of ArmRAL to `<install-dir>`. If you omit this option, ArmRAL is installed into `/usr/local`.

```
git clone https://git.gitlab.arm.com/networking/ral.git ~/ral
cd ~/ral
git checkout armral-25.01
mkdir build
cd build
cmake -GNinja -DBUILD_SHARED_LIBS=On ../
ninja
```

Once ArmRAL is configured at your convenience and built, you can install it:

```
ninja install
```

Build OAI gNB

You can now proceed building OAI. You build it the same way as for other radios, providing option `-t oran_fhlib_5g`. Additionally, you need to provide it the location of the FH library: `--cmake-opt -Dxran_LOCATION=PATH`. Note that since we cannot use

`~` here, we resort to `$HOME`, which is equivalent. Finally, if you needed to define `PKG_CONFIG_PATH` previously, you need to do so now, too.

```
# You should have already cloned above
cd ~/openairinterface5g/cmake_targets
# if you installed DPDK in a custom path as described above
export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/usr/local/lib64/pkgconfig/
./build_oai -I  # if you never installed OAI, use this command once before the next line
./build_oai --gNB --ninja -t oran_fhlib_5g --cmake-opt -Dxran_LOCATION=$HOME/phy/fhi_lib/lib
```

You can optionally check that everything has been linked properly with

```
ldd ran_build/build/liboran_fhlib_5g.so
```

► Possible output

```
#check if all the libraries are properly linked to liboai_transpro.so
ldd ran_build/build/liboran_fhlib_5g.so
    linux-vdso.so.1 (0x00007fffb459e000)
    librte_node.so.21 => /usr/local/lib64/librte_node.so.21 (0x00007fd358690000)
    librte_graph.so.21 => /usr/local/lib64/librte_graph.so.21 (0x00007fd358685000)
    librte_bpf.so.21 => /usr/local/lib64/librte_bpf.so.21 (0x00007fd358672000)
    librte_flow_classify.so.21 => /usr/local/lib64/librte_flow_classify.so.21
        (0x00007fd35866c000)
    librte_pipeline.so.21 => /usr/local/lib64/librte_pipeline.so.21 (0x00007fd35862f000)
    librte_table.so.21 => /usr/local/lib64/librte_table.so.21 (0x00007fd358612000)
    librte_port.so.21 => /usr/local/lib64/librte_port.so.21 (0x00007fd3585f8000)
    librte_fib.so.21 => /usr/local/lib64/librte_fib.so.21 (0x00007fd3585e9000)
...
    libm.so.6 => /lib64/libm.so.6 (0x00007fd357eb1000)
    libnuma.so.1 => /lib64/libnuma.so.1 (0x00007fd357ea1000)
    libc.so.6 => /lib64/libc.so.6 (0x00007fd357c98000)
    /lib64/ld-linux-x86-64.so.2 (0x00007fd3587c7000)
    libelf.so.1 => /lib64/libelf.so.1 (0x00007fd357c7d000)
    libz.so.1 => /lib64/libz.so.1 (0x00007fd357c61000)
```

Note that you might also call cmake directly instead of using `build_oai`:

```
cd ~/openairinterface5g
mkdir build && cd build
cmake .. -GNinja -DOAI_FHI72=ON -Dxran_LOCATION=$HOME/phy/fhi_lib/lib
ninja nr-softmodem oran_fhlib_5g params_libconfig
```

Note that in tags 2025.w06 and prior, the FHI72 driver used polling to wait for the next slot. This is inefficient as it burns CPU time, and has been replaced with a more efficient mechanism. Nevertheless, if you experience problems that did not occur previously, it is possible to re-enable polling, either with `build_oai` like this

```
./build_oai --gNB --ninja -t oran_fhlib_5g --cmake-opt -Dxran_LOCATION=$HOME/phy/fhi_lib/lib
```

or with `cmake` like so

```
cmake .. -GNinja -DOAI_FHI72=ON -Dxran_LOCATION=$HOME/phy/fhi_lib/lib -DOAI_FHI72_USE_POLLIN
ninja oran_fhlib_5g
```

Configuration

RU and DU configurations have a circular dependency: you have to configure DU MAC address in the RU configuration and the RU MAC address, VLAN and Timing advance parameters in the DU configuration.

Note: You may run OAI with O-RAN 7.2 Fronthaul without a RU attached (e.g. for benchmarking). In such case, skip RU configuration and only configure Network Interfaces, DPDK VFs and OAI configuration by using arbitrary values for RU MAC addresses and VLAN tags.

Configure the RU

Contact the RU vendor and get the configuration manual to understand the below commands. The below configuration only corresponds to the RU firmware version indicated at the start of this document. If your firmware version does not correspond to the indicated version, then please don't try these commands.

NOTE: Please understand all the changes you are doing at the RU, especially if you are manipulating anything related to output power.

Benetel 650

The OAI configuration file `gnb-du.sa.band77.273prb.fhi72.4x4-benetel650.conf` corresponds to:

- TDD pattern `DDDSU`, 2.5ms
- Bandwidth 100MHz
- MTU 9600
- 4TX4R

RU configuration

After switching on the radio or rebooting, wait for the radio bring up to complete, which you can follow using `tail -f /tmp/logs/radio_status`. Once you will see `[INFO] Radio bringup complete`, you can configure the RU via editing `/etc/ru_config.cfg`

```
cat /etc/ru_config.cfg

mimo_mode=1_2_3_4_4x4
downlink_scaling=0
prach_format=short
compression=static_compressed
lf_prach_compression_enable=true
cplane_per_symbol_workaround=disabled
cuplane_dl_coupling_sectionID=disabled
flexran_prach_workaround=disabled
dl_ul_tuning_special_slot=0xfd00000
```

Benetel 550

The OAI configuration file `gnb.sa.band78.273prb.fhi72.4x4-benetel550.conf` corresponds to:

- TDD pattern `DDDDDDDSUU`, 5ms
- Bandwidth 100MHz
- MTU 9600
- 4TX4R

RU configuration

After switching on the radio or rebooting, wait for the radio bring up to complete, which you can follow using `tail -f /tmp/logs/radio_status`. Once you will see `[INFO] Radio bringup complete`, you can configure the RU via editing `/etc/ru_config.cfg`

```
cat /etc/ru_config.cfg

mimo_mode=1_2_3_4_4x4
downlink_scaling=0
prach_format=short
compression=static_compressed
lf_prach_compression_enable=true
cplane_per_symbol_workaround=disabled
cuplane_dl_coupling_sectionID=disabled
flexran_prach_workaround=disabled
dl_tuning_special_slot=0x13b6
```

LITEON

The OAI configuration file `gnb.sa.band78.273prb.fhi72.4x4-liteon.conf` corresponds to:

- TDD pattern `DDDSU`, 2.5ms

- Bandwidth 100MHz
- MTU 1500
- MTU 9600: v02.00.10

RU configuration

SSH to the unit as user `user`. Write `enable` in the terminal to enter the configuration console; the password should be in the user guide. Use the command `show oru-status` to check the RU status. The output should be similar to:

```
# show oru-status
Sync State  : SYNCHRONIZED
RF State    : Ready
DPD         : Ready
DuConnected : notReady
```

Also, you can use `show running-config` to display the current RU configuration.

Once the RU is PTP synced, and RF state and DPD are `Ready`, write `configure terminal` to set:

- Center frequency
- Bandwidth
- Compression Bitwidth
- TX/RX attenuation
- PRACH eAxC IDs
- DU MAC address ...

The configuration mode example:

```
compression-bit 9 # set IQ bitwidth for PxSCH/PRACH
eAXC_id 4 5 6 7 # set PRACH eAxC IDs
jumboframe 1 # enable jumbo frame
...
```

VVDN LPRU

Version 3.x

The OAI configuration file `gnb.sa.band77.273prb.fhi72.4x4-vvdn.conf` corresponds to:

- TDD pattern `DDDSU`, 2.5ms

- Bandwidth 100MHz
- MTU 9600

RU configuration

Check in the RU user manual how to configure the center frequency. There are multiple ways to do it. We set the center frequency by editing `sysrepocfg` database. You can use `sysrepocfg --edit=vi -d running` to do the same. You can edit the `startup` database to make the center frequency change persistent.

To make any change in the RU, always wait for it to be PTP synced. You can check the sync status via `tail -f /var/log/synctimingptp2.log`.

To set the RU for 4TX and 4RX antennas with 9 bit compression, you need to create this XML file and apply it.

Login to the ru and create this file as `4x4-config.xml`.

```
<vvdn_config>
    <du_mac_address>YOUR-DU-MAC-ADDR</du_mac_address>
    <cu_plane_vlan>YOUR-RU-VLAN</cu_plane_vlan>
    <dl_compression_method>1</dl_compression_method>
    <dl_compression_value>9</dl_compression_value>
    <ul_compression_method>1</ul_compression_method>
    <ul_compression_value>9</ul_compression_value>
    <num_prb>273</num_prb>
    <prach_layer0_PCID>4</prach_layer0_PCID>
    <prach_layer1_PCID>5</prach_layer1_PCID>
    <prach_layer2_PCID>6</prach_layer2_PCID>
    <prach_layer3_PCID>7</prach_layer3_PCID>
    <pxsch_layer0_PCID>0</pxsch_layer0_PCID>
    <pxsch_layer1_PCID>1</pxsch_layer1_PCID>
    <pxsch_layer2_PCID>2</pxsch_layer2_PCID>
    <pxsch_layer3_PCID>3</pxsch_layer3_PCID>
</vvdn_config>
```

Execute the below commands on every restart

```
xml_parser 4x4-config.xml
## To enable prach compression
mw.l a0010024 1919 # format `<PRACH-comp-method><PRACH-compr-value><PUSCH-comp-
        method><PUSCH-compr-value>
## This will show the current configuration
/etc/scripts/lpru_configuration.sh
## Edit the sysrepo to ACTIVATE the carrier when you want to use the RU
## option 1 - activation by writing directly in register
mw.l a0050010 <YOUR-RU-VLAN>3 # e.g. VLAN = 4 => `mw.l a0050010 43`
## option 2 - activation via sysrepocfg command
sysrepocfg --edit=vi -d running
```

Metanoia RU

Version 2.0.6

The OAI configuration file `gnb.sa.band78.273prb.fhi72.4x4-metanoia.conf` corresponds to:

- TDD pattern `DDDSU`, 2.5ms (`DDDDDDDSUU`, 5ms, also supported)
- Bandwidth 100MHz
- 4TX4R

The RU configuration is stored in `/etc/rumanager.conf`. The required modifications:

1. `processing_element/vlan_id`
2. `processing_element/du_mac_address`
3. `low_level_tx_endpoint/compression_type` -> `STATIC`
4. `low_level_rx_endpoint/compression_type` -> `STATIC`
5. `low_level_rx_endpoint/compression/fs-offset` -> `8`
6. `center-of-channel-bandwidth` -> `3750000000`
7. `tx_gain_correction` -> tested with `6020` (please be careful to not fry the RU)
8. `rx_gain_correction` -> tested with `-903` (please be careful to not fry the RU)

At this stage, RU must be rebooted so the changes apply.

Foxconn RPQN RU

Version v3.1.15q.551_rc10

The OAI configuration file `gnb.sa.band78.273prb.fhi72.4X4-foxconn.conf` corresponds to:

- TDD pattern `DDDSU`, 2.5ms
- Bandwidth 100MHz
- MTU 9600

RU configuration

After switching on or rebooting the RU, the `/home/root/test/init_rrh_config_enable_cuplane` script should be run.

The RU configuration file is located in `/home/root/test/RRHconfig_xran.xml`. The required modifications:

1. `RRH_DST_MAC_ADDR`
2. `RRH_SRC_MAC_ADDR`
3. `RRH_EAXC_ID_TYPE1`
4. `RRH_EAXC_ID_TYPE3`
5. `RRH_CMPR_HDR_PRESENT` -> `0`
6. `RRH_C_PLANE_VLAN_TAG`
7. `RRH_U_PLANE_VLAN_TAG`
8. `RRH_LO_FREQUENCY_KHZ` -> `3750000, 0`
9. `RRH_DISABLE_USING_CAL_TABLES` -> `YES`
10. `RRH_TX_ATTENUATION` -> must be larger than 10dB
11. `RRH_RX_ATTENUATION` -> must be lower than 30dB

RU must be rebooted so the changes apply.

Note

- The RU was tested with the `2024.w30` tag of OAI.
- The measured throughput was **520 Mbps DL** and **40 Mbps UL**.
- With newer OAI versions, throughput degrades. This issue is currently under investigation.

Configure Network Interfaces and DPDK VFs

The 7.2 fronthaul uses the xran library, which requires DPDK. In this step, we need to configure network interfaces to send data to the RU, and configure DPDK to bind to the corresponding PCI interfaces. More specifically, in the following we use SR-IOV to create one or multiple virtual functions (VFs) through which Control plane (C plane) and User plane (U plane) traffic will flow. The following commands are not persistant, and have to be repeated after reboot.

In the following, we will use these short hands:

- `IF_NAME`: Physical network interface through which you can access the RU

- `VLAN` : the VLAN tag as recommended by the RU vendor
- `MTU` : this MTU must be higher than supported by the RU vendor due to additional ethernet header of 14 B and DPDK packet header `RTE_PKTMBUF_HEADROOM` of 128 B
- `DU_U_PLANE_MAC_ADD` : DU U plane MAC address
- `U_PLANE_PCI_BUS_ADD` : PCI bus address of the VF for U plane
- `DU_C_PLANE_MAC_ADD` : DU C plane MAC address
- `C_PLANE_PCI_BUS_ADD` : PCI bus address of the VF for C plane

In the configuration file, in option `fhi_72.dpdk_devices`, the first PCI address is for U-plane and the second for C-plane.

RU might support either one DU MAC address for both CU planes or two different. i.e. VVDN Gen3, Metanoia support only one, Benetel550 supports both cases

Note

- X710 NIC supports the same DU MAC address for multiple VFs
- E-810 NIC requires different DU MAC addresses for multiple VFs

If the RU vendor requires untagged traffic, remove the VLAN tagging in the below command and configure VLAN on the switch as "access VLAN". In case the MTU is different than 1500, you have to update the MTU on the switch interface as well.

Set maximum ring buffers:

As a first step, please set up the maximum allowed buffer size to your desired interface. To check the maximum value, please execute the following command:

```
sudo ethtool -g $IF_NAME
```

```
set -x
IF_NAME=<YOUR_PHYSICAL_INTERFACE_NAME>
MAX_RING_BUFFER_SIZE=<YOUR_PHYSICAL_INTERFACE_MAX_BUFFER_SIZE>

sudo ethtool -G $IF_NAME rx $MAX_RING_BUFFER_SIZE tx $MAX_RING_BUFFER_SIZE
```

Set the maximum MTU in the physical interface:

```
set -x
IF_NAME=<YOUR_PHYSICAL_INTERFACE_NAME>
MTU=<RU_MTU>

sudo ip link set $IF_NAME mtu $MTU
```

(Re-)create VF(s)

one VF

```
set -x
IF_NAME=<YOUR_PHYSICAL_INTERFACE_NAME>
DU_CU_PLANE_MAC_ADD=<YOUR_DU_CU_PLANE_MAC_ADDRESS>
VLAN=<RU_VLAN>
MTU=<RU_MTU>

sudo modprobe iavf
sudo sh -c 'echo 0 > /sys/class/net/$IF_NAME/device/sriov_numvfs'
sudo sh -c 'echo 1 > /sys/class/net/$IF_NAME/device/sriov_numvfs'
sudo ip link set $IF_NAME vf 0 mac $DU_CU_PLANE_MAC_ADD vlan $VLAN mtu $MTU spoofchk off #
        set CU planes PCI address
```

two VFs

```
set -x
IF_NAME=<YOUR_PHYSICAL_INTERFACE_NAME>
DU_U_PLANE_MAC_ADD=<YOUR_DU_U_PLANE_MAC_ADDRESS>
DU_C_PLANE_MAC_ADD=<YOUR_DU_C_PLANE_MAC_ADDRESS> # can be same as for U plane -> depends if
        the NIC supports the same MAC address
VLAN=<RU_VLAN>
MTU=<RU_MTU>

sudo modprobe iavf
sudo sh -c 'echo 0 > /sys/class/net/$IF_NAME/device/sriov_numvfs'
sudo sh -c 'echo 2 > /sys/class/net/$IF_NAME/device/sriov_numvfs'
sudo ip link set $IF_NAME vf 0 mac $DU_U_PLANE_MAC_ADD vlan $VLAN mtu $MTU spoofchk off #
        set U plane PCI address
sudo ip link set $IF_NAME vf 1 mac $DU_C_PLANE_MAC_ADD vlan $VLAN mtu $MTU spoofchk off #
        set C plane PCI address
```

After running the above commands, the kernel created VF(s) that have been assigned a PCI address under the same device and vendor ID. For instance, use `sudo lshw -c network -businfo` to get a list of PCI addresses and interface names, locate the PCI address of `$IF_NAME`, then use `lspci | grep Virtual` to get all virtual interfaces and use the ones with the same Device/Vendor ID parts (first two numbers).

► Example with two VFs

The machine in this example has an Intel X710 card. The interface <physical-interface> in question is `eno12409`. Running `lshw` gives:

```
$ sudo lshw -c network -businfo
Bus info          Device     Class          Description
========================================================
[...]
pci@0000:31:00.1  eno12409   network        Ethernet Controller X710 for 10GbE SFP+
```

```
[...]
```

We see the PCI address `31:00.1`. Listing the virtual interfaces through `lspci`, we get

```
$ lspci | grep Virtual
31:06.0 Ethernet controller: Intel Corporation Ethernet Virtual Function 700 Series (rev 02)
31:06.1 Ethernet controller: Intel Corporation Ethernet Virtual Function 700 Series (rev 02)
98:11.0 Ethernet controller: Intel Corporation Ethernet Adaptive Virtual Function (rev 02)
98:11.1 Ethernet controller: Intel Corporation Ethernet Adaptive Virtual Function (rev 02)
98:11.2 Ethernet controller: Intel Corporation Ethernet Adaptive Virtual Function (rev 02)
98:11.3 Ethernet controller: Intel Corporation Ethernet Adaptive Virtual Function (rev 02)
98:11.4 Ethernet controller: Intel Corporation Ethernet Adaptive Virtual Function (rev 02)
98:11.5 Ethernet controller: Intel Corporation Ethernet Adaptive Virtual Function (rev 02)
```

The hardware card `31:00.1` has two associated virtual functions `31:06.0` and `31:06.1`.

Bind VF(s)

Now, unbind any pre-existing DPDK devices, load the "Virtual Function I/O" driver `vfio_pci` or `mlx5_core`, and bind DPDK to these devices.

Bind one VF

```
set -x
CU_PLANE_PCI_BUS_ADD=<YOUR_CU_PLANE_PCI_BUS_ADDRESS>
DRIVER=<YOUR_DRIVER> # set to `vfio_pci` or `mlx5_core`, depending on your NIC

sudo /usr/local/bin/dpdk-devbind.py --unbind $CU_PLANE_PCI_BUS_ADD
sudo modprobe $DRIVER
sudo /usr/local/bin/dpdk-devbind.py --bind $DRIVER $CU_PLANE_PCI_BUS_ADD
```

Bind two VFs

```
set -x
U_PLANE_PCI_BUS_ADD=<YOUR_U_PLANE_PCI_BUS_ADDRESS>
C_PLANE_PCI_BUS_ADD=<YOUR_C_PLANE_PCI_BUS_ADDRESS>
DRIVER=<YOUR_DRIVER> # set to `vfio_pci` or `mlx5_core`, depending on your NIC

sudo /usr/local/bin/dpdk-devbind.py --unbind $U_PLANE_PCI_BUS_ADD
sudo /usr/local/bin/dpdk-devbind.py --unbind $C_PLANE_PCI_BUS_ADD
sudo modprobe $DRIVER
sudo /usr/local/bin/dpdk-devbind.py --bind $DRIVER $U_PLANE_PCI_BUS_ADD
sudo /usr/local/bin/dpdk-devbind.py --bind $DRIVER $C_PLANE_PCI_BUS_ADD
```

We recommand to put the above four steps into one script file to quickly repeat them.

