srsRAN Documentation

Release 21.04

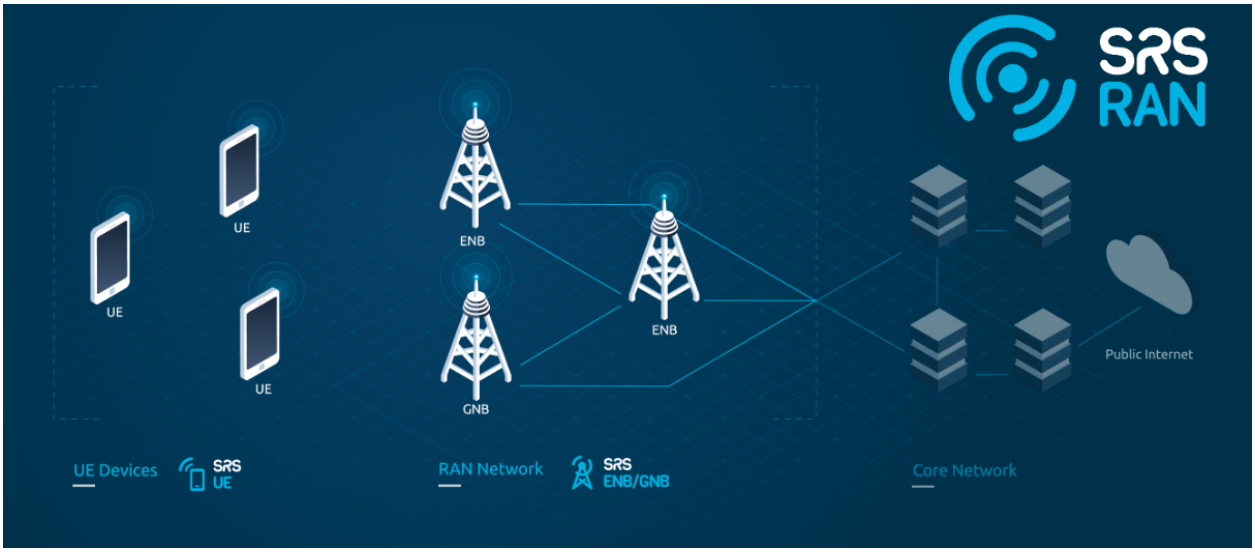
['Software Radio Systems']

Jul 17, 2023

CONTENTS

srsRAN is a free and open-source 4G and 5G software radio suite.

Featuring both UE and eNodeB/gNodeB applications, srsRAN can be used with third-party core network solutions to build complete end-to-end mobile wireless networks. For more information, see www.srsran.com.



The srsRAN suite currently includes:

- srsUE - a full-stack 4G and 5G NSA UE (User Equipment) application **(5G SA coming soon)**
- srsENB - a full-stack 4G eNodeB (Basestation) application **(5G NSA and SA coming soon)**
- srsEPC - a light-weight 4G EPC (Core Network) implementation with MME, HSS and S/P-GW

All srsRAN software runs in linux with off-the-shelf compute and radio hardware.

CHAPTER

ONE

FIRST STEPS

Get srsRAN installed on your computer:

- *Installation*

Get a network up and running:

- *End-to-end Network Setup*

Read the user manuals:

- *UE User Manual*
- *eNodeB User Manual*
- *EPC User Manual*

Read the srsRAN Application Notes:

- *Application Notes*

Take a look at the source code:

- srsRAN on GitHub

Learn about the team behind srsRAN:

- Software Radio Systems

CHAPTER

TWO

LINKS

2.1 General

2.1.1 Installation Guide

Which Installation Should I Use?

srsRAN can be installed from packages or from source. The following decision tree should help users decide which is best for them:

In short, users looking for a simple installation who only expect to run basic srsRAN applications with USRP front-ends should use the package installation. Users who wish to modify srsRAN and/or use alternative RF front-ends such as limeSDR and BladeRF should install from source.

Package Installation

The srsRAN software suite can be installed using packages on Ubuntu:

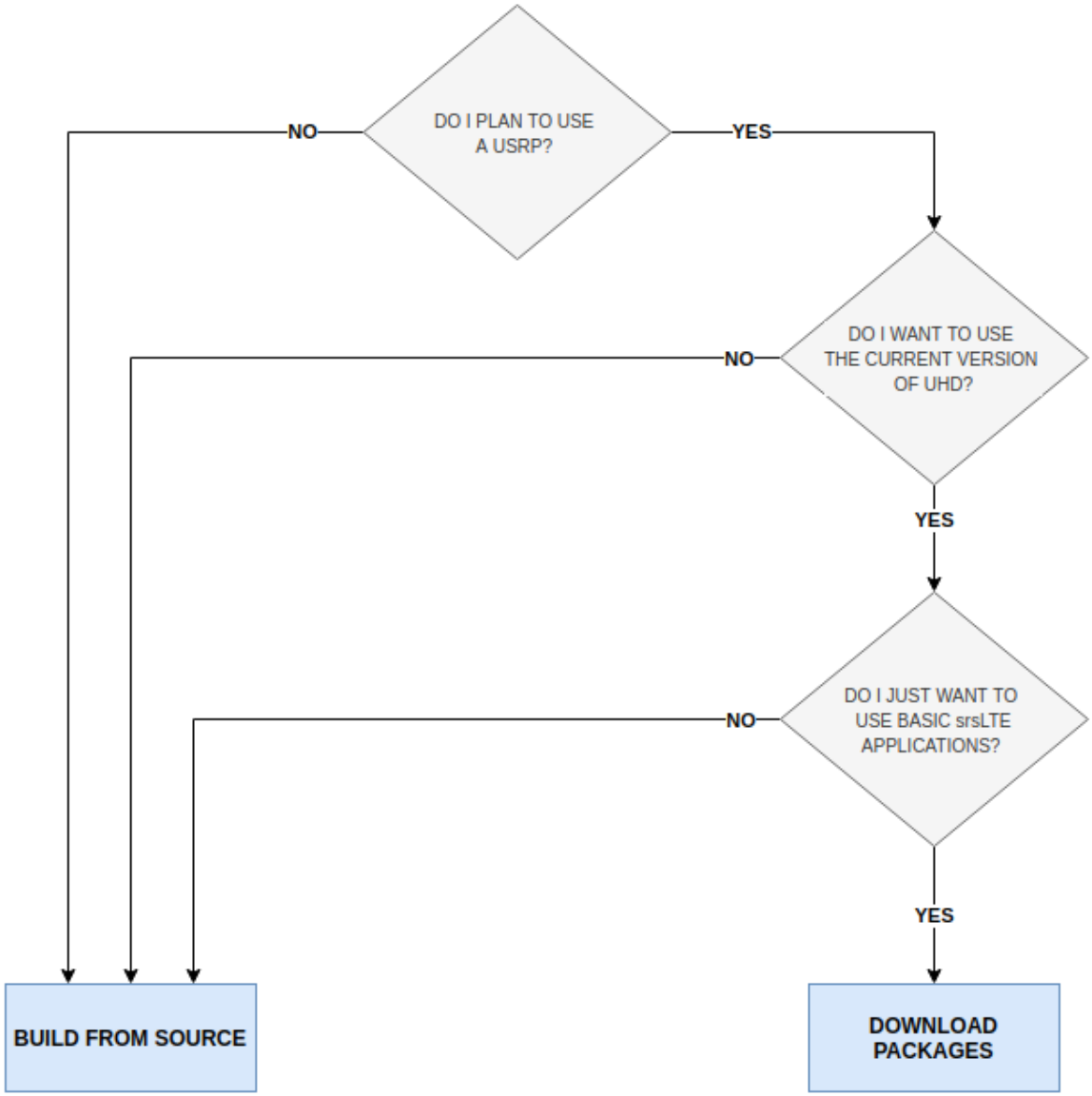
```
sudo add-apt-repository ppa:softwareradiosystems/srsran
sudo apt-get update
sudo apt-get install srsran -y
```

Package installs are also available for other distributions.

- openSUSE
- Arch Linux
- Debian (Pop OS, Mint, etc)

Note, only the Launchpad packages for Ubuntu are maintained by SRS. Different distributions will maintain their own packages for srsRAN, which may or may not be up to date. Check the available version before installing to ensure you are using the desired version of srsRAN.

Fedora does not yet have support for a package installation of srsRAN.


DO I PLAN TO USE A USRP?
NO
YES
DO I WANT TO USE THE CURRENT VERSION OF UHD?
NO
YES
DO I JUST WANT TO USE BASIC srsLTE APPLICATIONS?
NO
YES
BUILD FROM SOURCE
DOWNLOAD PACKAGES

Installation from Source

- Mandatory requirements:
  - Common:
    - cmake https://cmake.org/
    - libfftw http://www.fftw.org/
    - PolarSSL/mbedTLS https://tls.mbed.org
  - srsUE:
    - Boost: http://www.boost.org
  - srsENB:
    - Boost: http://www.boost.org
    - lksctp: http://lksctp.sourceforge.net/
    - config: http://www.hyperrealm.com/libconfig/
  - srsEPC:
    - Boost: http://www.boost.org
    - lksctp: http://lksctp.sourceforge.net/
    - config: http://www.hyperrealm.com/libconfig/

For example, on Ubuntu, one can install the required libraries with:

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

or on Fedora:

```
dnf install cmake fftw3-devel mbedtls-devel lksctp-tools-devel libconfig-devel boost-
↪devel
```

For CentOS, use the Fedora packages but replace *libconfig-devel* with just *libconfig*.

Note that depending on your flavor and version of Linux, the actual package names may be different.

- Optional requirements:
  - srsgui: https://github.com/srsran/srsgui - for real-time plotting.
  - libpcsclite-dev: https://pcsclite.apdu.fr/ - for accessing smart card readers
  - libdw-dev libdw - for truly informative backtraces using backward-cpp
- RF front-end driver:
  - UHD: https://github.com/EttusResearch/uhd
  - SoapySDR: https://github.com/pothosware/SoapySDR
  - BladeRF: https://github.com/Nuand/bladeRF
  - ZeroMQ: https://github.com/zeromq

Note: If using UHD we recommended the LTS version of UHD, i.e. either 3.9.7 or 3.15.

Download and build srsRAN:

```
git clone https://github.com/srsRAN/srsRAN.git
cd srsRAN
mkdir build
cd build
cmake ../
make
make test
```

Install srsRAN:

```
sudo make install
sudo srsran_install_configs.sh user
```

This installs srsRAN and also copies the default srsRAN config files to the user's home directory (~/.srs).

2.1.2 Release Roadmap

srsRAN follows a bi-annual release schedule, with new releases published in April and October each year. Each new release brings a range of new features, bugfixes and performance improvements. Minor intermediate releases to address priority bugfixes occur on an ad-hoc basis between releases.

Releases are labelled as (Year).(Month).(Minor), for example 20.10.1.

2021 Roadmap

This year will see the addition of 5G NR support to the SRS software suite. Initial releases will focus on Non-Standalone (NSA) mode support, building on top of our existing 4G UE and eNB applications. In addition to new 5G NR features, releases will include ongoing updates and improvements across all existing 4G applications.

21.04

21.04 will see the release of the first **Open-Source 5G NSA UE**. Building on top of the existing 4G UE application, key features of the 5G NSA UE will include:

- 5G-NR PHY layer for x86 including SIMD-optimized LDPC encoder/decoder
- Compatible & tested with 3rd-party RAN/Core solutions
- Data traffic over Secondary Cell Group (SCG) bearer for NR

21.10

21.10 will see the release of our 5G NSA eNB/gNB application. Further details available from Q3 2021.

2.1.3 Release Notes

- 21.04
  - Rename project from srsLTE to srsRAN
  - Add initial 5G NSA support to srsUE (including x86-optimized FEC and PHY layer)
  - Add PDCP discard support
  - Add UL power control, measurement gaps and a new proportional fair scheduler to srsENB
  - Extend GTP-U tunneling to support tunnel forwarding over S1
  - Optimize many data structures, remove dynamic memory allocations in data plane
  - Improved S1AP error handling and enhanced event reporting
  - Update ASN.1 packing/unpacking, RRC to Rel 15.11, S1AP to Rel 16.1
  - Update PCAP writer to use UDP framing
  - Other bug-fixes and improved stability and performance in all parts
- 20.10.1
  - Fix eNB issue relating to uplink hybrid ARQ
- 20.10
  - EUTRA mobility support
  - Fix for QAM256 support for eNB
  - New logging framework
  - PHY optimizations
  - Other performance and stability improvements
- 20.04.1
  - Fix for UE MIMO segfault issue
  - Fix for eNodeB SR configuration
  - Clang compilation warning fixes
  - Fix GPS tracking synchronization
- 20.04
  - Carrier Aggregation and Time Alignment in srsENB
  - Complete Sidelink PHY layer (all transmission modes)
  - Complete NB-IoT PHY downlink signals
  - New S1AP packing/unpacking library
  - EVM and EPRE measurements
  - Remove system timers in srsUE and srsENB
  - Refactor eNB to prepare for mobility support
  - Other bug-fixes and improved stability and performance in all parts
- 19.12
  - Add 5G NR RRC and NGAP ASN1 packing/unpacking

  - Add sync routines and broadcast channel for Sidelink
  - Add cell search and MIB decoder for NB-IoT
  - Add PDCP discard
  - Improve RRC Reestablishment handling
  - Improve RRC cell measurements and procedure handling
  - Add multi-carrier and MIMO support to ZMQ radio
  - Refactor eNB scheduler to support multiple carriers
  - Apply clang-format style on entire code base
  - Other bug-fixes and improved stability and performance in all parts
- 19.09
  - Add initial support for NR in MAC/RLC/PDCP
  - Add sync code for NB-IoT
  - Add support for EIA3/EEA3 (i.e. ZUC)
  - Add support for CSFB in srsENB
  - Add adaptation layer to run TTCN-3 conformance tests for srsUE
  - Add High Speed Train model to channel simulator
  - Rework RRC and NAS layer and make them non-blocking
  - Fixes in ZMQ, bladeRF and Soapy RF modules
  - Other bug-fixes and improved stability and performance in all parts
- 19.06
  - Add QAM256 support in srsUE
  - Add QoS support in srsUE
  - Add UL channel emulator
  - Refactor UE and eNB architecture
  - Many bug-fixes and improved stability and performance in all parts
- 19.03
  - PHY library refactor
  - TDD support for srsUE
  - Carrier Aggregation support for srsUE
  - Paging support for srsENB and srsEPC
  - User-plane encryption for srsENB
  - Channel simulator for EPA, EVA, and ETU 3GPP channels
  - ZeroMQ-based fake RF driver for I/Q over IPC/network
  - Many bug-fixes and improved stability and performance in all parts
- 18.12
  - Add new RRC ASN1 message pack/unpack library

  - Refactor EPC and add encryption support
  - Add IPv6 support to srsUE
  - Fixed compilation issue for ARM and AVX512
  - Add clang-format file
  - Many bug-fixes and improved stability and performance in all parts
- 18.09
  - Improved Turbo Decoder performance
  - Configurable SGi interface name and M1U params
  - Support for GPTU echo mechanism
  - Added UE detach capability
  - Refactor RLC/PDCP classes
  - Various fixes for ARM-based devices
  - Added support for bladeRF 2.0 micro
  - Many bug-fixes and improved stability and performance in all parts
- 18.06.1
  - Fixed RLC reestablish
  - Fixed aperiodic QCI retx
  - Fixed eNB instability
  - Fixed Debian packaging
- 18.06
  - Added eMBMS support in srsUE/srsENB/srsEPC
  - Added support for hard SIM cards
  - Many bug-fixes and improved stability and performance in all parts
- 18.03.1
  - Fixed compilation for NEON
  - Fixed logging and RLC AM issue
- 18.03
  - Many bug-fixes and improved stability and performance in all parts
- 17.12
  - Added support for MIMO 2x2 in srsENB (i.e. TM3/TM4)
  - Added srsEPC, a light-weight core network implementation
  - Added support for X2/S1 handover in srsUE
  - Added support for user-plane encryption in srsUE
  - Many bug-fixes and improved stability and performance in srsUE/srsENB
- 17.09
  - Added MIMO 2x2 in the PHY layer and srsUE (i.e. TM3/TM4)

  - eMBMS support in the PHY layer
  - Many bug-fixes and improved stability and performance in srsUE/srsENB
- 002.000.000
  - Added fully functional srsENB to srsRAN code
  - Merged srsUE code into srsRAN and reestructured PHY code
  - Added support for SoapySDR devices (eg LimeSDR)
  - Fixed issues in RLC AM
  - Added support for NEON and AVX in many kernels and Viterbi decoder
  - Added support for CPU affinity
  - Other minor bug-fixes and new features
- 001.004.000
  - Fixed issue in rv for format1C causing incorrect SIB1 decoding in some networks
  - Improved PDCCH decoding BER (fixed incorrect trellis initialization)
  - Improved PUCCH RX performance
- 001.003.000
  - Bugfixes:
    - x300 master clock rate
    - PHICH: fixed bug causing more NACKs
    - PBCH: fixed bug in encoding function
    - channel estimation: fixed issue in time interpolation
    - DCI: Fixed bug in Format1A packing
    - DCI: Fixed bug in Format1C for RA-RNTI
    - DCI: Fixed overflow in MIMO formats
  - Improvements:
    - Changed and cleaned DCI blind search API
    - Added eNodeB PHY processing functions
- 001.002.000
  - Bugfixes:
    - Estimation of extrapolated of out-of-band carriers
    - PDCCH REG interleaving for certain cell IDs
    - MIB decoding
    - Overflow in viterbi in PBCH
  - Improvements:
    - Synchronization in long multipath channels
    - Better calibration of synchronization and estimation
    - Averaging in channel estimation

        * Improved 2-port diversity decoding

- 001.001.000
  - Added support for BladeRF

2.1.4 Contributions

Contributions to the srsRAN software suite are always welcome. The easiest way to contribute is by issuing a pull request on the srsRAN repository. We use clang-format to maintain consistent code style - see our default format.

We ask srsRAN contributors to agree to a Copyright License Agreement. This provides us with necessary permissions to use contributed code. For more information, see the FAQ below. Viewing and accepting the CLA agreement is integrated into the GitHub pull request workflow using CLAassistant.

FAQ

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

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

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

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

2. How do I complete and submit the CLA?

The srsRAN CLAs can be found here. Download the appropriate CLA, then print, sign and scan the document before sending by email to licensing@softwareradiosystems.com.

3. How will my contributions to srsRAN be used?

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

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

If you have any questions about srsRAN licensing and contributions, please contact us at licensing@softwareradiosystems.com

2.1.5 Troubleshooting

Building with Debug Symbols

First make sure srsRAN has been downloaded, and you have created and navigated to the build folder:

```
git clone https://github.com/srsran/srsran.git
cd srsran
mkdir build
cd build
```

To build srsRAN with debug symbols, the following steps can be taken. If srsRAN has already been built, the original build folder should be cleared before proceeding. This can be done with the following command:

```
rm -rf *
make clean
```

The following command can then be used to build srsRAN with debug symbols enabled:

```
cmake -DCMAKE_BUILD_TYPE=RelWithDebInfo ../
make
make test
```

The log file containing the debug info can be found in the `srsran_backtrace.log` file.

Examining PCAPs with Wireshark

The srsRAN applications support packet capture at the MAC and NAS layers of the network stack.

Packet capture files (pcaps) can be viewed using Wireshark (www.wireshark.org). pcaps are encoded in compact MAC-LTE and MAC-NR form. They can be found in the */tmp* folder where other logs are located. To view in wireshark, edit the preferences of the DLT_USER dissector.

To decode MAC pcaps add an entry with the following:

- DLT=149
- Payload Protocol=udp

Further, enable the heuristic dissection in UDP under: *Analyze* > *Enabled Protocols* > *MAC-LTE* > *mac_lte_udp* and *MAC-NR* > *mac_nr_udp*

Using the same filename for mac_filename and mac_nr_filename writes both MAC-LTE and MAC-NR to the same file allowing a better analysis.

To decode NAS pcaps add and entry with the following:

- DLT=148
- Payload Protocol=nas-eps

For more information, see https://wiki.wireshark.org/MAC-LTE.

The srsEPC application supports packet capture (pcap) of S1AP messages between the MME and eNodeBs. Enable packet captures in *epc.conf* or on the command line, by setting the *pcap.enable* value to *true*. To view in wireshark, edit the preferences of the DLT_USER dissector.

To decode S1AP pcaps add an entry with:

- DLT=150

- Payload Protocol=s1ap

2.2 User Manuals

2.2.1 Setup Guide

Baseline Hardware Requirements

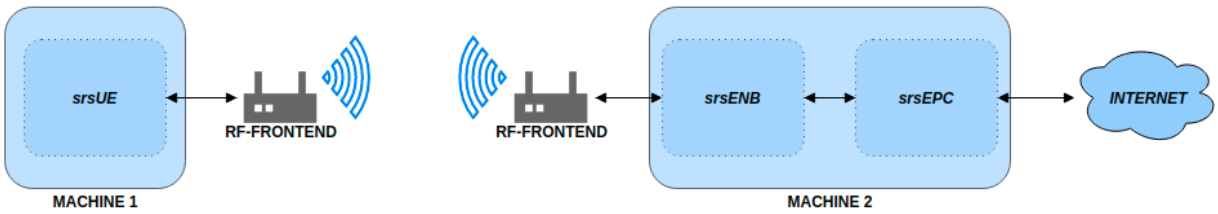


Fig. 1: Basic srsRAN System Architecture (4G LTE)

The overall system requires 2 x RF-frontends and 2 x PCs with a Linux based OS. This can be broken down as follows:

Table 1: System Hardware Requirements

Network Element	RF-Frontend	Linux based PC
srsUE	X	X
srsENB/ srsGNB	X	X
srsEPC		X

The UE will be instatiated on machine 1 with an RF-frontend attached. The eNB will run on machine 2 with an RF-frontend attached to communicate over the air with the UE. The EPC will be insantiated on the same machine as the eNB. See the following figure which outlines the overall system architecture.

A list of supported RF front-end drivers is outlined *here*. We also have some suggested hardware packages, which can be found *here*.

Running a 4G End-to-end System

The following execution instructions are for users that have the appropriate RF-hardware to simulate a network. If you would like to test the use of srsRAN without RF-hardware please see the ZeroMQ application note.

The srsUE, srsENB and srsEPC applications include example configuration files that should be copied (manually or by using the convenience script) and modified, if needed, to meet the system configuration. On many systems they should work out of the box.

By default, all applications will search for config files in the user's home directory (~/.srs) upon startup.

Note that you have to execute the applications with root privileges to enable real-time thread priorities and to permit creation of virtual network interfaces.

srsENB and srsEPC can run on the same machine as a network-in-the-box configuration. srsUE needs to run on a separate machine.

If you have installed the software suite using `sudo make install` and have installed the example config files using `sudo srsRAN_install_configs.sh`, you may just start all applications with their default parameters.

srsEPC

On machine 1, run srsEPC as follows:

```
sudo srsepc
```

Using the default configuration, this creates a virtual network interface named "srs_spgw_sgi" on machine 1 with IP 172.16.0.1. All connected UEs will be assigned an IP in this network.

srsENB

Also on machine 1, but in another console, run srsENB as follows:

```
sudo srsenb
```

srsUE

On machine 2, run srsUE as follows:

```
sudo srsue
```

Using the default configuration, this creates a virtual network interface named "tun_srsue" on machine 2 with an IP in the network 172.16.0.x. Assuming the UE has been assigned IP 172.16.0.2, you may now exchange IP traffic with machine 1 over the LTE link. For example, run a ping to the default SGi IP address:

```
ping 172.16.0.1
```

Examples

If srsRAN is build from source, then preconfigured example use-cases can be found in the following folder: `./srsRAN/build/lib/examples`

The following list outlines some of the use-cases covered:

- Cell Search
- NB-IoT Cell Search
- A UE capable of decoding PDSCH packets
- An eNB capable of creating and transmitting PDSCH packets

Note, the above examples require RF-hardware to run. These examples also support the use of srsGUI for real time plotting of data.

2.2.2 UE User Manual

Introduction

Overview

SrsUE is a 4G LTE and 5G NR NSA UE modem implemented entirely in software. Running as an application on a standard Linux-based operating system, srsUE connects to any LTE network and provides a standard network interface with high-speed mobile connectivity. To transmit and receive radio signals over the air, srsUE requires SDR hardware such as the Ettus Research USRP.

To provide a complete end-to-end LTE network, use srsUE with srsENB and srsEPC.

This User Guide provides all the information needed to get up and running with the srsUE application, to become familiar with all of the key features and to achieve optimal performance. For information on extending or modifying the srsUE source code, please see the srsUE Developers Guide.