► Example script for Benetel 550-A/650 with Intel X710 on host with two VFs

```
set -x
```

```
IF_NAME=eno12409
MAX_RING_BUFFER_SIZE=4096
MTU=9600
DU_U_PLANE_MAC_ADD=00:11:22:33:44:66
DU_C_PLANE_MAC_ADD=00:11:22:33:44:67
VLAN=3
U_PLANE_PCI_BUS_ADD=31:06.0
C_PLANE_PCI_BUS_ADD=31:06.1
DRIVER=vfio_pci

sudo ethtool -G $IF_NAME rx $MAX_RING_BUFFER_SIZE tx $MAX_RING_BUFFER_SIZE
sudo ip link set $IF_NAME mtu $MTU
sudo modprobe iavf
sudo sh -c 'echo 0 > /sys/class/net/$IF_NAME/device/sriov_numvfs'
sudo sh -c 'echo 2 > /sys/class/net/$IF_NAME/device/sriov_numvfs'
sudo ip link set $IF_NAME vf 0 mac $DU_U_PLANE_MAC_ADD vlan $VLAN mtu $MTU spoofchk off #
        set U plane PCI address
sudo ip link set $IF_NAME vf 1 mac $DU_C_PLANE_MAC_ADD vlan $VLAN mtu $MTU spoofchk off #
        set C plane PCI address
sleep 1
sudo /usr/local/bin/dpdk-devbind.py --unbind $U_PLANE_PCI_BUS_ADD
sudo /usr/local/bin/dpdk-devbind.py --unbind $C_PLANE_PCI_BUS_ADD
sudo modprobe $DRIVER
sudo /usr/local/bin/dpdk-devbind.py --bind $DRIVER $U_PLANE_PCI_BUS_ADD
sudo /usr/local/bin/dpdk-devbind.py --bind $DRIVER $C_PLANE_PCI_BUS_ADD
```

Configure OAI gNB

Beware in the following section to let in the range of isolated cores the parameters that should be (i.e. `L1s.L1_rx_thread_core`, `L1s.L1_tx_thread_core`, `RUs.ru_thread_core`, `fhi_72.io_core` and `fhi_72.worker_cores`)

Sample configuration files for OAI gNB, specific to the manufacturer of the radio unit, are available at:

1. LITEON RU: `gnb.sa.band78.273prb.fhi72.4x4-liteon.conf`
2. VVDN RU: `gnb.sa.band77.273prb.fhi72.4x4-vvdn.conf`
   `gnb.sa.band77.106prb.fhi72.4x4-vvdn.conf`
   `gnb.sa.band77.273prb.fhi72.2x2-vvdn.conf`
3. Benetel 650 RU: `gnb-du.sa.band77.273prb.fhi72.4x4-benetel650.conf`
4. Benetel 550 RU: `gnb.sa.band78.273prb.fhi72.4x4-benetel550.conf`
   `gnb.sa.band78.273prb.fhi72.4x2-benetel550.conf`
5. Metanoia RU: `gnb.sa.band78.273prb.fhi72.4x4-metanoia.conf`

Edit the sample OAI gNB configuration file and check following parameters:

- `gNBs` section
  - The PLMN section shall match the one defined in the AMF
  - `amf_ip_address` shall be the correct AMF IP address in your system
  - `GNB_IPV4_ADDRESS_FOR_NG_AMF` shall match your gNB N2 interface IP address
  - `GNB_IPV4_ADDRESS_FOR_NGU` shall match your gNB N3 interface IP address
  - `prach_ConfigurationIndex`
  - `prach_msg1_FrequencyStart`
  - Adjust the frequency, bandwidth and SSB position
- `L1s` section
  - Set an isolated core for L1 thread `L1_rx_thread_core`, in our environment we are using CPU 8
  - Set an isolated core for L1 thread `L1_tx_thread_core`, in our environment we are using CPU 10
  - `phase_compensation` should be set to 0 to disable when it is performed in the RU and set to 1 when it should be performed on the DU side
  - `tx_amp_backoff_dB` controls the output level of OAI with respect to a full-scale output. The exact value for `tx_amp_backoff_dB` should be obtained from the O-RU documentation. This documentation typically includes detailed sections on Downlink (DL) signal scaling and gain setup. **Warning:** Exceeding the recommended power limits may permanently damage the RU.
- `RUs` section
  - Set an isolated core for RU thread `ru_thread_core`, in our environment we are using CPU 6
- `fhi_72` (FrontHaul Interface) section: this config follows the structure that is employed by the xRAN library (`xran_fh_init` and `xran_fh_config` structs in the code):
  - `dpdk_devices`: PCI addresses of NIC VFs binded to the DPDK (not the physical NIC but the VFs, use `lspci | grep Virtual`) in the format

`{VF-U-plane, VF-C-plane}`; if one VF used per RU, U and C planes will share the same VF => depends on the RU capabilities

- `system_core`: absolute CPU core ID for DPDK control threads, it should be an isolated core, in our environment we are using CPU 0 (`rte_mp_handle`, `eal-intr-thread`, `iavf-event-thread`)
- `io_core`: absolute CPU core ID for XRAN library, it should be an isolated core, in our environment we are using CPU 4
- `worker_cores`: array of absolute CPU core IDs for XRAN library, they should be isolated cores, in our environment we are using CPU 2
- `ru_addr`: RU U- and C-plane MAC-addresses (format `UU:VV:WW:XX:YY:ZZ`, hexadecimal numbers)
- `mtu`: Maximum Transmission Unit for the RU, specified by RU vendor; either 1500 or 9600 B (Jumbo Frames); if not set, 1500 is used
- `file_prefix`: used to specify a unique prefix for shared memory and files created by multiple DPDK processes; if not set, default value of `wls_0` is used
- `dpdk_mem_size`: the huge page size that should be pre-allocated by DPDK for NUMA node 0; by default, this is 8192 MiB (corresponding to 8 huge pages à 1024 MiB each, see above). In the current implementation, you cannot preallocate memory on NUMA nodes other than 0; in this case, set this to 0 (no pre-allocation) and so that DPDK will allocate it on-demand on the right NUMA node.
- `dpdk_iova_mode`: Specifies DPDK IO Virtual Address (IOVA) mode:
  - `PA`: IOVA as Physical Address (PA) mode, where DPDK IOVA memory layout corresponds directly to the physical memory layout.
  - `VA`: IOVA as Virtual Address (VA) mode, where DPDK IOVA addresses do not follow the physical memory layout. Uses IOMMU to remap physical memory. Requires kernel support and IOMMU for address translation.
  - If not specified, default value of "PA" is used (for backwards compabilibity; it was hardcoded to PA in the past). However, we recommend using "VA" mode as it offers several benefits. For a detailed explanation of DPDK IOVA, including the advantages and disadvantages of each mode, refer to Memory in DPDK
- `owdm_enable`: used for eCPRI One-Way Delay Measurements; it depends if the RU supports it; if not set to 1 (enabled), default value is 0 (disabled)

- `fh_config`
  - DU delay profile (`T1a` and `Ta4`): pairs of numbers `(x, y)` specifying minimum and maximum delays
  - `ru_config` : RU-specific configuration:
    - `iq_width` : Width of DL/UL IQ samples: if 16, no compression, if <16, applies compression
    - `iq_width_prach` : Width of PRACH IQ samples: if 16, no compression, if <16, applies compression
  - `prach_config` : PRACH-specific configuration
    - `eAxC_offset` : PRACH antenna offset; if not set, default value of `N = max(Nrx,Ntx)` is used
    - `kbar` : the PRACH guard interval, provided in RU

Layer mapping (eAxC offsets) happens as follows:

- For PUSCH/PDSCH, the layers are mapped to `[0,1,...,Nrx-1]/[0,1,...,Ntx-1]` where `Nrx/Ntx` is the respective RX/TX number of antennas.
- For PRACH, the layers are mapped to `[No,No+1,...No+Nrx-1]` where `No` is the `fhi_72.fh_config.[0].prach_config.eAxC_offset` . xran assumes PRACH offset `No >= max(Nrx,Ntx)` . However, we made a workaround that xran supports PRACH eAxC IDs same as PUSCH eAxC IDs. This is achieved with `is_prach` and `filter_id` parameters in the patch. Please note that this approach only applies to the RUs that support this functionality, e.g. LITEON RU.

Note

- At the moment, OAI is compatible with CAT A O-RU only. Therefore, SRS is not supported.
- XRAN retrieves DU MAC address with `rte_eth_macaddr_get()` function. Hence, `fhi_72.du_addr` parameter is not taken into account.

Start and Operation of OAI gNB

Run the `nr-softmodem` from the build directory:

```
cd ~/openairinterface5g/cmake_targets/ran_build/build
sudo ./nr-softmodem -O <configuration file> --thread-pool <list of non isolated cpus>
```

Warning: Make sure that the configuration file you add after the `-O` option is adapted to your machine, especially to its isolated cores.

Note: You may run OAI with O-RAN 7.2 Fronthaul without a RU attached (e.g. for benchmarking). In such case, you would generate artificial traffic by adding the `--phy-test` option.

You have to set the thread pool option to non-isolated CPUs, since the thread pool is used for L1 processing which should not interfere with DPDK threads. For example if you have two NUMA nodes in your system (for example 18 CPUs per socket) and odd cores are non-isolated, then you can put the thread-pool on `1,3,5,7,9,11,13,15`. On the other hand, if you have one NUMA node, you can use either isolated cores or non isolated cores, but make sure that isolated cores are not the ones defined earlier for DPDK/xran.

► Once the gNB runs, you should see counters for PDSCH/PUSCH/PRACH per

antenna port, as follows (4x2 configuration):

```
[NR_PHY]   [o-du 0][rx   24604 pps   24520 kbps  455611][tx  126652 pps  126092 kbps
        2250645][Total Msgs_Rcvd 24604]
[NR_PHY]   [o_du0][pusch0   10766 prach0    1536]
[NR_PHY]   [o_du0][pusch1   10766 prach1    1536]
```

The first line show RX/TX packet counters, i.e., packets received from the RU (RX), and sent to the RU (TX). In the second and third line, it shows the counters for the PUSCH and PRACH ports (2 receive antennas, so two counters each). These numbers should be equal, otherwise it indicates that you don't receive enough packets on either port.

► If you see many zeroes, then it means that OAI does not receive

packets on the fronthaul from the RU (RX is almost 0, all PUSCH/PRACH counters are 0).

```
[NR_PHY]   [o-du 0][rx       2 pps       0 kbps       0][tx 1020100 pps  127488 kbps
        4717971][Total Msgs_Rcvd 2]
[NR_PHY]   [o_du0][pusch0       0 prach0       0]
[NR_PHY]   [o_du0][pusch1       0 prach1       0]
[NR_PHY]   [o_du0][pusch2       0 prach2       0]
[NR_PHY]   [o_du0][pusch3       0 prach3       0]
```

In this case, please make sure that the O-RU has been configured with the right ethernet address of the gNB, and has been activated. You might enable port mirroring at your switch to capture the fronthaul packets: check that you see (1) packets at all (2)

they have the right ethernet address (3) the right VLAN tag. Although we did not test this, you might make use of the DPDK packet capture feature

► If you see messages about `Received time doesn't correspond to the`

`time we think it is or Jump in frame counter, the S-plane is not working.`

[PHY] `Received Time doesn't correspond to the time we think it is (slot mismatch, received 480.5, expected 475.8)`

```
[PHY]   Received Time doesn't correspond to the time we think it is (frame mismatch,
        480.5 , expected 475.5)
[PHY]   Jump in frame counter last_frame 480 => 519, slot 19
[PHY]   Received Time doesn't correspond to the time we think it is (slot mismatch,
        received 519.19, expected 480.12)
[PHY]   Received Time doesn't correspond to the time we think it is (frame mismatch,
        519.19 , expected 480.19)
[PHY]   Received Time doesn't correspond to the time we think it is (slot mismatch,
        received 520.1, expected 520.0)
```

`You can see that the frame numbers jump around, by 5-40 frames (corresponding to 50-400ms!). This indicates the gNB receives packets on the fronthaul that don't match its internal time, and the synchronization between gNB and RU is not working!`

`In this case, you should reverify that ptp4l and phc2sys are working, e.g., do not do any jumps (during the last hour). While an occasional jump is not necessarily problematic for the gNB, many such messages mean that the system is not working, and UEs might not be able to attach or reach good performance. Also, you can try to compile with polling (see the build section) to see if it resolves the problem.`

Operation with multiple RUs

It is possible to connect up to 4 RUs to one DU at the same time and operate them either with a single antenna array or a distributed antenna array. This works since all RUs and the DU are synchronized onto a common clock using PTP. The assumed configuration is that with N RUs each having an M×M configuration, we effectively reach an (N×M)×(N×M) configuration.

Some caveats:

- Since it's a distributed antenna, this implies that this setup will deploy a single cell only – multiple cells on different RUs are not supported.
- All RUs should use the same MTU, so either "normal" (1500 byte) MTU or jumbo frames, but not a mix of both.
- We tested only two RUs as of now, i.e., an 8×8 configuration.
- Testing is currently limited to 4 logical antenna ports in DL; in UL, up to 8 can be used.

For two RUs each using a 4x4 configuration, make sure to configure the 8x8 configuration, i.e., set `nb_tx` and `nb_rx` under `RUs` to 8 each (NOT two `RUs`!). Also, set the antenna port information as listed above, i.e.,

```
pdsch_AntennaPorts_XP = 2;
pdsch_AntennaPorts_N1 = 2;
pusch_AntennaPorts    = 8;
maxMIMO_layers        = 2;
```

Once testing for 8 antenna ports in DL is complete, we will change pdsch_AntennaPorts_N1 to 4.

Next, configure the `fhi_72` section as indicated below:

```
fhi_72 = {
```

```
   dpdk_devices = ("ru1_up_vf_pci", "ru1_cp_vf_pci", "ru2_up_vf_pci", "ru2_cp_vf_pci"); #
        two VFs can be used as well
   // core config as always
   ru_addr = ("ru1_up_mac_addr", "ru1_cp_mac_addr", "ru2_up_mac_addr", "ru2_cp_mac_addr");
        # if two VFs, set two RU MAC addresses (one per RU)
   // mtu
   fh_config = (
     {
       // DU delay profile, ru_config, prach_config of RU1
     },
     {
       // DU delay profile, ru_config, prach_config of RU2
     }
  );
};
```

i.e., for `dpdk_devices`, and `ru_addr` is configured for both RUs in a (flat) array, and the individual radio configuration is given for each RU individually inside the `fh_config`.

Sample FHI 7.2 configuration for two RUs (2 x Benetel 650)

```
fhi_72 = {
  dpdk_devices = ("0000:01:01.0", "0000:01:01.1", "0000:01:01.2", "0000:01:01.3");
  system_core = 0;
  io_core = 1;
  worker_cores = (2);
  ru_addr = ("8c:1f:64:d1:10:46","8c:1f:64:d1:10:46","8c:1f:64:d1:10:43","8c:1f:
        64:d1:10:43")
  mtu = 9600;
  fh_config = (
# RAN650 #1
   {
    T1a_cp_dl = (419, 470);
    T1a_cp_ul = (285, 336);
    T1a_up = (294, 345);
    Ta4 = (0, 200);
    ru_config = {
      iq_width = 9;
      iq_width_prach = 9;
    };
  },
# RAN650 #2
  {
    T1a_cp_dl = (419, 470);
    T1a_cp_ul = (285, 336);
    T1a_up = (294, 345);
    Ta4 = (0, 200);
    ru_config = {
      iq_width = 9;
      iq_width_prach = 9;
    };
  });
```

Compare also with the example (DU) configuration in `gnb-du.sa.band77.273prb.fhi72.8x8-benetel650_650.conf`.

Afterwards, start the gNB with the modified configuration file. If everything went well, you should see the RU counters for both RUs go up:

```
[NR_PHY]   [o-du 0][rx   63488 pps   63264 kbps 2759808][tx  127684 pps
127116 kbps 4717971][Total Msgs_Rcvd 63488]
```

```
[NR_PHY]   [o_du0][pusch0   14336 prach0    1536]
[NR_PHY]   [o_du0][pusch1   14336 prach1    1536]
[NR_PHY]   [o_du0][pusch2   14336 prach2    1536]
[NR_PHY]   [o_du0][pusch3   14336 prach3    1536]
[NR_PHY]   [o-du 1][rx   63544 pps   63320 kbps 2763240][tx  127684 pps  127116 kbps
        4717971][Total Msgs_Rcvd 63544]
[NR_PHY]   [o_du1][pusch0   14350 prach0    1536]
[NR_PHY]   [o_du1][pusch1   14350 prach1    1536]
[NR_PHY]   [o_du1][pusch2   14350 prach2    1536]
[NR_PHY]   [o_du1][pusch3   14350 prach3    1536]
```

You can also verify that there is signal on all RX antennas like so:

```
$ cat nrL1_stats.log
```

```
[...]
max_IO = 55 (85), min_I0 = 0 (136), avg_I0 = 44 dB(43.44.43.45.44.43.43.45.)
PRACH I0 = 30.6 dB
```

Note the eight entries after `avg_IO`.

You should be able to connect a UE now.

OAI Management Plane

We support Configuration Management in OAI gNB, where gNB configures CU-planes, interfaces, TX/RX antennas, and TX/RX carriers for the RU. The reference specifications:

- `O-RAN.WG4.MP.0-v05.00`
- `O-RAN.WG4.MP-YANGs-v04.00`

M-plane prerequisites

Before proceeding, please make sure you have a support for 7.2 interface, as described in Prerequisites.

DHCP server

The M-plane requires a DHCP server, where the M-plane connection can be established over untagged or tagged VLAN. We tested with untagged (the default VLAN is 1). Please modify `/etc/dhcp/dhcpd.conf` configuration based on your testbed.

Example DHCP server configuration

```
class "vendor-class" {
  match option vendor-class-identifier;
}
subclass "vendor-class" "o-ran-ru2/Benetel" {
  vendor-option-space VC;
}
option space VC;
option VC.server-address code 129 = array of ip-address;
option VC.server-fqdn code 130 = string;
# Netconf client IP address - DHCP option 43
option VC.server-address 192.168.80.1;
option VC.server-fqdn "o_du_1.operator.com";
set vendor-string = option vendor-class-identifier;
# option 143 - DHCPv4 SZTP Redirect Option (RFC8572)
# 2 bytes of URI's length + URI
option sztp code 143 = { unsigned integer 16, string };
option sztp 15 "https://192.168.80.1";
# port is optional in URI, e.g. "https://192.168.80.1:222"
#option sztp 20 "https://192.168.80.1:2222";
subnet 192.168.80.0 netmask 255.255.255.0 {
  option routers 192.168.80.1;
  option subnet-mask 255.255.255.0;
  option domain-name "oai.com";
  option domain-name-servers 172.21.3.100;
  host benetel_ru {
    # RU MAC address
    hardware ethernet <ru-mac-address>;
    # RU IP address
    fixed-address <desired-ru-ip-address>;
  }
}
```

Please, configure the interface as:

```
sudo ip address add 192.168.80.1/24 dev <interface>
```

Mandatory packages

- On Fedora (we haven't yet tested RHEL):

```
sudo dnf install pcre-devel libssh-devel libxml2-devel libyang2-devel libnetconf2-
        devel
```

- On Ubuntu:

```
sudo apt-get install libpcre3-dev libssh-dev libxml2-dev
```

On Ubuntu, please note:
`sudo apt-get install libyang2-dev libnetconf2-dev` will install unsupported versions (i.e. v2.0.112/v2.0.24 for `libyang2-dev`/ `libnetconf2-dev`, but minimum required are v2.1.4/v2.1.25). Therefore, please compile these libraries from source, as following:

Installing latest v2 libyang2 and libnetconf2 libraries

```
rm -rf /tmp/build_mplane_v2
mkdir /tmp/build_mplane_v2

# libyang
cd /tmp/build_mplane_v2
git clone https://github.com/CESNET/libyang.git
cd libyang
git checkout v2.1.111
mkdir build && cd build
cmake -DENABLE_TESTS=OFF \
      -DENABLE_VALGRIND_TESTS=OFF \
      -DCMAKE_INSTALL_PREFIX=/usr/local \
      -DCMAKE_INSTALL_RPATH=/usr/local/lib \
      -DPLUGINS_DIR=/usr/local/lib/libyang \
      -DPLUGINS_DIR_EXTENSIONS=/usr/local/lib/libyang/extensions \
      -DPLUGINS_DIR_TYPES=/usr/local/lib/libyang/types \
      -DYANG_MODULE_DIR=/usr/local/share/yang/modules/libyang ..
make -j8
sudo make install
sudo ldconfig

#libnetconf
cd /tmp/build_mplane_v2
git clone https://github.com/CESNET/libnetconf2.git
cd libnetconf2
git checkout v2.1.37
mkdir build && cd build
cmake -DENABLE_TESTS=OFF \
      -DENABLE_EXAMPLES=OFF \
      -DENABLE_VALGRIND_TESTS=OFF \
      -DCLIENT_SEARCH_DIR=/usr/local/share/yang/modules \
      -DCMAKE_INSTALL_PREFIX=/usr/local \
      -DCMAKE_INSTALL_RPATH=/usr/local/lib \
      -DLIBYANG_INCLUDE_DIR=/usr/local/include \
      -DLIBYANG_LIBRARY=/usr/local/lib/libyang.so \
      -DLY_VERSION_PATH=/usr/local/include \
      -DYANG_MODULE_DIR=/usr/local/share/yang/modules/libnetconf2 ..
make -j8
sudo make install
sudo ldconfig

# to uninstall libraries
# cd /tmp/build_mplane_v2/libyang/build && sudo make uninstall
# cd /tmp/build_mplane_v2/libnetconf2/build && sudo make uninstall
# cd
# rm -rf /tmp/build_mplane_v2
```

If you would like to install these libraries in the custom path, please replace `/usr/local` default path to e.g. `/opt/mplane-v2`.