Features

The SRS UE includes the following features:

- LTE Release 10 aligned with features up to release 15
- 5G NSA support
- TDD and FDD configurations
- Tested bandwidths: 1.4, 3, 5, 10, 15 and 20 MHz
- Transmission modes 1 (single antenna), 2 (transmit diversity), 3 (CCD) and 4 (closed-loop spatial multiplexing)
- Manually configurable DL/UL carrier frequencies
- Soft USIM supporting XOR/Milenage authentication
- Hard USIM support via PC/SC
- Snow3G and AES integrity/ciphering support
- TUN virtual network kernel interface integration for Linux OS
- Detailed log system with per-layer log levels and hex dumps
- MAC and NAS layer wireshark packet captures
- Command-line trace metrics
- Detailed input configuration files
- Evolved multimedia broadcast and multicast service (eMBMS)

- Frequency-based ZF and MMSE equalizers
- Highly optimized Turbo Decoder available in Intel SSE4.1/AVX2 (+150 Mbps)
- Channel simulator for EPA, EVA, and ETU 3GPP channels
- QoS support
- 150 Mbps DL in 20 MHz MIMO TM3/TM4 or 2xCA configuration (195 Mbps with QAM256)
- 75 Mbps DL in 20 MHz SISO configuration (98 Mbps with QAM256)
- 36 Mbps DL in 10 MHz SISO configuration
- Supports Ettus USRP B2x0/X3x0 families, BladeRF, LimeSDR

srsUE has been fully tested and validated with the following network equipment:

- Amarisoft LTE100 eNodeB and EPC
- Nokia FlexiRadio family FSMF system module with 1800MHz FHED radio module and TravelHawk EPC simulator
- Huawei DBS3900
- Octasic Flexicell LTE-FDD NIB

UE architecture

Fig. 2: Basic UE Architecture

The srsUE application includes layers 1, 2 and 3 as shown in the figure above.

At the bottom of the UE protocol stack, the Physical (PHY) layer carries all information from the MAC over the air interface. It is responsible for link adaptation, power control, cell search and cell measurement.

The Medium Access Control (MAC) layer multiplexes data between one or more logical channels into Transport Blocks (TBs) which are passed to/from the PHY layer. The MAC is responsible for control and scheduling information exchange with the eNodeB, retransmission and error correction (HARQ) and priority handling between logical channels.

The Radio Link Control (RLC) layer can operate in one of three modes: Transparent Mode (TM), Unacknowledged Mode (UM) and Acknowledged Mode (AM). The RLC manages multiple logical channels or bearers, each of which operates in one of these three modes. Transparent Mode bearers simply pass data through the RLC. Unacknowledged Mode bearers perform concatenation, segmentation and reassembly of data units, reordering and duplication detection. Acknowledged Mode bearers additionally perform retransmission of missing data units and resegmentation.

The Packet Data Convergence Protocol (PDCP) layer is responsible for ciphering of control and data plane traffic, integrity protection of control plane traffic, duplicate discarding and in-sequence delivery of control and data plane traffic to/from the RRC and GW layers respectively. The PDCP layer also performs header compression (ROHC) of IP data if supported.

The Radio Resource Control (RRC) layer manages control plane exchanges between the UE and the eNodeB. It uses System Information broadcast by the network to configure the lower layers of the UE and handles the establishment, maintenance and release of the RRC connection with the eNodeB. The RRC manages cell search to support cell selection as well as cell measurement reporting and mobility control for handover between neighbouring cells. The RRC is also responsible for handling and responding to paging messages from the network. Finally, the RRC manages security functions for key management and the establishment, configuration, maintenance and release of radio bearers.

The Non-Access Stratum (NAS) layer manages control plane exchanges between the UE and entities within the core network (EPC). It controls PLMN selection and manages network attachment procedures, exchanging identification

and authentication information with the EPC. The NAS is responsible for establishing and maintaining IP connectivity between the UE and the PDN gateway within the EPC.

The Gateway (GW) layer within srsUE is responsible for the creation and maintenance of the TUN virtual network kernel interface, simulating a network layer device within the Linux operating system. The GW layer permits srsUE to run as a user-space application and operates with data plane IP packets.

Getting Started

To get started with srsUE you will require a PC with a GNU/Linux based operating system and an SDR RF front-end. An SDR RF front-end is a generic radio device such as the Ettus Research USRP that connects to your PC and supports transmission and reception of raw radio signals.

If you are using Ubuntu operating system, you can install srsUE from the binary packages provided:

```
sudo add-apt-repository ppa:srslte/releases
sudo apt-get update
sudo apt-get install srsue
```

If you are using a different distribution, you can install from source using the guide provided in the project's GitHub page.

After installing the software you can install the configuration files into the default location (`~/.config/srsran`), by running:

```
srsran_install_configs.sh user
```

Running the software

To run srsUE with default parameters, run `sudo srsue` on the command line. srsUE needs to run with sudo admin privileges in order to be able to create high-priority threads and to create a TUN device upon successful network attach. Upon starting, srsUE will attempt to find your RF front-end device and connect to a local cell.

If srsUE successfully attaches to a local network, it will start a TUN interface `tun_srsue`. The TUN interface can be used as any other network interface on your PC, supporting data traffic to and from the network.

Example console output for a successful network attach:

```
linux; GNU C++ version 6.3.0 20170618; Boost_106200; UHD_003.009.007-release
Reading configuration file /.config/srsran/ue.conf...
Built in RelWithDebInfo mode using commit 6b2961fce on branch next.

Opening 1 RF devices with 2 RF channels...
Opening USRP with args: type=b200,master_clock_rate=30.72e6
Waiting PHY to initialize ... done!

Attaching UE...
Searching cell in DL EARFCN=2850, f_dl=2630.0 MHz, f_ul=2510.0 MHz
Found Cell:  Mode=FDD, PCI=1, PRB=6, Ports=2, CFO=1.3 KHz
Found PLMN:  Id=00101, TAC=7
Random Access Transmission: seq=42, ra-rnti=0x2
RRC Connected
Random Access Complete.     c-rnti=0x46, ta=0
Network attach successful. IP: 192.168.3.2
```

With the UE attached to the network, type `t` in the console to enable the metrics trace. Example metrics trace:

```
----Signal-------------DL-----------------------------------UL----------------------
cc  rsrp    pl    cfo    mcs   snr turbo  brate   bler   ta_us  mcs   buff  brate   bler
 0   -77    77   1.2k     24    33  0.84   5.8M     0%    0.0    18   193k   624k     0%
 0   -77    77   1.2k     24    31  0.80   5.7M     0%    0.0    18   193k   631k     0%
 0   -77    77   1.2k     24    32  0.80   5.8M     0%    0.0    18   192k   633k     0%
 0   -77    77   1.2k     25    34  0.93   6.0M     0%    0.0    18   194k   636k     0%
 0   -77    77   1.2k     24    33  0.83   5.8M     0%    0.0    19   193k   632k     0%
 0   -77    77   1.2k     24    31  0.82   5.8M     0%    0.0    18   194k   632k     0%
 0   -77    77   1.2k     24    32  0.82   5.8M     0%    0.0    18   193k   635k     0%
 0   -77    77   1.2k     25    34  0.91   6.0M     0%    0.0    18   194k   629k     0%
 0   -77    77   1.2k     24    33  0.85   5.8M     0%    0.0    19   193k   634k     0%
 0   -77    77   1.2k     24    31  0.82   5.7M     0%    0.0    19   194k   647k     0%
 0   -77    77   1.2k     24    32  0.84   5.8M     0%    0.0    18   192k   629k     0%
```

Configuration

The UE can be configured through the configuration file: `ue.conf`. This configuration file provides parameters relating to operating frequencies, transmit power levels, USIM properties, logging levels and much more. To run srsUE with the installed configuration file, use `sudo srsue ~/.config/srsran/ue.conf`.

All parameters specified in the configuration file can also be overwritten on the command line. For example, to run the UE with a different EARFCN, use `sudo srsue ~/.config/srsran/ue.conf --rf.dl_earfcn 3350`.

By default, srsUE uses a virtual USIM card, with parameters from `ue.conf`. These parameters are:

- ALGO - the authentication algorithm to use (MILENAGE or XOR)
- IMSI - the unique identifier of the USIM
- K - the secret key shared with the HSS in the EPC
- OP or OPc - the Operator Code (only used with MILENAGE algorithm)

To connect successfully to a network, these parameters will need to match those in the HSS of the EPC. MILENAGE is the algorithm used in most networks, the XOR algorithm is used primarily by test equipment and test USIM cards. OP is the network-wide operator code and OPc is the USIM-specific encrypted operator code - both are supported by srsUE.

Hardware Setup

To use srsUE to connect over-the-air to a local network, you will need an RF front-end and suitable antennas. The default EARFCN is 3400 (2565MHz uplink, 2685MHz downlink). To reduce TX-RX crosstalk, we recommend orienting TX and RX antennas at a 90 degree angle to each other.

The srsUE can also be used over a cabled connection. The cable configuration and required RF components will depend upon your RF front-end. For RF front-ends such as the USRP, connect TX to RX and ensure at least 30dB of attenuation to avoid damage to your devices. For more detailed information about cabled connections, see *Advanced Usage*.

Operating System Setup

The srsUE runs in user-space with standard linux kernels. For best performance, we recommend disabling CPU frequency scaling. To disable frequency scaling use:

```
for f in /sys/devices/system/cpu/cpu[0-9]*/cpufreq/scaling_governor ; do
  echo performance > $f
done
```

Observing results

To observe srsUE results, use the generated log files and packet captures.

Log files are created by default at /tmp/ue.log. The srsUE configuration file can be used to specify log levels for each layer of the network stack and to enable hex message output. Supported log levels are debug, info, warning, error and none.

Log messages take the following format:

```
Timestamp  [Layer ]  Level    Content
```

e.g.:

```
17:52:25.246 [RLC ]  Info    DRB1 Tx SDU
```

or with hex message output enabled:

```
17:52:25.246 [RLC ]  Info    DRB1 Tx SDU
         0000: 8b 45 00 00 c7 f3 8b 40 00 01 11 d1 f6 c0 a8 03
         0010: 01 ef ff ff fa 92 55 07 6c 00 b3 ee 41 4d 2d 53
```

PHY-layer log messages have additional details:

```
Timestamp  [Layer]  Level  [Subframe] Channel:  Content
```

e.g.:

```
17:52:26.094 [PHY1]  Info  [05788]  PDSCH:    l_crb= 1, harq=0, snr=22.1 dB, CW0: tbs=55,
↪ mcs=22, rv=0, crc=OK, it=1, dec_time=  12 us
```

The srsUE application supports packet capture at two levels - MAC layer and NAS layer. MAC layer captures include both control and data traffic and will be encrypted if configured by the network. NAS layer captures include control traffic only and will not be encrypted. Packet capture (pcap) files can be viewed using Wireshark (www.wireshark.org).

See the explanation *here* on setting up wireshark to decdode pcaps.

Troubleshooting

RF Configuration

The srsUE software application generates and consumes raw radio signals in the form of baseband I/Q samples. In order to transmit and receive these signals over the air, an RF front end device is needed. Devices supported by srsRAN include e.g. USRP, BladeRF and LimeSDR.

When using an RF front-end, the *dl_earfcn* field in `ue.conf` should be populated. This field provides the UE with a list (comma separated) of DL EARFCNs. The UE will perform the cell search procedure using the specified EARFCNs. The UL EARFCN is normally deduced from the DL EARFCN but it could optionally forced by setting *ul_earfcn*:

```
[rf]
dl_earfcn = 3400
#ul_earfcn = 21400
...
```

In some cases, one may use some custom bands which do not mach with any EARFCN. In these cases, the downlink and uplink frequencies can be forced by setting *dl_frequency* and *ul_frequency* respectively in Hertzs:

```
...
dl_freq = 400e6
ul_freq = 450e6
...
```

Attention: the eNB and UE DL EARFCNs calculate some security sequences using the DL EARFCN. If they do not match, the UE may fail to perform some actions.

Most off-the-shelf RF front-ends have relatively low-accuracy clocks, resulting in high frequency offsets (> 1kHz) from base stations (which use high-accuracy GPS disciplined clock sources). A large frequency offset deteriorates the UE receiver performance. It is recommended setting the parameter *freq_offset* (Hz) in order manually correct large offsets. This parameter applies an offset to the received DL signal and will mitigate the impairment caused by the carrier frequency offset. Also, the UE will apply a proportional correction in the UL frequency.

```
...
freq_offset = -6600
...
```

The current UE does not support open or closed loop power control. The RF front end gain is controlled by the user before running the UE. The transmit gain (*tx_gain*) is specified in dB and maximum transmit power range varies between brands and models.

At the receiver side, an Automatic Gain Control (AGC) module is activated when the receiver gain (*rx_gain* in dB) is not specified. Otherwise, it sets a fixed receive gain. Once again, the range of gain varies between brands and models.

```
...
tx_gain = 80
#rx_gain = 40
...
```

When transmitting, the srsUE application provides a radio signal to the front-end and specifies the time at which the signal should be transmitted. Typically, an RF front-end will have a small fixed timing offset caused by delays in the RF chain. This offset is usually in the order of microseconds and can vary between different devices. To calibrate this offset, it is possible to use the *time_adv_nsamples* parameter. This compensates the delay and will ensure that the UE transmits at the correct time.

Network Attach

There are two main reasons for a network attach failing:

- A misconfigured network
- RF issues

Either may stop the UE being able to see the eNB, cause the UE to fail to connect, or cause the UE to connect but with poor stability.

Misconfigured Network

A misconfigured network may stop the UE being able to see the eNB and/ or connect to the EPC. It may be helpful to reference the EPC user manual, namely the *configuration section* to ensure the EPC has been configured correctly. The UE configuration file should also be checked to ensure the relevant information is reflected across the two files. See the *configuration section* of the UE documentation for notes on this.

An unsuccessful attach can be down to how the UE's credentials are reflected in the EPC's config file and database. See the *COTS UE* Application Note for info on how to add a UE to the EPC's database and ensure the correct network configuration. Note, this is for connecting a COTS UE but may also be useful for troubleshooting issues when connecting srsUE using an SDR.

Users should also keep an eye on the console outputs of the UE, eNB and EPC to ensure no errors were given when starting up the network. Errors during network instantiation may lead to elements not connecting properly.

RF Issues

The RF hardware and configuration should also be checked if a network attach continues to fail.

First check that the hardware is correctly connected and running over USB 3.0, also check the drivers for your HW are up to date. The latest drivers can be found *here*.

The antenna choice and position is important to ensure the correct operation of the SDR and overall network. We recommend using the Vert2450 antenna from Ettus (or similar). The antennae should be positioned at 90° to each other. You should also ensure the correct ports are used for the antennae. For example, on the b200 mini the *TRX* and *RX2* ports are used.

It is also important that the correct configuration settings are used as described *above*.

If possible you should use a spectrum analyser or other such piece of equipment to check the state of the signal(s) being transmitted by the RF-hardware. If the signal is too weak or malformed then an attach will not be successful. The gr-fosphor tool is a very useful SDR spectrum analyzer which can be used to check the properties of transmitted RF signals.

Carrier frequency offset (CFO) may also result in a UE not being able to sucessfully attach to an eNB. Check the configration files so that EARFCNs and carrier frequencies match. You may also need to calibrate your SDR, as low clock accuracy may result in the CFO being outside of the accepted tolerance. Multiple open source tools like Kalibrate-RTL can be used to calculate the oscillator offset of your SDR and help to minimize CFO. An external clock reference or GPSODO can also be used to increase clock accuracy. Calibrating your SDR may also help with peak throughput and stability.

Peak Throughput

Maximum achievable srsUE peak throughput may be limited for a number of different reasons. These include limitations in the PC being used, the network configuration, the RF-hardware and the physical network conditions.

Computational Power

In order to achieve peak throughput, we recommend using a PC with an 8th Gen i7 processor or above, running Ubuntu 16.04 OS or higher. Machines with lower specs can also run srsRAN sucessfully but with lower maximum throughput.

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

```
echo "performance" | sudo tee /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
```

Again, you should also ensure your SDR drivers are up to date and that you are running over USB 3.0, as this will also affect maximum throughput.

If using a laptop, users should keep the PC connected to a power-source at all times while running srsRAN, as this will avoid performance loss due to CPU frequency scaling on the machine.

The computational requirements of the srsUE application are closely tied to the bandwidth of the LTE or NR carrier being used. For example, maximum throughput using 100-PRB carrier will require a more powerful CPU than maximum throughput using a 25-PRB carrier. If your machine is not powerful enough to support srsUE with a given network configuration, you will see Late and/or Overflow packet reports from the SDR front-end.

RF Hardware

The RF-signal itself can also affect the peak throughput a network can achieve. Ensure the radio being used is correctly calibrated and that the appropriate gain settings are used. The health of an RF-signal can be quickly checked using the console trace output by srsUE.

The following is an example of a "healthy" console trace from srsUE. This trace is for a 50-PRB network configuration. Note the relatively low CFO of 3.1kHz, the high SNR value, the high MCS values and the 0% BLER on both DL and UL:

```
--------Signal--------------DL-----------------------------------UL---------------------
↪--
cc pci  rsrp    pl    cfo   mcs   snr turbo  brate   bler   ta_us  mcs   buff  brate  ␣
↪bler
0   1   -62    62  -3.1k   3.7    39  0.42   3.5k     0%   0.0    14    0.0    33k     0%
0   1   -62    62  -3.1k   3.5    39  0.50    0.0     0%  0.52    22    0.0    0.0     0%
0   1   -62    62  -3.1k   3.5    39  0.50    0.0     0%  0.52    22    0.0    0.0     0%
0   1   -62    62  -3.1k    16    37  0.73    33M     0%  0.52    22    0.0    57k     0%
0   1   -62    62  -3.1k    28    34   1.0    72M     0%  0.52    22    0.0    69k     0%
0   1   -62    62  -3.1k    28    34   1.0    72M     0%  0.52    22    2.0    65k     0%
0   1   -62    62  -3.1k    28    34   1.0    72M     0%  0.52    22    0.0    69k     0%
0   1   -62    62  -3.1k    28    34   1.0    72M     0%  0.52    22    0.0    69k     0%
0   1   -62    62  -3.1k    28    34   1.0    72M     0%  0.52    22    2.0    65k     0%
0   1   -62    62  -3.1k    28    34   1.0    72M     0%  0.52    22    0.0    69k     0%
0   1   -62    62  -3.1k    28    34   1.0    72M     0%  0.52    22    0.0    69k     0%
```

The SNR, CFO and BLER can be used to debug the health of a signal connection. See the section on UE *command line reference* for information regarding the console trace.

Advanced Usage

External USIM

This section is only needed if you do not have access to the USIM credentials, or have no control over the network. Note, most programmable or test USIM cards ship with their credentials.

Using an actual SIM card to authenticate the user against the network is an advanced feature. It requires a SIM card reader attached to the PC running srsUE that is supported by PCSClite.

Before using a SIM card, please make sure to disable PIN activation using a regular phone.

In order to compile srsUE with support for it, the pcsclite development headers as well as the pcsc daemon need to be installed and running. On Ubuntu (or other Debian derivates), this can be done with:

```
sudo apt-get install libpcsclite-dev pcscd pcsc-tools
```

After this is done, please verify you've got a PCSC-compatible reader by running 'pcsc_scan'.

Now, CMake should pick up the pcsc libraries and build the support code for it. If that is not the case, try with a clean build folder or remove your exisiting `CMakeCache.txt`:

```
$ cmake ..
...
-- PCSC LIBRARIES: /usr/lib/x86_64-linux-gnu/libpcsclite.so
-- PCSC INCLUDE DIRS: /usr/include/PCSC
-- Building with PCSC support.
...
$ make
```

After the build is complete, you can verify the correct operation with the `pcsc_usim_test` application. Please verify that the IMSI can be read from the card:

```
$ ./srsue/test/upper/pcsc_usim_test
..
09:06:36.064073 [USIM] [D] SCARD: MNC length=2
09:06:36.064079 [USIM] [D] MNC length 2
IMSI: 21XXXXXXXXXXX
09:06:36.064095 [USIM] [D] SCARD: UMTS auth - RAND
             0000: bc 4c b0 27 b3 4b 7f 51 21 5e 56 5f 67 3f de 4f
09:06:36.064102 [USIM] [D] SCARD: UMTS auth - AUTN
             0000: 5a 17 77 3c 62 57 90 01 cf 47 f7 6d b3 a0 19 46
09:06:36.064107 [USIM] [D] SCARD: scard_transmit: send
             0000: 00 11 00 81 22 10 bc ac b1 17 13 4b 6f 51 21 5e
             0010: 47 47 6d b3 a0 19 46
09:06:36.119675 [USIM] [D] SCARD: SCardTransmit: recv
             0000: 98 62
09:06:36.119707 [USIM] [D] SCARD: UMTS alg response
             0000: 98 62
09:06:36.119717 [USIM] [W] SCARD: UMTS auth failed - MAC != XMAC
09:06:36.119725 [USIM] [E] SCARD: Failure during USIM UMTS authentication
09:06:36.119732 [USIM] [D] SCARD: deinitializing smart card interface
```

If those steps completed successfully we can now start srsUE by either enabling the PCSC USIM in the config file or by passing the option as an command line argument, e.g., run:

```
$ ./srsue/src/srsue --usim.mode=pcsc
```

Channel Emulator

The srsUE application includes an internal channel emulator in the downlink receive path which can emulate uncorrelated fading channels, propagation delay and Radio-Link failure.

The channel emulator can be enabled and disabled with the parameter *channel.dl.enable*.

```
[channel]
dl.enable = true
...
```

As mentioned above, the channel emulator can simulate fading channels. It supports 4 different models:

- none: single tap with no delay, doppler dispersion can be applied if specified.
- epa: Extended Pedestrian A, described in 3GPP 36.101 Section B.2.1
- eva: Extended Vehicular A model, described in 3GPP 36.101 Section B.2.1
- etu: Extended Typical Urban model, described in 3GPP 36.101 Section B.2.1

The fading emulator has two parameters: *enable* and *model*. The parameter *model* is the channel model mentioned above, followed by the maximum Doppler dispersion (e.g. eva5). The following example enables the fading submodule with a EVA fading model and a maximum doppler dispersion of 5 Hz.

```
...
dl.fading.enable = true
dl.fading.model  = eva5
...
```

The delay simulator generates the delay according to the next formula:

$$d(t) = delay.min_us + (delay.max_us - delay.min_us) * (1 + sin(2 * pi * t/delay.period))/2$$

Where *delay.min_us* and *delay.max_us* are specified in microseconds while *delay.period* must be in seconds.

Hence, the maximum simulated speed is given by:

$$v_max = (delay.max_us - delay.min_us) * pi * 300/delay.period$$

The following example enables the delay simulator for having a period of 1h with a minimum delay of 10 microseconds and a maximum delay of 100 microseconds:

```
...
dl.delay.enable     = true
dl.delay.period     = 3600
dl.delay.max_us = 100
dl.delay.min_us = 10
...
```

Finally, the Radio-Link Failure (RLF) simulator has two states:

- on: the UE receives baseband signal, unaffected by the simulator.
- off: the UE does not receive any signal, the simulator substitutes the baseband with zeros.

The time the emulator spends in *on* is parametrized by *rlf.t_on_ms* and *rlf.t_off_ms* for *off*. Both parameters are expected to be in milliseconds.

The following example enables the RLF simulator for having 2 seconds of blackout every 10 seconds of full baseband signal:

```
...
dl.rlf.enable       = true
dl.rlf.t_on_ms      = 10000
dl.rlf.t_off_ms     = 2000
...
```

MIMO

The srsUE supports MIMO operation for transmission modes 1, 2, 3 and 4. The user can select the number of select antennas in the `ue.conf`:

```
...
[rf]
...
nof_rx_ant = 2
...
```

Do you want to attach to a 2 port eNb and you have only one receive channel? No problem. The UE can attach to 2 port cell and be in TM3 or TM4 without having a second receive antenna. Nevertheless, it will not take advantage of spatial multiplexing and it will not achieve the maximum throughput.

Configuration Reference

The srsUE example configuration file contains detailed descriptions of all UE configuration parameters.

Command Line Reference

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

```
----Signal-------------DL-----------------------------------UL----------------------
cc  rsrp    pl    cfo   mcs   snr turbo  brate   bler   ta_us  mcs   buff  brate   bler
 0   -77    77   1.2k    24    33  0.84   5.8M     0%   0.0    18   193k   624k     0%
 0   -77    77   1.2k    24    31  0.80   5.7M     0%   0.0    18   193k   631k     0%
 0   -77    77   1.2k    24    32  0.80   5.8M     0%   0.0    18   192k   633k     0%
 0   -77    77   1.2k    25    34  0.93   6.0M     0%   0.0    18   194k   636k     0%
 0   -77    77   1.2k    24    33  0.83   5.8M     0%   0.0    19   193k   632k     0%
 0   -77    77   1.2k    24    31  0.82   5.8M     0%   0.0    18   194k   632k     0%
 0   -77    77   1.2k    24    32  0.82   5.8M     0%   0.0    18   193k   635k     0%
 0   -77    77   1.2k    25    34  0.91   6.0M     0%   0.0    18   194k   629k     0%
 0   -77    77   1.2k    24    33  0.85   5.8M     0%   0.0    19   193k   634k     0%
 0   -77    77   1.2k    24    31  0.82   5.7M     0%   0.0    19   194k   647k     0%
 0   -77    77   1.2k    24    32  0.84   5.8M     0%   0.0    18   192k   629k     0%
```

Metrics are generated once per second by default. This can be configured using the *expert.metrics_period_secs* parameter in `ue.conf`.

Metrics are provided for the received signal (Signal), downlink (DL) and uplink (UL) respectively. The following metrics are provided:

cc Component carrier

rsrp Reference Signal Receive Power (dBm)

pl Pathloss (dB)

cfo Carrier Frequency Offset (Hz)

mcs Modulation and coding scheme (0-28)

snr Signal-to-Noise Ratio (dB)

turbo Average number of turbo decoder iterations

brate Bitrate (bits/sec)

bler Block error rate

ta_us Timing advance (uS)

buff Uplink buffer status - data waiting to be transmitted (bytes)

2.2.3 eNodeB User Manual

Introduction

Overview

SrsENB is an LTE eNodeB basestation implemented entirely in software. Running as an application on a standard Linux-based operating system, srsENB connects to any LTE core network (EPC) and creates a local LTE cell. To transmit and receive radio signals over the air, srsENB requires SDR hardware such as the Ettus Research USRP.

To provide an end-to-end LTE network, use srsENB with srsEPC and srsUE.

This User Guide provides all the information needed to get up and running with the srsENB application, to become familiar with all of the key features and to achieve optimal performance. For information on extending or modifying the srsENB source code, please see the srsENB Developers Guide.

Features

The srsENB LTE eNodeB includes the following features:

- LTE Release 10 aligned
- FDD configuration
- Tested bandwidths: 1.4, 3, 5, 10, 15 and 20 MHz
- Transmission mode 1 (single antenna), 2 (transmit diversity), 3 (CCD) and 4 (closed-loop spatial multiplexing)
- Frequency-based ZF and MMSE equalizer
- Evolved multimedia broadcast and multicast service (eMBMS)
- Highly optimized Turbo Decoder available in Intel SSE4.1/AVX2 (+150 Mbps)
- Detailed log system with per-layer log levels and hex dumps
- MAC layer wireshark packet capture
- Command-line trace metrics
- Detailed input configuration files
- Channel simulator for EPA, EVA, and ETU 3GPP channels
- ZeroMQ-based fake RF driver for I/Q over IPC/network
- Intra-ENB and Inter-ENB (S1) mobility support
- Proportional-fair and round-robin MAC scheduler with FAPI-like C++ API
- SR support
- Periodic and Aperiodic CQI feedback support
- Standard S1AP and GTP-U interfaces to the Core Network
- 150 Mbps DL in 20 MHz MIMO TM3/TM4 with commercial UEs (195 Mbps with QAM256)
- 75 Mbps DL in SISO configuration with commercial UEs
- 50 Mbps UL in 20 MHz with commercial UEs
- User-plane encryption

srsENB has been tested and validated with a broad range of COTS handsets including:

- LG Nexus 5 and 4
- Motorola Moto G4 plus and G5
- Huawei P9/P9lite, P10/P10lite, P20/P20lite
- Huawei dongles: E3276 and E398

eNodeB architecture

Fig. 3: Basic eNodeB Architecture

The srsENB application includes layers 1, 2 and 3 as shown in the figure above.

At the bottom of the protocol stack, the Physical (PHY) layer carries all information from the MAC over the air interface. It is responsible for link adaptation and power control.

The Medium Access Control (MAC) layer multiplexes data between one or more logical channels into Transport Blocks (TBs) which are passed to/from the PHY layer. The MAC is responsible for scheduling uplink and downlink transmissions for connected UEs via control signalling, retransmission and error correction (HARQ) and priority handling between logical channels.

The Radio Link Control (RLC) layer can operate in one of three modes: Transparent Mode (TM), Unacknowledged Mode (UM) and Acknowledged Mode (AM). The RLC manages multiple logical channels or bearers for each connected UE. Each bearer operates in one of these three modes. Transparent Mode bearers simply pass data through the RLC. Unacknowledged Mode bearers perform concatenation, segmentation and reassembly of data units, reordering and duplication detection. Acknowledged Mode bearers additionally perform retransmission of missing data units and resegmentation.

The Packet Data Convergence Protocol (PDCP) layer is responsible for ciphering of control and data plane traffic, integrity protection of control plane traffic, duplicate discarding and in-sequence delivery of control and data plane traffic to/from the RRC and GTP-U layers respectively. The PDCP layer also performs header compression (ROHC) of IP data if supported.

The Radio Resource Control (RRC) layer manages control plane exchanges between the eNodeB and connected UEs. It generates the System Information Blocks (SIBs) broadcast by the eNodeB and handles the establishment, maintenance and release of RRC connections with the UEs. The RRC also manages security functions for ciphering and integrity protection between the eNodeB and UEs.

Above the RRC, the S1 Application Protocol (S1-AP) layer provides the control plane connection between the eNodeB and the core network (EPC). The S1-AP connects to the Mobility Management Entity (MME) in the core network. Messages from the MME to UEs are forwarded by S1-AP to the RRC layer, where they are encapsulated in RRC messages and sent down the stack for transmission. Messages from UEs to the MME are similarly encapsulated by the UE RRC and extracted at the eNodeB RRC before being passed to the S1-AP and on to the MME.

The GPRS Tunnelling Protocol User Plane (GTP-U) layer within srsENB provides the data plane connection between the eNodeB and the core network (EPC). The GTP-U layer connects to the Serving Gateway (S-GW) in the core network. Data plane IP traffic is encapsulated in GTP packets at the GTP-U layer and these GTP packets are tunneled through the EPC. That IP traffic is extracted from the tunnel at the Packet Data Network Gateway (P-GW) and passed out into the internet.

Getting Started

To get started with srsENB you will require a PC with a GNU/Linux based operating system and an SDR RF front-end. An SDR RF front-end is a generic radio device such as the Ettus Research USRP that connects to your PC and supports transmission and reception of raw radio signals.

If you are using Ubuntu operating system, you can install srsENB from the binary packages provided:

```
sudo add-apt-repository ppa:srslte/releases
sudo apt-get update
sudo apt-get install srsenb
```

If you are using a different distribution, you can install from source using the guide provided in the project's GitHub page.

After installing the software you can install the configuration files into the default location (`~/.config/srsran`), by running:

```
srsran_install_configs.sh user
```

Running the software

To run srsENB with default parameters, run `sudo srsenb` on the command line. srsENB needs to run with sudo admin privileges in order to be able to create high-priority threads. Upon starting, srsENB will attempt to find your RF front-end device, attempt to attach to the core network (EPC) and start broadcasting.

Example console output:

```
linux; GNU C++ version 6.3.0 20170618; Boost_106200; UHD_003.009.007-release
Built in RelWithDebInfo mode using commit 6b2961fce on branch next.

---  Software Radio Systems LTE eNodeB  ---

Reading configuration file /conf/enb.conf...
Setting number of control symbols to 3 for 25 PRB cell.
Opening USRP with args: type=b200,master_clock_rate=30.72e6
Setting frequency: DL=2630.0 Mhz, UL=2510.0 MHz
Setting Sampling frequency 5.76 MHz
Enter t to stop trace.

==== eNodeB started ===
```

Upon receiving a UE connection:

```
RACH:  tti=3381, preamble=3, offset=1, temp_crnti=0x46
User 0x46 connected
```

With the eNodeB running and one or more UEs connected, type `t` in the console to enable the metrics trace. Example metrics trace:

```
------DL------------------------------UL-----------------------------------
rnti  cqi    ri   mcs  brate   bler   snr   phr   mcs  brate   bler    bsr
46   14.1   2.0  25.1  28.4M   0.8%  24.8   0.0  23.1  9.60M   2.2%   140k
46   14.8   2.0  26.6  30.7M     0%  24.9   0.0  23.2  9.92M     0%   140k
46   14.7   2.0  26.3  30.1M   0.8%  24.9   0.0  23.1  9.90M     0%   140k
46   14.8   2.0  26.5  30.6M     0%  24.9   0.0  23.1  9.90M     0%   140k
46   15.0   2.0  26.7  30.9M     0%  24.8   0.0  23.1  9.83M     0%   140k
46   14.5   2.0  26.1  30.0M     0%  24.9   0.0  23.1  9.88M     0%   140k
46   14.8   2.0  26.3  30.3M     0%  24.8   0.0  23.1  9.84M     0%   140k
46   14.7   2.0  26.4  30.4M     0%  24.9   0.0  23.1  9.89M     0%   140k
46   14.7   2.0  26.4  30.4M     0%  24.9   0.0  23.2  9.91M     0%   140k
46   14.7   2.0  26.3  30.4M     0%  24.9   0.0  23.1  9.87M     0%   140k
46   14.8   2.0  26.4  30.4M     0%  24.9   0.0  23.1  9.88M     0%   140k
```

Configuration

The eNodeb can be configured through the configuration file: `enb.conf`. This configuration file provides parameters relating to the cell configuration, operating frequencies, transmit power levels, logging levels and much more. To run srsENB with the installed configuration file, use `sudo srsenb ~/.config/srsran/enb.conf`.

All parameters specified in the configuration file can also be overwritten on the command line. For example, to run the eNodeB with a different EARFCN, use `sudo srsenb ~/.config/srsran/enb.conf --rf.dl_earfcn 3350`.

In addition to the top-level configuration file, srsENB uses separate files to configure SIBs (sib.conf), radio resources (rr.conf) and data bearers (drb.conf). These additional configuration files are listed under [enb_files] in the top-level enb.conf and defaults are provided for each.

A key eNodeB parameter is enb.mme_addr, which specifies the IP address of the core network MME. The default configuration assumes that srsEPC is running on the same machine. For more information, as well instructions for using an EPC on a separate machine, see the EPC user manual.

Hardware Setup

To use srsENB to create an over-the-air local cell, you will need an RF front-end and suitable antennas. The default EARFCN is 3400 (2565MHz uplink, 2685MHz downlink). To reduce TX-RX crosstalk, we recommend orienting TX and RX antennas at a 90 degree angle to each other.

The srsENB can also be used over a cabled connection. The cable configuration and required RF components will depend upon your RF front-end. For RF front-ends such as the USRP, connect TX to RX and ensure at least 30dB of attenuation to avoid damage to your devices. For more detailed information about cabled connections, see *Advanced Usage*.

Operating System Setup

The srsENB runs in user-space with standard linux kernels. For best performance, we recommend disabling CPU frequency scaling. To disable frequency scaling use:

```
for f in /sys/devices/system/cpu/cpu[0-9]*/cpufreq/scaling_governor ; do
  echo performance > $f
done
```

Observing results

To observe srsENB results, use the generated log files and packet captures.

Log files are created by default at /tmp/enb.log. The srsENB configuration file can be used to specify log levels for each layer of the network stack and to enable hex message output. Supported log levels are debug, info, warning, error and none.

Log messages take the following format:

```
Timestamp  [Layer ]  Level    Content
```

e.g.:

```
17:52:25.246 [RLC ]  Info    DRB1 Tx SDU
```

or with hex message output enabled:

```
17:52:25.246 [RLC ]  Info    DRB1 Tx SDU
         0000: 8b 45 00 00 c7 f3 8b 40 00 01 11 d1 f6 c0 a8 03
         0010: 01 ef ff ff fa 92 55 07 6c 00 b3 ee 41 4d 2d 53
```

PHY-layer log messages have additional details:

```
Timestamp  [Layer]  Level  [Subframe] Channel:  Content
```

e.g.:

```
17:52:26.094 [PHY1]  Info  [05788]  PDSCH:    l_crb= 1, harq=0, snr=22.1 dB, CW0: tbs=55,
↪ mcs=22, rv=0, crc=OK, it=1, dec_time=  12 us
```

See the explanation *here* on setting up wireshark to decdode the pcaps captured by srsENB.

Troubleshooting

COTS UE Issues

The following are the most common issues when using a COTS UE with srsENB. A full application note on this can be found *here*. Reference this for more detail on the following issues.

UE Can't See The Network

The most likely reasons for a UE not seeing the network are the eNB/EPC configuration, the RF conditions and the frequency accuracy of the RF frontend being used.

The first thing to check is that the LTE frequency band and EARFCN which you have configured are supported by the UE which you are using. Most UE devices support a subset of the bands allocated for LTE. Ensure also that the full bandwidth of the configured LTE carrier is within the frequency band which you are using.

Some UE devices fail to see networks configured with test PLMN MCC/MNC values. For example, commonly used MCC/MNC values of 901/70 or 001/01 may not work, particularly with iPhone devices using Intel baseband chipsets. Instead, try setting the MCC of your network to your country specific value (e.g. 272 for Ireland). A full list of MCC codes can be found here. The MNC value can then be set to any value that is not currently in use by a Mobile Network Operator in your country.

The RF conditions can be affected by the antenna being used, we recommend the Vert2450 antenna from Ettus (or similar). Ensure the antennae are placed at a 90° angle to each other to minimize cross-talk. If possible you should use a spectrum analyser or other such piece of equipment to check the quality of the signal(s) being transmitted by the RF-hardware. If signals are too weak or malformed then a UE may not successfully receive them and will not attempt to attach. The gr-fosphor tool is a very useful SDR spectrum analyzer which can be used to check the properties of transmitted RF signals.

Low carrier frequency accuracy in the RF front-end may also cause a UE to fail to see the network. The clock accuracy in most SDR front-ends is quite low without the use of an external reference clock input. It may be possible to use lab equipment or open source tools such as Kalibrate-RTL to estimate the CFO of your RF front-end and to manually compensate by setting small frequency offsets in the Downlink and Uplink carrier frequency settings of the eNodeB configuration file.

UE Won't Attach

If the UE sees the network but cannot successfully attach, you can check the MAC-layer PCAP provided by srsENB using Wireshark to see at which point in the attach procedure it fails. For more information about the MAC-layer PCAP and using Wireshark, see *here* in the documentation.

Can't Access Internet

If an attached UE cannot access the internet, this may be due to a misconfigured APN in the UE and/ or eNB. See the *app note* for information on how to configure this.

Another common reason is misconfigured IP routing at the EPC. If using srsEPC, make sure to follow the instructions on IP Masquerading in the *app note*.

Peak Throughput

Maximum achievable srsENB peak throughput may be limited for a number of different reasons. These include limitations in the PC being used, the network configuration, the RF-hardware and the physical network conditions.

Computational Power

In order to achieve peak throughput, we recommend using a PC with an 8th Gen i7 processor or above, running Ubuntu 16.04 OS or highe. Machines with lower specs can also run srsENB sucessfully but with lower maximum achievable throughput.

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

```
echo "performance" | sudo tee /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
```

Again, you should also ensure your SDR drivers are up to date and that you are running over USB 3.0, as this will also affect maximum throughput.

If using a laptop, users should keep the PC connected to a power-source at all times while running srsENB, as this will avoid performance loss due to CPU frequency scaling on the machine.

The computational requirements of the srsENB application are closely tied to the bandwidth of the LTE carrier being used. For example, maximum throughput using 100-PRB carrier will require a more powerful CPU than maximum throughput using a 25-PRB carrier. If your machine is not powerful enough to support srsENB with a given network configuration, you will see Late and/or Overflow packet reports from the SDR front-end.

RF Hardware

The RF-signal itself can also affect the peak throughput a network can achieve. Ensure the radio being used is correctly calibrated and that the appropriate gain settings are used. The health of an RF-signal can be quickly checked using the console trace output by srsENB.

Advanced Usage

MIMO

The srsENB supports MIMO transmission modes 2, 3, and 4. You only need to set up the transmission mode and the number of eNb ports in the `enb.conf` file:

```
...
[enb]
...
tm = 3
nof_ports = 2
...
```

The eNb configures the UE for reporting the Rank Indicator for transmission modes 3 and 4. You can set the rank indicator periodic report in the file `rr.conf` field `m_ri`. This value is multiples of CQI report period. For example, if the CQI period is 40ms and `m_ri` is 8, the rank indicator will be reported every 320ms.

Configuration Reference

The srsENB example configuration file contains detailed descriptions of all eNodeB configuration parameters.

In addition to the top-level configuration file, srsENB uses separate files to configure SIBs sib.conf, radio resources rr.conf and data bearers drb.conf. These files use libconfig format.

Command Line Reference

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

```
------DL------------------------------UL-----------------------------------------
rnti cqi  ri mcs brate   ok  nok  (%)  pusch pucch phr mcs brate   ok  nok  (%)   bsr
  47  15   0  27   49M 1000    0   0%   29.7  0.0  23.0  23   23M 1000    0   0%  130k
  47  15   0  27   49M 1000    0   0%   29.7  0.0  23.0  23   23M 1000    0   0%  130k
  47  15   0  27   49M 1000    0   0%   29.7  0.0  23.0  23   23M 1000    0   0%  128k
  47  15   0  27   49M 1000    0   0%   29.6  0.0  23.0  23   23M 1000    0   0%  128k
  47  15   0  27   49M 1000    0   0%   29.7  0.0  23.0  23   23M 1000    0   0%  128k
```

Metrics are generated once per second by default. This can be configured using the *expert.metrics_period_secs* parameter in `enb.conf`.

Metrics are provided on a per-UE basis for the downlink (DL) and uplink (UL) respectively. The following metrics are provided:

rnti Radio Network Temporary Identifier (UE identifier)

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

ri Rank Indicator reported by the UE (dB)

mcs Modulation and coding scheme (0-28)

brate Bitrate (bits/sec)

ok Number of packets succesfully sent

nok Number of packets dropped

(%) % of packets dropped

pusch PUSCH SNIR (Signal-to-Interference-plus-Noise Ratio)

pucch PUCCH SNIR

phr Power Headroom (dB)

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

2.2.4 EPC User Manual

Introduction

Overview

srsEPC is a lightweight implementation of a complete LTE core network (EPC). The srsEPC application runs as a single binary but provides the key EPC components of Home Subscriber Service (HSS), Mobility Management Entity (MME), Service Gateway (S-GW) and Packet Data Network Gateway (P-GW).

Fig. 4: EPC overall architecture

The figure above illustrates the main components of the EPC, along with the main interfaces between them.

- *HSS*: The Home Subscriber Service (HSS) is the user database. It stores information such as the user's id, key, usage limits, etc. It is responsible for authenticating an authorizing the user's access to the network.
- *MME*: Mobility Managment Entity (MME) is the main control element in the network. It handles mobility and attach control messages. It is also responsible for paging UEs in idle mode.
- *S-GW*: The S-GW is the main dataplane gateway for the users, as it provides the mobility anchor for the UEs. It works as an IP router and helps setting up GTP sessions between the eNB and the P-GW.
- *P-GW*: The Packet Gateway (P-GW) is the point of contact with external networks. It enforces the QoS parameters for subscriber sessions.

To provide a complete end-to-end LTE network, use srsEPC with srsENB and srsUE.

This User Guide provides all the information needed to get up and running with the srsEPC application, to become familiar with all of the key features and to achieve optimal performance. For information on extending or modifying the srsEPC source code, please see the srsEPC Developers Guide.

Features

The srsEPC LTE core network includes the implementation of the MME, HSS and SPGW entities. The features of each of these entities is further described below.

MME Features

The srsEPC MME entity provides support for standard compliant NAS and S1AP protocols to provide control plane communication between the EPC and the UEs and eNBs.

At the NAS level, this includes:

- Attach procedure, detach procedure, service request procedure
- NAS Security Mode Command, Identity request/response, authentication
- Support for the setup of integrity protection (EIA1 and EIA2) and ciphering (EEA0, EEA1 and EEA2)

At the S1AP level, this includes:

- S1-MME Setup/Tear-down
- Transport of NAS messages
- Context setup/release procedures
- Paging procedures

HSS Features

The srsEPC HSS entity provides support for configuring UE's authentication parameters and other parameters that can be configured on a per-UE basis. The HSS entity includes the following features:

- Simple CSV based database
- XOR and MILENAGE authentication algorithms, specified per UE.
- QCI information
- Dynamic or static IP configuration of UEs

SPGW Features

The srsEPC SPGW entity provides support for to user plane communication between the EPC and the and eNBs, using S1-U and SGi interfaces.

The SPGW supports the following features:

- SGi interface exposed as a virtual network interface (TUN device)
- SGi < > S1-U Forwarding using standard compliant GTP-U protocol
- Support of GTP-C procedures to setup/teardown GTP-U tunnels
- Support for Downlink Data Notification procedures

Getting Started

To get started with srsEPC you will require a PC with a GNU/Linux based operating system. This can be a distribution of your preference, such as Ubuntu, Debian, Fedora, etc.

If you are using Ubuntu, you can install from the binary packages provided:

```
sudo add-apt-repository ppa:srslte/releases
sudo apt-get update
sudo apt-get install srsepc
```

If you are using a different distribution, you can install from source using the guide provided in the project's GitHub page.

After installing the software you can install the configuration files into the default location (`~/.config/srsran`), by running:

```
srsran_install_configs.sh user
```

Running the software

To run srsEPC with default parameters, run `sudo srsepc` on the command line. srsEPC needs to run with sudo admin privileges in order to create a TUN device. This will start the EPC and it will wait for eNBs and UEs to connect to it.

srsEPC will start a TUN interface `srs_spgw_sgi` that will allow user-plane packets to reach the UEs.

Configuration

The EPC can be configured through two configuration files: `epc.conf` and `user_db.csv`. The `epc.conf` will hold general configuration parameters of the MME, SPGW and the HSS. This includes PLMN value, integrity/ciphering algorithms, APN, SGi IP address, GTP-U bind address, etc.

The `user_db.csv` is used to keep UE specific parameters for the HSS. This will include IMSI, authentication algorithms, K, OP or OPc, etc.

In the following subsections, we will cover a few common configuration cases with srsEPC: adding a new UE to the HSS database, running the eNB and EPC on separate machines, and setting up network routing to enable UEs to connect to the Internet.

Adding an UE to HSS database

When adding a UE to be able to the `user_db.csv` database that the HSS will use, you must make sure that that parameters in that file match the parameters stored in the UE's USIM card.

Of particular relevance are the IMSI, authentication algorithm, K and OP or OPc (if using the MILENAGE algorithm). The IMSI is the unique identifier of the SIM card, the K the secret key that the HSS and the UE use to authenticate each other.

The usual authentication algorithm used by SIM cards is MILENAGE, but there are also test SIMs that use XOR authentication. If you are using the MILENAGE algorithm, you must also know whether you are using OP or OPc and the corresponding value of this parameter.

Once you know these parameters you can replace then in the user_db.csv which has the following format:

```
(ue_name),(algo),(imsi),(K),(OP/OPc_type),(OP/OPc_value),(AMF),(SQN),(QCI),(IP_alloc)
```

So, if you have a SIM card with the following parameters:

- MILENAGE algorithm
- IMSI = 901700000000001
- K = 00112233445566778899aabbccddeeff
- Using OPc
- OPc = 63bfa50ee6523365ff14c1f45f88737d

You can configure the `user_db.csv` like this:

```
ue1,mil,901700000000001,00112233445566778899aabbccddeeff,opc,
↪63bfa50ee6523365ff14c1f45f88737d,9000,000000000000,9,dynamic
```

eNBs and srsEPC on separate machines

By default, srsEPC is configured to run with srsENB on the same machine. When running srsEPC with an eNB on a separate machine, all that is necessary to configure is the `mme_bind_addr` and the `gtpu_bind_addr`.

The MME bind address will specify where the MME will listen for eNB S1AP connections. The GTP-U bind address should be the same as the MME bind address, unless you want to run the user-plane on a different sub-net then the S1AP connection.

So if you want to listen to eNB on the interface with IP *10.0.1.10*, you can do:

```
sudo srsepc --mme.mme_bind_addr 10.0.1.10 --spgw.gtpu_bind_addr 10.0.1.10
```

Connecting UEs to the Internet

To allow UEs to connect to the Internet, it is necessary to perform IP masquerading. Without masquerading, the Linux kernel will not do packet forwarding from one subnet to another.

To enable this, you can run a convenience script `sudo srsepc_if_masq <out_interface>`, where *out_interface* is the interface that connects the PC to the Internet.

Warning: *out_interface* is NOT the *srs_spgw_sgi* interface, but the Ethernet or WiFi ethernet that connects the PC to the Internet.

Observing results

By default, log files are stored in `/tmp/epc.log`. This file can be inspected to troubleshoot any issues related to srsEPC. Log files can have multiple verbosity levels, which can be configured in the `epc.conf` or through the command line. They can also be enabled on a per-layer capacity, which is useful when troubleshooting a specific layer.

Troubleshooting

This section describes some of the most common issues with srsEPC and how to troubleshoot them.