Benetel O-RU

Note: Only v1.2.2 RAN550 and RAN650 have been successfully tested.

One time steps

Connect to the RU as user `root`, enable the mplane service, and reboot:

```
ssh root@<ru-ip-address>
```

```
systemctl enable mplane
reboot
```

Once the mplane service is successfully enabled on the RU, two new users are being added in `/etc/passwd`:

```
...
```

```
oranbenetel:x:1000:1000::/home/oranbenetel:/bin/sh
oranext:x:1001:1001::/home/oranext:/bin/sh
```

Create `oranbenetel` home directory:

```
mkdir /home/oranbenetel && chown oranbenetel:oranbenetel /home/oranbenetel
```

Connect to the RU as user `oranbenetel`, generate ssh keys, and copy DU public key into RU for NETCONF authentication:

```
ssh oranbenetel@<ru-ip-address>
```

```
ssh-keygen
echo "<DU-pub-key>" >>  ~/.ssh/authorized_keys
```

gNB configuration

The reference gNB configuration file for one Benetel RAN550: `gnb.sa.band78.273prb.fhi72.4x4-benetel550-mplane.conf` The reference DU configuration file for two Benetel RAN650: gnb-du.sa.band77.273prb.fhi72.8x8-benetel650_650-mplane.conf

In order to run gNB/DU with M-plane, we need to modify the `fhi_72` section in the configuration file. Example for one RU:

```
fhi_72 = {
  dpdk_devices = ("0000:c3:11.0", "0000:c3:11.1"); # one VF can be used as well
  system_core = 0;
  io_core = 1;
  worker_cores = (2);
  du_key_pair = ("<path-to>/.ssh/id_rsa.pub", "<path-to>/.ssh/id_rsa");
  du_addr = ("00:11:22:33:44:66", "00:11:22:33:44:67"); # only one needed if one VF
        configured
  vlan_tag = (9, 9); # only one needed if one VF configured
  ru_ip_addr = ("192.168.80.9");
  fh_config = ({
    T1a_cp_dl = (419, 470);
    T1a_cp_ul = (285, 336);
    T1a_up = (294, 345);
    Ta4 = (0, 200);
  });
};
```

Example for two RUs:

```
fhi_72 = {
  dpdk_devices = ("0000:c3:11.0", "0000:c3:11.1", "0000:c3:11.2", "0000:c3:11.3"); # two
        VFs can be used as well
  system_core = 0;
  io_core = 1;
  worker_cores = (2);
  du_key_pair = ("/home/oaicicd/.ssh/id_rsa.pub", "/home/oaicicd/.ssh/id_rsa");
  du_addr = ("00:11:22:33:44:66", "00:11:22:33:44:67", "00:11:22:33:44:68",
        "00:11:22:33:44:69"); # only two needed if two VFs configured
  vlan_tag = (9, 9, 11, 11); # only two needed if two VFs configured
  ru_ip_addr = ("192.168.80.9", "192.168.80.10");
  fh_config = (
# RAN550 #1
  {
    T1a_cp_dl = (419, 470);
    T1a_cp_ul = (285, 336);
    T1a_up = (294, 345);
    Ta4 = (0, 200);
  },
# RAN550 #2
  {
    T1a_cp_dl = (419, 470);
    T1a_cp_ul = (285, 336);
    T1a_up = (294, 345);
    Ta4 = (0, 200);
  });
};
```

- `fhi_72` :
  - `dpdk_devices` : [*]
  - `system_core` : [*]
  - `io_core` : [*]
  - `worker_cores` : [*]
  - `file_prefix` : [*]
  - `du_key_pair` : ssh public and private keys to authenticate RU with NETCONF
  - `du_addr` : DU MAC address(es) to create CU-plane interface(s) in the RU
  - `vlan_tag` : VLAN U and C plane tags to create CU-plane interface(s) in the RU
  - `ru_ip_addr` : RU IP address to connect to the RU via M-plane
  - `dpdk_mem_size` : [*]
  - `dpdk_iova_mode` : [*]
  - `owdm_enable` : [*]
  - `fh_config` : only DU delay profile (`T1a` and `Ta4`)

`[*] see Configure OAI gNB for more details`

The following parameters are retrieved from the RU and forwarded to the xran:

- `MTU`
- `RU MAC address`
- `IQ compression`: if RU supports multiple, the first value in the list is taken; please note that the same value is used for PxSCH/PRACH
- `PRACH offset`: hardcoded based on the RU vendor (i.e. for Benetel `max(Nrx,Ntx)`)

Build and compile gNB

The following cmake options are available:

- `OAI_FHI72` = CUS support
- `OAI_FHI72_MPLANE` = M support

Compiled libraries:

- `OAI_FHI72` <=> `oran_fhlib_5g`
- `OAI_FHI72` && `OAI_FHI72_MPLANE` <=> `oran_fhlib_5g` (CUS) && `oran_fhlib_5g_mplane` (CUSM)

Using build_oai script

```
git clone https://gitlab.eurecom.fr/oai/openairinterface5g.git ~/
        openairinterface5g
```

```
cd ~/openairinterface5g/cmake_targets/
./build_oai -I  # if you never installed OAI, use this command once before the next line
./build_oai --install-optional-packages  # for pcre/libpcre3, libssh, and libxml2 library
        installation
./build_oai --gNB --ninja -t oran_fhlib_5g_mplane --cmake-opt -Dxran_LOCATION=$HOME/phy/
        fhi_lib/lib
# if libyang2 and libnetconf2 are installed in `/opt/mplane-v2`, please use the following
        command:
PKG_CONFIG_PATH=/opt/mplane-v2/lib/pkgconfig ./build_oai --gNB --ninja -t
        oran_fhlib_5g_mplane --cmake-opt -Dxran_LOCATION=$HOME/phy/fhi_lib/lib
```

Using cmake directly

```
git clone https://gitlab.eurecom.fr/oai/openairinterface5g.git ~/
        openairinterface5g
```

```
cd ~/openairinterface5g/
mkdir build && cd build
cmake .. -GNinja -DOAI_FHI72=ON -DOAI_FHI72_MPLANE=ON -Dxran_LOCATION=$HOME/phy/fhi_lib/lib
# if libyang2 and libnetconf2 are installed in `/opt/mplane-v2`, please use the following
        command:
PKG_CONFIG_PATH=/opt/mplane-v2/lib/pkgconfig cmake .. -GNinja -DOAI_FHI72=ON -
        DOAI_FHI72_MPLANE=ON -Dxran_LOCATION=$HOME/phy/fhi_lib/lib
ninja nr-softmodem oran_fhlib_5g_mplane params_libconfig
```

Start the gNB

Run the `nr-softmodem` from the build directory:

```
cd ~/openairinterface5g/cmake_targets/ran_build/build
```

```
sudo ./nr-softmodem -O <mplane-configuration file> --thread-pool <list of non isolated
        cpus>
```

Warning: Make sure that the configuration file you add after the `-O` option is adapted to your machine, especially to its isolated cores.

M-plane sequence diagram:

```
sequenceDiagram
```

```
participant dhcp as DHCP server
participant du as O-DU
participant ru as O-RU

dhcp->ru: 1. Transport Layer Initialization
note over dhcp,ru: (a) perform VLAN scan (untagged or tagged VLAN)<br/>(b) DHCP assigns
      IP address to RU<br/>(c) DHCP sends DU IP address to RU

du->ru: 2. NETCONF/SSH connect

du->>ru: 3. DU retrieves RU info
note left of du: <get>

note over du: Check if RU is PTP synced

du->>ru: 4. DU subscribes to all RU notifications
note left of du: <subscribe>

du->>ru: 5. DU updates the supervision timer
note left of du: <supervision-watchdog-reset>

note over du: Store RU MAC, MTU, IQ bitwidth, and PRACH offset info for xran

note over du: Store all the RU U-plane info - interface name, TX/RX carrier, and TX/RX
      endpoint names

du->>ru: 6. DU loads yang models
note left of du: <get-schema>
note right of ru: ietf-netconf-monitoring.yang

du->>ru: 7. DU performs CU-plane configuration
note left of du: <edit-config>

note over ru: (1) o-ran-interface.yang - create new interface with VLAN tag, DU and RU
      MAC addresses<br/>(2)(opt.) o-ran-transceiver.yang - for file upload<br/>(3) o-
      ran-processing-elements.yang - element with which RU ports should be assigned<br/
      >(4) o-ran-uplane-conf.yang<br/>#8193;(4a) low-level-tx/rx-endpoints - PxSCH,
      PRACH<br/>#8193;(4b) tx/rx-array-carriers - center frequency, BW, gain,
      ACTIVE,...<br/>#8193;(4c) low-level-tx/rx-links - mapping endpoints, carriers and
      processing element<br/>#8193;(4d)(opt.) if CAT B, SRS configuration<br/>#8193;(4e)
      (opt.) TDD configuration

du->>ru: 8. DU checks if CU-plane configuration is valid
note left of du: <validate>

du->>ru: 9. If valid, DU commits the changes
note left of du: <commit>

du->>ru: 10. DU retrieves RU states
note right of ru: ietf-hardware.yang
note left of du: <get>
note over ru: admin-state, power-state, oper-state,<br/>availability-state, usage-state

ru->>du: 11. RU notifies DU about the configuration change

note over du: DU configures xran

du->ru: 12. DU and RU exchange packets
```

4x4 MIMO and 100MHz BW with Benetel 550 RU example run

```
[HW]   [MPLANE] RPC request to RU "192.168.80.9" = <connect> with username "oranbenetel"
        and port ID "830".
[HW]   [MPLANE] Successfuly connected to RU "192.168.80.9" with username "oranbenetel" and
        port ID "830".
[HW]   [MPLANE] RPC request to RU "192.168.80.9" = <get> operational datastore.
[HW]   [MPLANE] Successfully retrieved operational datastore from RU "192.168.80.9".
[HW]   [MPLANE] RU is already PTP synchronized.
[HW]   [MPLANE] RPC request to RU "192.168.80.9" = <subscribe> with stream "NETCONF" and
        filter "(null)".
[HW]   [MPLANE] RPC reply = OK.
[HW]   [MPLANE] Successfully subscribed to all notifications from RU "192.168.80.9".
[HW]   [MPLANE] RPC request to RU "192.168.80.9" = "<supervision-watchdog-reset
        xmlns="urn:o-ran:supervision:1.0">
<supervision-notification-interval>65535</supervision-notification-interval>
<guard-timer-overhead>65535</guard-timer-overhead>
</supervision-watchdog-reset>".
[HW]   [MPLANE] Successfully updated supervision timer to (65535+65535)[s] for RU
        "192.168.80.9".
[HW]   [MPLANE] Watchdog timer answer:
    <next-update-at xmlns="urn:o-ran:supervision:1.0">2025-03-30T08:52:31+02:00</next-
        update-at>

[HW]   [MPLANE] Interface MTU 1500 unreliable/not correctly reported by Benetel O-RU,
        hardcoding to 9600.
[HW]   [MPLANE] IQ bitwidth 16 unreliable/not correctly reported by Benetel O-RU,
        hardcoding to 9.
[HW]   [MPLANE] Storing the following information to forward to xran:
    RU MAC address 8c:1f:64:d1:11:c0
    MTU 9600
    IQ bitwidth 9
    PRACH offset 4
    DU port bitmask 61440
    Band sector bitmask 3840
    CC ID bitmask 240
    RU port ID bitmask 15
    DU port ID 0
    Band sector ID 0
    CC ID 0
    RU port ID 0
[HW]   [MPLANE] Successfully retrieved all the U-plane info - interface name, TX/RX carrier
        names, and TX/RX endpoint names.
[HW]   [MPLANE] RPC request to RU "192.168.80.9" = <get-schema> for module "ietf-yang-
        metadata".
[HW]   [MPLANE] RPC request to RU "192.168.80.9" = <get-schema> for module "yang".
[HW]   [MPLANE] RPC request to RU "192.168.80.9" = <get-schema> for module "ietf-inet-
        types".
[HW]   [MPLANE] RPC request to RU "192.168.80.9" = <get-schema> for module "ietf-yang-
        types".
[HW]   [MPLANE] RPC request to RU "192.168.80.9" = <get-schema> for module "ietf-yang-
        schema-mount".
[HW]   [MPLANE] RPC request to RU "192.168.80.9" = <get-schema> for module "ietf-yang-
        structure-ext".
[HW]   [MPLANE] RPC request to RU "192.168.80.9" = <get-schema> for module "ietf-
        datastores".
[HW]   [MPLANE] RPC request to RU "192.168.80.9" = <get-schema> for module "sysrepo".
[HW]   [MPLANE] RPC request to RU "192.168.80.9" = <get-schema> for module "ietf-netconf-
        acm".
[HW]   [MPLANE] RPC request to RU "192.168.80.9" = <get-schema> for module "ietf-factory-
        default".
[HW]   [MPLANE] RPC request to RU "192.168.80.9" = <get-schema> for module "sysrepo-
        factory-default".
[HW]   [MPLANE] RPC request to RU "192.168.80.9" = <get-schema> for module "ietf-yang-
        library".
[HW]   [MPLANE] RPC request to RU "192.168.80.9" = <get-schema> for module "sysrepo-
```

Note: If you wish to run the fronthaul without M-plane, no need for recompilation, as the library `oran_fhlib_5g` already exists. The only mandatory step is to link `oran_fhlib_5g` to `oai_transpro` library.

```
cd ~/openairinterface5g/cmake_targets/ran_build/build
rm liboai_transpro.so
ln -s liboran_fhlib_5g.so liboai_transpro.so
sudo ./nr-softmodem -O <without-mplane-configuration file> --thread-pool <list of non
        isolated cpus>
```

Contact in case of questions

You can ask your question on the mailing lists.

Your email should contain below information:

- A clear subject in your email.
- For all the queries there should be [Query] in the subject of the email and for problems there should be [Problem].
- In case of a problem, add a small description.
- Do not share any screenshots/photos unless you want to share a diagram.
- OAI gNB/DU/CU/CU-CP/CU-UP configuration file in `.conf` format only.
- Logs of OAI gNB/DU/CU/CU-CP/CU-UP in `.log` or `.txt` format only.
- RU Vendor and Version.
- In case your question is related to performance, include a small description of the machine (CPU, RAM and networking card) and diagram of your testing environment.
- If you have any issues related to PTP or synchronization, then first check the section "Debugging PTP issues". Then share your problem with PTP version you are using, switch details and master clock.
- Known/open issues are present on GitLab, so keep checking.

LDPC_T2_OFFLOAD_SETUP

OAI T1/T2 LDPC offload

Table of Contents

[[TOC]]

This documentation aims to provide a tutorial for AMD Xilinx T2 Telco card integration into OAI and its usage.

Requirements

- bitstream image and PMD driver for the T2 card provided by AccelerComm
- DPDK 20.11.9 with patch from Accelercomm: version ACCL_BBDEV_DPDK20.11.3_ldpc_3.1.918.patch
- tested on RHEL7.9, RHEL9.2, Ubuntu 22.04

DPDK setup

DPDK installation

Note: Following instructions are valid for ACCL_BBDEV_DPDK20.11.3_ldpc_3.1.918.patch version, which is compatible with DPDK 20.11.9. Installation steps, which should be followed for older versions of the patch file (for example ACL_BBDEV_DPDK20.11.3_BL_1006_build_1105_dev_branch_MCT_optimisations_1106_physical_std.p are present in older version of this documentation, under the tag 2023.w48.

```
# Get DPDK source code
git clone https://github.com/DPDK/dpdk-stable.git ~/dpdk-stable
cd ~/dpdk-stable
git checkout v20.11.9
git apply ~/ACL_BBDEV_DPDK20.11.3_ldpc_3.1.918.patch
```

Replace `~/ACL_BBDEV_DPDK20.11.3_ldpc_3.1.918.patch` by patch file provided by Accelercomm.

```
cd ~/dpdk-stable
meson setup build
# meson setup --prefix=/opt/dpdk-t2 build for installation with non-default installation
        prefix
cd build
ninja
sudo ninja install
```

```
sudo ldconfig
```

DPDK configuration

- load required kernel module

  ```
  sudo modprobe vfio-pci
  ```

- check presence of the card and its PCI addres on the host machine

  ```
  lspci | grep "Xilinx"
  ```

- bind the card with vfio-pci driver

  ```
  sudo python3 ~/dpdk-stable/usertools/dpdk-devbind.py --bind=vfio-pci 41:00.0
  ```

  Replace PCI address of the card 41:00.0 by address detected by lspci | grep "Xilinx" command

- hugepages setup (10 x 1GB hugepages)

  ```
  sudo python3 ~/dpdk-stable/usertools/dpdk-hugepages.py -p 1G --setup 10G
  ```

Note: device binding and hugepages setup has to be done after every reboot of the host machine

Modifications in the OAI code

DPDK lib and PMD path specification

If DPDK library was installed into custom path, you have to point to the right directory with `PKG_CONFIG_PATH`, prior to the OAI build. Sample command to set the DPDK path to `/opt/dpdk-t2/lib64/pkgconfig/`:

```
export PKG_CONFIG_PATH=/opt/dpdk-t2/lib64/pkgconfig/:$PKG_CONFIG_PATH
```

OAI Build

OTA deployment is precisely described in the following tutorial:

- NR_SA_Tutorial_COTS_UE Instead of section 3.2 Build OAI gNB from the tutorial, run following commands:

```
# Get openairinterface5g source code
git clone https://gitlab.eurecom.fr/oai/openairinterface5g.git ~/openairinterface5g
```

```
cd ~/openairinterface5g
git checkout develop

# Install OAI dependencies
cd ~/openairinterface5g/cmake_targets
./build_oai -I

# Build OAI gNB
cd ~/openairinterface5g
source oaienv
cd cmake_targets
./build_oai -w USRP --ninja --gNB -P --build-lib "ldpc_t2" -C
```

Shared object file libldpc_t2.so is created during the compilation. This object is conditionally compiled. Selection of the library to compile is done using –build-lib ldpc_t2.

Required poll mode driver has to be present on the host machine and required DPDK version has to be installed on the host, prior to the build of OAI

Setup of T2-related DPDK EAL parameters

To configure T2-related DPDK Environment Abstraction Layer (EAL) parameters, you can set the following parameters via the command line of PHY simulators or softmodem:

- `nrLDPC_coding_t2.dpdk_dev` - **mandatory** parameter, specifies PCI address of the T2 card. PCI address of the T2 card can be detected by `lspci | grep "Xilinx"` command.
- `nrLDPC_coding_t2.dpdk_core_list` - **mandatory** parameter, specifies CPU cores assigned to DPDK for T2 processing. Ensure that the CPU cores specified in nrLDPC_coding_t2.dpdk_core_list are available and not used by other processes to avoid conflicts.
- `nrLDPC_coding_t2.dpdk_prefix` - DPDK shared data file prefix, by default set to b6.

Note: These parameters can also be provided in a configuration file:

```
nrLDPC_coding_t2 : {
  dpdk_dev : "41:00.0";
  dpdk_core_list : "14-15";
};

loader : {
  ldpc : {
    shlibversion : "_t2";
  };
};
```

5G PHY simulators

nr_ulsim test

Offload of the channel decoding to the T2 card is in nr_ulsim specified by –loader.ldpc.shlibversion _t2 option. Example command for running nr_ulsim with LDPC decoding offload to the T2 card:

```
cd ~/openairinterface5g
source oaienv
cd cmake_targets/ran_build/build
sudo ./nr_ulsim -n100 -s20 -m20 -r273 -R273 --loader.ldpc.shlibversion _t2 --
        nrLDPC_coding_t2.dpdk_dev 01:00.0 --nrLDPC_coding_t2.dpdk_core_list 0-1
```

nr_dlsim test

Offload of the channel encoding to the AMD Xilinx T2 card is in nr_dlsim specified by -c option. Example command for running nr_dlsim with LDPC encoding offload to the T2 card:

```
cd ~/openairinterface5g
source oaienv
cd cmake_targets/ran_build/build
sudo ./nr_dlsim -n300 -s30 -R 106 -e 27 --loader.ldpc.shlibversion _t2 --
        nrLDPC_coding_t2.dpdk_dev 01:00.0 --nrLDPC_coding_t2.dpdk_core_list 0-1
```

OTA test

Offload of the channel encoding and decoding to the AMD Xilinx T2 card is enabled by –loader.ldpc.shlibversion _t2 option.