UE did not attach

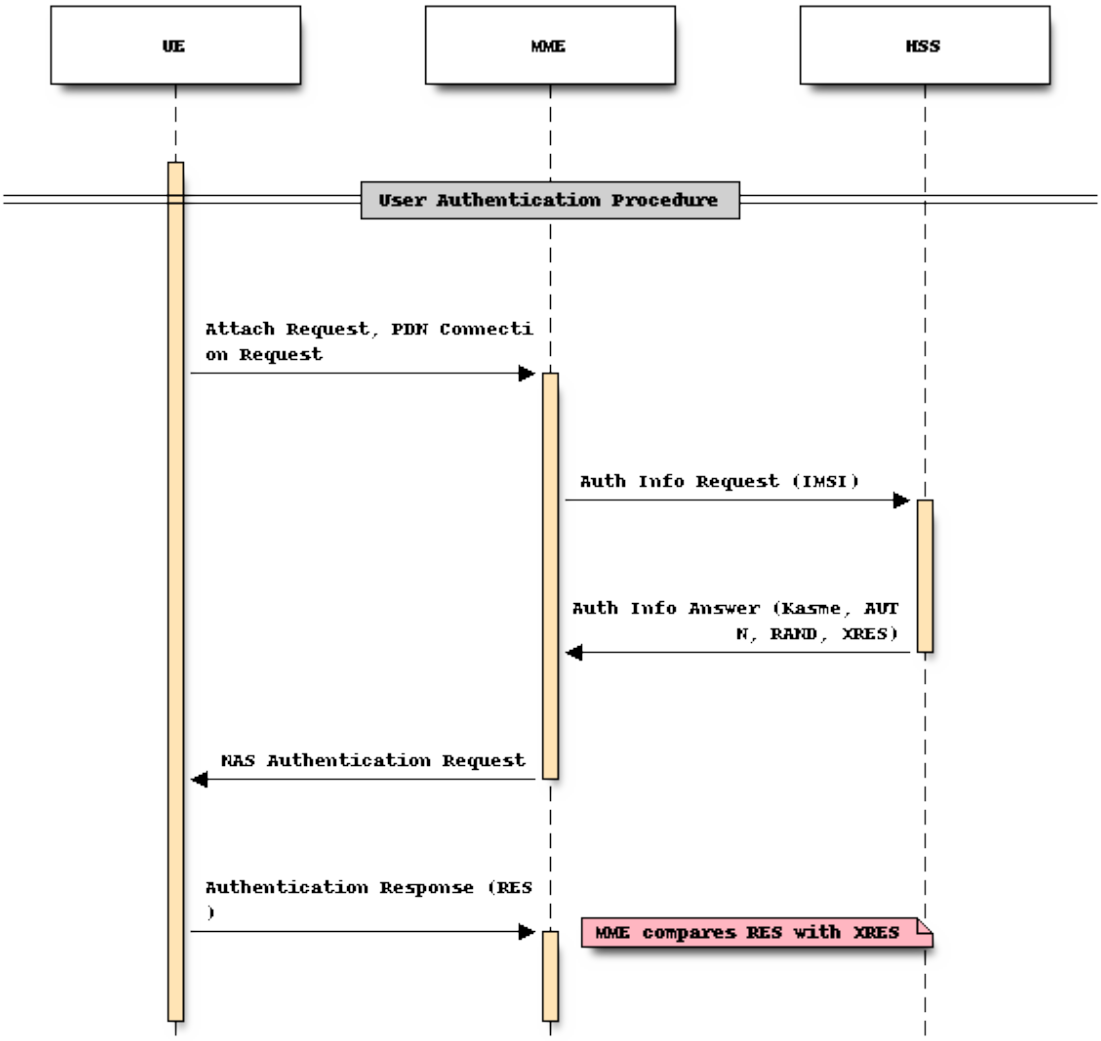
If the UE could not attach it is important to see at what point the attach procedure broke down. The easiest way to do this is to inspect the NAS messages on the EPC PCAP. See the *Observing results* section for instructions on how to obtain a PCAP from srsEPC.

The most common reasons for an attach failure are either an *Authentication failure* or a *Mismatched APN*. Some instructions on addressing these issues can be found on the subsections below.

Authentication failure

The most common case of attach failure is authentication failure. In LTE, not only the network must authenticate the UE, but the UE must also authenticate the network. For that reason, there is an authentication procedure within the attach procedure.

An simplified illustration of the messages involved in the authentication procedure can be found bellow:



If when the MME compares the RES and XRES and these values do not match, that means that the keys used to generate those values are different and authentication fails.

For authentication, there are four important parameters that must be configured correctly both at the UE and the HSS: the IMSI, the authentication algorithm, the UE key and OP/OPc. If you misconfigure your IMSI, you will see an *User not found. IMSI <Your_IMSI>* message in the epc.log. If you misconfigure the other parameters, you will see a "NAS Authentication Failure" message in the epc.pcap, with the failure code "MAC Code Failure."

Instructions on how to configure these parameters can be found in the *Adding an UE to HSS database* section.

Mismatched APN

Within the attach procedure, the UEs sends an APN setting, either in the "PDN connectivity request" message or in the "ESM information transfer" message. It is necessary that the configuration of the APN in the UE and the EPC match. Important parameters to check are the APN name, the PDN type (must be IPv4), and that no PAP/CHAP authentication is being used.

In srsUE you can configure these parameters in the NAS section of the ue.conf. If using a COTS UE, go to your APN settings and make sure that the APN configured in the UE matches the one configured in the EPC.

I cannot access the Internet

If the UE attached successfully and can ping the SPGW, that means that the attach procedure went well and that the UE was able to obtain the IP.

That means that not being able to access the Internet is a problem not with srsRAN, but with the network configuration of the system. The most likely issue is that, by default, Linux will not forward packets from one subnet to another. See the *Connecting UEs to the Internet* section on how to enable IP packet forwarding in Linux.

Advanced Usage

Configuration Reference

The srsEPC example configuration file contains detailed descriptions of all EPC configuration parameters.

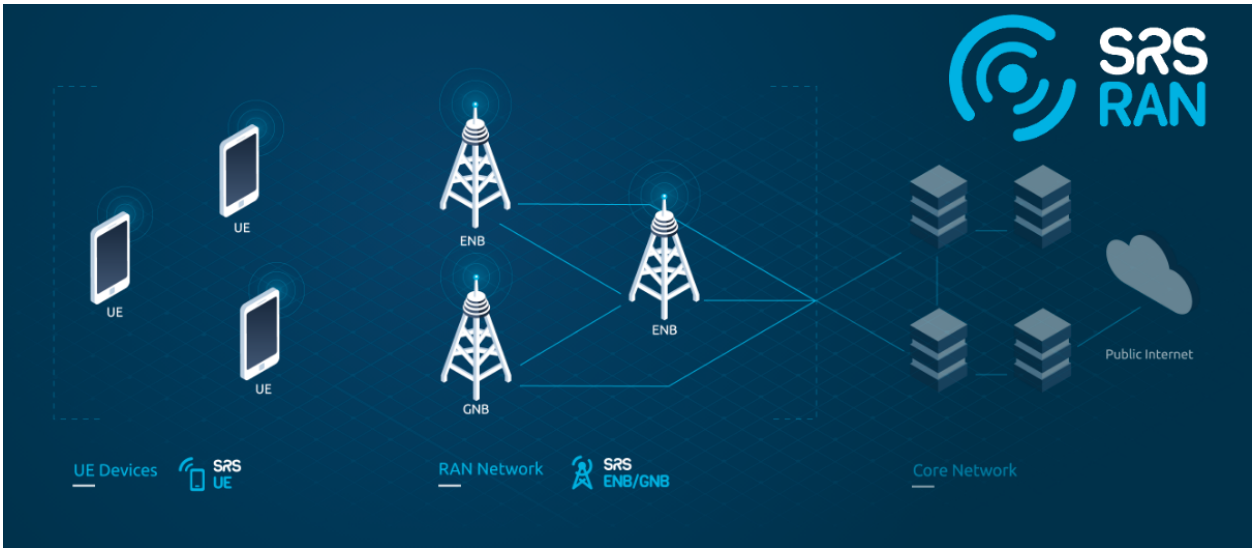
In addition to the top-level configuration file, srsEPC uses a separate file user_db.csv to store user details in the HSS. This user database file uses CSV format.

2.3 srsRAN Application Notes

srsRAN is a free and open-source 4G and 5G software radio suite.

Featuring both UE and eNodeB/gNodeB applications, srsRAN can be used with third-party core network solutions to build complete end-to-end mobile wireless networks. For more information, see www.srsran.com.

These application notes provide guides for specific srsRAN use-cases, using external applications, and guides on hardware choices and use.



Use srsRAN without RF hardware in the loop:

- *ZeroMQ Application Note*

Carrier Aggregation:

- *Carrier Aggregation Application Note*

Use eMBMS to support multicast/broadcast traffic using srsRAN:

- *eMBMS Application Note*

Use srsRAN to explore NB-IoT deployments:

- *NB-IoT Application Note*

Running srsRAN on the Raspberry Pi 4:

- *Raspberry Pi 4 Application Note*

Experiment with CV2X signalling with srsRAN:

- *CV2X Application Note*

Connect a COTS UE to srsRAN:

- *COTS UE Application Note*

Simulate Intra-eNB & S1 Handover using ZMQ:

- *Handover Application Note*

Set-up and test your first 5G NSA network:

- *5G NSA Application Note*

Suggested hardware packages for experimentation & development:

- *Suggested Hardware Packages*

2.3.1 ZeroMQ Application note

Introduction

srsRAN is a 4G and 5g software radio suite. The 4G network consists of a core network, an eNodeB, and a UE implementation. Usually eNodeB and UE are used with actual RF front-ends for over-the-air transmissions. There are, however, a number of use-cases for which RF front-ends might not be needed or wanted. Those use-cases include (but are not limited to) the use of srsRAN for educational or research purposes, continuous integration and delivery (CI/CD) or development and debugging.

With srsRAN this can be achieved by replacing the radio link between eNodeB and UE with a mechanism that allows to exchange baseband IQ samples over an alternative transport. For this purpose, we've implemented a ZeroMQ-based RF driver that essentially acts as a transmit and receive pipe for exchanging IQ samples over TCP or IPC.

ZeroMQ Installation

First thing is to install ZeroMQ and build srsRAN. On Ubuntu, ZeroMQ development libraries can be installed with:

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

Alternatively, installing from sources can also be done.

First, one needs to install libzmq:

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

Second, install czmq:

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

Finally, you need to compile srsRAN (assuming you have already installed all the required dependencies). Note, if you have already built and installed srsRAN prior to installing ZMQ and other dependencies you will have to re-run the make command to ensure srsRAN recognises the addition of ZMQ:

```
git clone https://github.com/srsRAN/srsRAN.git
cd srsRAN
mkdir build
cd build
cmake ../
make
```

Put extra attention in the cmake console output. Make sure you read the following line:

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

Running a full end-to-end LTE network on a single computer

Before launching the LTE network components on a single machine we need to make sure that both UE and EPC are in different network namespaces. This is because both EPC and UE will be sharing the same network configuration, i.e. routing tables etc. Because the UE receives an IP address from the EPC's subnet, the Linux kernel would bypass the TUN interfaces when routing traffic between both ends. Therefore, we create a separate network namespace (netns) that the UE uses to create its TUN interface in.

We only require TUN interfaces for the UE and EPC as they are the only IP endpoints in the network and need to communicate over the TCP/IP stack.

We will run each srsRAN application in a seperate terminal instance. Applications such as ping and iperf used to generate traffic will be run in separate terminals.

Note, the examples used here can be found in the following directory: `./srsRAN/build/`. With the UE, eNB and EPC then being called from their associated directory.

Network Namespace Creation

Let's start with creating a new network namespace called "ue1" for the (first) UE:

```
sudo ip netns add ue1
```

To verify the new "ue1" netns exists, run:

```
sudo ip netns list
```

Running the EPC

Now let's start the EPC. This will create a TUN device in the default network namespace and therefore needs root permissions.

```
sudo ./srsepc/src/srsepc
```

Running the eNodeB

Let's now launch the eNodeB. We use the default configuration in this example and pass all parameters that need to be tweaked for ZMQ through as command line arguments. If you want to make those persistent just add them to your local enb.conf. The eNB can be launched without root permissions.

```
./srsenb/src/srsenb --rf.device_name=zmq --rf.device_args="fail_on_disconnect=true,tx_
↪port=tcp://*:2000,rx_port=tcp://localhost:2001,id=enb,base_srate=23.04e6"
```

Running the UE

Lastly we can launch the UE, again with root permissions to create the TUN device.

```
sudo ./srsue/src/srsue --rf.device_name=zmq --rf.device_args="tx_port=tcp://*:2001,rx_
↪port=tcp://localhost:2000,id=ue,base_srate=23.04e6" --gw.netns=ue1
```

The last command should start the UE and attach it to the core network. The UE will be assigned an IP address in the configured range (e.g. 172.16.0.2).

Traffic Generation

To exchange traffic in the downlink direction, i.e. from the the EPC, just run ping or iperf as usual on the command line, e.g.:

```
ping 172.16.0.2
```

In order to generate traffic in the uplink direction it is important to run the ping command in the UE's network namespace.

```
sudo ip netns exec ue1 ping 172.16.0.1
```

Namespace Deletion

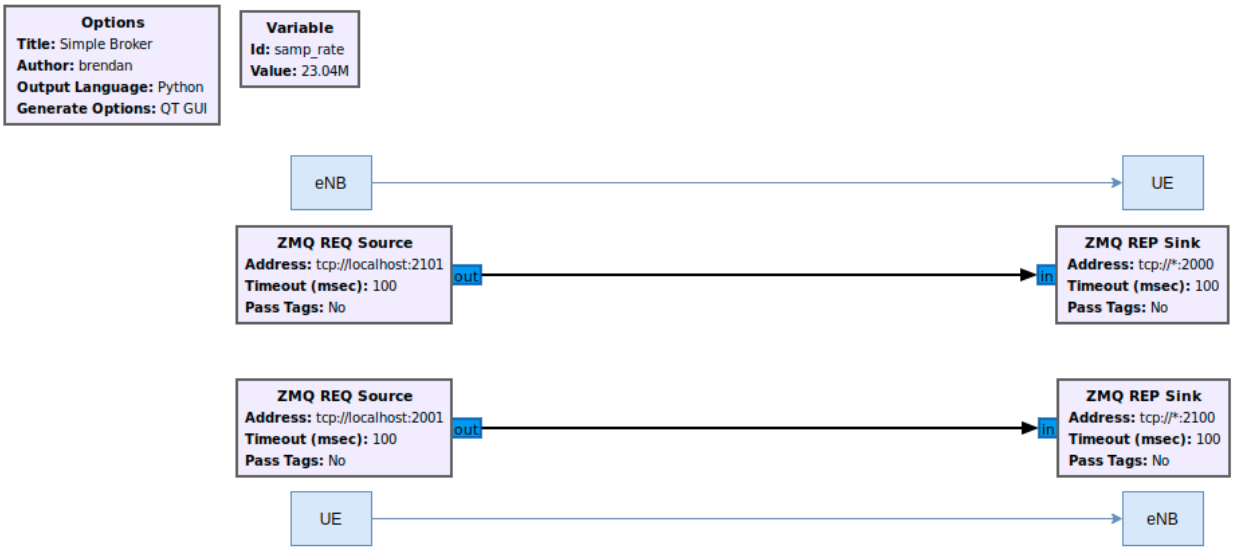
After finishing, make sure to remove the netns again.

```
sudo ip netns delete ue1
```

GNU-Radio Companion Integration

GNU-Radio Companion can be easily integrated with a ZMQ based instance of srsRAN. This can be used to manipulate, and/ or visualise baseband I/Q data as it is sent between the UE and eNB. It does this by using the ZMQ-compatible blocks within GRC connected to the TCP ports used to transmit data between the two network elements. Data going both from the UE to the eNB, and from the eNB to the UE can be handled via a GRC Broker.

The following figure shows a basic GRC Broker:



The above figure shows how the broker acts as a man-in-the-middle between the UE and the eNB. The blue boxes and arrows show the direction of data between the network elements. The following ports are used in this example:

Table 2: Ports Used

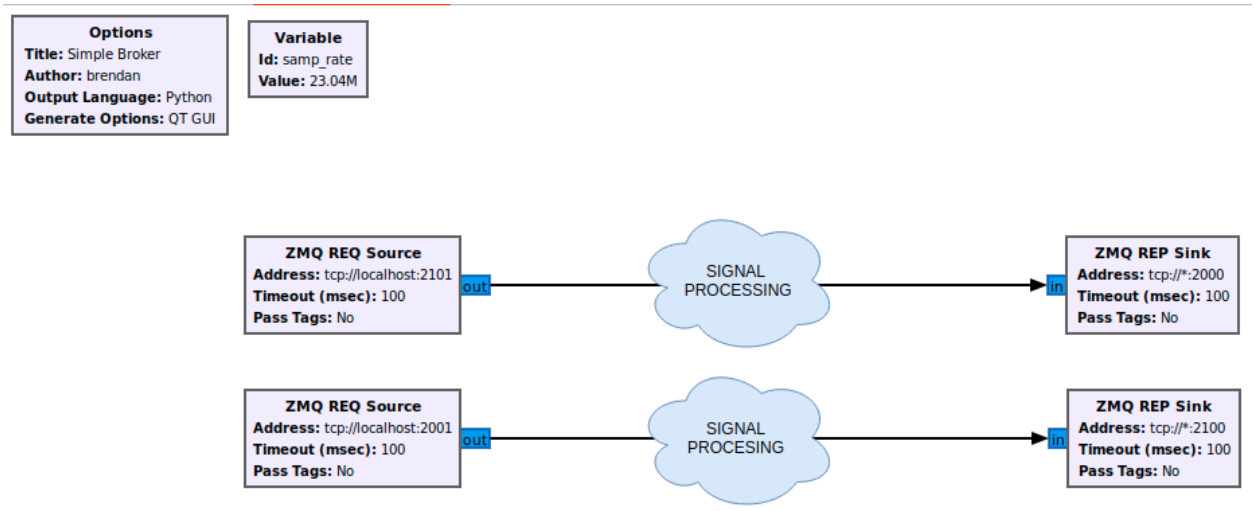
Port Direction	srsUE	srsENB
TX	2001	2101
Rx	2000	2100

Building on this simple example, the I/Q data sent between elements can be processed, manipulated and/ or visualised as needed. This would lead to a GRC architecture simliar to what is shown in the following figure.

The signal processing clouds between elements here represent where any processing of the data would take place.

When running an E2E Network with a Broker between elements the following steps must be taken when spinning up the network:

1. Start up the EPC
2. Start the eNB using ZMQ
3. Start the UE using ZMQ



4. Run the GRC Flowgraph associated with the broker.

Note, the UE will not connect to the eNB until the broker has been started, as the UL and DL channels are not directly connected between the UE and eNB. You will also need to restart the GRC Broker each time the network is restarted.

Known issues

- For a clean tear down, the UE needs to be terminated first, then the eNB.
- eNB and UE can only run once, after the UE has been detached, the eNB needs to be restarted.
- We currently only support a single eNB and a single UE.

2.3.2 Carrier Aggregation Application note

Introduction

Before getting hands-on we recommend reading about Carrier Aggregation.

The srsRAN software suite supports 2-carrier aggregation in both srsENB and srsUE. To experiment with carrier aggregation using srsRAN over-the-air, you will need an RF device that can tune different frequencies in different channels, for example the USRP X300 series from Ettus Research (NI). We've tested with UHD 3.15 LTS and UHD 4.0.

Alternatively, experiment with carrier aggregation without SDR hardware using our ZeroMQ-based RF layer emulation. See our *ZeroMQ Application Note* for more information about RF layer emulation.

Carrier Aggregation using SDR Hardware

eNodeB Configuration

To configure the eNodeB for carrier aggregation, we must first configure the RF front-end. We must then configure srsENB for multiple cells and define the primary/secondary relationships between them.

If you're using a real RF device such as the USRP X300 it's advisable to use an external clock reference, either using the 10 MHz/1 PPS input (`clock=external`) or the GPSDO (`clock=gpsdo`). For the X300, especially for newer UHD versions, it's also required to specific the sample rate upon radio initialization. For example, if you're planning to use 10 MHz cells (50 PRB) the sample rate of the radio will be 11.52 Msamples/s, hence a `sampling_rate=11.52e6` shall be used. For 20 MHz cells (100 PRB) the sample rate will be 23.04 Msamples/s, hence `sampling_rate=23.04e6` shall be used.

```
[rf]
device_name = uhd
device_args = type=x300,clock=external,sampling_rate=23.04e6
```

The second step is to configure srsENB with two cells. For this, one needs to modify `rr.conf`:

```
cell_list =
(
  {
    rf_port = 0;
    cell_id = 0x01;
    tac = 0x0001;
    pci = 1;
    root_seq_idx = 204;
    dl_earfcn = 2850;

    // CA cells
    scell_list = (
      {cell_id = 0x02; cross_carrier_scheduling = false; scheduling_cell_id = 0x01; ul_
↪allowed = true}
    )
  },
  {
    rf_port = 1;
    cell_id = 0x02;
    tac = 0x0001;
    pci = 4;
    root_seq_idx = 205;
    dl_earfcn = 3050;

    // CA cells
    scell_list = (
      {cell_id = 0x01; cross_carrier_scheduling = false; scheduling_cell_id = 0x02; ul_
↪allowed = true}
    )
  }
)
```

With these changes, simply run srsENB as usual.

UE Configuration

In the UE, we must again set the RF configuration and configure the UE capabilities.

For the RF configuration, we need to set the list of EARFCNs according to the cells configured in the eNodeB and set the number of carriers to 2:

```
[rf]
dl_earfcn = 2850,3050
nof_carriers = 2
```

Adding more EARFCNs in the list makes the UE scan these frequencies and the number of carriers makes the UE use more RF channels.

For the UE capabilities, we need to report at least release 10 and category 7:

```
[rrc]
ue_category         = 7
ue_category_dl      = 10
```

With these changes, simply run srsUE as usual.

Carrier Aggregation using ZeroMQ RF emulation

To experiment with carrier aggregation using the ZeroMQ RF emulation instead of SDR hardware, we simply need to configure srsENB and srsUE to use the `zmq` RF device.

eNodeB Configuration

For srsENB, configure the `zmq` RF device as follows:

```
[rf]
device_name = zmq
device_args = fail_on_disconnect=true,id=enb,tx_port0=tcp://*:2000,tx_port1=tcp://*:2002,
↪rx_port0=tcp://localhost:2001,rx_port1=tcp://localhost:2003
```

UE Configuration

For srsUE, configure the `zmq` RF device as follows:

```
[rf]
device_name = zmq
device_args = tx_port0=tcp://*:2001,tx_port1=tcp://*:2003,rx_port0=tcp://localhost:2000,
↪rx_port1=tcp://localhost:2002,id=ue,tx_freq0=2510e6,tx_freq1=2530e6,rx_freq0=2630e6,rx_
↪freq1=2650e6
```

Since the ZMQ module is frequency agnostic, it is important that Tx and Rx frequencies are set in ZMQ config. This makes internal carrier switching possible.

Known issues

- The eNodeB ignores UE's band capabilities
- CPU hungry and real time errors for more than 10 MHz

2.3.3 eMBMS Application note

Introduction

enhanced Multimedia Broadcast Multicast Services (eMBMS) is the broadcast mode of LTE. Using eMBMS, an eNodeB can efficiently broadcast the same data to all users attached to the cell. srsRAN supports eMBMS in the end-to-end system including srsUE, srsENB and srsEPC. In addition to these, a new application is introduced - srsMBMS. srsMBMS is the SRS MBMS gateway, an additional network component which receives multicast data on a TUN virtual network interface and provides it to the eMBMS bearer in the eNodeB.

Setup

To run an end-to-end srsRAN system with eMBMS, some additional configuration of the srsENB and srsUE applications are required. In the sample configurations provided, it is assumed that srsmbms, srsepc and srsenb run on the same physical machine.

srsENB configuration

At the eNodeB, additional configuration is required in order to support eMBMS transmission. First, instead of using the default `sib.conf.example`, the alternative `sib.conf.mbsfn.example` should be used. This version of the sib configuration adds eMBMS parameters to SIB2 and includes SIB 13 which is specific to eMBMS. These SIB modifications define the following key eMBMS network parameters:

- eMBMS Subframe Allocation
- MCCH Scheduling Period
- MCCH Modulation Order
- Non-eMBMS Subframe Region Length
- eMBMS Area Id
- MCCH Subframe Allocation
- MCCH Repetition Period

In addition to using the eMBMS SIB configuration file, a number of further configurations must be changed in the `enb.conf`:

```
[enb_files]
sib_config = sib.conf.mbsfn

[embms]
enable = true

[scheduler]
min_nof_ctrl_symbols = 2
max_nof_ctrl_symbols = 2
```

(continues on next page)

(continued from previous page)

```
[expert]
nof_phy_threads = 2
```

Once these setting adjustments have been made, the eNodeB should be ready to run in eMBMS mode.

srsUE configuration

For the UE, the presence of an eMBMS transmission will be automatically detected from the SIBs and the MCCH present in the downlink signal. To receive an active eMBMS service, the following parameter must be set in `ue.conf`:

```
[rrc]
mbms_service_id = 0
```

Note this service id must match the service id in use by the network.

In addition, we recommend the following settings for best performance with eMBMS:

```
[phy]
interpolate_subframe_enabled = true
snr_estim_alg = empty
nof_phy_threads = 2
```

Once these configurations have been made, your UE should be ready to run eMBMS.

Usage

First, run srsMBMS (the MBMS gateway), srsEPC and srsENB on the same machine:

```
sudo ./srsmbms ~/.config/srsran/mbms.conf
```

```
sudo ./srsepc ~/.config/srsran/epc.conf
```

```
sudo ./srsenb ~/.config/srsran/enb.conf
```

The MBMS gateway will create a TUN interface to which all traffic intended for multicast should be written. It will then forward this traffic to the eNodeB via a seperate GTPU tunnel that is dedicated to eMBMS traffic.

To test eMBMS with srsMBMS, srsEPC and srsENB, we can use Iperf. At the MBMS gateway, create a route and start downlink traffic:

```
sudo route add -net 239.255.1.0 netmask 255.255.255.0 dev sgi_mb
```

```
iperf -u -c 239.255.1.1 -b 10M -T 64 -t 60
```

Next, we can run srsUE on a separate machine to receive the eMBMS data:

```
sudo ./srsue ~/.config/srsran/ue.conf
```

Upon running srsUE with an eMBMS enabled eNodeB you should see the following output at the terminal of the UE:

```
Searching cell in DL EARFCN=3400, f_dl=2685.0 MHz, f_ul=2565.0 MHz
Found Cell:  Mode=FDD, PCI=1, PRB=50, Ports=1, CFO=-0.0 KHz
Found PLMN:  Id=00101, TAC=7
Random Access Transmission: seq=20, ra-rnti=0x2
```

(continues on next page)

(continued from previous page)

```
Random Access Complete.     c-rnti=0x46, ta=1
RRC Connected
MBMS service started. Service id:0, port: 4321
Network attach successful. IP: 172.16.0.2
Software Radio Systems LTE (srsRAN)
```

the *MBMS service started. Service id:0, port: 4321* indicates the eMBMS service has successfully started.

To receive the multicast iperf data, add a route to the UE and start an iperf server:

```
sudo route add -net 239.255.1.0 netmask 255.255.255.0 dev tun_srsue
```

```
iperf -s -u -B 239.255.1.1 -i 1
```

2.3.4 NB-IoT Application note

Introduction

Narrowband Internet of Things (NB-IoT) is the 3GPP alternative to other Low Power Wide Area Network (LPWAN) technologies, such as SigFox and LoRa. Technically it uses similar ideas and reuses some of the components of LTE. But the bandwidth is significantly reduced to a single PRB (180 kHz) in order to achieve the low-complexity, low-cost, long battery life requirements. It was first standardized in Release 13.

This application note shows how to spot and decode commercial NB-IoT transmissions in the first part. The second part shows how to transmit and receive your own NB-IoT downlink signal.

Requirements

The NB-IoT examples require a radio that can sample at 1.92 Msps. Since the bandwidth of an NB-IoT carrier is very small, even very cheaply available devices are sufficient to receive and decode the signal. For example, popular RTL-SDR USB dongles available for around 15-20 Euro are fine for decoding the signal.

The eNB transmitter example requires a radio with transmitting capabilities. For example, an Ettus B200mini can be used as the eNB transmitter and an RTL-SDR as UE receiver. In principle, any device supported by either UHD or SoapySDR should work.

The following application also supports srsGUI for real time visualization of data.

All of the examples used here can be found in the following directory: `./srsRAN/build/lib/examples`

Spotting local NB-IoT deployments

Most NB-IoT deployments can be found in the sub-GHz bands. In Europe especially band 20 (Downlink 791-821 MHz). To run a NB-IoT cell search on band 20 one can simply run:

```
$ ./lib/examples/cell_search_nbiot -b 20
Opening RF device...
[INFO] [UHD] linux; GNU C++ version 8.3.0; Boost_106700; UHD_3.13.1.0-3build1
[INFO] [LOGGING] Fastpath logging disabled at runtime.
Opening USRP channels=1, args: type=b200,master_clock_rate=23.04e6
[INFO] [B200] Detected Device: B200mini
[INFO] [B200] Operating over USB 3.
[INFO] [B200] Initialize CODEC control...
```

(continues on next page)

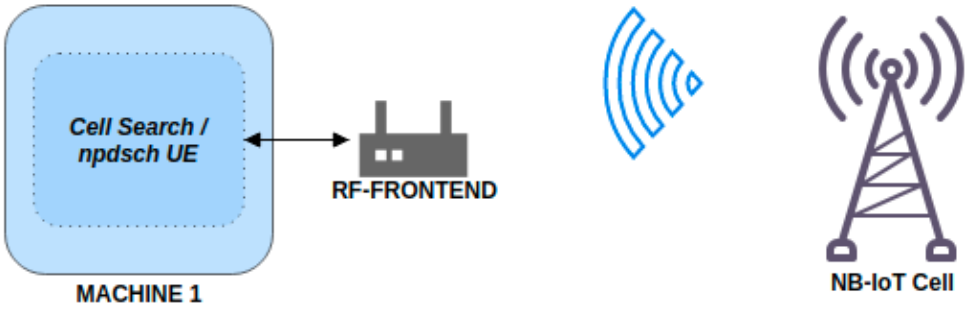


Fig. 5: Basic system architecture required to perform a cell search and decode transmissions.

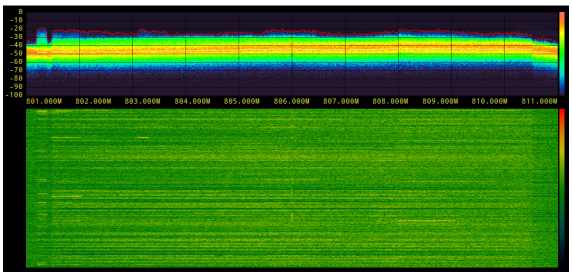
(continued from previous page)

```
[INFO] [B200] Initialize Radio control...
[INFO] [B200] Performing register loopback test...
[INFO] [B200] Register loopback test passed
[INFO] [B200] Asking for clock rate 23.040000 MHz...
[INFO] [B200] Actually got clock rate 23.040000 MHz.
[  0/299]: EARFCN 6150, 791.00 MHz looking for NPSS.
[  1/299]: EARFCN 6151, 791.10 MHz looking for NPSS.
[  2/299]: EARFCN 6152, 791.20 MHz looking for NPSS.
...
[105/299]: EARFCN 6253, 801.30 MHz looking for NPSS.
NSSS with peak=95.885849, cell-id: 257, partial SFN: 0
Found CELL ID 257.
...
[295/299]: EARFCN 6445, 820.50 MHz looking for NPSS.
[296/299]: EARFCN 6446, 820.60 MHz looking for NPSS.
[297/299]: EARFCN 6447, 820.70 MHz looking for NPSS.
[298/299]: EARFCN 6448, 820.80 MHz looking for NPSS.

Found 1 cells
Found CELL 801.3 MHz, EARFCN=6253, PHYID=257, NPSS power=31.0 dBm

Bye
```

In this example, we've found a NB-IoT carrier at 801.3 MHz. We can now use the *npdsch_ue* example (see next section) to decode the transmission.

It's also possible to just have a look at the spectrum and check for an NB-IoT carrier there. Most of the time the carrier is clearly visible, it's close to a LTE carrier in most cases and usually even a bit stronger than the LTE signal itself.

The example below, shows a 10 MHz Downlink LTE signal at 806 MHz. One can spot the NB-IoT carrier on the left hand side (the guardband) of the LTE spectrum.



The table below shows some examples of known NB-IoT deployments in Europe.

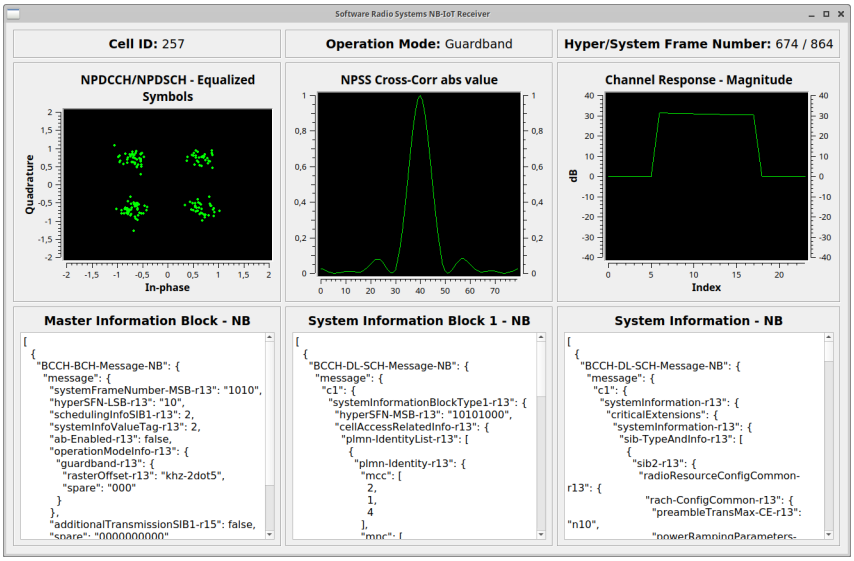
Country	Operator	EARFCN	Frequency (MHz)
Spain	Vodafone	6253	801.3
Germany	Vodafone	6346	810.6
Ireland	Vodafone	6354	811.4

Decoding the NB-IoT transmission

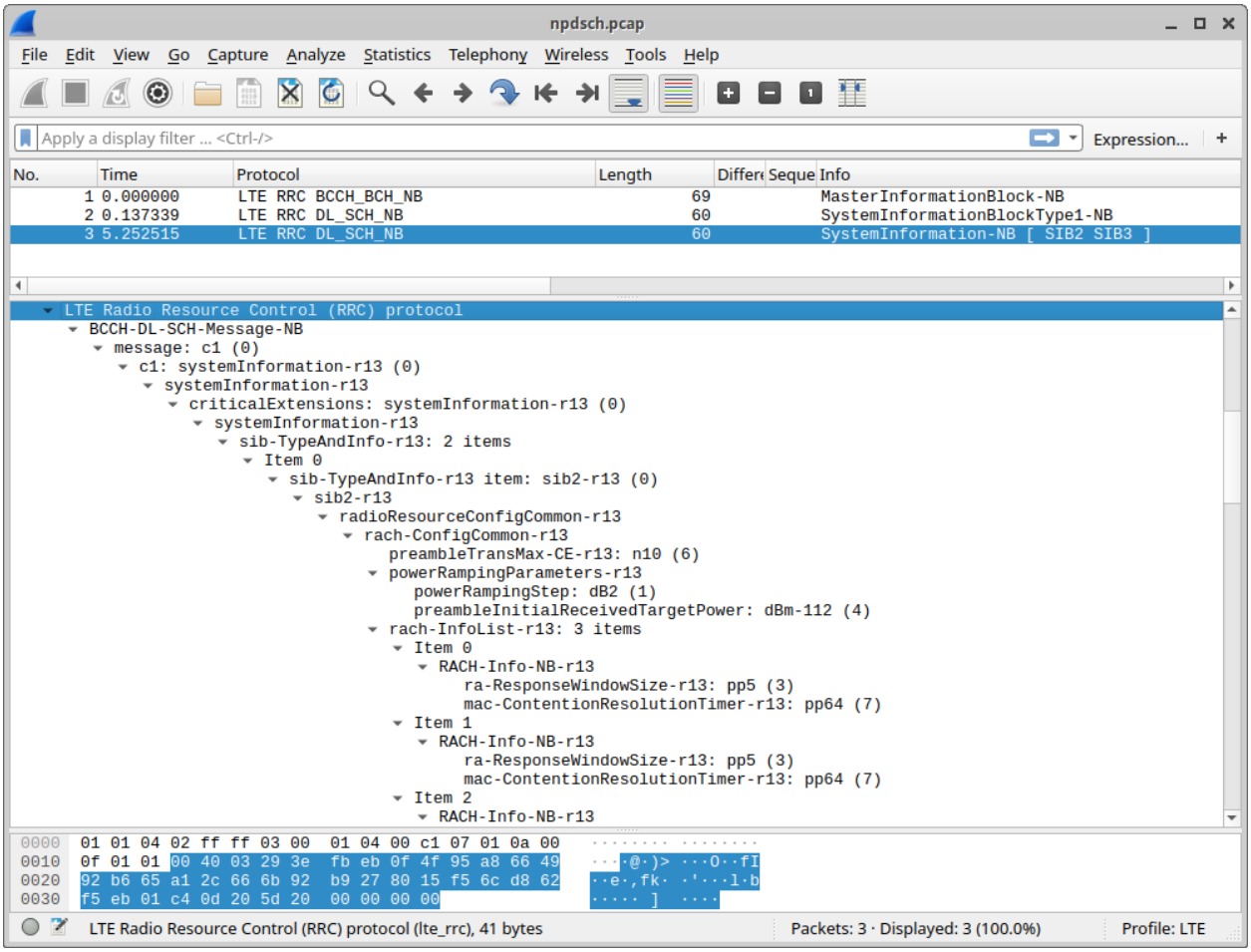
Once we've found the downlink frequency of an NB-IoT carrier, we can use the *npdsch_ue* example to decode the signal. The application should synchronize on the carrier, detect the cell ID and start to decode MIB, SIB and SIB2.

```
$ ./lib/examples/npdsch_ue -f 801.3e6
Opening RF device...
Soapy has found device #0: driver=rtlsdr, label=Generic RTL2832U OEM :: 00000001,␣
↪manufacturer=Realtek, product=RTL2838UHIDIR, serial=00000001, tuner=Rafael Micro R820T,
Selecting Soapy device: 0
..
Set RX freq: 801.300000 MHz
Setting sampling rate 1.92 MHz
NSSS with peak=65.811836, cell-id: 257, partial SFN: 0
*Found n_id_ncell: 257 DetectRatio= 0% PSR=10.57, Power=111.7 dBm
MIB received (CFO:  -2,82 kHz) FrameCnt: 0, State: 10
SIB1 received
SIB2 received
CFO:  -2,76 kHz, RSRP: 28,0 dBm SNR:  5,0 dB, RSRQ: -11,5 dB, NPDCCH detected: 0, NPDSCH-
↪BLER:  0,00% (0 of total 2), NPDSCH-Rate:  0,10
..
```

If you've compiled srsRAN with GUI support you should see something like this on your screen.



You can stop the UE decoder with Ctrl+C. Upon exit, the application writes a PCAP file of the decoded signal to */tmp/npdsch.pcap*. This file can be inspected with Wireshark. The screenshot below shows Wireshark decoding the received signal.



Transmit and Receive Downlink Signal

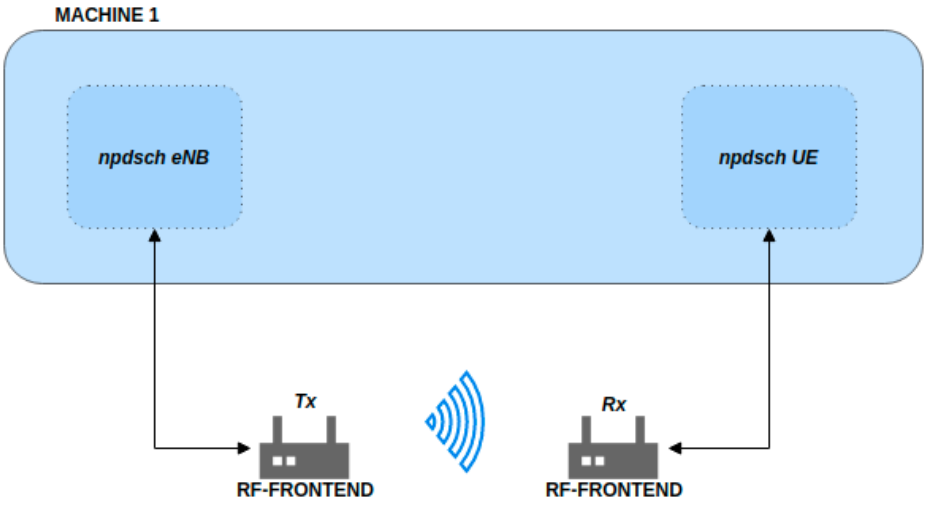


Fig. 6: Basic system architecture required to transmit and recieve downlink signal.

In this part of the tutorial we will show how we can use the provided example applications to transmit and receive our own NB-IoT signal. Please note that you should only do that in a cabled setup or Faraday cage in order to comply with emission rules of your country.

Please check that the RF-frontend hardware you are using meets the *requirements* previously outlined.

To start the eNB example, simply execute the command shown below. This will launch the eNB which by default picks the first available RF device and transmits the signal. With the *-o* option the signal can also be written to file for offline processing.

```
$ ./lib/examples/npdsch_enodeb -f 868e6
Opening RF device...
[INFO] [UHD] linux; GNU C++ version 8.3.0; Boost_106700; UHD_3.13.1.0-3build1
[INFO] [LOGGING] Fastpath logging disabled at runtime.
[INFO] [B200] Loading firmware image: /usr/share/uhd/images/usrp_b200_fw.hex...
Opening USRP channels=1, args: type=b200,master_clock_rate=23.04e6
[INFO] [B200] Detected Device: B200mini
[INFO] [B200] Loading FPGA image: /usr/share/uhd/images/usrp_b200mini_fpga.bin...
[INFO] [B200] Operating over USB 3.
[INFO] [B200] Initialize CODEC control...
[INFO] [B200] Initialize Radio control...
[INFO] [B200] Performing register loopback test...
[INFO] [B200] Register loopback test passed
[INFO] [B200] Asking for clock rate 23.040000 MHz...
[INFO] [B200] Actually got clock rate 23.040000 MHz.
Setting sampling rate 1.92 MHz
Set TX gain: 70.0 dB
Set TX freq: 868.00 MHz
NB-IoT DL DCI:
 - Format flag:                        1
   + FormatN1 DCI:                     Downlink
```

(continues on next page)

(continued from previous page)

```
 - PDCCH Order:                          0
 - Scheduling delay:                     0 (0 subframes)
 - Resource assignment:                  0
   + Number of subframes:                1
 - Modulation and coding scheme index:   1
 - Repetition number:                    0
   + Number of repetitions:              1
 - New data indicator:                   0
 - HARQ-ACK resource:                    1
 - DCI subframe repetition number:       0
DL grant config:
 - Number of subframes:                  1
 - Number of repetitions:                1
 - Total number of subframes:            1
 - Starting SFN:                         0
 - Starting SF index:                    6
 - Modulation type:                      QPSK
 - Transport block size:                 24
```

The eNB example will transmit a standard-compliant downlink signal with MIB-NB and SIB1-NB. It does not transmit SIB2 though. In all empty downlink subframes not used for MIB or SIB transmissions it does transmit a NPDSCH signal for test purposes to RNTI 0x1234. One can modify the transport block size of the test transmission by typing the MCS value (e.g. 20) on the eNB console and hitting Enter.

This test transmission can be decoded with the UE example. For this, we need to run the UE example by telling it to decode RNTI 0x1234 and skip SIB2 decoding (because it's not transmitted by eNB):

```
$ ./lib/examples/npdsch_ue -f 868e6 -r 0x1234 -s
Opening RF device...
Found Rafael Micro R820T tuner
Soapy has found device #0: driver=rtlsdr, label=Generic RTL2832U OEM :: 00000001,␣
↪manufacturer=Realtek, product=RTL2838UHIDIR, serial=00000001, tuner=Rafael Micro R820T,
Selecting Soapy device: 0
[INFO] Opening Generic RTL2832U OEM :: 00000001...
Found Rafael Micro R820T tuner
Setting up Rx stream with 1 channel(s)
[INFO] Using format CF32.
[R82XX] PLL not locked!
Available device sensors:
Available sensors for Rx channel 0:
State of gain elements for Rx channel 0 (AGC supported):
 - TUNER: 0.00 dB
State of gain elements for Tx channel 0 (AGC supported):
 - TUNER: 0.00 dB
Rx antenna set to RX
Tx antenna set to RX
Set RX gain: 40.0 dB
Set RX freq: 868.000000 MHz
Setting sampling rate 1.92 MHz
NSSS with peak=24.363365, cell-id: 0, partial SFN: 0
*Found n_id_ncell:   0 DetectRatio= 0% PSR=8.66, Power=86.4 dBm
MIB received (CFO:  -1,55 kHz) FrameCnt: 0, State: 10
SIB1 received
```

(continues on next page)

(continued from previous page)

```
CFO:  -1,41 kHz, RSRP: 12,0 dBm SNR: 19,0 dB, RSRQ: -3,7 dB, NPDCCH detected: 510,␣
↪NPDSCH-BLER:  0,20% (1 of total 511), NPDSCH-Rate: 10,36 kbit/s
```

The outlook should look similar except that no SIB2 is decoded. If you've compiled with GUI support you should again see a similar application like above. Please note the constellation diagram is updated a lot more frequently because now all NPDSCH transmissions to the test user are also received.

Known issues

- Cell ID detection isn't reliable.

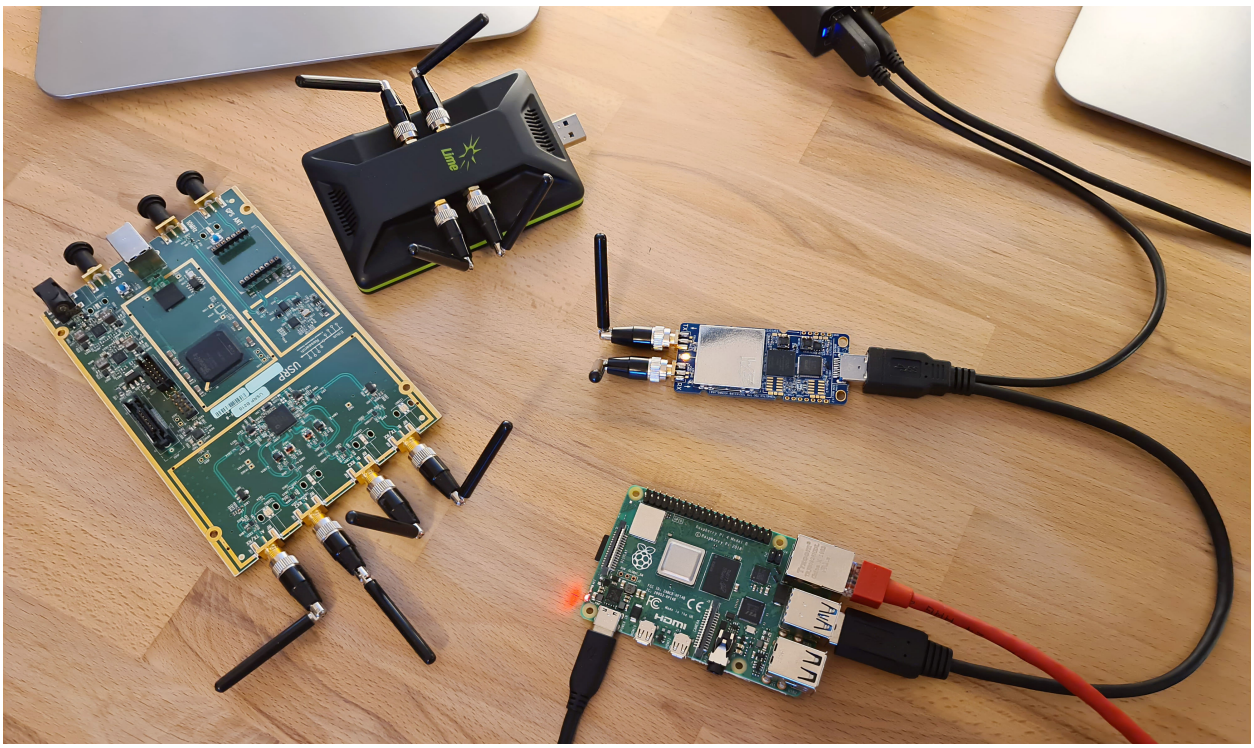
  In some cases the cell ID detection using the NSSS signal isn't working reliably. In case the *npdsch_ue* application clearly synchronizes to the downlink signal (you see a strong correlation peak in the middle graph in the GUI) but the MIB is never decoded, it is very likely that the cell ID wasn't detected correctly. In this case, try to restart the application again and see if the cell ID can be detected. If the problem still persists, one can also try to set the cell ID manually with the *-l* parameter. For this you need to first figure out the correct value, which sometimes can be done by decoding the default LTE carrier with *pdsch_ue* and use the same cell ID for the NB-IoT carrier.

2.3.5 Raspberry Pi 4 Application note

Introduction

srsRAN is a 4G and 5G software radio suite. The 4G LTE systems includes a core network and an eNodeB. Most people in the srsRAN community run the software on high performance computers, however the eNodeB can also be run on the low power Raspberry Pi 4 with a variety of SDRs.

The concept of an ultra low cost, low power and open source SDR LTE femtocell has a lot of people excited!



Pi4 eNodeB Hardware Requirements

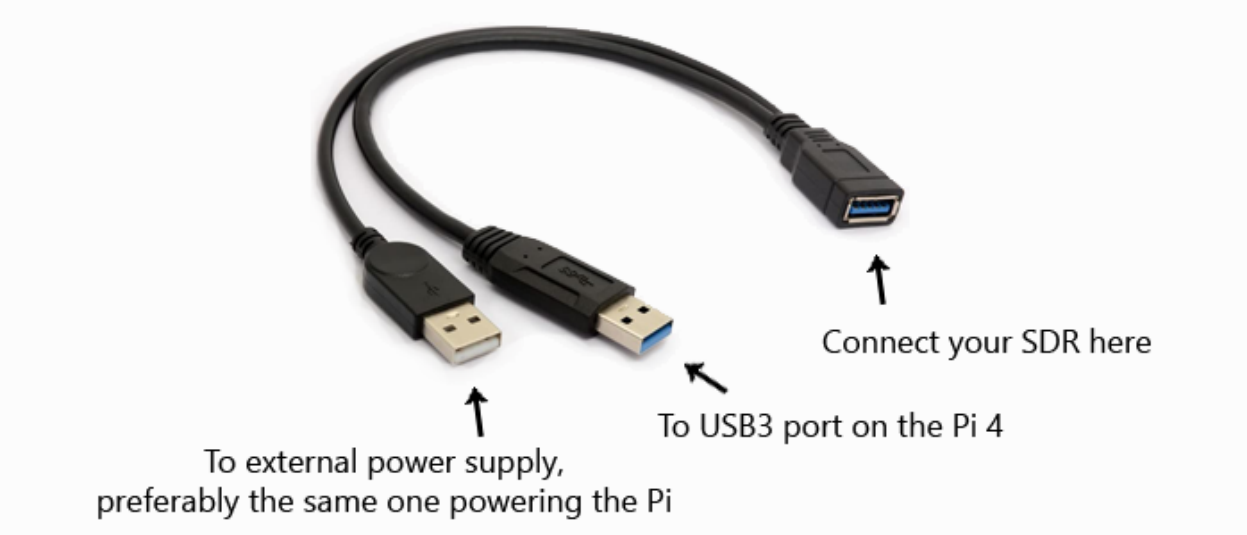
The setup instructions provided below have been tested with a **Raspberry Pi 4B /4GB rev 1.2** running the Ubuntu Server 20.04 LTS aarch64 image. It has not been tested with the rev 1.1 board, boards with 2GB of RAM or alternative operating systems. The Ubuntu image can be downloaded from the official Ubuntu website. You can visually identify your Pi4 hardware revision – this doc from Cytron shows you how.