Run OAI gNB with USRP B210

```
cd ~/openairinterface5g
source oaienv
cd cmake_targets/ran_build/build
sudo ./nr-softmodem -O ../../../targets/PROJECTS/GENERIC-NR-5GC/CONF/
        gnb.sa.band78.fr1.106PRB.usrpb210.conf --loader.ldpc.shlibversion _t2 --
        nrLDPC_coding_t2.dpdk_dev 01:00.0 --nrLDPC_coding_t2.dpdk_core_list 0-1
```

Limitations

AMD Xilinx T2 card

- functionality of the LDPC encoding and decoding offload verified in OTA SISO setup with USRP N310 and Quectel RM500Q, blocking of the card reported for MIMO setup (2 layers)

Note: AMD Xilinx T1 Telco card is not supported anymore.

Aerial_FAPI_Split_Tutorial

OAI 7.2 Fronthaul Interface 5G SA Tutorial

Table of Contents

[[TOC]]

Prerequisites

The hardware on which we have tried this tutorial:

Hardware (CPU,RAM)	Operating System (kernel)	NIC (Vendor,Driver,Firmware)
Gigabyte Edge E251-U70 (Intel Xeon Gold 6240R, 2.4GHz, 24C48T, 96GB DDR4)	Ubuntu 22.04.3 LTS (5.15.0-72-lowlatency)	NVIDIA ConnectX®-6 Dx 22.38.1002
Dell PowerEdge R750 (Dual Intel Xeon Gold 6336Y CPU @ 2.4G, 24C/48T (185W), 512GB RDIMM, 3200MT/s)	Ubuntu 22.04.3 LTS (5.15.0-72-lowlatency)	NVIDIA Converged Accelerator A100X (24.39.2048)
Supermicro Grace Hopper MGX ARS-111GL-NHR (Neoverse-V2, 3.4GHz, 72C/72T, 576GB LPDDR5)	Ubuntu 22.04.5 LTS (6.5.0-1019-nvidia-64k)	NVIDIA BlueField3 (32.41.1000)

NOTE:

- These are not minimum hardware requirements. This is the configuration of our servers. The NIC card should support hardware PTP time stamping.
- Starting from tag 2025.w13 of OAI, we are only testing with the Grace Hopper server.

PTP enabled switches and grandmaster clock we have tested with:

Vendor	Software Version
Fibrolan Falcon-RX/812/G	8.0.25.4
CISCO C93180YC-FX3	10.2(4)
Qulsar Qg2 (Grandmaster)	12.1.27

These are the radio units we've used for testing:

Vendor	Software Version
Foxconn RPQN-7801E RU	2.6.9r254
Foxconn RPQN-7801E RU	3.1.15_0p4
Foxconn RPQN-7801E RU	v3.2.0q.551.12.E.rc2.srs-AIO

The UEs that have been tested and confirmed working with Aerial are the following:

Vendor	Model
Sierra Wireless	EM9191
Quectel	RM500Q-GL
Quectel	RM520N-GL
Apaltec	Tributo 5G-Dongle
OnePlus	Nord (AC2003)
Apple iPhone	14 Pro (MQ0G3RX/A) (iOS 17.3)
Samsung	S23 Ultra

Configure your server

To set up the L1 and install the components manually refer to this instructions page.

Note:

- To configure the Gigabyte server please refer to these instructions
- The last release to support the Gigabyte server is **Aerial CUDA-Accelerated RAN 24-1**.

CPU allocation

Server brand	Model	Nº of CPU Cores	Isolated CPUs
Grace Hopper MGX	ARS-111GL-NHR	72	4-64

Grace Hopper MGX ARS-111GL-NHR

Applicative Threads	Allocated CPUs
PTP & PHC2SYS Services	41
OAI `nr-softmodem`	13-20

PTP configuration

1. You need to install the `linuxptp` debian package. It will install both ptp4l and phc2sys.

```
#Ubuntu
sudo apt install linuxptp -y
```

Once installed you can use this configuration file for ptp4l (`/etc/ptp4l.conf`). Here the clock domain is 24 so you can adjust it according to your PTP GM clock domain

```
[global]
domainNumber            24
slaveOnly               1
time_stamping           hardware
tx_timestamp_timeout    30

[your_PTP_ENABLED_NIC]
network_transport       L2
```

The service of ptp4l (`/lib/systemd/system/ptp4l.service`) should be configured as below:

```
[Unit]
Description=Precision Time Protocol (PTP) service
Documentation=man:ptp4l

[Service]
Restart=always
RestartSec=5s
Type=simple
ExecStart=taskset -c 41 /usr/sbin/ptp4l -f /etc/ptp.conf

[Install]
WantedBy=multi-user.target
```

and service of phc2sys (`/lib/systemd/system/phc2sys.service`) should be configured as below:

```
[Unit]
Description=Synchronize system clock or PTP hardware clock (PHC)
Documentation=man:phc2sys
After=ntpdate.service
Requires=ptp4l.service
After=ptp4l.service

[Service]
Restart=always
RestartSec=5s
Type=simple
ExecStart=/usr/sbin/phc2sys -a -r -n 24
```

```
[Install]
WantedBy=multi-user.target
```

Build OAI gNB

If it's not already cloned, the first step is to clone OAI repository

```
git clone https://gitlab.eurecom.fr/oai/openairinterface5g.git ~/openairinterface5g
cd ~/openairinterface5g/
```

Get nvIPC sources from the L1 container

The library used for communication between L1 and L2 components is called nvIPC, and is developed by NVIDIA. It is not open-source and can't be freely distributed. In order to achieve this communication, we need to obtain the nvIPC source files from the L1 container (cuBB) and place it in the gNB project directory `~/openairinterface5g`. This allows us to build and install this library when building the L2 docker container.

Check whether your L1 container is running:

```
~$ docker container ls -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
        STATUS                      PORTS               NAMES
a9681a0c4a10        14dca2002237        "/opt/nvidia/nvidia_…"   3 months ago        Exited
        (137) 10 days ago                       cuBB
```

If it is not running, you may start it and logging into the container by running the following:

```
~$ docker start cuBB
cuBB
~$ docker exec -it cuBB bash
aerial@c_aerial_aerial:/opt/nvidia/cuBB#
```

After logging into the container, you need to pack the nvIPC sources and copy them to the host (the command creates a `tar.gz` file with the following name format: `nvipc_src.YYYY.MM.DD.tar.gz`)

```
~$ docker exec -it cuBB bash
aerial@c_aerial_aerial:/opt/nvidia/cuBB# cd cuPHY-CP/gt_common_libs
aerial@c_aerial_aerial:/opt/nvidia/cuBB/cuPHY-CP/gt_common_libs#./pack_nvipc.sh
nvipc_src.YYYY.MM.DD/
...
-------------------------------------------
Pack nvipc source code finished:
/opt/nvidia/cuBB/cuPHY-CP/gt_common_libs/nvipc_src.YYYY.MM.DD.tar.gz
```

```
aerial@c_aerial_aerial:/opt/nvidia/cuBB/cuPHY-CP/gt_common_libs# sudo cp
        nvipc_src.yyyy.mm.dd.tar.gz /opt/cuBB/share/
aerial@c_aerial_aerial:/opt/nvidia/cuBB/cuPHY-CP/gt_common_libs# exit
```

The file should now be present in the `~/openairinterface5g/ci-scripts/yaml_files/sa_gnb_aerial/` directory, from where it is moved into `~/openairinterface5g`

```
~$ mv ~/openairinterface5g/ci-scripts/yaml_files/sa_gnb_aerial/nvipc_src.YYYY.MM.DD.tar.gz
        ~/openairinterface5g/
```

Alternatively, after running `./pack_nvipc.sh` , you can exit the container and copy `nvipc_src.YYYY.MM.DD.tar.gz` using `docker cp`

```
~$ docker exec -it cuBB bash
aerial@c_aerial_aerial:/opt/nvidia/cuBB# cd cuPHY-CP/gt_common_libs
aerial@c_aerial_aerial:/opt/nvidia/cuBB/cuPHY-CP/gt_common_libs#./pack_nvipc.sh
nvipc_src.YYYY.MM.DD/
...
-------------------------------------------
Pack nvipc source code finished:
/opt/nvidia/cuBB/cuPHY-CP/gt_common_libs/nvipc_src.YYYY.MM.DD.tar.gz
aerial@c_aerial_aerial:/opt/nvidia/cuBB/cuPHY-CP/gt_common_libs# exit
~$ docker cp nv-cubb:/opt/nvidia/cuBB/cuPHY-CP/gt_common_libs/nvipc_src.YYYY.MM.DD.tar.gz ~/
        openairinterface5g
```

Important note: For using docker cp, make sure to copy the entire name of the created nvipc_src tar.gz file.

With the nvIPC sources in the project directory, the docker image can be built.

Building OAI gNB docker image

In order to build the target image (`oai-gnb-aerial`), first you should build a common shared image (`ran-base`)

```
~$ cd ~/openairinterface5g/
~/openairinterface5g$ docker build . -f docker/Dockerfile.base.ubuntu22 --tag ran-
        base:latest
~/openairinterface5g$ docker build . -f docker/Dockerfile.gNB.aerial.ubuntu22 --tag oai-gnb-
        aerial:latest
```

Running the setup

In order to use Docker Compose to automatically start and stop the setup, we first need to create a pre-built image of L1 after compiling the L1 software and making the

necessary adjustments to its configuration files. The process of preparing L1 is covered in NVIDIA's documentation and falls outside the scope of this document.

Prepare the L1 image

After preparing the L1 software, the container needs to be committed to create an image where L1 is ready for execution, which can later be referenced by a `docker-compose.yaml` file.

Note: The default L1 configuration file is `cuphycontroller_P5G_FXN_GH.yaml`, located in `/opt/nvidia/cuBB/cuPHY-CP/cuphycontroller/config/`. In this file the RU MAC address needs to be specified before commiting the image.

```
~$ docker commit nv-cubb cubb-build:24-3
~$ docker image ls
..
cubb-build                                      24-3
        824156e0334c   2 weeks ago    23.9GB
-..
```

Adapt the OAI-gNB configuration file to your system/workspace

Edit the sample OAI gNB configuration file and check following parameters:

- `gNBs` section
  - The PLMN section shall match the one defined in the AMF
  - `amf_ip_address` shall be the correct AMF IP address in your system
  - `GNB_IPV4_ADDRESS_FOR_NG_AMF` shall match your DU N2 interface IP address
  - `GNB_IPV4_ADDRESS_FOR_NGU` shall match your DU N3 interface IP address

The default amf_ip_address:ipv4 value is 192.168.70.132, when installing the CN5G following this tutorial Both 'GNB_IPV4_ADDRESS_FOR_NG_AMF' and 'GNB_IPV4_ADDRESS_FOR_NGU' need to be set to the IP address of the NIC referenced previously.

Running docker compose

Aerial L1 entrypoint script

The `aerial_l1_entrypoint` script is used by the L1 container to start the L1 software and is called in the Docker Compose file. It begins by setting up environment variables, restarting NVIDIA MPS, and finally running `cuphycontroller_scf`.

The L1 software is executed with an argument that specifies which configuration file to use. If not modified, the default argument is set to `P5G_FXN_GH`.

After building the gNB image, and preparing the configuration file, the setup can be run with the following command:

```
cd ci-scripts/yaml_files/sa_gnb_aerial/
docker compose up -d
```

This will start both containers, beginning with `nv-cubb`, and `oai-gnb-aerial` will start only after it is ready.

The logs can be followed using these commands:

```
docker logs -f oai-gnb-aerial
docker logs -f nv-cubb
```

Running with multiple L2s

One L1 instance can support multiple L2 instances. See also the aerial documentation for more details.

In OAI the shared memory prefix must be configured in the configuration file.

```
    tr_s_preference = "aerial";
tr_s_shm_prefix = "nvipc";
```

Stopping the setup

Run the following command to stop and remove both containers, leaving the system ready to be restarted later:

```
cd ci-scripts/yaml_files/sa_gnb_aerial/
docker compose down
```

mac-usage

This document describes the basic functioning of the 5G MAC scheduler, describes the periodic output, and explains out the various configuration options that influence its behavior.

[[TOC]]

General

The 5G MAC scheduler is a proportional fair (PF) scheduler, "approximating wide-band CQI" (for lack of a better term, but CQI is typically used for PF) through the selection of an MCS to use.

Concretely, the scheduler loops through all UEs and calculates the PF coefficient using the currently selected MCS, and the historical achieved rate. The UE with highest coefficient wins and is scheduled RBs until all resources are used, or until it has no more data to fill RBs, in which the scheduler continues with the UE with the second-best coefficient.

UEs with retransmissions are allocated first; similarly, UEs that have not been scheduled for some time in UL are scheduled automatically in UL and have therefore priority over data with "normal" traffic.

The MCS selection is done in `get_mcs_from_bler()` in file `gNB_scheduler_primitives.c`. It consider two thresholds for a "BLER" that is computed from the number of first-round retransmissions over total transmissions in the last window (50ms). If that ratio is higher than an "upper" threshold (see `dl/ul_bler_target_upper` in the configuration section below), it is interpreted as "bad channel" and MCS is decremented by 1. If the ratio is lower than a "lower" threshold (see `dl/ul_bler_target_lower`), it is interpreted as "good channel" and MCS is incremented by 1. This happens each window.

The actual scheduler implementation can be found in functions `pf_dl()` and `pf_ul()` in files `gNB_scheduler_dlsch.c` (for DL) and `gNB_scheduler_ulsch.c` (for UL), respectively.

PDDCH aggregation level

PDCCH aggregation level is selected using closed loop controller, where DL HARQ feedback is the controller feedback signal. It is used to increment `pdcch_cl_adjust`

variable if no feedback is detected and decrement the variable when feedback is detected. `pdcch_cl_adjust` is later mapped to the PDCCH aggregation level range.

The value of `pdcch_cl_adjust` is clamped to range <0,1>, the increment value is 0.05 while the decrement value is 0.01. These values are selected to ensure PDCCH success rate is high. See Examples below for futher explaination.

The possible values of aggregation level on UE SS can be configured via `uess_agg_levels` configuration option. By default the gNB uses only aggregation level 2 which translates to `uess_agg_levels` set to `[0, 1, 0, 0, 0]`. For example, to enable aggregation level 2 and 4 set `uess_agg_levels` to `[0, 1, 1, 0, 0]`.

Examples:

Example 1:

Say we have 90% PDCCH success rate at aggregation level 1, `pdcch_cl_adjust` will stay at 0 for most of the time. 2 consecutive PDCCH failures will not result in increasing the aggregation level (because (0.05 + 0.05) * 4 = 0.4 which is closer to 0 than to 1). If PDCCH fails 3 times in a row the aggregation level will change to 2 and hopefully back to 1 once more PDDCH successes happen.

Example 2

Say we have 0% PDCCH success rate (radio link failure scenario) but `pdcch_cl_adjust` is 0 indicating perfect PDCCH channel. it would take ~18 PDCCH failures to reach maximum aggregation level.

Periodic output and interpretation

The scheduler periodically outputs statistics that can help you judge the radio channel quality, i.e., why a UE does not perform as you would expect. The scheduler outputs this info in log level "INFO". The same information is also available in the the file `nrMAC-stats.log` in the same directory in which `nr-softmodem` is run.

Example:

```
UE RNTI 2460 CU-UE-ID 2 in-sync PH 28 dB PCMAX 24 dBm, average RSRP -74 (8 meas)
UE 2460: CQI 15, RI 2, PMI (14,1)
UE 2460: UL-RI 2 TPMI 0
UE 2460: dlsch_rounds 32917/5113/1504/560, dlsch_errors 211, pucch0_DTX 1385, BLER 0.19557
        MCS (1) 23
UE 2460: ulsch_rounds 3756/353/182/179, ulsch_errors 170, ulsch_DTX 285, BLER 0.33021 MCS
        (1) 27 (Qm 8  dB) NPRB 5  SNR 31.0 dB
```

```
UE 2460: MAC:    TX     1530943191 RX         194148 bytes
UE 2460: LCID 1: TX            651 RX           3031 bytes
UE 2460: LCID 2: TX              0 RX              0 bytes
UE 2460: LCID 4: TX     1526169592 RX          16152 bytes
```

In the first line,

- `UE RNTI` (here `2460`): this is also used as the DU UE ID over F1; each line is prepended with the RNTI
- `CU UE ID` (`2`): separate identifier from the RNTI to handle multiple UEs across DUs (in which case RNTI conflicts are possible)
- whether a UE is `in-sync` (actively being scheduled) or `out-of-sync` (the UE is not being scheduled, because it did not respond when being scheduled in UL or since it has radio-link failure
- `PH` (`28`): Power Headroom, the amount of power the UE has left. If it is > 40 you can achieve full UL throughput. `PCMAX` (`24 dBm`) is what the UE reported as maximum UL transmit power it can output in the channel.
- `RSRP` (`-74`): measured power of the DL reference signals at the UE. >-80dBm you should have full DL throughput. <-95 dBm, you are very limited in terms of connectivity.

The second and third line reflect channel state information (CSI) as reported by the UE, and only appear if CSI-RS/SRS are enabled and received (for some bands, they cannot be enabled):

- `CQI` (`15`): the channel quality indicator is a number between 0 and 15. It indicates the achievable spectral efficiency of the UE. 15 means highest, 0 is lowest. This corresponds to a 4-bit table in 38.214 with actual spectral efficiencies in bits/s/Hz
- `RI`: Rank indicator 1-4. If 1 it means limited to 1 layer transmission on DL. 2 two layers, etc.
- `PMI`: precoding matrix indicator. It's a measure of the spatial-direction of the gNBs transmit array as seen by the UE. It indicates the precoding that the gNB applies. It should be more or less stationary unless the UE is moving around quickly. It can jump when objects move around the UE
- `UL-RI`, `TPMI`: same as DL.

The fourth and fifth line show HARQ-related information:

- `dlsch_rounds A/B/C/D` (`32917/5113/1504/560`). This is the number of transmissions by the gNB for each round of the HARQ protocol. `A` is the first round,

B is the second, etc. If `A` is high and `B` is low, `C` is lower and `D` is around 0, then things work well. If `B` is around the same as `A`, then the first round is almost always in error. This is not usually good unless the UE is far from the gNB.

- `dlsch_errors` (`211`) is the number of errors, meaning that after 4 transmissions the MAC transport block was not received by the UE. A high number is not good/it should be a very low number, again unless there are severe radio conditions.
- `pucch0_DTX` : this is the number of PUCCH format 0 (or 1 later) missed detections (unreceived ack/nak). This means that either the UL is very bad and ACK/NAK cannot be conveyed properly or DL DCIs are missed by the UE. This is also something that should be very small compared to `A` in `dlsch_rounds` .
- `DLSCH BLER` is the current measured block-error rate of the DLSCH. Basically a moving average of `B` / `A` in `dlsch_rounds` . This is something that should always be close to the target bler that the MAC scheduler uses. typically 10-30% if we want a high throughput scheduling policy.
- `MCS (Q) M` : M (0-28) is the current MCS used by the MAC scheduler and Q is the mcs table: 0=64QAM, 1=256QAM, 2=low SE table for URLLC
- `ulsch_rounds` / `ulsch_errors` : same as DLSCH but for UL.
- `ulsch_DTX` (`285`): number of PUSCH missed detection (i.e. signal energy below threshold in configuration file, `L1s.pusch_dtx_threshold` , which is 10 times the actual unnormalized digital signal level). This is an indication of either very poor radio conditions (UE cannot reach the gNB), power control problems, or missed DCI 0_x DCIs. It should be low compared to `A` in `ulsch_rounds` when things are working properly.
- ULSCH `BLER/MCS` : same as DLSCH but for UL.
- ULSCH `Qm X deltaMCS Y dB` : modulation order: 2=QPSK, 4=16QAM, 6=64QAM, 8=256QAM and the dB offset for deltaMCS component in PUSCH power control law. If deltaMCS is disabled (this is the default) then it indicates 0. When deltaMCS is enabled it indicates the current power offset applied by the UE on the UL corresponding to the scheduled MCS
- ULSCH `NPRB` : current number of PRBs scheduled by the gNB. This will fluctuate a lot but when doing high throughput with iperf, it indicates the number of PRBs that the UE is actually able to use with its power budget and should be high.
- ULSCH `SNR` : the current SNR that the gNB receives the UE signal with. It should be close to the target in the gNB configuration file, `pusch_TargetSNRx10` , which should be around 10 times the value shown in the log

In the last lines:

- `MAC` shows the amount of MAC PDU bytes scheduled in transmit (`TX` , `1530943191`) and receive (`RX` , `194148`) directions
- `LCID X` shows the amount of MAC SDU/RLC PDU data for Logical Channel ID with ID `X` in transmit and receive directions. LCIDs 1 and 2 are mapped to SRBs 1 and 2. LCIDs 4 and onward are mapped to DRBs 1 onward. If you have an LCID 4, it means you have a PDU session.

Configuration of the MAC

Split-related options (running in a DU)

See **nFAPI documentation** or **Aerial**tutorial" for information about the (n)FAPI split. See @ref "../F1-design.md" "F1 documentation" for information about the F1 split. @subsection autotoc_md434 MAC scheduler-related configuration options The following options have an influence on the MAC scheduler operation (for all UEs, if applicable), either on the MAC scheduler operation directly or how a UE is configured. In the <tt>MACRLCs</tt> section of the gNB/DU configuration file: * <tt>ulsch_max_frame_inactivity</tt> (default 10): number of frames before a UE is scheduled automatically in UL. Lowering can improve latency at the expense of more resources in idle (which limits the number of UEs, and increases UE power consumption) * <tt>pusch_TargetSNRx10</tt> (default 200): target SNR in PUSCH times 10 (200 corresponds to 20dB target SNR) * <tt>pucch_TargetSNRx10</tt> (default 150): target SNR in PUCCH times 10 (as above) * <tt>ul_prbblack_SNR_threshold</tt> (default 10): target SNR for disabling RB scheduling in uplink on affected RBs. * <tt>pucch_FailureThres</tt> (default 10): number of DTX on PUCCH after which scheduler declares UE in radio link failure and moves it to "out-of-sync state". Currently not used. * <tt>pusch_FailureThres</tt> (default 10): number of DTX on PUSCH after which scheduler declares UE in radio link failure and moves it to "out-of-sync state" * <tt>dl_bler_target_upper</tt> (default 0.15): upper threshold of BLER (first round retransmission over initial transmission) to decrease MCS by 1 * <tt>dl_bler_target_lower</tt> (default 0.05): lower threshold of BLER (first round retransmission over initial transmission) to increase MCS by 1 * <tt>dl_max_mcs</tt> (default 28): maximum MCS to use * <tt>ul_bler_target_upper</tt> (default 0.15): as <tt>dl_bler_target_upper</tt> * <tt>ul_bler_target_lower</tt> (default 0.05): as <tt>dl_bler_target_lower</tt> * <tt>ul_max_mcs</tt> (default 28): as <tt>dl_max_mcs</tt> * <tt>dl_harq_round_max</tt> (default 4): maximum number of HARQ rounds, i.e., retransmissions to perform, in DL * <tt>ul_harq_round_max</tt> (default 4): as <tt>dl_harq_round_max</tt> * <tt>min_grant_prb</tt> (default 5): number of PRBs to schedule for UE after activity (see <tt>ulsch_max_frame_inactivity</tt>) or after scheduling request (SR) * <tt>min_grant_mcs</tt> (default 9): MCS to use

for UE after activity (see <tt>ulsch_max_frame_inactivity</tt>) or after scheduling request (SR) * <tt>identity_precoding_matrix</tt> (default 0=false): flag to enable to use only the identity precoding matrix in DL precoding * <tt>set_analog_beamforming</tt> (default 0=false): flag to enable analog beamforming (for more information @ref "/tmp/tmpqbhrntcs/doc/analog_beamforming.md" "`analog_beamforming.md`") * <tt>beam_duration</tt> (default 1): duration/number of consecutive slots for a given set of beams, depending on hardware switching performance * <tt>beams_per_period</tt> (default 1): set of beams that can be simultaneously allocated in a period (<tt>beam_duration</tt>) * <tt>pusch_RSSI_Threshold</tt>: Value between -1280 and 0 which maps to range from -128.0 dBm/dBFS to 0.0 dBm/dBFS. This limits PUSCH TPC commands in case RSSI reaches the threshold and prevents ADC railing. Unit depends on RSSI reporting config. * <tt>pucch_RSSI_Threshold</tt>: Same as above but for PUCCH In the <tt>gNBs</tt> section of the gNB/DU configuration file: some of the parameters affect RRC configuration (CellGroupConfig) of a UE, and are therefore listed here, also they pertain to the DU, i.e., the scheduler. Note also that some SIBs are configured at the DU and some at the CU; please consult the @ref "/tmp/tmpqbhrntcs/doc/RRC/rrc-usage.md" "RRC configuration" as well for SIB configuration. * <tt>pdsch_AntennaPorts_XP</tt> (default 1): number of XP logical antenna ports in PDSCH (see also @ref "/tmp/tmpqbhrntcs/doc/RUNMODEM.md" "`RUNMODEM.md`") * <tt>pdsch_AntennaPorts_N1</tt> (default 1): number of horizontal logical antenna ports in PDSCH (see also @ref "/tmp/tmpqbhrntcs/doc/RUNMODEM.md" "`RUNMODEM.md`") * <tt>pdsch_AntennaPorts_N2</tt> (default 1): number of vertical logical antenna ports in PDSCH (see also @ref "/tmp/tmpqbhrntcs/doc/RUNMODEM.md" "`RUNMODEM.md`") * <tt>pusch_AntennaPorts</tt> (default 1): number of antenna ports in PUSCH * <tt>maxMIMO_layers</tt> (default -1=unlimited): maximum number of MIMO layers to use in downlink * <tt>sib1_tda</tt> (default 1): time domain allocation (TDA) indices to use for SIB1 (38.214 section 5.1.2.1.1) * <tt>do_CSIRS</tt> (default 0): flag whether to use channel-state information reference signal (CSI-RS) * <tt>do_SRS</tt> (default 0): flag whether to use sounding reference signal (SRS) * <tt>do_SINR</tt> (default 0): flag whether to enable CSI reporting of SSB-SINR (introduced in rel16) Default setting of CSI reporting quantity is SSB-RSRP. * <tt>min_rxtxtime</tt> (default 2): minimum feedback time for UE to respond to transmissions (k1 and k2 in 3GPP spec) * <tt>ul_prbblacklist</tt>: PRBs that should not be used for UL scheduling. Cf with parameter <tt>ul_prbblack_SNR_threshold</tt> that does a similar thing based on measurements * <tt>force_256qam_off</tt> (default 0=false): flag whether to disable 256QAM (limit to 64QAM) in DL * <tt>force_UL256qam_off</tt> (default 0=false): flag whether to disable 256QAM (limit to 64QAM) in DL * <tt>disable_harq</tt> (default 0=false): flag whether to disable HARQ completely (useful for NTN operation, see <../RUNMODEM.md>). this is a Rel-17 feature and you need to have a capable UE for this * <tt>use_deltaMCS</tt> (default 0=false): flag whether to enable deltaMCS (this is not fully tested and might not work and you might need to

adjust other parameters such as target SNRs) * <tt>num_dlharq</tt> (default 16): number of HARQ processes to use in DL (other valid options are 2, 4, 6, 8, 10, 12, 32; 32 is a Rel-17 features) * <tt>num_ulharq</tt> (default 16): as <tt>num_dlharq</tt> for UL (other valid option is 32; 32 is i Rel-17 feature) - <tt>du_sibs</tt> (default <tt>[]</tt>): list of SIBs to transmit in the cell. Currently, SIB19 (for NTN) is supported. <table class="markdownTable"> <tr class="markdownTableHead"> <th class="markdownTableHeadNone"> DL MIMO

`do_CSIRS`

`do_SINR`

CSI report Quantity

OFF (pdsch_AntennaPorts = 1)

0

0

SSB-RSRP

OFF (pdsch_AntennaPorts = 1)

0

1

SSB-SINR

OFF (pdsch_AntennaPorts = 1)

1

0

CSI-Reference signal RSRP

OFF (pdsch_AntennaPorts = 1)

1

1

CSI-Reference signal SINR (not supported yet)

ON (pdsch_AntennaPorts > 1)

1

0

cri-RI-PMI-CQI

DL-MIMO is configured using following parameters: `pdsch_AntennaPorts_XP` , `pdsch_AntennaPorts_N1` , `pdsch_AntennaPorts_N2` , `maxMIMO_layers` (see also **`RUNMODEM.md`**)

ServingCellConfigCommon parameters

The `gNBs` configuration section has a big structure `servingCellConfigCommon` that has an influence on the overall behavior of MAC and L1. As the name says, this structure is a flat representation of the ServingCellConfigCommon structure, specified by 3GPP. Describing all of these parameters would be too exhaustive; more information about the individual fields can be found in 3GPP TS 38.331, section 6.3.2 "Radio resource control information elements".

Below is a description of some of these parameters.

Frequency configuration

There are many parameters, such as `absoluteFrequencySSB` , etc., that have an impact on the frequency used by the gNB. For more information, please check the **corresponding document**.

TDD pattern configuration

The TDD configuration parameters allow to use one or two TDD patterns.

Single TDD pattern

Configure the TDD pattern through these options:

- `dl_UL_TransmissionPeriodicity` : Refers to the UL/DL slots periodicity for the TDD pattern. See below for valid numbers.
- `nrofDownlinkSlots` : Refers to the number of consecutive DL slots in the TDD pattern. The number of DL slots depends on the TDD period.
- `nrofDownlinkSymbols` : Indicates the number of consecutive DL symbols within the special slot that follows the downlink slots in the TDD pattern. The special slot is needed only to switch from DL to UL. The maximum number of symbols in this slot is 14.
- `nrofUplinkSlots` : Refers to the number of consecutive UL slots in the TDD pattern, depending on the desired TDD period.

- `nrofUplinkSymbols`: Indicates the number of consecutive UL symbols within the special slot that follows the downlink slots in the TDD pattern. The sum of downlink and uplink symbols should not exceed 14.
- `prach_ConfigurationIndex`: index for PRACH according to tables 6.3.3.2-2 to 6.3.3.2-4 in TS 38.211.

As an example, the below figure shows a single TDD pattern, consisting of 3 DL slots, 1 mixed slots (with 10 DL, 2 guard, 2 UL symbols), and 1 UL slot.

TDD Frame Structure

To configure this pattern in the configuration file, use

```
#dl_UL_TransmissionPeriodicity  0=ms0p5, 1=ms0p625, 2=ms1, 3=ms1p25, 4=ms2, 5=ms2p5, 6=ms5,
        7=ms10, 8=ms3, 9=ms4

dl_UL_TransmissionPeriodicity = 5;
nrofDownlinkSlots             = 3;
nrofDownlinkSymbols           = 10;
nrofUplinkSlots               = 1;
nrofUplinkSymbols             = 2;
```

The `dl_UL_TransmissionPeriodicity` is set to `5` (2.5ms). The above figure shows two TDD periods over 5ms. The 10 ms frame period must be strictly divisible by the sum of the TDD pattern periods.

Two TDD patterns

An optional `pattern2` structure is used to signal a TDD pattern 2. In this case, the TDD pattern may have two extended values of 3 ms and 4 ms. These values are standardized as `dl-UL-TransmissionPeriodicity-v1530`. For the sake of simplicity of the gNB configuration file, we have extended the set of `dl-UL-TransmissionPeriodicity` values to support the new TDD periods, as explained in the table below. However, these values will later be encoded as `dl-UL-TransmissionPeriodicity-v1530` to match the specifications.

dl_UL_TransmissionPeriodicity	TDD period
0	0.5 ms
1	0.625 ms
2	1 ms
3	1.25 ms
4	2 ms
5	2.5 ms
6	5 ms
7	10 ms
8	3 ms
9	4 ms

The 10 ms frame period must be strictly divisible by the sum of the TDD pattern periods. For example, the 3 ms TDD pattern should be used with a second TDD pattern of 2 ms. Additionally, the 4 ms TDD pattern should be used with a second TDD pattern of 1 ms.

As an example, this configuration might be used, where the first TDD pattern is followed by a second, consisting of 4 DL slots.

```
# pattern1
# dl_UL_TransmissionPeriodicity
# 0=ms0p5, 1=ms0p625, 2=ms1, 3=ms1p25, 4=ms2, 5=ms2p5, 6=ms5, 7=ms10
# ext: 8=ms3, 9=ms4
dl_UL_TransmissionPeriodicity  = 8;
nrofDownlinkSlots              = 3;
nrofDownlinkSymbols            = 6;
nrofUplinkSlots                = 2;
nrofUplinkSymbols              = 4;

# pattern2
pattern2: {
    dl_UL_TransmissionPeriodicity2 = 4;
    nrofDownlinkSlots2             = 4;
    nrofDownlinkSymbols2           = 0;
    nrofUplinkSlots2               = 0;
    nrofUplinkSymbols2             = 0;
};
```

rrc-usage

This document describes the basic functioning of the 5G RRC layer, describes the periodic output, and explains the various configuration options that influence its behavior.

Developer documentation, such as UE connection control flow, reestablishment, or handover, are described in **a separate page**.

[[TOC]]

General

The RRC layer controls the basic connection setup of UEs as well as additional procedures. It is the fundamental building block of OAI's CU-CP, and interacts with lower layers (DU, basically MAC and RLC) through F1AP messages, and with the CU-UP through E1AP messages. More information can be found in the respective F1AP page and **E1AP page**.

Periodic output and interpretation

Similarly to the scheduler, the RRC periodically prints information about connected UEs and DUs into file `nrRRC_stats.log` in the current working directory of the executable running the RRC (typically, `nr-softmodem`). The output lists first all UEs that are currently connected, and then all DUs, in order.

For each UE, it prints:

```
UE 0 CU UE ID 1 DU UE ID 40352 RNTI 9da0 random identity c0f1ac9824000000:
    last RRC activity: 5 seconds ago
    PDU session 0 ID 10 status established
    associated DU:  DU assoc ID 8
```

where `UE 0` is the UE index, CU UE ID and DU UE IDs are the IDs used to exchange over F1 (cf. scheduler logs). Further, it shows RNTI, when the last RRC activity happened, the status of PDU sessions and which DU is associated (through the SCTP association ID).

For each DU, it prints:

```
1 connected DUs
[1] DU ID 3584 (gNB-OAI-DU) assoc_id 8: nrCellID 12345678, PCI 0, SSB ARFCN 641280
    TDD: band 78 ARFCN 640008 SCS 30 (kHz) PRB 106
```

i.e., an index (`[1]`), the DU ID and it's name, the SCTP association ID (`assoc_id 8` , cf. UE information), and DU specific information for the cell (cell ID, physical cell identity/ PCI, the SSB frequency in ARFCN notation, the band and Point A ARFCN, subcarrier spacing/SCS, and the number of resource blocks/PRB). Only one cell per DU is supported.

As of now, it does not print information about connected CU-UPs or AMFs.

Configuration of the RRC

Split-related options (when running in a CU or CU-CP)

See F1 documentation for information about the F1 split. See **E1 documentation** for information about the E1 split.

RRC-specific configuration options

In the `gNBs` section of the gNB/CU/CU-CP configuration file is the RRC-specific configuration

cell-specific options

Note that some SIBS are configured at the CU and some at the DU; please consult the **MAC configuration** as well for SIB configuration.

- `gNB_ID` and `gNB_name` : ID and name of the gNB
- `tracking_area_code` : the current tracking area code in the range `[0x0001, 0xfffd]`
- `plmn` : the PLMN, which is a list of entries consisting of:
  - `mcc` : mobile country code
  - `mnc` : mobile network code
  - `mnc_length` : length of mobile network code, allowed values: 2, 3
  - `snssaiList` : list of NSSAI (network selection slice assistence information, "slice ID"), which itself consists in
    - `sst` : slice service type, in `[1,255]`

    - `sd` (default `0xffffff`): slice differentiator, in `[0,0xffffff]`, `0xffffff` is a reserved value and means "no SD" Note that: SST=1, no SD is "eMBB"; SST=2, no SD is "URLLC"; SST=3, no SD is "mMTC"

- `enable_sdap` (default: false): enable the use of the SDAP layer. If deactivated, a transparent SDAP header is prepended to packets, but no further processing is being done.
- `cu_sibs` (default: `[]`) list of SIBs to give to the DU for transmission. Currently, SIB2 is supported.

UE-specific configuration

- `um_on_default_drb` (default: false): use RLC UM instead of RLC AM on default bearers
- `drbs` (default: 0): the number of DRBs to allocate for a UE, only useful for do-ra or phy-test testing

Neighbor-gNB configuration

```
TBD
```

Refer to the **handover tutorial** for more information.

E1-design

[[TOC]]

1. Introduction

E1 is the interface that lies between the nodes CU Control Plane (CUCP) and CU User Plane (CUUP). Once the nodes are configured, all user plane traffic flows through CUUP.

The E1 design in OAI follows the 3GPP specification in TS 38.460/463. The code design on E1 in OAI is very similar to F1: OAI uses E1 internally to set up bearers, even in monolithic gNB, where the actual "way" the messages are passed is abstracted through function pointers. More specifically, there are handlers that handle the E1 messages as shown below, which are the same no matter if the CU is integrated or separate in CU-CP/UP. The below sequence diagram lists the handlers that are executed and the current E1AP message flow (SCTP connection setup only in split-mode, the rest is the same for CU):

```
sequenceDiagram
  participant c as CUCP
  participant u as CUUP

  Note over c,u: E1 Setup Procedure
  u->c: SCTP connection setup (in split mode)
  u->>c: E1 Setup Request
  Note over c: Execute rrc_gNB_process_e1_setup_req()
  c->>u: E1 Setup Response
  c-->>u: E1 Setup Failure

  Note over c,u: E1 Bearer Context Setup Procedure
  Note over c: Receives PDU session setup request from AMF
  c->>u: Bearer Context Setup Request
  Note over u: Execute e1_bearer_context_setup()<br/>Configure DRBs and create GTP Tunnels
        for F1-U and N3
  u->>c: Bearer Context Setup Response
  Note over c: Execute rrc_gNB_process_e1_setup_req()<br/>Sends F1-U UL TNL info to DU and
        receives DL TNL info

  Note over c,u: E1 Bearer Context Modification (CU-CP Initiated) Procedure
  c->>u: Bearer Context Modification Request
  Note over u: Execute e1_bearer_context_modif()<br/>Updates GTP Tunnels with received info
  u->>c: Bearer Context Modification Response
  Note over c: Execute rrc_gNB_process_e1_bearer_context_modif_resp()

  Note over c: UE release triggered
  c->>u: Bearer Context Release Request
  Note over u: Execute e1_bearer_release_cmd()<br/>Frees GTP tunnels, PDCP data etc
  u->>c: Bearer Context Release Complete
  Note over c: Execute rrc_gNB_process_e1_bearer_context_release_cplt()
```

The implementation of callbacks towards these handlers is distributed across the following file, depending on the operating mode:

Mode	CP=>UP messages	UP=>CP messages
Integrated CU	`cucp_cuup_direct.c`	`cuup_cucp_direct.c`
E1 Split	`cucp_cuup_e1ap.c`	`cuup_cucp_e1ap.c`

As long as concerns E1 Interface Management Procedures, the code flow of request messages towards northbound looks like this:

```
sequenceDiagram
  participant r as RRC
  participant c as CUCP
  participant u as CUUP

  Note over u: E1AP_CUUP_task (SCTP Handler)
  Note over u: ASN1 encoder
  u->>c: e.g. E1 Setup Request (SCTP)
  Note over c: E1AP_CUCP_task (SCTP Handler)
  Note over c: ASN1 decoder
  c->>r: E1AP_SETUP_REQ (ITTI)
  Note over r: TASK_RRC_GNB (RRC Handler)
  r->>c: E1AP_SETUP_RESP (ITTI)
  Note over c: E1AP_CUCP_task (E1AP Callback)
  Note over c: ASN1 encoder

  c->>u: e.g. E1 Setup Response/Failure
  Note over u: E1AP_CUUP_task (SCTP Handler)
  Note over u: ASN1 decoder
```

More details about the E1AP procedures in OAI are available in this document: **E1 Procedures**.

2. Running the E1 Split

The setup is assuming that all modules are running on the same machine. The user can refer to the **F1 design document** for local deployment of the DU.

2.1 Configuration File

The gNB is started based on the node type that is specified in the configuration file. The following parameters must be configured accordingly.