This setup has been tested with a USRP B210, a LimeSDR-USB and a LimeSDR-Mini.

Note: When using the USRP B210, you can create a 2x2 MIMO cell with srsenb. It is also possible to run the srsepc core network on the Pi too.

When using either of the LimeSDRs, you can only create a 1x1 SISO cell with srsenb. The core network must be run on a separate device.

Due to the power requirements of the SDRs, you must use an external power source. This can be achieved with a 'Y' cable, such as this:



Software Setup

First thing is to install the SDR drivers and build srsRAN. UHD drivers are required for USRPs, SoapySDR/LimeSuite are required for the LimeSDRs.

```
sudo apt update
sudo apt upgrade
sudo apt install cmake
```

UHD Drivers can be installed with:

```
sudo apt install libuhd-dev libuhd3.15.0 uhd-host
sudo /usr/lib/uhd/utils/uhd_images_downloader.py

## Then test the connection by typing:
sudo uhd_usrp_probe
```

SoapySDR and **LimeSuite** can be installed with:

```
git clone https://github.com/pothosware/SoapySDR.git
cd SoapySDR
git checkout tags/soapy-sdr-0.7.2
mkdir build && cd build
cmake ..
make -j4
sudo make install
sudo ldconfig
```

```
sudo apt install libusb-1.0-0-dev
git clone https://github.com/myriadrf/LimeSuite.git
cd LimeSuite
git checkout tags/v20.01.0
mkdir builddir && cd builddir
cmake ../
make -j4
sudo make install
sudo ldconfig
cd ..
cd udev-rules
sudo ./install.sh

## Then test the connection by typing:
LimeUtil --find
LimeUtil --update
SoapySDRUtil --find
```

Next, **srsRAN** can be compiled:

```
sudo apt install libfftw3-dev libmbedtls-dev libboost-program-options-dev libconfig++-
↪dev libsctp-dev
git clone https://github.com/srsRAN/srsRAN.git
cd srsran
git checkout tags/release_19_12
mkdir build && cd build
cmake ../
make -j4
sudo make install
sudo ldconfig

## copy configs to /root
sudo ./srsran_install_configs.sh user
```

And finally, modify the **Pi CPU scaling_governor** to ensure it is running in performance mode:

```
sudo systemctl disable ondemand
sudo apt install linux-tools-raspi

sudo nano /etc/default/cpufrequtils
* insert:
* GOVERNOR="performance"

## reboot
```

(continues on next page)

(continued from previous page)

```
sudo cpupower frequency-info
* should show that the CPU is running in performance mode, at maxiumum clock speed
```

Pi4 eNodeB Config

During testing, the following eNodeB config options have been shown to be stable for 24hr+ when running with the USRP B210, and stable for 2hr+ when running with the LimeSDRs, so should be a good starting point for you.

The Pi4 eNodeB has been tested with a 3MHz wide cell in LTE B3 (1800MHz band), DL=1878.40 UL=1783.40. This sits inside the UK's new "1800MHz shared access band", for which you can legally obtain a low cost, low power shared access spectrum licence from Ofcom if you are working in the UK.

Changes to default enb.conf for **USRP B210**:

```
sudo nano /root/.config/srsran/enb.conf

[enb]
mcc = <yourMCC>
mnc = <yourMNC>
mme_addr = 127.0.1.100     ## or IP for external MME, eg. 192.168.1.10
gtp_bind_addr = 127.0.1.1  ## or local interface IP for external S1-U, eg. 192.168.1.3
s1c_bind_addr = 127.0.1.1  ## or local interface IP for external S1-MME, eg. 192.168.1.3
n_prb = 15
tm = 2
nof_ports = 2

[rf]
dl_earfcn = 1934
tx_gain = 80               ## this power seems to work best
rx_gain = 40
device_name = UHD
device_args = auto         ## does not work with anything other than 'auto'
```

Changes to default enb.conf for **LimeSDR-USB or LimeSDR-Mini**:

```
sudo nano /root/.config/srsran/enb.conf

[enb]
mcc = <yourMCC>
mnc = <yourMNC>
mme_addr = <ipaddr>        ## IP for external MME, eg. 192.168.1.10
gtp_bind_addr = <ipaddr>   ## local interface IP for external S1-U, eg. 192.168.1.3
s1c_bind_addr = <ipaddr>   ## local interface IP for external S1-MME, eg. 192.168.1.3
n_prb = 15
tm = 1
nof_ports = 1

[rf]
dl_earfcn = 1934
tx_gain = 60               ## this power seems to work best
rx_gain = 40
```

(continues on next page)

(continued from previous page)

```
device_name = soapy
device_args = auto          ## does not work with anything other than 'auto'
```

Changes to default configs for srsRAN core network:

```
sudo nano /root/.config/srsran/epc.conf

[mme]
mcc = <yourMCC>
mnc = <yourMNC>
mme_bind_addr = 127.0.1.100  ## or local interface IP for external S1-MME, eg. 192.168.1.
↪10
```

```
sudo nano /root/.config/srsran/user_db.csv

* add details of your SIM cards
```

Note: When running the srsRAN core network (srsepc) on an external device (eg. another Pi), you must open incoming firewall ports to allow the S1-MME and S1-U connections from srsenb.

S1-MME = sctp, port 36412 || S1-U = udp, port 2152

If using iptables,

```
sudo iptables -A INPUT -p sctp -m sctp --dport 36412 -j ACCEPT
sudo iptables -A INPUT -p udp -m udp --dport 2152 -j ACCEPT
```

Running the Pi4 eNodeB

Launch the software in separate ssh windows or using screen. Remember to use an external power source for your SDR. **The first time you run the srsenb software, you will need to wait a few minutes for it to finish setting up**. After the first time it will start without delay.

Launch Pi4 eNodeB:

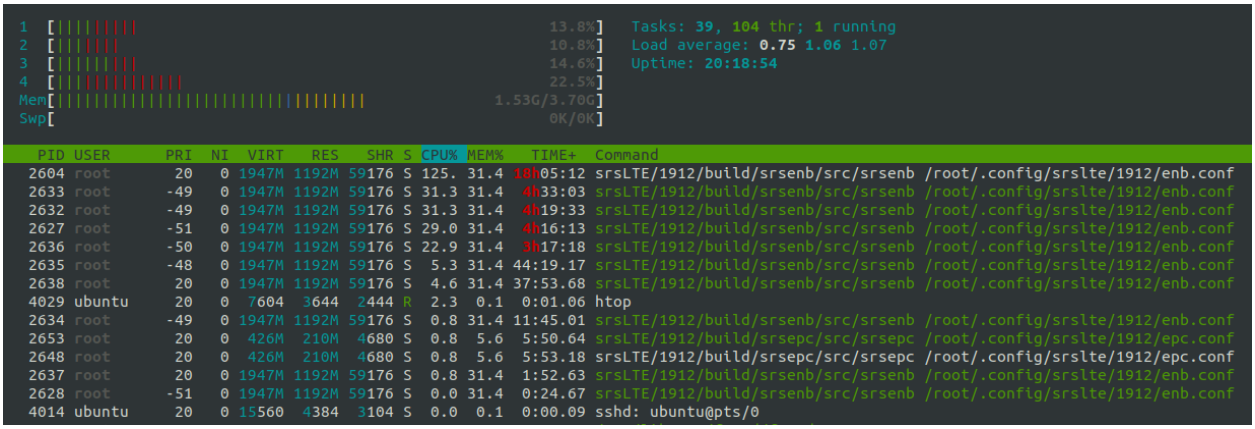
```
sudo srsenb /root/.config/srsran/enb.conf
```

Note: Between runs when using the LimeSDR-USB, you sometimes need to physically unplug and reconnect the SDR to power cycle it.

Launch core network (on separate device, or on the Pi4 eNodeB when using USRP B210):

```
sudo srsepc /root/.config/srsran/epc.conf
sudo /usr/local/bin/srsepc_if_masq.sh eth0
```

The following htop screenshot shows the resource utilisation when running the software on the Pi 4B /4GB RAM with x2 UEs attached to the USRP B210 cell. The srsRAN software has been running here for more than 18 hours without any problems. Only half of the RAM is used, and the CPU cores are sitting at around 25%. There is a chance, therefore, that this software configuration will work with the Pi 4B /2GB RAM version, and maybe also on other recent Arm based dev boards. If you can get a working cell going with alternative hardware, let the srsran-users mailing list know!



Known issues

- For bandwidths above 6 PRB it is recommended to use srsRAN 19.12 instead of the most recent release 20.04. We have identified the issue in the PRACH handling mainly affecting low-power devices. The fix will be included in the upcoming release.

2.3.6 C-V2X Application Note

Introduction

Cellular-V2X (C-V2X), or Cellular Vehicle to Everything, is a 3GPP standard to facilitate automated and (cooperative) intelligent transportation systems (C-ITS). With C-V2X, vehicles or other devices will be able to directly communicate with each other without having to go through the cellular infrastructure. This so called **Sidelink** communication, has a couple of advantages such as reducing communication delay when peers are in close vicinity, but may also increase network capacity when communication resources can be reused in different locations. The vehicular extensions have first been introduced in 3GPP Release 14 but are in fact based on earlier attempts to support direct device to device (D2D) communication within cellular networks. Although C-V2X is considered a key enabler for future transportation systems and the key market players, chip manufactures, operators and infrastructure providers, are heavily pushing the technology, only few devices are available. But even if they are officially announced it is extremely difficult to purchase them for developers or researchers, especially in small quantities.

As of version 20.04, srsRAN includes a complete implementation of the 3GPP Sidelink physical layer standardized in Release 14 licensed under AGPL v3. This includes all control and data channels and signals for all transmission modes for both receive and transmit operation. This allows to build complete and fully compatible C-V2X modems using software radios.

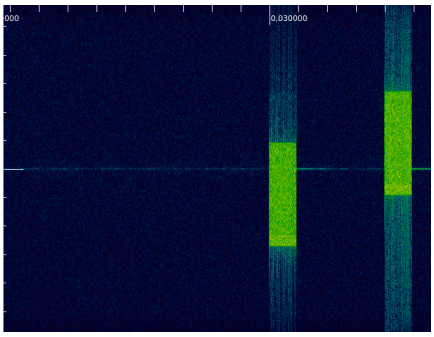
This application note shows how to use the receive-only example provided in srsRAN 20.04 to decode transmissions from a third-party commercial C-V2X device.

Requirements

The C-V2X example requires a radio that can process 10 or 20 MHz wide channels. Furthermore, the device needs to be capable of deriving timing information from GNSS signals, e.g. a GPS signal. We have tested with a Ettus Research B210 with GPSDO module.

Anatomy of a C-V2X Signal

Let's first have a look at a typical signal as it will be transmitted and received by C-V2X devices. The image below shows a signal captures from a commercial C-V2X modem. This signal has been captured at 5.92 GHz (channel 184) with a sample rate of 11.52 MHz.



Two identical subframes are transmitted one after each other with a gap of three empty subframes. The second transmission is a actually a retransmission of the first subframe. Retransmissions occur with a fixed time offset but may occupy different frequency resources. Note that there are no acknowledgments to provide the sender feedback as to whether the transmission has been received or not.

It's also noteworthy to say that no dedicated synchronization signals are transmitted as timing is solely derived from the GNSS signal.

For more information about the Sidelink signal structure have a look at this excellent (albeit not focusing on C-V2X) white paper from Rohde+Schwarz.

Decoding C-V2X Signals

The COTS C-V2X device used in this app note by default uses channel 184 centered at 5.92 GHz for transmission. Also it uses the default channel bandwidth of 10 MHz (or 50 PRB). In preparation for this, make sure to turn the device on and assure it has good GPS reception. Then, enable the transmit example. Make sure that you can observe the transmissions using a spectrum analyzer for example.

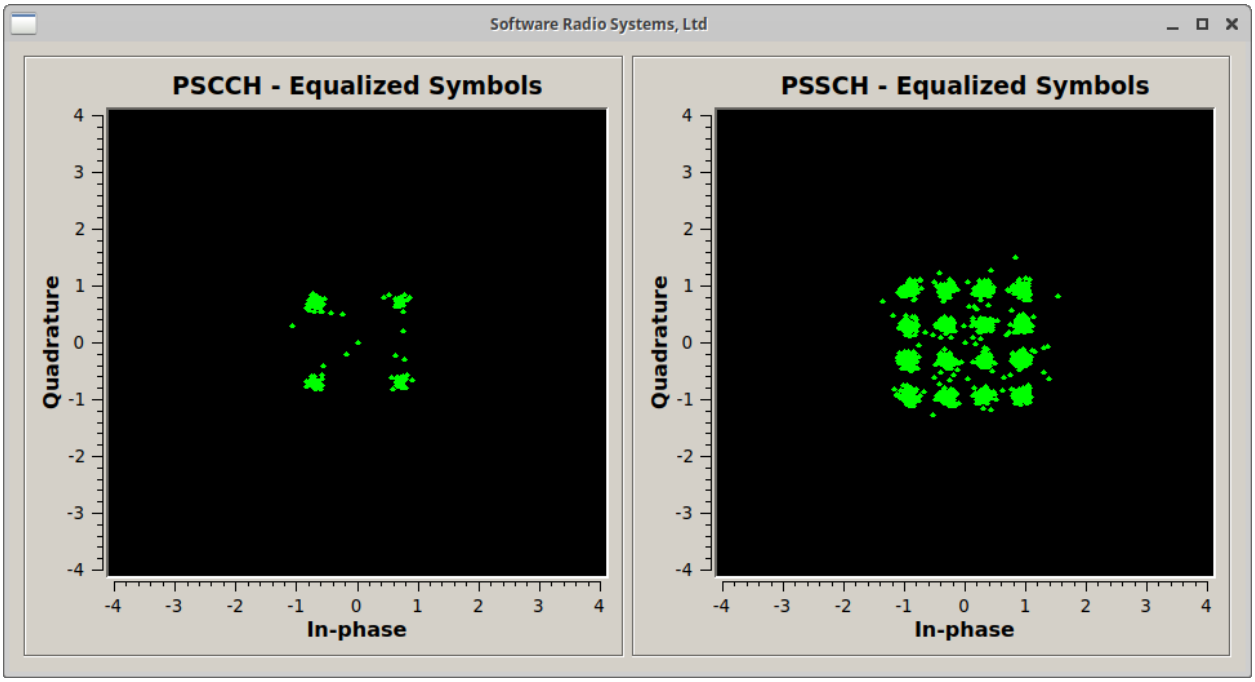
We have to options to decode the signal, we either capture the signal first and save it into a file and process the file, or we capture a live and decode it real-time. Let's start with the second option and decode the live signal, which is also the default case for *pssch_ue*.

Capture and Decode in Real-time

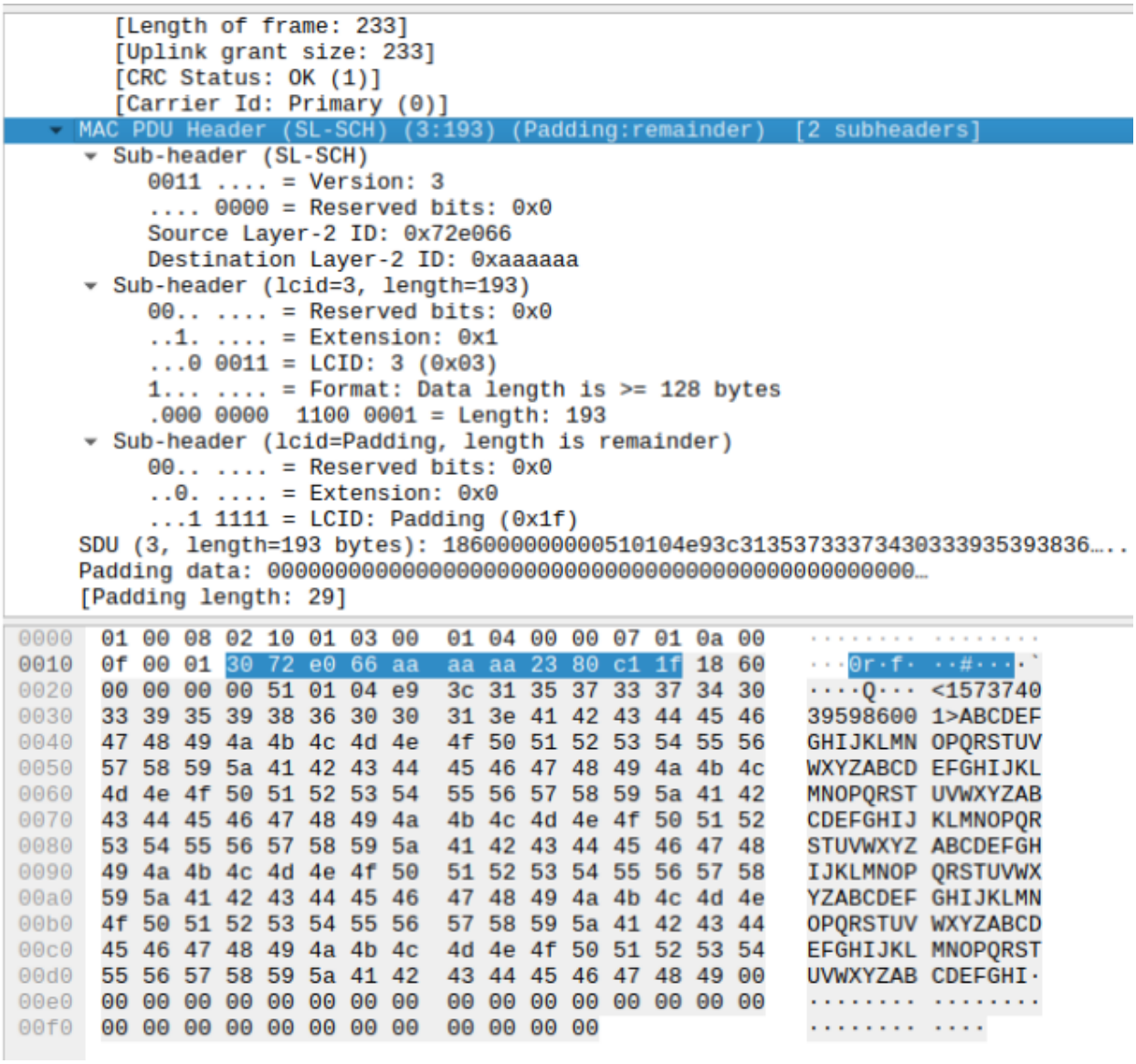
For this, we can simply run the *pssch_ue* example. It uses 5.92 GHz by default, but the frequency can be changed using the *-f* parameter. We have to make sure we use the device in GPS-sync mode via parameter though.

```
$ ./lib/examples/pssch_ue -a clock=gpsdo
open file to write
Opening RF device...
[INFO] [UHD] linux; GNU C++ version 7.4.0; Boost_106501; UHD_3.14.1.1-release
[INFO] [LOGGING] Fastpath logging disabled at runtime.
Using GPSDO clock
Opening USRP channels=1, args: type=b200,master_clock_rate=23.04e6
[INFO] [B200] Detected Device: B210
[INFO] [B200] Operating over USB 3.
[INFO] [B200] Detecting internal GPSDO....
[INFO] [GPS] Found an internal GPSDO: GPSTCXO , Firmware Rev 0.929a
[INFO] [B200] Initialize CODEC control...
[INFO] [B200] Initialize Radio control...
[INFO] [B200] Performing register loopback test...
[INFO] [B200] Register loopback test passed
[INFO] [B200] Performing register loopback test...
[INFO] [B200] Register loopback test passed
[INFO] [B200] Asking for clock rate 23.040000 MHz...
[INFO] [B200] Actually got clock rate 23.040000 MHz.
Setting USRP time to 1588858638s
[INFO] [MULTI_USRP]     1) catch time transition at pps edge
[INFO] [MULTI_USRP]     2) set times next pps (synchronously)
Setting sampling rate 11.52 MHz
Set RX freq: 5920.00 MHz
Set RX gain: 50.0 dB
Using a SF len of 11520 samples
OOSCI1: riv=4, mcs=5, priority=2, res_rsrv=0, t_gap=3, rtx=1, txformat=0
SCI1: riv=1, mcs=5, priority=2, res_rsrv=0, t_gap=11, rtx=1, txformat=0
SCI1: riv=1, mcs=5, priority=2, res_rsrv=0, t_gap=8, rtx=0, txformat=0
SCI1: riv=0, mcs=5, priority=2, res_rsrv=0, t_gap=8, rtx=1, txformat=0
^CSIGINT received. Exiting...
num_decoded_sci=4 num_decoded_tb=4
Saving PCAP file to /tmp/pssch.pcap
```

If you've compiled srsRAN with GUI support you should see something like this on your screen. In this particular examples we can see the QPSK constellation of the control channel (PSCCH) and the 16-QAM constellation of the data channel (PSSCH).



You can stop the decoder with Ctrl+C. Upon exit, the application writes a PCAP file of the decoded signal to */tmp/pssch.pcap*. This file can be inspected with Wireshark. The screenshot below shows Wireshark decoding the received signal. In this examples just random data is being transmitted but if you're device transmits actual ITS traffic, you should be able to see that there too.



Capture Signal to File and Post-Process

As a second option, we can also capture the signal first, save it into file and then post-process the capture. For example, the command below writes 200 subframes to */tmp/usrp.dat*.

```
$ ./lib/examples/usrp_capture_sync -l 0 -f 5.92e9 -o /tmp/usrp.dat -a clock=gpsdo -p 50 -
↪m -n 200
Opening RF device...
[INFO] [UHD] linux; GNU C++ version 7.4.0; Boost_106501; UHD_3.14.1.1-release
[INFO] [LOGGING] Fastpath logging disabled at runtime.
Using GPSDO clock
Opening USRP channels=1, args: type=b200,master_clock_rate=23.04e6
[INFO] [B200] Detected Device: B210
[INFO] [B200] Operating over USB 3.
[INFO] [B200] Detecting internal GPSDO....
```

(continues on next page)

(continued from previous page)

```
[INFO] [GPS] Found an internal GPSDO: GPSTCXO , Firmware Rev 0.929a
[INFO] [B200] Initialize CODEC control...
[INFO] [B200] Initialize Radio control...
[INFO] [B200] Performing register loopback test...
[INFO] [B200] Register loopback test passed
[INFO] [B200] Performing register loopback test...
[INFO] [B200] Register loopback test passed
[INFO] [B200] Asking for clock rate 23.040000 MHz...
[INFO] [B200] Actually got clock rate 23.040000 MHz.
Setting USRP time to 1588858960s
[INFO] [MULTI_USRP]     1) catch time transition at pps edge
[INFO] [MULTI_USRP]     2) set times next pps (synchronously)
Set RX freq: 5920.000000 MHz
Set RX gain: 60.0 dB
Setting sampling rate 11.52 MHz
Writing to file    199 subframes...
Ok - wrote 200 subframes
Start of capture at 1588858963+0.010. TTI=108.6
```

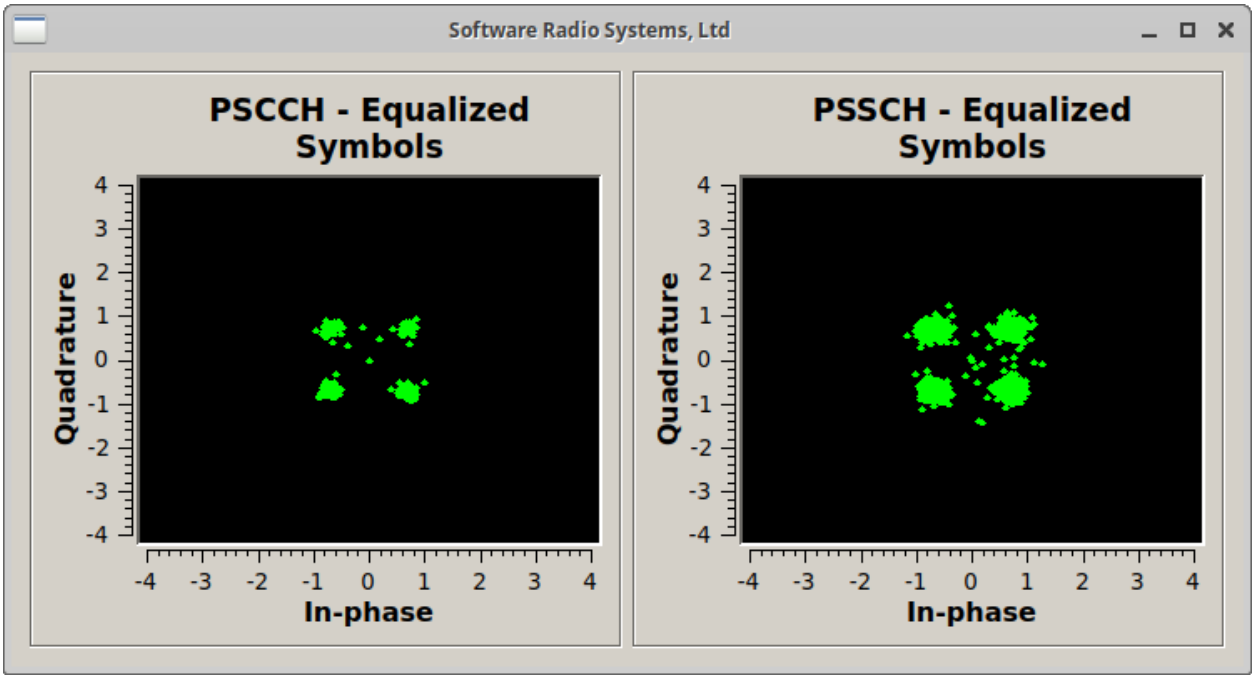
Similar to the above shown example, those subframes can now be decoded with *pssch_ue* by specifying the input file name with parameter *-i*.

```
$ ./lib/examples/pssch_ue -i /tmp/usrp.dat
...
```

We can also use the example to decode one of the test vectors:

```
$./lib/examples/pssch_ue -i ../lib/src/phy/phch/test/signal_sidelink_cmw500_f5.92e9_s11.
↪52e6_50prb_0offset_1ms.dat
Using a SF len of 11520 samples
SCI1: riv=0, mcs=5, priority=0, res_rsrv=1, t_gap=0, rtx=0, txformat=0
num_decoded_sci=1 num_decoded_tb=1
Saving PCAP file to /tmp/pssch.pcap
```

In this example, we can see that both PSCCH and PSSCH use QPSK as modulation scheme.