On either CUCP and CUUP:

- The southbound transport preference `gNBs.[0].tr_s_preference` set to `f1`
- config section `E1_INTERFACE` should be present

On the CU-CP:

- `type` parameter within the `E1_INTERFACE` should be set to `cp`

On the CU-UP:

- `type` parameter within the `E1_INTERFACE` should be set to `up`

Executables:

- executable `nr-softmodem` to run a CU-CP
- executable `nr-cuup` to run the CU-UP

In the `E1_INTERFACE` configuration section, the parameters `ipv4_cucp` and `ipv4_cuup` must be configured to specify the IP addresses of the respective network functions.

For CUCP, a typical `E1_INTERFACE` config looks like:

```
E1_INTERFACE =
(
  {
    type = "cp";
    ipv4_cucp = "127.0.0.4";
    ipv4_cuup = "127.0.0.5";
  }
)
```

For CUUP, it is:

```
E1_INTERFACE =
(
  {
    type = "up";
    ipv4_cucp = "127.0.0.4";
    ipv4_cuup = "127.0.0.5";
  }
)
```

One could take an existing CU configuration file and add the above parameters to run the gNB as CUCP or CUUP.

The CUUP uses the IP address specified in `gNBs.[0].local_s_address` for F1-U and `GNB_IPV4_ADDRESS_FOR_NGU` for N3 links. Note that `local_s_address` is under `gNBs` and `GNB_IPV4_ADDRESS_FOR_NGU` is part of the `NETWORK_INTERFACES` config member.

Alternatively, you can use the config files `ci-scripts/conf_files/gnb-cucp.sa.f1.conf` and `ci-scripts/conf_files/gnb-cuup.sa.f1.conf`.

2.2 Steps to Run the Split in rfsimulator with OAI UE

Note: A 5G core must be running at this point. Steps to start the OAI 5G core can be found in the oai-cn5g-fed repository or here.

1. Open wireshark to capture the E1AP messages. You might set the capture filter to `sctp` to limit the number of captured packages.
2. Start the CUCP first by running the following command

```
sudo ./nr-softmodem -O ../../../ci-scripts/conf_files/gnb-cucp.sa.f1.conf --gNBs.
       [0].min_rxtxtime 6
```

Note that `min_rxtxtime` should be set to `6` only when you are connecting an OAI UE to the gNB.

1. Start the CUUP and DU (in any order)

CUUP (has its own executable):

```
sudo ./nr-cuup -O ../../../ci-scripts/conf_files/gnb-cuup.sa.f1.conf
```

DU:

```
sudo ./nr-softmodem -O ../../../ci-scripts/conf_files/gNB_SA_DU.conf --rfsim
```

You need to use `--rfsim` if you are running the test with rfsimulator.

1. Start OAI UE or COTS UE.

OAI UE:

```
sudo ./nr-uesoftmodem -r 106 --numerology 1 --band 78 -C 3619200000 --rfsim --
       rfsimulator.serveraddr 127.0.0.1
```

3. Configuration file IP addresses of network functions

You can also run the nodes on different machines. If you do so please change the interface parameters accordingly and make sure the interfaces are reachable. Please refer to the following figure for an overview of all parameters.

E1/F1/NG parameters

PDF version | LaTeX/TikZ version if you want to modify to reflect your setup

4. Multiple CU-UP

It is possible to connect multiple CU-UP to the same CU-CP. In the handler of the E1 Setup Request, the CU-CP verifies that the **PLMN(s)** (MCC, MNC) between both nodes match, and that no CU-UP with the same ID exists. Note that the NSSAIs for the CU-UPs can be different.

During attach, the following rules are used to select a CU-UP at the CU-CP, in order:

- Select a CU-UP based on slice identifiers:
  - If both the SST and SD match exactly, that CU-UP is selected. (If multiple have the same NSSAI, the first is always selected.)
  - Otherwise, the first CU-UP with the same SST but different SD is selected.
- Round-robin across all CU-UPs.
- The first CU-UP.
- If no CU-UP is selected, the CU-CP asserts.

Note that CU-UPs are not released from CU-CP internal structures. That means that you have to restart the CU-CP if you want to connect the CU-UP again (e.g., after a crash). The CU-CP might also misfunction during attach if a CU-UP was connected, but disconnected in the meantime.

5. Abnormal conditions

- The CU-UP goes offline during normal operation (e.g. UEs have a valid PDU Session and are exchanging data on the UP): after restarting the CU-UP, the UP is not restored and the user will notice GTP errors. In this case the UEs have to reconnect.

F1-design

F1 split design

[[TOC]]

Introduction

The F1 interface is the functional split of 3GPP between the CU (centralized unit: PDCP, RRC, SDAP) and the DU (distributed unit: RLC, MAC, PHY). It is standardized in TS 38.470 - 38.473 for 5G NR.

F1 specs:

- 3GPP TS 38.470 F1 general aspects and principles
- 3GPP TS 38.471 F1 layer 1

F1-C:

- 3GPP TS 38.472 F1 signalling transport
- 3GPP TS 38.473 F1 **Application** Protocol (F1AP)

F1-U:

- 3GPP TS 38.474 F1 Data Transport

No equivalent for 4G exists.

Control plane (F1-C)

The interface F1-C is designed for the exchange of signalling messages between the Radio Network Layer (RNL) and the Transport Network Layer (TNL). It consists of F1 **Application** Protocol messages (F1-AP) exchanged over SCTP.

Data plane (F1-U)

F1-U uses GTP-U for information exchange.

OAI Implementation Status

The implementation of F1AP messages is seamlessly integrated into OAI, supporting both Monolithic SA and CU/DU functional split modes. The F1 code is therefore always compiled with nr-softmodem. This architecture ensures that even when operating a

monolithic gNB, internal information exchange always utilizes F1AP messages. The major difference lies in the CU/DU split scenario, where ASN.1-encoded F1AP messages (F1-C) are exchanges over SCTP, via a socket interface.

This is the current status:

- gNB-CU/gNB-DU split
- Supported deployments:
  - SA
  - Single cell per DU
  - Multiple DUs connected to one CU (both CP and UP)
  - **Mobility between DUs connected to a single CU**
- Not supported:
  - NSA

F1-C

F1AP messages

Refer to FEATURE_SET.md to learn about the current F1AP implementation status.

High-level F1-C code structure

The F1 interface is used internally between CU (mostly RRC) and DU (mostly MAC) to exchange information. In DL, the CU sends messages as defined by the callbacks in `mac_rrc_dl.h`, whose implementation is defined in files `mac_rrc_dl_direct.c` (monolithic) and `mac_rrc_dl_f1ap.c` (for F1AP). In the monolithic case, the RRC calls directly into the message handler on the DU side (`mac_rrc_dl_handler.c`). In the F1 case, an ITTI message is sent to the CU task, sending an ASN.1-encoded F1AP message. The DU side's DU task decodes the message, and then calls the corresponding handler in `mac_rrc_dl_handler.c`. Thus, the message flow is the same in both F1 and monolithic cases, with the difference that F1AP encodes the messages using ASN.1 and sends over a socket.

In UL, the callbacks defined in `mac_rrc_ul.h` are implemented by `mac_rrc_ul_direct.c` (monolithic) and `mac_rrc_ul_f1ap.c` (F1). In the direct case, an ITTI message is directly sent to the RRC task (hence, there is no dedicated handler). In F1, the DU task receives the ITTI message, encodes using ASN.1, and sends it over a network socket. The CU task decodes, and sends the same ITTI message to the RRC task as done directly in the monolithic case.

Summary of callbacks and handlers:

Entity	Callback definition	Callback F1 Implementation	Callback Monolithic SA Implementation	Handler
CU/ RRC	`mac_rrc_dl.h`	`mac_rrc_dl_f1ap.c`	`mac_rrc_dl_direct.c`	No handler (use ITTI)
DU/ MAC	`mac_rrc_ul.h`	`mac_rrc_ul_f1ap.c`	`mac_rrc_ul_direct.c`	`mac_rrc_dl_handler`

A sequence diagram for downlink F1AP messages over the OAI CU/DU functional split:

```
sequenceDiagram
    box rgba(17,158,189,255) CU/RRC
    participant TASK_RRC_GNB
    participant TASK_CU_F1
    end
    Note over TASK_RRC_GNB: MAC/RRC callback
    TASK_RRC_GNB->>+TASK_CU_F1: F1AP message (ITTI)
    Note over TASK_CU_F1: F1 message encoding
    Note over TASK_CU_F1: ASN.1 encoding
    box Grey DU/MAC
    participant TASK_DU_F1
    participant MAC
    end
    Note over TASK_DU_F1: F1AP DL message handler
    TASK_CU_F1->>+TASK_DU_F1: SCTP (ITTI)
    Note over TASK_DU_F1: F1 message decoding
    Note over TASK_DU_F1: ASN.1 decoding
    TASK_DU_F1->>+MAC: F1AP message (function call)
    Note over MAC: MAC DL message handler
```

and for the uplink F1AP messages:

```
sequenceDiagram
    box rgba(17,158,189,255) CU/RRC
    participant TASK_RRC_GNB
    participant TASK_CU_F1
    end
    box Grey DU/MAC
    participant TASK_DU_F1
    participant TASK_MAC_GNB
    end
    Note over TASK_MAC_GNB: MAC/RRC callback
    TASK_MAC_GNB->>+TASK_DU_F1: F1AP message (ITTI)
    Note over TASK_DU_F1: F1 message encoding
    Note over TASK_DU_F1: ASN.1 encoding
    Note over TASK_CU_F1: F1AP UL message handler
    TASK_DU_F1->>+TASK_CU_F1: SCTP (ITTI)
    Note over TASK_CU_F1: F1 message decoding
    Note over TASK_CU_F1: ASN.1 decoding
    TASK_CU_F1->>+TASK_RRC_GNB: F1AP message (ITTI)
```

Alternative sequence handling (e.g. Monolithic), for downlink:

```
sequenceDiagram
    box rgba(17,158,189,255) RRC
    participant TASK_RRC_GNB
    end
    Note over TASK_RRC_GNB: mac_rrc_dl_direct.c callback
    box Grey MAC
    participant TASK_MAC_GNB
    end
    TASK_RRC_GNB->>+TASK_MAC_GNB: raw F1AP message
    Note over TASK_MAC_GNB: mac_rrc_dl_handler.c
```

and for the uplink:

```
sequenceDiagram
    box rgba(17,158,189,255) RRC
    participant TASK_RRC_GNB
    end
    box Grey MAC
    participant TASK_MAC_GNB
    end
    Note over TASK_MAC_GNB: mac_rrc_ul_direct.c callback
    TASK_MAC_GNB->>+TASK_RRC_GNB: raw F1AP message (ITTI)
```

F1-U

Current status:

- Buffer status report over GTP-U
- Each packet is acknowledged individually
- Support of multiple DUs per CU

How to run

As mentioned earlier, OAI uses F1 internally. It is always compiled in. To start CU/DU, you use `./nr-softmodem` with the appropriate configuration files, for instance:

```
sudo cmake_targets/ran_build/build/nr-softmodem -O ci-scripts/conf_files/gnb-
        cu.sa.band78.106prb.conf
sudo cmake_targets/ran_build/build/nr-softmodem -O ci-scripts/conf_files/gnb-
        du.sa.band78.106prb.rfsim.conf
```

These files are tested in the CI, and are configured for use in docker, see this `docker-compose` file.

The rules to decide if a config triggers a start of a DU, CU, or monolithic gNB, are, in order:

1. If the `MACRLCs` section lists `f1` as **northbound transport preference** (`tr_n_preference`), it is a DU.
2. If the `gNBs` section lists `f1` as a **southound transport preference** (`tr_s_preference`), it is a CU.
3. It is a (monolithic) gNB.

F1 IP configuration for Local network deployment of F1

The following paragraphs explain the IP configuration for F1 in the OAI config files on the example of a a local deployment. We assume that the CU will bind on `192.168.70.129` towards the core, `127.0.0.3` towards the DU, and the DU `127.0.0.4` towards the CU.

In the CU file:

- Update the `gNBs.[0].amf_ip_address` and `gNBs.[0].NETWORK_INTERFACES` section towards the core (typically, OAI CN is configured to provide a docker bridge on `192.168.70.129` and the AMF is on `192.168.70.132`):
  - `gNBs.[0].amf_ip_address.[0].ipv4 192.168.70.132`
  - `gNBs.[0].NETWORK_INTERFACES.GNB_IPV4_ADDRESS_FOR_NG_AMF 192.168.70.129`
  - `gNBs.[0].NETWORK_INTERFACES.GNB_IPV4_ADDRESS_FOR_NGU 192.168.70.132`
- Set `gNBs.[0].tr_s_preference` (transport south-bound) to `f1`
- Update the `gNBs.[0].local_s_address` (CU-local south address) for the F1-C/SCTP and F1-U/GTP/UDP south-bound interfaces: `127.0.0.3`
- Note: the `gNBs.[0].remote_s_address` (CU-remote/DU south address) is ignored, but we recommend to put `0.0.0.0` ("any")
- Ports should match the ones in the DU config, but for simplicity and standards-conformity, simply set all to 2152:
  - `local_s_portc` in CU should match `remote_n_portc` in DU
  - `local_s_portd` in CU should match `remote_n_portd` in DU

    - `remote_s_portc` in CU should match `local_n_portc` in DU
    - `remote_s_portd` in CU should match `local_n_portd` in DU

In the DU file:

- Set `MACRLCs.[0].tr_n_preference` to `f1`
- Update `MACRLCs.[0].local_n_address` (local north-bound address of the DU) to `127.0.0.4`. This IP address is used to bind the F1-C/SCTP and F1-U/GTP/UDP socket.
- If you need to bind to different addresses for F1-C and F1-U, there is a second option `MACRLCs.[0].local_n_address_f1u` which is used to bind F1-U/GTP/UDP instead; F1-C/SCTP will still bind to the address in `MACRLCs.[0].local_n_address`.
- Update `MACRLCs.[].remote_n_address` (remote north-bound address of the CU) to `127.0.0.3`. This IP address is used as the CU destination IP address for F1AP communication.

Note: at the DU, you can bind to different interfaces for F1-C and F1-U with the options `MACRLCs.[0].local_n_address` and `MACRLCs.[0].local_n_address_f1u`, respectively. Note that this is not foreseen for the CU; in the case of the CU, please use separate CU-UP and CU-CP.

Configuration of multiple DUs

Upon F1 Setup Request of a new DU, the CU cross-checks that

- no DUs have the same gNB-DU IDs, taken from `gNBs.[0].gNB_ID`, and
- no cells have the same NR Cell ID, taken from `gNBs.[0].nr_cellid`, and
- no cells have the same physical cell IDs, taken from `gNBs.[0].servingCellConfigCommon.[0].physCellId`.

If any of these are the same, the CU will reject the DU with an F1 Setup Failure. Thus, you should make sure that in the DU's configs, these parameters are set differently.

You have to of course make sure that the local interface of the DU (`MACRLCs.[0].local_n_address`) is different as well.

Assuming you use RFsim, you should make the RFsimulator server side (typically the gNB) bind on different hosts (`rfsimulator.serverport`).

Code documentation

Common multi-threading architecture

The CU and DU interfaces are based on ITTI threads (see `common/utils/ocp_itti/itti.md`) adopted by all OAI upper layers to run isolated threads dedicated to one feature.

```
graph TD
Linux[configuration file] --> APP[gNB_app_task]
Socks[Linux UDP] --> GTP[nr_gtpv1u_gNB_task]
APP --> CU
APP --> DU
A[Linux SCTP] -->|sockets| SCTP(OAI itti thread TASK_SCTP)
SCTP-->|TASK_NGAP| NG[ngap_gNB_task]
NG-->|TASK_SCTP| SCTP
SCTP-->|TASK_DU_F1| DU[F1AP_DU_task]
SCTP-->|TASK_CU_F1| CU[F1AP_CU_task]
CU-->|TASK_RRC_GNB| RRC
RRC -->|non ITTI| PHY[OAI L1+L2]
RRC -->|API calls to create tunnels| GTP
PHY --> |user plane| GTP
CU --> |TASK_GTPV1_U| GTP
DU --> |TASK_GTPV1_U| GTP
```

A "task" is a Linux thread running an infinite waiting loop on one ITTI queue.

The `app` task manages the initial configuration; the `sctp` task manages the SCTP Linux sockets.

The CU encoding/decoding runs in the task executing `F1AP_CU_task()`. It stores its private data context in a static variable. The DU encoding/decoding runs in the task executing `F1AP_DU_task()`. The design is similar to CU task.

All GTP-U tunnels are managed in a Linux Thread, that have partially ITTI design:

1. tunnel creation/deletion is by C API functions direct calls (a mutex protects it)
2. outgoing packets are pushed in a ITTI queue to the gtp thread
3. incoming packets are sent to the tunnel creator using a C callback (the callback function is given in tunnel creation order). The callback should not block

F1-C messages towards the CU

The CU thread starts when the CU starts. It opens listening socket on the configuration-specified IP/port by sending the appropriate message to `TASK_SCTP`. It will accept any incoming connection and forward any F1 message to RRC.

You can trace any message by looking up the handler `cu_task_handle_sctp_data_ind()`, which ultimately triggers a corresponding ITTI message sent to RRC. All ITTI messages arriving are dispatched in the ITTI message handling function in `rrc_gNB.c`. For your convenience, find below a table of same messages that might arrive from the DU and how they are forwarded.

After handling messages, the RRC uses a callback that determines if messages go directly to the MAC or are sent to the CU ITTI task (see above)

You might also want to consult TS 38.401 regarding the message exchange.

Incoming F1 message	ITTI message to RRC	RRC handler
F1 Setup Request	`F1AP_SETUP_REQ`	`rrc_gNB_process_f1_setup_req()`
Initial UL RRC Message Transfer	`F1AP_INITIAL_UL_RRC_MESSAGE`	`rrc_gNB_process_initial_ul_rrc_`
UL RRC Message Transfer	`F1AP_UL_RRC_MESSAGE`	`rrc_gNB_decode_dcch()`
UE Context Response	`F1AP_UE_CONTEXT_SETUP_RESP`	`rrc_CU_process_ue_context_setup`
UE Context Modification Response	`F1AP_UE_CONTEXT_MODIFICATION_RESP`	`rrc_CU_process_ue_context_modif`
UE Context Modification Required	`F1AP_UE_CONTEXT_MODIFICATION_REQUIRED`	`rrc_CU_process_ue_modification_`
UE Context Release Request	`F1AP_UE_CONTEXT_RELEASE_REQ`	`rrc_CU_process_ue_context_relea`
UE Context Release Complete	`F1AP_UE_CONTEXT_RELEASE_COMPLETE`	`rrc_CU_process_ue_context_relea`

F1-C Messages towards the DU

The task "gNB app", after reading the configuration file, sends a message `F1AP_DU_REGISTER_REQ` to the DU task. This message contains network configuration for the establishment of the SCTP connection, as well as the F1 Setup Request to be sent to the CU.

Using the network configuration, the DU task sets up an SCTP connection via the SCTP task. When it receives from the SCTP task the socket creation success, the DU task encodes and sends the F1 setup message to the CU.

Similarly as for the CU, you can trace any message by looking up the handler `du_task_handle_sctp_data_ind()`. Unlike in the CU, the decoders do NOT send an ITTI message, but directly call into the scheduler via a message handling function. This handler acquires the scheduler mutex and executes the corresponding message handling functionality. All handlers are in `mac_rrc_dl_handler.c`.

After handling messages, the MAC uses a callback that determines if messages are sent back to RRC via ITTI, or are sent to the DU ITTI task (see above)

You might also want to consult TS 38.401 regarding the message exchange.

Incoming F1 message	MAC handler	Comments
F1 Setup Response	`f1_setup_response()`	the DU does not start the radio before receiving this message
DL RRC Message Transfer	`dl_rrc_message_transfer()`	Processing timer started if reconfiguration is present[^footnote-reconfig]
UE Context Setup Request	`ue_context_setup_request()`	
UE Context Modification Request	`ue_context_modification_request()`	
UE Context Modification Confirm	`ue_context_modification_confirm()`	
UE Context Modification Refuse	`ue_context_modification_refuse()`	Will trigger release request
UE Context Release Command	`ue_context_release_command()`	Starts timer for release if UE in sync

[^footnote-reconfig]: The DU does not decode RRC messages, and does not know whether an RRC message is a reconfiguration. However, the spec mandates that a

reconfiguration has to be triggered if the CU receives a CellGroupConfig, originating at the DU. See also flag `expect_reconfiguration`.

F1-U messages

General

In the DU in UL, RLC checks in `deliver_sdu()` if we are operating in split mode, and either (direct) calls `pdcp_data_ind` (DRB) or (f1ap) sends a GTP message through the GTP API.

In the CU in UL, assuming the tunnel is in place, GTP decapsulates the packet and calls the callback `cu_f1u_data_req()`, which calls `pdcp_data_ind()` in CU.

In the CU in DL, the PDCP function `deliver_pdu_drb_gnb()` either (direct) calls into the RLC via `enqueue_rlc_data_req()`, or (f1ap) sends a GTP message through the GTP API.

In the DU in DL, assuming the GTP-U tunnel exists, GTP decapsulates the packet and calls the reception call back `du_rlc_data_req()`, which calls `enqueue_rlc_data_req()` in turn.

Tunnel Setup

In GTP-U, TS 29.281 specifies a optional header (NR RAN Container). This extension header may be transmitted in a G-PDU over the X2-U, Xn-U and F1-U user plane interfaces, but it is not mandatory.

Note that the GTP-U uses internal tables. Instead of exposing TEIDs, it uses the UE ID (RNTI, RRC UE ID) and Radio Bearer ID to map to a TEID for outgoing GTP-U packets. Upon receiving, it does the reverse and calls the callbacks. In the CU case, the DRB tunnel to DU and the tunnel on N3 have the same keys (RRC UE ID, RB ID), but they run in two different GTP-U instances. Each instance binds on diffrent sockets, which is the reason you cannot have the same interface for F1-U and N3/NG-U interfaces (this can be considered a design flaw).

In F1AP, for each "DRB to Be Setup Item IEs", we have a field TNL (transport network layer) to set a specific GTP tunnel (@IP, TEid). This is the same for each message related to DRBs.

So, For each F1AP containing DRB setup/modification/deletion, the related GTP-U tunnels will be modified one to one. The exception is the intialisation of new tunnel: in the call to tunnel creation, we need to send the remote TEID, but we don't know yet if we are the initial source. In this case, we issue a dummy (0xFFFF) remote TEID; when

we receive the remote answer, we get the source teid, with which the GTP-U endpoint is updadet.

doc/README.md File Reference

FEATURE_SET

Table of Contents

[[TOC]]

Functional Split Architecture

- RCC: Radio-Cloud Center
- RAU: Radio-Access Unit
- RRU: Remote Radio-Unit
- IF4.5 / IF5 : similar to IEEE P1914.1
- FAPI (IF2) : specified by Small Cell Forum (open-nFAPI implementation)
- IF1 : F1 in 3GPP Release 15

Functional Split Architecture

OpenAirInterface Block Diagram

Block Diagram

OpenAirInterface 5G-NR Feature Set

General Parameters

The following features are valid for the gNB and the 5G-NR UE.

- Static TDD
    - Multi TDD pattern supported refer TDD Configuration
- Static FDD
- Normal CP
- Subcarrier spacings: 15 and 30kHz (FR1), 120kHz (FR2)
- Bandwidths: 10, 20, 40, 60, 80, 100MHz
- Intermediate downlink and uplink frequencies to interface with IF equipment
- Procedures for 2-layer DL and UL MIMO
- Slot format: 14 OFDM symbols in UL or DL

- Highly efficient 3GPP compliant LDPC encoder and decoder (BG1 and BG2 supported)
- Highly efficient 3GPP compliant polar encoder and decoder
- Encoder and decoder for short blocks
- Support for UL transform precoding (SC-FDMA)

These modes of operation are supported:

- "phy-test" mode (gNB, nrUE):
  - gNB and nrUE have hardcoded RNTI and radio configuration
  - gNB schedules the nrUE all the time, even if no UE connected
  - can be used for performance evaluation
- "noS1" mode (DL and UL, gNB, nrUE):
  - Connection setup stops after RA; RRC configuration is exchanged through files
  - Creates TUN interface to PDCP to inject and receive user-place traffic
  - No connection to the core network
- Standalone (SA) mode (gNB, nrUE):
  - UE can register with the 5G Core Network through the gNB, establish a PDU Session and exchange user-plane traffic
  - Reestablishment supported
- Non-standalone (NSA) mode (gNB):
  - UE can use the gNB for user plane traffic while connected to the 4G eNB
  - is unstable (only one UE connection)

gNB PHY

- 15kHz and 30kHz SCS for FR1 and 120kHz SCS for FR2
- Generation of NR-PSS/NR-SSS
- NR-PBCH supports multiple SSBs and flexible periodicity
- Generation of NR-PDCCH (including generation of DCI, polar encoding, scrambling, modulation, RB mapping, etc)
  - common search space
  - user-specific search space
  - DCI formats: 00, 10, 01 and 11

- Generation of NR-PDSCH (including Segmentation, LDPC encoding, rate matching, scrambling, modulation, RB mapping, etc).
    - PDSCH mapping type A and B
    - DMRS configuration type 1 and 2
    - Single and multiple DMRS symbols
    - PTRS support
    - Support for up to 4 TX antennas
    - Support for up to 2 layers
    - Support for 256 QAM
- NR-CSIRS Generation of sequence at PHY
- NR-PUSCH (including Segmentation, LDPC encoding, rate matching, scrambling, modulation, RB mapping, etc).
    - PUSCH mapping type A and B
    - DMRS configuration type 1 and 2
    - Single and multiple DMRS symbols
    - PTRS support
    - Support for up to 4 RX antennas
    - Support for up to 2 layers
    - Support for 256 QAM
- NR-PUCCH
    - Format 0 (2 bits, for ACK/NACK and SR)
    - Format 2 (mainly for CSI feedback)
- NR-SRS
    - SRS signal reception
    - Channel estimation (with T tracer real time monitoring)
    - Power noise estimation
- NR-PRS
    - Rel16 Positioning reference signal(PRS) generation and modulation
    - Multiple PRS resources, one per beam is supported in FR2 TDD mode
    - FR1 and FR2 support with config file

- NR-PRACH
    - Formats 0,1,2,3, A1-A3, B1-B3
- Highly efficient 3GPP compliant LDPC encoder and decoder (BG1 and BG2 are supported)
- Highly efficient 3GPP compliant polar encoder and decoder
- Encoder and decoder for short block

gNB MAC

- MAC -> PHY configuration using NR FAPI P5 interface
- MAC <-> PHY data interface using FAPI P7 interface for BCH PDU, DCI PDU, PDSCH PDU
- Generation of and scheduler procedures for MIB/SIB1
- Scheduler procedures for RA
    - 4-Step RA
        - Contention Free RA procedure
        - Contention Based RA procedure
            - Msg3 can transfer uplink CCCH, DTCH or DCCH messages
            - CBRA can be performed using MAC CE for C-RNTI
            - Is not possible to use 2-Step RA and 4-Step RA at the same time
    - 2-Step RA
        - Contention Based RA procedure
            - MsgA can transfer uplink CCCH, DTCH or DCCH messages
            - CBRA can be performed using MAC CE for C-RNTI
            - Is not possible to use 2-Step RA and 4-Step RA at the same time
            - Fallback not supported
- Scheduler procedures for CSI-RS
- MAC downlink scheduler
    - phy-test scheduler (fixed allocation and usable also without UE)
    - regular scheduler with dynamic proportionally-fair allocation
    - MCS adaptation from HARQ BLER

- MAC header generation (including timing advance)
- ACK / NACK handling and HARQ procedures for downlink
- MAC uplink scheduler
    - phy-test scheduler (fixed allocation)
    - regular scheduler with dynamic proportionally-fair allocation
    - HARQ procedures for uplink
- Scheduler procedures for SRS reception
    - Periodic SRS reception
    - Channel rank computation up to 2x2 scenario
    - TPMI computation based on SRS up 4 antenna ports and 2 layers
- MAC procedures to handle CSI measurement report
    - evalution of RSRP report
    - evaluation of CQI report
- MAC scheduling of SR reception
- Intra-frequency handover
- Initial support for RedCap
- Scheduling of SIBs (2, 19)

gNB RLC

- Send/Receive operations according to 38.322 Rel.16
    - Segmentation and reassembly procedures
    - RLC Acknowledged mode supporting PDU retransmissions
    - RLC Unacknowledged mode
    - DRBs and SRBs establishment/handling and association with RLC entities
    - Timers implementation
    - Interfaces with PDCP, MAC
    - Interfaces with gtp-u (data Tx/Rx over F1-U at the DU)

gNB PDCP

- Send/Receive operations according to 38.323 Rel.16
    - Integrity protection and ciphering procedures

- Sequence number management, SDU dicard and in-order delivery
- Radio bearer establishment/handling and association with PDCP entities
- Interfaces with RRC, RLC
- Interfaces with gtp-u (data Tx/Rx over N3 and F1-U interfaces)

gNB SDAP

- Send/Receive operations according to 37.324 Rel.15
  - Establishment/Handling of SDAP entities.
  - Transfer of User Plane Data
  - Mapping between a QoS flow and a DRB for both DL and UL
  - Marking QoS flow ID in both DL and UL packets
  - Reflective QoS flow to DRB mapping for UL SDAP data PDUs

gNB RRC

- NR RRC (38.331) Rel 17 messages using new `asn1c`
- LTE RRC (36.331) also updated to Rel 15
- Generation of system information (SIB2)
- RRC can configure PDCP, RLC, MAC
- Interface with GTP-U (tunnel creation/handling for S1-U (NSA), N3 (SA), F1 interfaces)
- Integration of RRC messages and procedures supporting UE 5G SA connection
  - RRCSetupRequest/RRCSetup/RRCSetupComplete
  - RRC Uplink/Downlink Information transfer carrying NAS messages transparently
  - RRC Reconfiguration/Reconfiguration complete
  - RRC Reestablishment/Reestablishment complete
  - Support for MasterCellGroup configuration (from DU)
  - Interface with NGAP for the interactions with the AMF
  - Interface with F1AP for CU/DU split deployment option
  - Periodic RRC measurements of serving cell (no A/B events)
- Initial support for RedCap

gNB X2AP

- Integration of X2AP messages and procedures for the exchanges with the eNB over X2 interface supporting the NSA setup according to 36.423 Rel. 15
    - X2 setup with eNB
    - Handling of SgNB Addition Request/Addition Request Acknowledge/ Reconfiguration Complete

gNB NGAP

- Integration of NGAP messages and procedures for the exchanges with the AMF over N2 interface according to 38.413 Rel. 15
    - NGAP Setup request/response
    - NGAP Initial UE message
    - NGAP Initial context setup request/response
    - NGAP Downlink/Uplink NAS transfer
    - NGAP UE context release request/complete
    - NGAP UE radio capability info indication
    - NGAP PDU session resource setup request/response
- Interface with RRC

gNB F1AP

- Integration of F1AP messages and procedures for the control plane exchanges between the CU and DU entities according to 38.473 Rel. 16
    - F1 Interface Management:
        - F1 Setup request/response/failure
    - F1 RRC Message Transfer:
        - F1 Initial UL RRC Message Transfer
        - F1 DL RRC Message Transfer
        - F1 UL RRC Message Transfer
    - F1 UE Context Management:
        - F1 UE Context setup request/response
        - F1 UE Context modification request/response

        - F1 UE Context modification required
        - F1 UE Context release req/cmd/complete
    - F1 gNB CU configuration update
    - F1 Reset (handled at DU only, full reset only)
- Interface with RRC
- Interface with GTP-u (tunnel creation/handling for F1-U interface)
- One CU(-CP) can handle multiple DUs
- Support for intra-CU mobility (across DUs)

gNB E1AP

- Integration of E1AP messages and procedures for exchange between CU-CP and CU-UP according to TS 38.463 Rel. 16
    - E1 Setup (gNB-CU-UP initiated)
        - E1 Setup Request
        - E1 Setup Response
        - E1 Setup Failure
    - E1 Bearer Context Setup (gNB-CU-CP initiated)
        - E1 Bearer Context Setup Request
        - E1 Bearer Context Setup Response
    - Bearer Context Modification (gNB-CU-CP initiated)
        - E1 Bearer Context Modification Request
        - E1 Bearer Context Modification Response
- Interface with RRC and PDCP
- One CU-CP can handle multiple CU-UPs

gNB GTP-U

- New GTP-U implementation supporting both N3 and F1-U interfaces according to 29.281 Rel.15
    - Interfaces with RRC, F1AP for tunnel creation
    - Interfaces with PDCP and RLC for data send/receive at the CU and DU respectively (F1-U interface)

- Interface with SDAP for data send/receive, capture of GTP-U Optional Header, GTP-U Extension Header and PDU Session Container.

Number of supported UEs

- 16 by default (as defined in `MAX_MOBILES_PER_GNB`)
- up to 64 if the configured bandwidth is sufficient (at leat 40 MHz)

OpenAirInterface 5G-NR UE Feature Set

NR UE PHY Layer

- Initial synchronization
  - non-blind synchronization (information required: carrier frequency, bandwidth, numerology)
  - option to search SSB inside the bandwidth available
- Time tracking based on PBCH DMRS
- Frequency offset estimation based on PSS and SSS
- 15kHz and 30kHz SCS for FR1 and 120 kHz SCS for FR2
- Reception of NR-PSS/NR-SSS
- NR-PBCH supports multiple SSBs and flexible periodicity
  - RSRP measurement for the strongest SSB
- Reception of NR-PDCCH (including reception of DCI, polar decoding, de-scrambling, de-modulation, RB de-mapping, etc)
  - common search space configured by MIB
  - user-specific search space configured by RRC
  - DCI formats: 00, 10, 01 and 11
- Reception of NR-PDSCH (including Segmentation, LDPC decoding, rate de-matching, de-scrambling, de-modulation, RB de-mapping, etc).
  - PDSCH mapping type A and B
  - DMRS configuration type 1 and 2
  - Single and multiple DMRS symbols
  - PTRS support

    - Support for 256 QAM
    - Support for 1, 2 and 4 RX antennas
    - Support for up to 2 layers (currently limited to DMRS configuration type 2)
- Measurements based on NR-CSIRS
    - RSRP measurements
    - RI, PMI and CQI computation
    - Support for up to 4 RX antennas
    - Support for up to 2 layers
- NR-PUSCH (including Segmentation, LDPC encoding, rate matching, scrambling, modulation, RB mapping, etc).
    - PUSCH mapping type A and B
    - DMRS configuration type 1 and 2
    - Single and multiple DMRS symbols
    - PTRS support
    - Support for 256 QAM
    - Support for up to 2 TX antenna
    - Support for up to 2 layers
- NR-PUCCH
    - Format 0 (2 bits for ACK/NACK and SR)
    - Format 2 (mainly for CSI feedback)
    - Format 1 (limited testing)
    - Format 3 and 4 present but old code never tested (need restructuring before verification)
- NR-SRS
    - Generation of sequence at PHY
    - SRS signal transmission
- NR-PRS
    - PRS based Channel estimation with T tracer dumps
    - Time of arrival(ToA) estimation based on channel impulse response(CIR)
    - Finer ToA estimation by 16x oversampled IDFT for CIR

    - Support for multiple gNB reception with gNBs synced via GPSDO
- NR-PRACH
    - Formats 0,1,2,3, A1-A3, B1-B3
- Highly efficient 3GPP compliant LDPC encoder and decoder (BG1 and BG2 are supported)
- Highly efficient 3GPP compliant polar encoder and decoder
- Encoder and decoder for short block

NR UE FAPI

- MAC -> PHY configuration via UE FAPI P5 interface
- Basic MAC to control PHY via UE FAPI P7 interface
- PHY -> MAC indication

NR UE Higher Layers

UE MAC

- Minimum system information (MSI)
    - MIB processing
    - Scheduling of system information block 1 (SIB1) reception
- Random access procedure (needs improvement, there is still not a clear separation between MAC and PHY)
    - Mapping SSBs to multiple ROs
    - Scheduling of PRACH

4-Step RA

- Processing of RAR
- Transmission and re-transmission of Msg3
- Msg4 and contention resolution

2-Step RA

- Transmission of MsgA-PUSCH
- Reception of MsgB
- Processing of SuccessRAR

- Fallback not supported

DCI processing

- format 10 (RA-RNTI, C-RNTI, SI-RNTI, TC-RNTI)
- format 00 (C-RNTI, TC-RNTI)
- format 11 (C-RNTI)
- format 01 (C-RNTI)

UCI processing

- ACK/NACK processing
- Scheduling request procedures
- CSI measurement reporting (periodic and aperiodic)

DLSCH scheduler

- Configuration of fapi PDU according to DCI
- HARQ procedures

ULSCH scheduler

- Configuration of fapi PDU according to DCI
- Buffer status reporting procedures
- Logical channel prioritization of 'data from any logical channel'
- UCI on PUSCH

NR-CSIRS scheduler

- Scheduling of NR-CSIRS reception
- Fill UCI for CSI measurement reporting

Scheduler procedures for SRS transmission

- Periodic and aperiodic SRS transmission

Bandwidth part (BWP) operation

- Operation in configured dedicated BWP through RRCSetup or RRCReconfiguration

UE RLC

- Tx/Rx operations according to 38.322 Rel.16
    - Segmentation and reassembly procedures

    - RLC Acknowledged mode supporting PDU retransmissions
    - RLC Unacknowledged mode
    - DRBs and SRBs establishment and handling
    - Timers implementation
    - Interfaces with PDCP, MAC

UE PDCP

- Tx/Rx operations according to 38.323 Rel.16
    - Integrity protection and ciphering procedures
    - Sequence number management, SDU dicard and in-order delivery
    - Radio bearer establishment/handling and association with PDCP entities
    - Interfaces with RRC, RLC

UE SDAP

- Tx/Rx operations operations according to 37.324 Rel.15
    - Establishment/Handling of SDAP entities.
    - Transfer of User Plane Data
    - Reflective Mapping
    - RRC Signaling Mapping

UE RRC

- Integration of RRC messages and procedures supporting UE 5G SA connection according to 38.331 Rel.16
    - RRCSetupRequest/RRCSetup/RRCSetupComplete
    - RRC Uplink/Downlink Information transfer carrying NAS messages transparently
    - RRC Reconfiguration/Reconfiguration complete
    - RRCReestablishmentRequest/RRC Reestablishment/Reestablishment complete
    - Support for master cell group configuration
    - Reception of UECapabilityEnquiry, encoding and transmission of UECapability
- Interface with PDCP: configuration, DCCH and CCCH message handling
- Interface with RLC and MAC for configuration

UE NAS

- Transfer of NAS messages between the AMF and the UE supporting the UE registration with the core network and the PDU session establishment according to 24.501 Rel.16
    - Identity Request/Response
    - Authentication Request/Response
    - Security Mode Command/Complete
    - Registration Request/Accept/Complete
    - PDU Session Establishment Request/Accept
    - NAS configuration and basic interfacing with RRC

OpenAirInterface 4G LTE eNB Feature Set

eNB PHY Layer

The Physical layer implements **3GPP 36.211**, **36.212**, **36.213** and provides the following features:

- LTE release 8.6 compliant, and implements a subset of release 10
- FDD and TDD configurations: 1 (experimental) and 3
- Bandwidth: 5, 10, and 20 MHz
- Transmission modes: 1, 2 (stable), 3, 4, 5, 6, 7 (experimental)
- Max number of antennas: 2
- CQI/PMI reporting: aperiodic, feedback mode 3 - 0 and 3 - 1
- PRACH preamble format 0
- Downlink (DL) channels are supported: PSS, SSS, PBCH, PCFICH, PHICH, PDCCH, PDSCH, PMCH, MPDCCH
- Uplink (UL) channels are supported: PRACH, PUSCH, PUCCH (format 1/1a/1b), SRS, DRS
- HARQ support (UL and DL)
- Highly optimized base band processing (including turbo decoder)
- Multi-RRU support: over the air synchro b/ multi RRU in TDD mode
- Support for CE-modeA for LTE-M. Limited support for repeatition, single-LTE-M connection, legacy-LTE UE attach is disabled.