2.3.7 COTS UE Application Note

Please note, operating a private LTE network on cellular frequency bands may be tightly regulated in your jurisdiction. Seek the approval of your telecommunications regulator before doing so.

Introduction

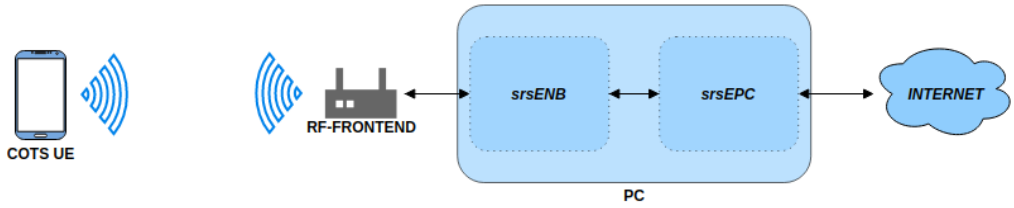
This application note aims to demonstrate how to set up your own LTE network using srsENB, srsEPC and a COTS UE. There are two options for network set-up when connecting a COTS UE: The network can be left as is, and the UE can communicate locally within the network, or the EPC can be connected to the internet through the P-GW, allowing the UE to access the internet for web-browsing, email etc.

Hardware Required

Creating a network and connecting a COTS UE requires the following:

- PC with a Linux based OS, with srsRAN installed and built
- An RF-frontend capable of both Tx & Rx
- A COTS UE
- USIM/ SIM card (This must be a test card or a programmable card, with known keys)

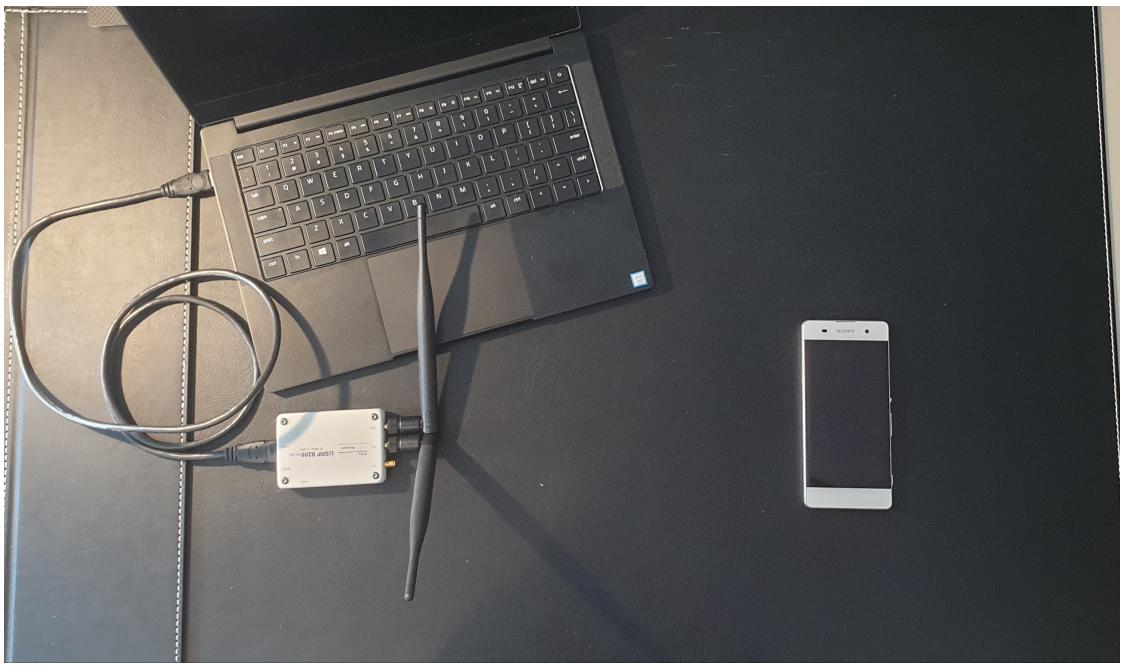
The following diagram outlines the set-up:



For this implementation the following equipment was used:

- Razer Blade Stealth running Ubuntu 18.04
- B200 mini USRP
- Sony Xperia XA with a Sysmocom USIM

The following photo shows the real world implementation of the equipment for this use case:



Note, this is for illustrative purposes, this orientation of USRP and UE may not give the best stability & throughput.

Driver & Conf. File Set-Up

Before instantiating the network and connecting the UE you need to first ensure you have the correct drivers installed and that the configuration files are edited appropriately.

Drivers

Firstly, check that you have the appropriate drivers for your SDR installed. If not they must be downloaded from the relevant source. If the drivers are already installed ensure they are up to date and are from a stable release. This step can be skipped if you have the correct drivers and know them to be working.

- RF front-end drivers:
  - UHD: https://github.com/EttusResearch/uhd
  - SoapySDR: https://github.com/pothosware/SoapySDR
  - BladeRF: https://github.com/Nuand/bladeRF

Note: This app note was tested using a b200-mini and UHD v4.0, but we recommend using UHD v3.15 where possible.

When the drivers have been installed/ updated you should connect your hardware and check that everything is working correctly. To do this for a USRP using the UHD drivers run the following command:

```
uhd_usrp_probe
```

This should be done anytime you are using a USRP before carrying out any testing or implementation to check a stable connection to the radio. Note, you should be using a USB 3.0 interface when using an SDR for this use case.

If you have had to install or update your drivers and everything is working as intended, then you will need to rebuild srsRAN to ensure it picks up on the new/ updated drivers.

To make a clean build execute the following commands in your terminal:

```
cd ./srsRAN/build
rm CMakeCache.txt
make clean
cmake ..
make
```

Your hardware and drivers should now be working correctly and be ready to use for connecting a COTS UE to srsRAN.

Conf. Files

The base configuration files for srsRAN can be installed by running the following command in the build folder:

```
sudo srsRAN_install_configs.sh <user/service>
```

You have the option to install the configurations files to the user directory or for all users. For this example the configuration files have been installed for all users by running the following command `sudo srsran_install_configs.sh service`. The config files can then be found in the following folder: `~./etc/srsran`

You will need to edit the following files before you can run a COTS UE over the network:

- epc.conf

- enb.conf
- user_db.csv

The eNB & EPC config files will need to be edited such that the MCC & MNC values are the same across both files. The user DB file needs to be updated so that it contains the credentials associated with the USIM card being used in the UE.

EPC:

The following snippet shows where to change the MCC & MNC values in the EPC config file:

```
22 | ###########################################################################
23 | [mme]
24 | mme_code = 0x1a
25 | mme_group = 0x0001
26 | tac = 0x0007
27 | mcc = 901
28 | mnc = 70
29 | mme_bind_addr = 127.0.1.100
30 | apn = srsapn
31 | dns_addr = 8.8.8.8
32 | encryption_algo = EEA0
33 | integrity_algo = EIA1
34 | paging_timer = 2
35 |
36 | ###########################################################################
```

Line 27 and 28 must be changed, for Sysmocom USIMS these values are 901 & 70. These values will be dependent on the USIM being used.

eNB:

The above changes must be mirrored in the eNB config. file. The following snippet shows this:

```
18 | ###########################################################################
19 | [enb]
20 | enb_id = 0x19B
21 | mcc = 901
22 | mnc = 70
23 | mme_addr = 127.0.1.100
24 | gtp_bind_addr = 127.0.1.1
25 | s1c_bind_addr = 127.0.1.1
26 | n_prb = 50
27 | #tm = 4
28 | #nof_ports = 2
29 |
30 | ###########################################################################
```

Here, the MCC and MNC values at lines 21 & 22 are changed to the values used in the EPC.

For both of the config files the rest of the values can be left at the default values. They may be changed as needed, but further customization is not necessary to enable the successful connection of a COTS UE.

User DB:

The following list describes the fields contained in the `user_db.csv` file, found in the same folder as the .conf files. As standard, this file will come with two dummy UEs entered into the CSV, these help to provide an example of how the file should be filled in.

- Name: Any human readable value
- Auth: Authentication algorithm (xor/ mil)
- IMSI: UE's IMSI value
- Key: UE's key, hex value
- OP Type: Operator's code type (OP/ OPc)
- OP: OP/ OPc code, hex value
- AMF: Authentication management field, hex value must be above 8000
- SQN: UE's Sequence number for freshness of the authentication
- QCI: QoS Class Identifier for the UE's default bearer
- IP Alloc: IP allocation strategy for the SPGW

The AMF, SQN, QCI and IP Alloc fields can be populated with the following values:

- 9000, 000000000000, 9, dynamic

This will result in a user_db.csv file that should look something like the following:

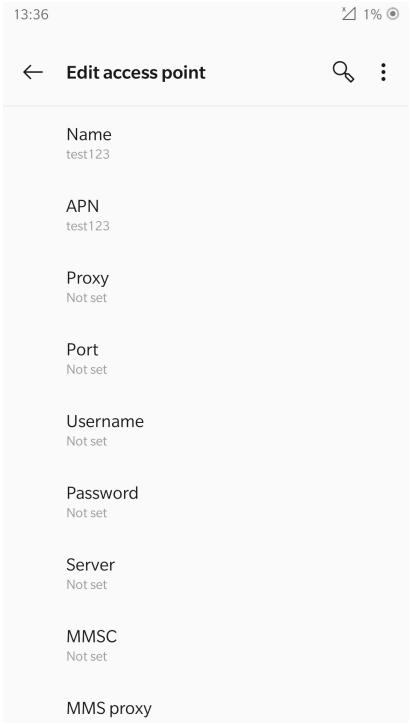
```
1 | #
2 | # .csv to store UE's information in HSS
3 | # Kept in the following format: "Name,Auth,IMSI,Key,OP_Type,OP,AMF,SQN,QCI,IP_alloc"
4 | #
5 | # Name:     Human readable name to help distinguish UE's. Ignored by the HSS
6 | # IMSI:     UE's IMSI value
7 | # Auth:     Authentication algorithm used by the UE. Valid algorithms are XOR
8 | #           (xor) and MILENAGE (mil)
9 | # Key:      UE's key, where other keys are derived from. Stored in hexadecimal
10| # OP_Type:  Operator's code type, either OP or OPc
11| # OP/OPc:   Operator Code/Cyphered Operator Code, stored in hexadecimal
12| # AMF:      Authentication management field, stored in hexadecimal
13| # SQN:      UE's Sequence number for freshness of the authentication
14| # QCI:      QoS Class Identifier for the UE's default bearer.
15| # IP_alloc: IP allocation stratagy for the SPGW.
16| #           With 'dynamic' the SPGW will automatically allocate IPs
17| #           With a valid IPv4 (e.g. '172.16.0.2') the UE will have a statically␣
↪assigned IP.
18| #
19| # Note: Lines starting by '#' are ignored and will be overwritten
20| ue3,mil,901700000020936,4933f9c5a83e5718c52e54066dc78dcf,opc,
↪fc632f97bd249ce0d16ba79e6505d300,9000,0000000060f8,9,dynamic
```

Line 20 shows the entry for the USIM being used in the COTS UE. The values assigned to the AMF, SQN, QCI & IP Alloc are default values above, as outlined *here* in the EPC documentation. Ensure there is no white space between the values in each entry, as this will cause the file to be read incorrectly.

Adding an APN

An APN is needed to allow the UE to access the internet. This is created from the UE and then a change is made to the EPC config file to reflect this.

From the UE navigate to the Network settings for the SIM being used. From here an APN can be added, usually under *"Access point names"*. Create a new APN with the name and APN "test123", as shown in the following figure.



The addition of this APN must be reflected in the EPC config file, to do this add the APN to the config. This is shown in the following snippet:

```
22 | ####################################################################
23 | [mme]
24 | mme_code = 0x1a
25 | mme_group = 0x0001
26 | tac = 0x0007
27 | mcc = 901
28 | mnc = 70
29 | mme_bind_addr = 127.0.1.100
30 | apn = test123
31 | dns_addr = 8.8.8.8
32 | encryption_algo = EEA0
33 | integrity_algo = EIA1
34 | paging_timer = 2
35 |
36 | ####################################################################
```

The APN has been added at line 30 above. This must match the APN on the UE to enable a successful connection.

Run Masquerading Script

To allow UE to connect to the internet via the EPC, the pre-configured masquerading script must be run. This can be found in `srsRAN/srsepc`. The masquerading script enables IP forwarding and sets up Network Address Translation to pass traffic between the srsRAN network and the external network. The script must be run each time the machine is re-booted, and can be done before or while the network is running. The UE will not be able to communicate with the interet until this script has been run.

Before running the script it is important to identify the interface being used to connect your PC to the internet. As the script requires this to be passed in as an argument. This can be done by running the following command:

```
route
```

You will see an output similar to the following:

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags   Metric  Ref    Use     Iface
default         192.168.1.1     0.0.0.0         UG      600     0        0     wlp2s0
link-local      0.0.0.0         255.255.0.0     U       1000    0        0     wlp2s0
192.168.1.0     0.0.0.0         255.255.255.0   U       600     0        0     wlp2s0
```

The interface (Iface) associated with the *default* destination is one which must be passed into the masq. script. In the above output that is the wlp2s0 interface.

The masq. script can now be run from the follow folder: `srsRAN/srsEPC`:

```
sudo ./srsepc_if_masq.sh <interface>
```

If it has executed successfully you will see the following message:

```
Masquerading Interface <interface>
```

The configuration files, user DB and UE should now be set up appropriately to allow the COTS UE to connect to the eNB and Core.

Connecting a COTS UE to srsRAN

The final step in connecting a COTS UE to srsRAN is to first spin up the network and then connect to that network from the UE. The following sections will outline how this is achieved.

Running srsEPC & srsENB

First navigate to the srsRAN folder. Then initialise the EPC by running:

```
sudo srsepc
```

The following output should be displayed on the console:

```
Built in Release mode using commit c892ae56b on branch master.

---  Software Radio Systems EPC  ---

Reading configuration file /etc/srsran/epc.conf...
```

(continues on next page)

(continued from previous page)

```
HSS Initialized.
MME S11 Initialized
MME GTP-C Initialized
MME Initialized. MCC: 0xf901, MNC: 0xff70
SPGW GTP-U Initialized.
SPGW S11 Initialized.
SP-GW Initialized.
```

The eNB can then be brought online in a separate console by running:

```
sudo srsenb
```

The console should display the following:

```
---  Software Radio Systems LTE eNodeB  ---

Reading configuration file /etc/srsran/enb.conf...

Built in Release mode using commit c892ae56b on branch master.

Opening 1 channels in RF device=UHD with args=default
[INFO] [UHD] linux; GNU C++ version 9.3.0; Boost_107100; UHD_4.0.0.0-666-g676c3a37
[INFO] [LOGGING] Fastpath logging disabled at runtime.
Opening USRP channels=1, args: type=b200,master_clock_rate=23.04e6
[INFO] [B200] Detected Device: B200mini
[INFO] [B200] Operating over USB 3.
[INFO] [B200] Initialize CODEC control...
[INFO] [B200] Initialize Radio control...
[INFO] [B200] Performing register loopback test...
[INFO] [B200] Register loopback test passed
[INFO] [B200] Asking for clock rate 23.040000 MHz...
[INFO] [B200] Actually got clock rate 23.040000 MHz.
Setting frequency: DL=2685.0 Mhz, UL=2565.0 MHz for cc_idx=0

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

The EPC console should now print an update if the eNB has successfully connected to the core:

```
Received S1 Setup Request.
S1 Setup Request - eNB Name: srsenb01, eNB id: 0x19b
S1 Setup Request - MCC:901, MNC:70, PLMN: 651527
S1 Setup Request - TAC 0, B-PLMN 0
S1 Setup Request - Paging DRX v128
Sending S1 Setup Response
```

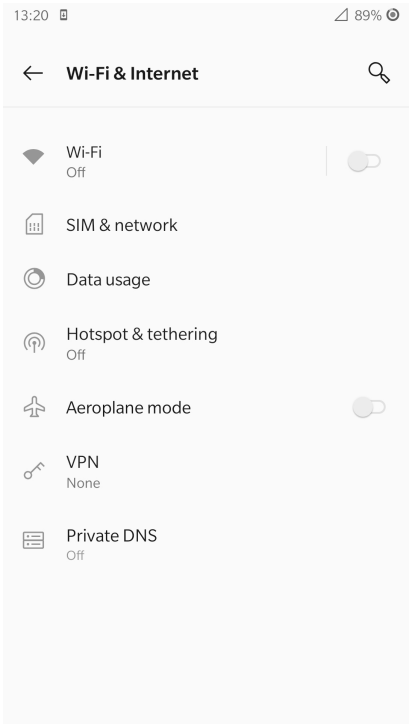
The network is now ready for the COTS UE to connect.

Connecting the UE

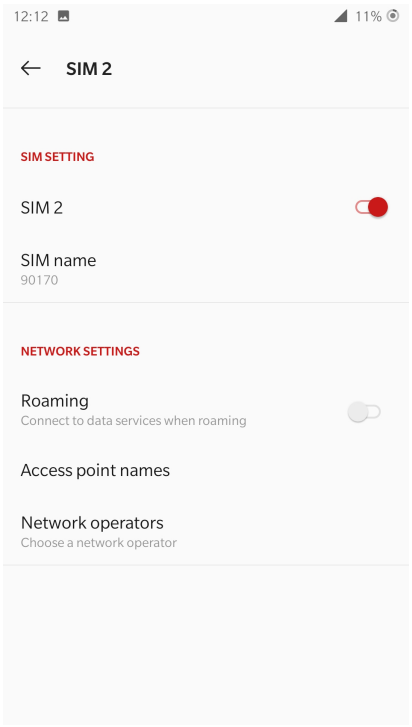
Connecting the UE to the network is a quick and easy process if the above steps have been completed successfully.

You can now connect the UE to the network by taking the following steps:

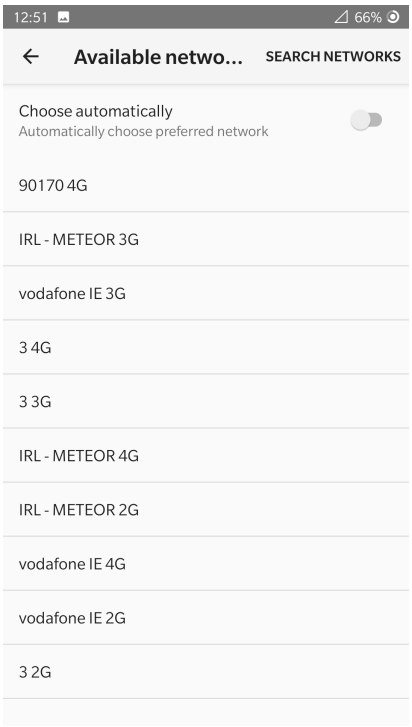
- Open the Settings menu and navigate to the Sim & Network options



- Open this menu and proceed to the sub-menu associated with the USIM being used. It should look something like the following:



- Under the Network Operators find the network which you have just instantiated using srsRAN



- Select the network that is a combination of your MMC & MNC values. For this example it is the network labelled 90170 4G. The UE should then automatically connect to the network.

The UE should now be connected to the network. To check for a successful connection use the logs output to the console.

Confirming Connection

Once the UE has connected to the network, the console outputs of the srsENB and srsEPC can be used to confirm a successful connection.

EPC Console:

The following output is shown for the EPC after a successful attach. First a confirmation message in the form of *UL NAS: Received Attach Complete* will be displayed, secondly the EPS bearers will be given out and the ID confirmed on the output, and lastly the *Sending EMM Information Message* output will be shown. If all of these are displayed in the logs, then an attach is successful. These messages are seen in the last five lines of the console output in the following console output:

```
Built in Release mode using commit c892ae56b on branch master.


---  Software Radio Systems EPC  ---

Reading configuration file /etc/srsran/epc.conf...
HSS Initialized.
MME S11 Initialized
MME GTP-C Initialized
MME Initialized. MCC: 0xf901, MNC: 0xff70
SPGW GTP-U Initialized.
SPGW S11 Initialized.
SP-GW Initialized.
Received S1 Setup Request.
S1 Setup Request - eNB Name: srsenb01, eNB id: 0x19b
S1 Setup Request - MCC:901, MNC:70, PLMN: 651527
S1 Setup Request - TAC 0, B-PLMN 0
S1 Setup Request - Paging DRX v128
Sending S1 Setup Response
Initial UE message: LIBLTE_MME_MSG_TYPE_ATTACH_REQUEST
Received Initial UE message -- Attach Request
Attach request -- IMSI: 901700000020936
Attach request -- eNB-UE S1AP Id: 1
Attach request -- Attach type: 2
Attach Request -- UE Network Capabilities EEA: 11110000
Attach Request -- UE Network Capabilities EIA: 11110000
Attach Request -- MS Network Capabilities Present: true
PDN Connectivity Request -- EPS Bearer Identity requested: 0
PDN Connectivity Request -- Procedure Transaction Id: 2
PDN Connectivity Request -- ESM Information Transfer requested: true
Downlink NAS: Sending Authentication Request
UL NAS: Authentication Failure
Authentication Failure -- Synchronization Failure
Downlink NAS: Sent Authentication Request
UL NAS: Received Authentication Response
Authentication Response -- IMSI 901700000020936
UE Authentication Accepted.
Generating KeNB with UL NAS COUNT: 0
Downlink NAS: Sending NAS Security Mode Command.
UL NAS: Received Security Mode Complete
Security Mode Command Complete -- IMSI: 901700000020936
```

(continues on next page)

(continued from previous page)

```
Sending ESM information request
UL NAS: Received ESM Information Response
ESM Info: APN srsapn
ESM Info: 6 Protocol Configuration Options
Getting subscription information -- QCI 9
Sending Create Session Request.
Creating Session Response -- IMSI: 901700000020936
Creating Session Response -- MME control TEID: 1
Received GTP-C PDU. Message type: GTPC_MSG_TYPE_CREATE_SESSION_REQUEST
SPGW: Allocated Ctrl TEID 1
SPGW: Allocated User TEID 1
SPGW: Allocate UE IP 192.168.0.2
Received Create Session Response
Create Session Response -- SPGW control TEID 1
Create Session Response -- SPGW S1-U Address: 127.0.1.100
SPGW Allocated IP 192.168.0.2 to IMSI 901700000020936
Adding attach accept to Initial Context Setup Request
Sent Initial Context Setup Request. E-RAB id 5
Received Initial Context Setup Response
E-RAB Context Setup. E-RAB id 5
E-RAB Context -- eNB TEID 0x460003; eNB GTP-U Address 127.0.1.1
UL NAS: Received Attach Complete
Unpacked Attached Complete Message. IMSI 901700000020936
Unpacked Activate Default EPS Bearer message. EPS Bearer id 5
Received GTP-C PDU. Message type: GTPC_MSG_TYPE_MODIFY_BEARER_REQUEST
Sending EMM Information
```

eNB Console:

The eNB console also display messages to confirm an attach. A *RACH* message should be seen followed by a *USER 0xX connected* message. Where "*0xX*" is a hex ID representing the UE.

NOTE, you may see some other RACHs and *Disconnecting rtni=0xX* messages. This may be from other devices trying to connect to the network, if you have seen a clear connection between the UE and network these can be ignored.