Performances

Transmission Mode, Bandwidth	Expected Throughput	Measured Throughput	Measurement Conditions
FDD DL: 5 MHz, 25 PRBS/ MCS 28	16 - 17 Mbit/s	TM1: 17.0 Mbits/s	COTS-UE Cat 4 (150/50 Mbps)
FDD DL: 10 MHz, 50 PRBS/ MCS 28	34 - 35 Mbit/s	TM1: 34.0 Mbits/s	COTS-UE Cat 4 (150/50 Mbps)
FDD DL: 20 MHz, 100 PRBS/ MCS 28	70 Mbit/s	TM1: 69.9 Mbits/s	COTS-UE Cat 4 (150/50 Mbps)

| | | FDD UL: 5 MHz, 25 PRBS/ MCS 20 | 9 Mbit/s | TM1: 8.28 Mbits/s | COTS-UE Cat 4 (150/50 Mbps) FDD UL: 10 MHz, 50 PRBS/ MCS 20 | 17 Mbit/s | TM1: 18.3 Mbits/s | COTS-UE Cat 4 (150/50 Mbps) FDD UL: 20 MHz, 100 PRBS/ MCS 20 | 35 Mbit/s | TM1: 18.6 Mbits/s | COTS-UE Cat 4 (150/50 Mbps) | | TDD DL: 5 MHz, 25 PRBS/ MCS **XX** | 6.5 Mbit/s | TM1: 6.71 Mbits/s | COTS-UE Cat 4 (150/50 Mbps) TDD DL: 10 MHz, 50 PRBS/ MCS **XX** | 13.5 Mbit/s | TM1: 13.6 Mbits/s | COTS-UE Cat 4 (150/50 Mbps) TDD DL: 20 MHz, 100 PRBS/ MCS **XX** | 28.0 Mbit/s | TM1: 27.2 Mbits/s | COTS-UE Cat 4 (150/50 Mbps) | | | TDD UL: 5 MHz, 25 PRBS/ MCS **XX** | 2.0 Mbit/s | TM1: 3.31 Mbits/s | COTS-UE Cat 4 (150/50 Mbps) TDD UL: 10 MHz, 50 PRBS/ MCS **XX** | 2.0 Mbit/s | TM1: 7.25 Mbits/s | COTS-UE Cat 4 (150/50 Mbps) TDD UL: 20 MHz, 100 PRBS/ MCS **XX** | 3.0 Mbit/s | TM1: 4.21 Mbits/s | COTS-UE Cat 4 (150/50 Mbps)

Number of supported UEs

- 16 by default
- up to 256 when compiling with dedicated compile flag
- was tested with 40 COTS-UE

eNB MAC Layer

The MAC layer implements a subset of the **3GPP 36.321** release v8.6 in support of BCH, DLSCH, RACH, and ULSCH channels.

- RRC interface for CCCH, DCCH, and DTCH
- Proportional fair scheduler (round robin scheduler soon), with the following improvements:
    - Up to 30 users tested in the L2 simulator, CCE allocation in the preprocessor ; the scheduler was also simplified and made more modular
    - Adaptative UL-HARQ
    - Remove out-of-sync UEs

    - No use of the `first_rb` in the UL scheduler ; respects `vrb_map_UL` and `vrb_map` in the DL

- DCI generation
- HARQ Support
- RA procedures and RNTI management
- RLC interface (AM, UM)
- UL power control
- Link adaptation
- Connected DRX (CDRX) support for FDD LTE UE. Compatible with R13 from 3GPP. Support for Cat-M1 UE comming soon.

eNB RLC Layer

The RLC layer implements a full specification of the 3GPP 36.322 release v9.3.

- RLC TM (mainly used for BCCH and CCCH)
    - Neither segment nor concatenate RLC SDUs
    - Do not include a RLC header in the RLC PDU
    - Delivery of received RLC PDUs to upper layers
- RLC UM (mainly used for DTCH)
    - Segment or concatenate RLC SDUs according to the TB size selected by MAC
    - Include a RLC header in the RLC PDU
    - Duplication detection
    - PDU reordering and reassembly
- RLC AM, compatible with 9.3
    - Segmentation, re-segmentation, concatenation, and reassembly
    - Padding
    - Data transfer to the user
    - RLC PDU retransmission in support of error control and correction
    - Generation of data/control PDUs

eNB PDCP Layer

The current PDCP layer is header compliant with **3GPP 36.323** Rel 10.1.0 and implements the following functions:

- User and control data transfer
- Sequence number management
- RB association with PDCP entity
- PDCP entity association with one or two RLC entities
- Integrity check and encryption using the AES and Snow3G algorithms

eNB RRC Layer

The RRC layer is based on **3GPP 36.331** v15.6 and implements the following functions:

- System Information broadcast (SIB 1, 2, 3, and 13)
    - SIB1: Up to 6 PLMN IDs broadcast
- RRC connection establishment
- RRC connection reconfiguration (addition and removal of radio bearers, connection release)
- RRC connection release
- RRC connection re-establishment
- Inter-frequency measurement collection and reporting (experimental)
- eMBMS for multicast and broadcast (experimental)
- Handover (experimental)
- Paging (soon)
- RRC inactivity timer (release of UE after a period of data inactivity)

eNB X2AP

The X2AP layer is based on **3GPP 36.423** v14.6.0 and implements the following functions:

- X2 Setup Request
- X2 Setup Response
- X2 Setup Failure
- Handover Request
- Handover Request Acknowledge

- UE Context Release
- X2 timers (t_reloc_prep, tx2_reloc_overall)
- Handover Cancel
- X2-U interface implemented
- EN-DC is implemented
- X2AP : Handling of SgNB Addition Request / Addition Request Acknowledge / Reconfiguration Complete
- RRC : Handling of RRC Connection Reconfiguration with 5G cell info, configuration of 5G-NR measurements
- S1AP : Handling of E-RAB Modification Indication / Confirmation

eNB/MCE M2AP

The M2AP layer is based on **3GPP 36.443** v14.0.1:

- M2 Setup Request
- M2 Setup Response
- M2 Setup Failure
- M2 Scheduling Information
- M2 Scheduling Information Response
- M2 Session Start Request
- M2 Session Start Response

MCE/MME M3AP

The M3AP layer is based on **3GPP 36.444** v14.0.1:

- M3 Setup Request
- M3 Setup Response
- M3 Setup Failure
- M3 Session Start Request
- M3 Session Start Response

OpenAirInterface 4G LTE UE Feature Set

LTE UE PHY Layer

The Physical layer implements **3GPP 36.211**, **36.212**, **36.213** and provides the following features:

- LTE release 8.6 compliant, and implements a subset of release 10
- FDD and TDD configurations: 1 (experimental) and 3
- Bandwidth: 5, 10, and 20 MHz
- Transmission modes: 1, 2 (stable)
- Max number of antennas: 2
- CQI/PMI reporting: aperiodic, feedback mode 3 - 0 and 3 - 1
- PRACH preamble format 0
- All downlink (DL) channels are supported: PSS, SSS, PBCH, PCFICH, PHICH, PDCCH, PDSCH, PMCH
- All uplink (UL) channels are supported: PRACH, PUSCH, PUCCH (format 1/1a/1b), SRS, DRS
- LTE MBMS-dedicated cell (feMBMS) procedures subset for LTE release 14 (experimental)
- LTE non-MBSFN subframe (feMBMS) Carrier Adquistion Subframe-CAS procedures (PSS/SSS/PBCH/PDSH) (experimental)
- LTE MBSFN MBSFN subframe channel (feMBMS): PMCH (CS@1.25KHz) (channel estimation for 25MHz bandwidth) (experimental)

LTE UE MAC Layer

The MAC layer implements a subset of the **3GPP 36.321** release v8.6 in support of BCH, DLSCH, RACH, and ULSCH channels.

- RRC interface for CCCH, DCCH, and DTCH
- HARQ Support
- RA procedures and RNTI management
- RLC interface (AM, UM)
- UL power control
- Link adaptation
- MBMS-dedicated cell (feMBMS) RRC interface for BCCH

- eMBMS and MBMS-dedicated cell (feMBMS) RRC interface for MCCH, MTCH

LTE UE RLC Layer

The RLC layer implements a full specification of the 3GPP 36.322 release v9.3.

LTE UE PDCP Layer

The current PDCP layer is header compliant with **3GPP 36.323** Rel 10.1.0.

LTE UE RRC Layer

The RRC layer is based on **3GPP 36.331** v14.3.0 and implements the following functions:

- System Information decoding
- RRC connection establishment
- MBMS-dedicated cell (feMBMS) SI-MBMS/SIB1-MBMS management

LTE UE NAS Layer

The NAS layer is based on **3GPP 24.301** and implements the following functions:

- EMM attach/detach, authentication, tracking area update, and more
- ESM default/dedicated bearer, PDN connectivity, and more

OAI wiki home

OAI softmodem build procedure

Running the OAI softmodem

Open Network Function Application Platform Interface (Open-nFAPI)

Introduction

Open-nFAPI is implementation of the Small Cell Forum's network femto API or nFAPI for short. nFAPI defines a network protocol that is used to connect a Physical Network Function (PNF) running LTE Layer 1 to a Virtual Network Function (VNF) running LTE Layer 2 and above. The specification can be found at http://scf.io/documents/082.

The aim of Open-nFAPI is to provide an open interface between LTE Layer 1 and Layer 2 to allow for interoperability between the PNF and VNF & also to facilitate the sharing of PNF's between different VNF's

Open-nFAPI implements the P4, P5 and P7 interfaces as defined by the nFAPI specification.

- The P5 interface allows the VNF to query and configure the 'resources' of the PNF; slicing it into 1 or more phy instances.
- The P7 interface is used to send the subframe information between the PNF and VNF for 1 or more phy instances
- The P4 interface allows the VNF to request the PNF phy instances to perform measurements of the surrounding network

The remaining interfaces are currently outside of the scope of this project.

The best place to start is by reviewing the detailed nFAPI call flow which can be **here**

Contibuting

The Small Cell Forum cordially requests that any derivative work that looks to extend the nFAPI libraries use the specified vendor extension techniques, so ensuring the widest interoperability of the baseline nFAPI specification in those derivative works.

Installation

Step 1: Opening the box

etc...

Directory Structure

```
nfapi
|- common                    Common functions for the nfapi project
|  |- src
|  |- public-inc
|- nfapi                     The NFAPI lib
|  |- inc
|  |- src
|  |- public-inc             Public interface for the nfapi library
|  |- tests                  Unit test for the nfapi lib
|- pnf                       The PNF lib
|  |- inc
|  |- src
|  |- public-inc             Public interface for the pnf library
|  |- tests                  Unit test for the pnf lib
|- vnf                       The VNF lib
|  |- inc
|  |- src
|  |- public-inc             Public interface for the vnf library
|  |- tests                  Unit test for the vnf lib
|- sim_common                Common functions for the nfapi simulators
|  |- inc
|  |- src
|- pnf_sim                   The PNF simulator
|  |- inc
|  |- src
|- vnf_sim                   The VNF simulator
|  |- inc
|  |- src
|- integration_tests         Integration tests that run both VNF & PNF simulators
|  |- inc
|  |- src
|- docs                      Documentation
|- xml                       Xml configuration for the simulator
```

Building

To build the nfapi project and run the unit test you will need to

```
autoreconf -i
./configure
make
make check
```

The following dependencies will be required

- Boost. Need to build the simulators
- STCP. Need to run the simulators
-

Running the simulators

Once you have build the nfapi project you can run the PNF/VNF simulators on either the same of seperate linux machines.

To run the VNF from the vnf_sim directory.

```
    ./vnfsim 4242 ../xml/vnf_A.xml
[MAC] Rx Data from 8891
[MAC] Tx Data to 10.231.16.80.8892
Calling nfapi_vnf_start
2035.854438: 0x04:  773068608: nfapi_vnf_start()
2035.854450: 0x04:  773068608: Starting P5 VNF connection on port 4242
2035.854472: 0x04:  773068608: P5 socket created... 3
2035.854478: 0x03:  773068608: VNF Setting the SCTP_INITMSG
2035.854481: 0x04:  773068608: IPV4 binding to port 4242
2035.854485: 0x04:  773068608: bind succeeded..3.
2035.854497: 0x04:  773068608: listen succeeded...
```

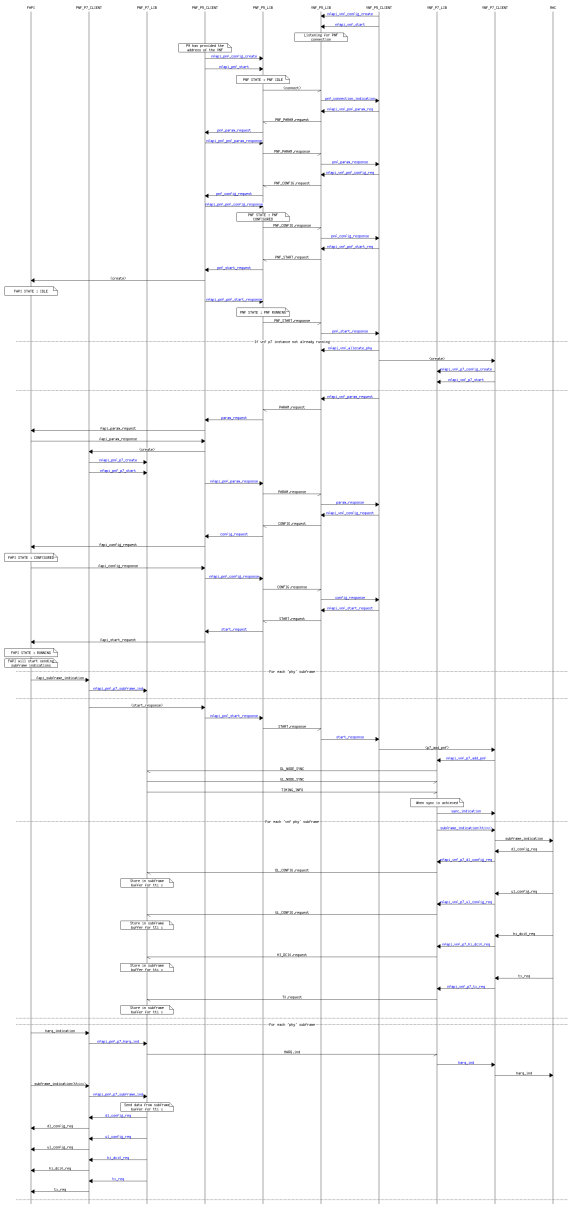
To run the PNF from the pnf_sim directory

```
    ./pnfsim 127.0.01 4242 ../xml/pnf_phy_1_A.xml
nfapi_pnf_start
Starting P5 PNF connection to VNF at 127.0.0.1:4242
Host address info  0 Family:IPV4 Address:127.0.0.1
PNF Setting the SCTP_INITMSG
P5 socket created...
Socket CONNECTED
PNF_PARAM.request received
[PNF_SIM] pnf param request
PNF_CONFIG.request received
[PNF_SIM] pnf config request
.... and so on
```

You can Ctrl-C to exit the simulators.

Sequence diagram

The follow sequence digram show how the nFAPI api will be used to bring up the PNF and then the PNF PHY and initiate subframe message exchange The names of the CLIENT callbacks are placeholders for the functions that the CLIENT will provide.



BCA

1. The VNF_P5_CLIENT is created. If it left to the VRAN partner to define how that is done. The client is responsiable to creation of the thread with the correct priorities within the wider system.
2. The VNF_P5_CLIENT creates and initializes the VNF_P5_LIB (nfapi_vnf_config_create & nfapi_vnf_start) providing the address that it should listen on for incoming SCTP connections. It is expected that this would be provided by some VNF management system. The VNF_P5_CLIENT also provides information on the callbacks that the VNF_P5_LIB will use to inform the VNF_P5_CLIENT of the received messages

```
nfapi_vnf_config_t* vnf_config = nfapi_vnf_config_create();
vnf_config->pnf_connection_indication = &pnf_connection_indication;
vnf_config->pnf_param_resp = &pnf_param_resp;
nfapi_vnf_start(vnf_config);
```

3. It is FFS how the address information of the VNF P5 is passed to the PNF over P9, but assuming that has been done
4. The PNF_P5_CLIENT is created and then creates the PNF_P5_LIB (nfapi_pnf_init & nfapi_pnf_start) passing the address of the VNF to connect to. The PNF_P5_CLIENT also provides information on the callbacks that the PNF_P5_LIB will use to inform the PNF_P5_CLIENT of the received messages

```
nfapi_pnf_config_t* pnf_config = nfapi_pnf_config_create();
pnf_config->pnf_param_req = &pnf_param_req;
nfapi_pnf_start(pnf_config);
```

5. The PNF_P5_LIB attempts to establish connection to the VNF_P5 endpoint
6. The VNF_P5_LIB receives the STCP connection request and indicates this to the VNF_P5_CLIENT (pnf_connection_indication). The VNF_P5_CLIENT can decide to accept or reject the connection.

```
int pnf_connection_indication(nfapi_vnf_config_t* config, int p5_idx) {
  // send the PNF_PARAM.request
  nfapi_pnf_param_request_t req;
  memset(&req, 0, sizeof(req));
  req.header.message_id = NFAPI_PNF_PARAM_REQUEST;
  nfapi_vnf_pnf_param_req(config, p5_idx, req);
  return 0;
}
```

7. The VNF and PNF then proceed to exchange the NFAPI PNF messages (PNF_PARAM, PNF_CONFIG and PNF_START). At each stage the LIB's invoke callbacks on the CLIENT's for them to handle message the return the appropriate response.
8. Finally the VNF will send the PNF_START.req which will invoke the PNF_P5_CLIENT nfapi_pnf_start_request callback. This is the point at which it is expect that the PNF_CLIENT will create the FAPI module. The PNF_P5_CLIENT will need to perform

the translation between NFAPI structures and vendor specific FAPI structures for messages sent between the PNF_P5_LIB and the FAPI interface.

9. The PNF_START.response will sent back to the VNF and at this point the VNF_P5_CLIENT will need to decide which VNF_P7 instance will handle the P7 connection. i.e. it needs to determine the IP address and port number of the P7 VNF instance.
10. A new VNF_P7_CLIENT & VNF_P7_LIB should be configured and started if necessary.
11. The VNF_P5 will then send the PARAM.request to the PNF_P5 which will forward it onto the FAPI interface. This PARAM.request includes the VNF_P7 address. The PNF_P7_CLIENT must decide on the PNF P7 address. This information is used to create and initialize the PNF_P7_LIB. The PNF_P7 address is then returned by the PNF_P5 to the VNF_P5 in the PARAM.response.
12. The VNF_P5 will then exchange with the PNF_P5 the CONFIG.request/response which will be used to configure the PNF FAPI instance.
13. The VNF_P5 will then decide to start the PNF_P7 by sending the START.request.
14. The PNF_P5_CLIENT need to send the start request to the FAPI instance. The response which is the start of subframe indications from the FAPI. In receipt of the first subframe indication the PNF_P5_CLIENT will send the START.response to the VNF_P5 via the PNF_P5_LIB
15. The FAPI subframe indications should be forwarded to the PNF_P7_LIB by the PNF_P7_CLIENT. Until the VNF_P7 instance has sent the subframe configuration messages (dl_config, ul_config, etc) the PNF_P7_LIB will send ‘dummy’ subframe configuration messages. The contents of which are configurable by the PNF_P7_CLIENT.
16. When the VNF_P5_CLIENT receives the START.response it will need to ‘communicate’ with the VNF_P7_CLIENT to inform it that the PNF_P7 instance has started. The VNF_P7_CLIENT will call the nfapi_vnf_P7_add_pnf function passing the address details of the PNF_P7 instance.
17. The VNF_P7_LIB will then start the sync procedure to establish sub frame synchronization between the PNF_P7 and VNF_P7 instances. This involves sending the DL_NODE_SYNC and UL_NODE_SYNC to determine network latency and PNF processing latency to be able to request MAC generate sub frames in advance on when they are required by the FAPI interface.
18. When sync is achieved the VNF_P7_LIB will send the nfapi_sync_indication to the VNF_P7_CLIENT.
19. The VNF_P7_LIB will then start issuing sub frame indications to the VNF_P7_CLIENT. The logical intent is that they indications are sent every millisecond. However due to the scheduling jitter that may be seen by the VNF these subframe indications may be less than or more than 1ms apart. How this is handled is one of the critical

functions of the VNF_P7_LIB and may require specialization or requirements on the VNF environment i.e. CPU pinning.

20. The VNF_P7_LIB will send the subframe_indication to the VNF_P7_CLIENT for ‘x’ subframe’s inadvance of the current TTI at the PNF. The delta ‘x’ will be determined by the sync procedure and how far in advance the FAPI needs to receive messages before RF transmission.
21. The VNF_P7_CLIENT is then responsible for communicating to the MAC layer to prepare the dl_config_request, ul_config_request, hi_dci0_request, tx_req in a timely manner and sending them to the VNF_P7_LIB
22. The VNF_P7_LIB will send them to the VNF_P7_LIB.
23. The PNF_P7_LIB will store these messages in a subframe buffer or playout buffer in advance of when they are required by FAPI.
    a. The PNF_P7_LIB will monitor to see if these messages arrive too late and if they do trigger a TIMING_INFO response to the VNF_P7_LIB to reassess if the sync is still valid.
24. As some time in the future the FAPI will send a subframe indication for the TTI that the VNF_P7_LIB had previously requested.
25. The PNF_P7_LIB will then use the messages in the subframe buffer and send them too the FAPI interface for transmission. The PNF_P7_CLIENT will need to perform the translation between NFAPI structure and vendor specific FAPI structures
26. This subframe exchange will continue and allow high layer MAC and RRC function to bringup the cell and connect UE’s