The following shows an output from the eNB that indicates a successful attach:

```
---  Software Radio Systems LTE eNodeB  ---

Reading configuration file /etc/srsran/enb.conf...

Built in Release mode using commit c892ae56b on branch master.

Opening 1 channels in RF device=UHD with args=default
[INFO] [UHD] linux; GNU C++ version 9.3.0; Boost_107100; UHD_4.0.0.0-666-g676c3a37
[INFO] [LOGGING] Fastpath logging disabled at runtime.
Opening USRP channels=1, args: type=b200,master_clock_rate=23.04e6
[INFO] [B200] Detected Device: B200mini
[INFO] [B200] Operating over USB 3.
[INFO] [B200] Initialize CODEC control...
[INFO] [B200] Initialize Radio control...
[INFO] [B200] Performing register loopback test...
[INFO] [B200] Register loopback test passed
[INFO] [B200] Asking for clock rate 23.040000 MHz...
```

(continues on next page)

(continued from previous page)

```
[INFO] [B200] Actually got clock rate 23.040000 MHz.
Setting frequency: DL=2685.0 Mhz, UL=2565.0 MHz for cc_idx=0

==== eNodeB started ===
Type <t> to view trace
RACH:  tti=521, preamble=44, offset=1, temp_crnti=0x46
User 0x46 connected
```

The UE is now connected to the network. and should now automatically connect to this network each time it is powered on. You should keep the UE in aeroplane mode until you want to connect it to the network. The UE should now also have access to the internet - as if connected to a commercial 4G network.

Troubleshooting

- Some users may experience trouble connecting to the internet, even after running the masquerading script. Ensure that IP forwarding is enabled, and check your network configuration as this may be stopping the UE from connecting successfully.
- Users may also have trouble connecting to the network. Firstly check all information in the configuration and user DB files are correct. You may also need to adjust the gain parameters in the eNB config. file - without high enough power (pmax threshold), the UE won't PRACH.
- Note that some USIM cards may not be compatible in UEs that are "locked" to certain network operators.

2.3.8 Handover Application Note

srsRAN Release 20.10 or later is required to run the following applications

Introduction

This application note focuses on mobility and handover. Specifically, we show how to configure an end-to-end network to support user-controlled handover. We address both intra-eNB and S1 handover using srsRAN with ZeroMQ-based RF emulation and we use the GNURadio Companion as a broker for controlling cell gains to trigger handover. Creating an E2E network using ZMQ and adding GRC functionality is demonstrated in our *ZMQ App Note*.

Hardware & Software Required

Both Intra-eNB and S1 handover have the following hardware and software requirements:

- A PC/ Laptop running a Linux based OS with the latest version of srsRAN installed and built.
- ZMQ installed and working with srsRAN.
- GNU-Radio Companion, which can be downloaded from this link.
- Fully up to date drivers & dependencies.

The following command will ensure the correct dependencies are installed (Ubuntu only):

```
sudo apt-get install cmake libfftw3-dev libmbedtls-dev libboost-program-options-dev␣
↪libconfig++-dev libsctp-dev
```

For a full guide on installing srsRAN see the *installation guide*. The *ZMQ app note* shows how to correctly install and run ZMQ.

If you have had to install or update your drivers and/or dependencies without having re-built srsRAN, then you will need to do so to ensure srsRAN picks up on the new/ updated drivers.

To make a clean build execute the following commands in your terminal:

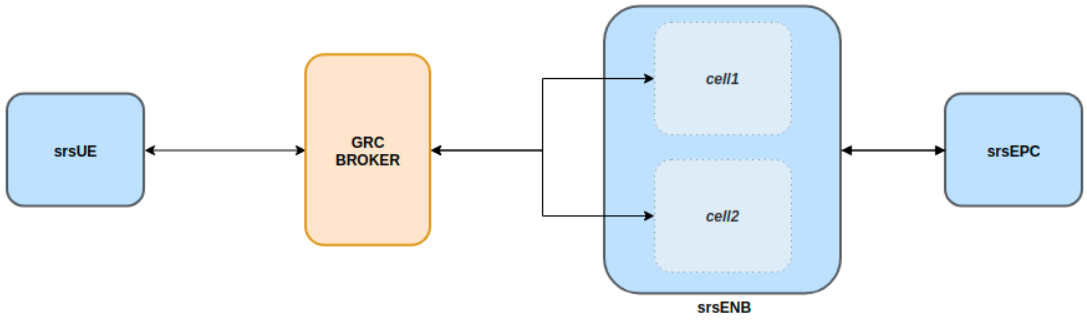
```
cd ./srsran/build
rm CMakeCache.txt
make clean
cmake ..
make
```

Your hardware and drivers should now be working correctly and be ready to use correctly with srsRAN.

Intra-eNB Handover

Intra-eNB Handover describes the handover between cells when a UE moves from one sector to another sector which are managed by the same eNB. The following steps show how ZMQ and GRC can be used with srsRAN to demonstrate such a handover.

The following figure shows the overall architecture used:



This set-up will allow intra-frequency intra-enb handover.

Note, ZMQ elements have not been included here so as to simplify the diagram, although they do form a key aspect of this implementation.

srsRAN Set-Up

To enable the successful execution of intra-eNB handover the configuration files of the eNB, radio resources and the UE must be modified.

eNB:

In the eNB the RF Device and Args should be set so that ZMQ is used and two Tx/Rx TCP port pairings are created for the UL & DL of each cell.

The following example shows how this is done:

```
######################################################################
[rf]
tx_gain = 80
```

(continues on next page)

(continued from previous page)

```
rx_gain = 40

device_name = zmq
device_args = fail_on_disconnect=true,id=enb,tx_port0=tcp://*:2101,tx_port1=tcp://*:2201,
↪rx_port0=tcp://localhost:2100,rx_port1=tcp://localhost:2200,id=enb,base_srate=23.04e6
###################################################################
```

The following table should make clear how the TCP ports are allocated across the cells:

Table 3: Cell Ports Used

Port Direction	cell1 Port #	cell2 Port #
Rx	2100	2200
Tx	2101	2201

The use of a clear labelling system for the ports is employed to allow for easier implementation of the GRC broker. By having the least significant unit of each Rx port be 0 and Tx port be 1 the flowgraph becomes easier to debug. The second most significant unit is used to indicate which cell the port belongs to.

Radio Resource (RR):

The rr.conf is where the cells (sectors) are added to the eNB, this is also where the handover flags are enabled. The following shows how this is done:

```
cell_list =
(
 {
        rf_port = 0;
        cell_id = 0x01;
        tac = 0x0007;
        pci = 1;
        root_seq_idx = 204;
        dl_earfcn = 2850;
        ho_active = true;

        // Cells available for handover
        meas_cell_list =
        (
        );

        // ReportCfg (only A3 supported)
        meas_report_desc = {
        a3_report_type = "RSRP";
        a3_offset = 6;
        a3_hysteresis = 0;
        a3_time_to_trigger = 480;
        rsrq_config = 4;
        rsrp_config = 4;
        };
 },
 {
        rf_port = 1;
        cell_id = 0x02;
```

(continues on next page)

(continued from previous page)

```
        tac = 0x0007;
        pci = 6;
        root_seq_idx = 268;
        dl_earfcn = 2850;
        ho_active = true;

        // Cells available for handover
        meas_cell_list =
        (
        );

        // ReportCfg (only A3 supported)
        meas_report_desc = {
        a3_report_type = "RSRP";
        a3_offset = 6;
        a3_hysteresis = 0;
        a3_time_to_trigger = 480;
        rsrq_config = 4;
        rsrp_config = 4;
                };
 }
);
```

Note, the TAC of the cells must match that of the MME, and the EARFCN must be the same across both cells and the UE. The PCI of each cell with the same EARFCN must be different, such that *PCI%3* for the cells is not equal.

UE:

For the UE configuration, ZMQ must be set as the default device and the appropriate TCP ports set for Tx & Rx. As well as this the EARFCN value must be checked to ensure it is the same as that set for the cells in rr.conf. The following example shows how the ue.conf file must be modified:

```
#####################################################################
[rf]
dl_earfcn = 2850
freq_offset = 0
tx_gain = 80
#rx_gain = 40

device_name = zmq
device_args = tx_port=tcp://*:2001,rx_port=tcp://localhost:2000,id=ue,base_srate=23.04e6
#####################################################################
```

The default USIM configuration can be used, as it is already present in the user_db.csv file used by the EPC to authenticate the UE. If you want to use a custom USIM set up this will need to be added to the relevant section in the ue.conf file and reflected in the user_db.csv to ensure the UE is authenticated correctly.

Table 4: UE Ports Used

Port Direction	Port #
Rx	2000
Tx	2001

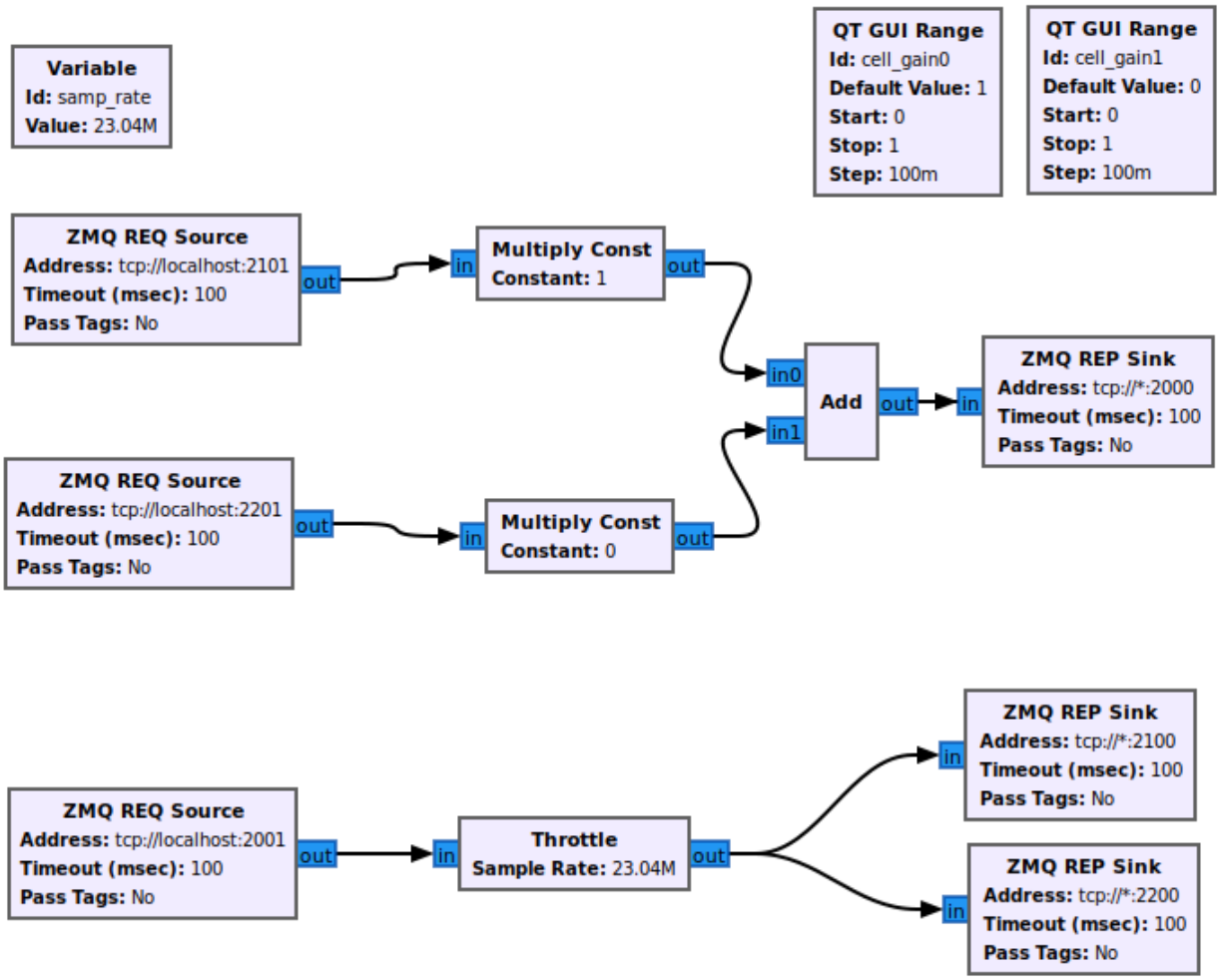
Again for these ports the least significant unit is used to indicate whether the port is being used for Tx or Rx.

In short, the EARFCN values must be the same across the eNB, both cells and the UE, handover must be enabled in the RR config file and ZMQ made the default device for both the eNB and UE.

GNU-Radio Companion

The GRC file can be downloaded `here`. Download and/ or save the file as a *.grc* file. Run with GNU-Radio Companion when needed.

The GRC Broker will be used to force handover between cells. This will be done by manually controlling the gain of each cell using variables and a slider. ZMQ REQ Source and REP Sink blocks will be used to link the flowgraph to the ZMQ instances of srsENB and srsUE. The following figure illustrated how this is done:



The following table again shows the clear breakdown of how the ports are assigned to each of the network elements:

Table 5: Ports Used

Port Direction	cell1 Port #	cell2 Port #	UE Port #
Rx	2100	2200	2000
Tx	2101	2201	2001

The gain of cell2 is first set to 0, and cell1 to 1. These are then controlled via sliders and increased in steps of 0.1 to

force handover once a connection has been established. Handover should occur once the gain of a cell is higher than the other, i.e. when the signal is stronger.

Running the Network

To instantiate the network correctly srsEPC is first run, then srsENB and finally srsUE. Once all three are running the GRC Broker should be run from GNU-Radio. The UE should then connect to the network, with the UL & DL passing through the broker. You should have already set up a network namespace for the UE, as described in the *ZMQ App Note*.

EPC:

To initiate the EPC, simply run the following command:

```
sudo srsepc
```

The EPC should display the following:

```
Built in Release mode using commit 7e60d8aae on branch next.


---  Software Radio Systems EPC  ---

Reading configuration file /etc/srsran/epc.conf...
HSS Initialized.
MME S11 Initialized
MME GTP-C Initialized
MME Initialized. MCC: 0xf901, MNC: 0xff70
SPGW GTP-U Initialized.
SPGW S11 Initialized.
SP-GW Initialized.
```

eNB:

Once the EPC is running, the eNB can by run using this command:

```
sudo srsenb
```

You should then see the following in the console:

```
---  Software Radio Systems LTE eNodeB  ---

Reading configuration file /etc/srsran/enb.conf...

Built in Release mode using commit 7e60d8aae on branch next.

Opening 2 channels in RF device=zmq with args=fail_on_disconnect=true,id=enb,tx_
↪port0=tcp://*:2101,tx_port1=tcp://*:2201,rx_port0=tcp://localhost:2100,rx_port1=tcp://
↪localhost:2200,id=enb,base_srate=23.04e6
CHx base_srate=23.04e6
CHx id=enb
Current sample rate is 1.92 MHz with a base rate of 23.04 MHz (x12 decimation)
CH0 rx_port=tcp://localhost:2100
CH0 tx_port=tcp://*:2101
```

(continues on next page)

(continued from previous page)

```
CH0 fail_on_disconnect=true
CH1 rx_port=tcp://localhost:2200
CH1 tx_port=tcp://*:2201
Current sample rate is 11.52 MHz with a base rate of 23.04 MHz (x2 decimation)
Current sample rate is 11.52 MHz with a base rate of 23.04 MHz (x2 decimation)
Setting frequency: DL=2630.0 Mhz, UL=2510.0 MHz for cc_idx=0
Setting frequency: DL=2630.0 Mhz, UL=2510.0 MHz for cc_idx=1

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

The EPC console should then display a confirmation that the eNB cas connected:

```
Received S1 Setup Request.
S1 Setup Request - eNB Name: srsenb01, eNB id: 0x19b
S1 Setup Request - MCC:901, MNC:70
S1 Setup Request - TAC 7, B-PLMN 0x9f107
S1 Setup Request - Paging DRX v128
Sending S1 Setup Response
```

UE:

The UE now needs to be run, this can be done with the following command:

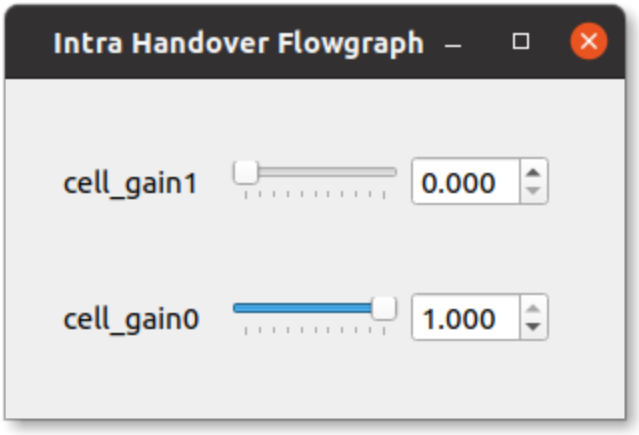
```
sudo srsue --gw.netns=ue1
```

The UE console should then display this:

```
Reading configuration file /etc/srsran/ue.conf...

Built in Release mode using commit 7e60d8aae on branch next.

Opening 1 channels in RF device=zmq with args=tx_port=tcp://*:2001,rx_port=tcp://
↪localhost:2000,id=ue,base_srate=23.04e6
CHx base_srate=23.04e6
CHx id=ue
Current sample rate is 1.92 MHz with a base rate of 23.04 MHz (x12 decimation)
CH0 rx_port=tcp://localhost:2000
CH0 tx_port=tcp://*:2001
Waiting PHY to initialize ... done!
Attaching UE...
Current sample rate is 1.92 MHz with a base rate of 23.04 MHz (x12 decimation)
Current sample rate is 1.92 MHz with a base rate of 23.04 MHz (x12 decimation)
```

GRC:

Once all three network elements have been successfully initiated, the Broker can be run. This is done in the same way as any other GRC Flowgraph. Once successful, a pop up window should display the interactive slider for controlling the gain of the two cells.



Confirming Connection

Once the broker has been run, a successful attach should be made and the network should be up and running fully. To confirm this, check the appropriate messages are displayed in the console.

EPC Attach:

If the attach is successful the EPC should give the following readout:

```
Initial UE message: LIBLTE_MME_MSG_TYPE_ATTACH_REQUEST
Received Initial UE message -- Attach Request
Attach request -- M-TMSI: 0xd1006989
Attach request -- eNB-UE S1AP Id: 1
Attach request -- Attach type: 1
Attach Request -- UE Network Capabilities EEA: 11110000
Attach Request -- UE Network Capabilities EIA: 01110000
Attach Request -- MS Network Capabilities Present: false
PDN Connectivity Request -- EPS Bearer Identity requested: 0
PDN Connectivity Request -- Procedure Transaction Id: 1
PDN Connectivity Request -- ESM Information Transfer requested: false
UL NAS: Received Identity Response
ID Response -- IMSI: 901700123456789
Downlink NAS: Sent Authentication Request
UL NAS: Received Authentication Response
Authentication Response -- IMSI 901700123456789
UE Authentication Accepted.
Generating KeNB with UL NAS COUNT: 0
Downlink NAS: Sending NAS Security Mode Command.
UL NAS: Received Security Mode Complete
Security Mode Command Complete -- IMSI: 901700123456789
Getting subscription information -- QCI 7
Sending Create Session Request.
Creating Session Response -- IMSI: 901700123456789
Creating Session Response -- MME control TEID: 1
Received GTP-C PDU. Message type: GTPC_MSG_TYPE_CREATE_SESSION_REQUEST
SPGW: Allocated Ctrl TEID 1
SPGW: Allocated User TEID 1
```

(continues on next page)

(continued from previous page)

```
SPGW: Allocate UE IP 172.16.0.2
Received Create Session Response
Create Session Response -- SPGW control TEID 1
Create Session Response -- SPGW S1-U Address: 127.0.1.100
SPGW Allocated IP 172.16.0.2 to IMSI 901700123456789
Adding attach accept to Initial Context Setup Request
Sent Initial Context Setup Request. E-RAB id 5
Received Initial Context Setup Response
E-RAB Context Setup. E-RAB id 5
E-RAB Context -- eNB TEID 0x1; eNB GTP-U Address 127.0.1.1
UL NAS: Received Attach Complete
Unpacked Attached Complete Message. IMSI 901700123456789
Unpacked Activate Default EPS Bearer message. EPS Bearer id 5
Received GTP-C PDU. Message type: GTPC_MSG_TYPE_MODIFY_BEARER_REQUEST
Sending EMM Information
```

eNB Attach:

You will see the RACH and connection message on the eNB:

```
RACH:  tti=341, cc=0, preamble=14, offset=0, temp_crnti=0x46
User 0x46 connected
```

UE Attach:

The UE console will display the following:

```
Found Cell:  Mode=FDD, PCI=1, PRB=50, Ports=1, CFO=-0.2 KHz
Current sample rate is 11.52 MHz with a base rate of 23.04 MHz (x2 decimation)
Current sample rate is 11.52 MHz with a base rate of 23.04 MHz (x2 decimation)
Found PLMN:  Id=90170, TAC=7
Random Access Transmission: seq=14, ra-rnti=0x2
Random Access Complete.     c-rnti=0x46, ta=0
RRC Connected
Network attach successful. IP: 172.16.0.2
Software Radio Systems LTE (srsRAN) 21/10/2020 12:47:43 TZ:0
```

The network is now ready for handover to be initiated and tested. To keep the UE from entering idle, you should send traffic between the UE and the eNB. This can be done with the following command:

```
sudo ip netns exec ue1 ping 172.16.0.1
```

Forcing Handover

Handover is simply forced by using the slider to change the gain variables within GRC. Once the handover is successful a message should be displayed by the UE acknowledging a successful handover.

GRC:

The Following steps outline how handover can be forced with GRC. Aagain, this is done using the sliders for the gain variables:

1. Set the gain of *cell1* to 0.5
2. Slowly increase the gain of *cell2* to above 0.5 and on to 1.

3. Wait for handover to be acknowledged.
4. Move the gain of *cell1* to 0.

UE Console:

If handover is successful you should see the following read out in the UE console:

```
Received HO command to target PCell=6, NCC=0
Random Access Transmission: seq=3, ra-rnti=0x2
Random Access Complete.     c-rnti=0x47, ta=0
HO successful
```

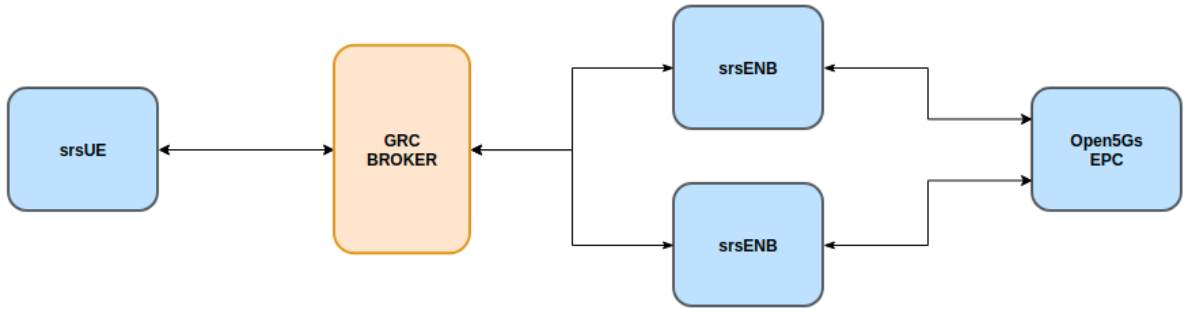
Handover can now be repeated as many times as needed by repeating the above steps.

S1 Handover

Note, srsEPC does not support handover via the S1 interface, as it is designed to be a lightweight core for network-in-a-box type deployments. To support S1 handover, a third party EPC must be used. We will use Open5GS for the purposes of this note, however any third-party EPC supporting S1 handover can be used.

S1 handover takes place over the S1-interface as a UE transitions from the coverage of one eNB to the next. This differs from intra-enb handover as the UE is leaving the coverage of all sectors in an eNBs coverage, it is a handover to a new eNB. The following steps outline how this can be demonstrated using srsUE, srsENB and a third-party open source core. In this case the EPC from Open5GS is used. Other third party options would also work in this case, so long as they support S1 handover.

The following diagram outlines the network architecture:



Open5GS EPC

The Open5GS EPC is an open source core network solution which is inter-operable with srsRAN. The software can be installed from packages if using Ubuntu, as shown via the open5GS docs. The EPC, and the rest of the Open5GS applications, run out of the box and only require minor configuration for use with srsRAN.

EPC Set-Up

The EPC needs to be configured for use with srsRAN. The only changes required are to the MME configuration and adding the UE to the user database.

MME Config:

In the file mme.yaml, the TAC must be changed to 7, this is the standard configuration for srsRAN. You could also leave these settings as they are and configure the srsRAN elements instead.

The following shows the MME configuration used:

```
mme:
        freeDiameter: /etc/freeDiameter/mme.conf
        s1ap:
          - addr: 127.0.0.2
        gtpc:
          - addr: 127.0.0.2
        gummei:
          plmn_id:
                mcc: 901
                mnc: 70
          mme_gid: 2
          mme_code: 1
        tai:
          plmn_id:
                mcc: 901
                mnc: 70
          tac: 7
        security:
          integrity_order : [ EIA2, EIA1, EIA0 ]
          ciphering_order : [ EEA0, EEA1, EEA2 ]
        network_name:
          full: Open5GS
        mme_name: open5gs-mme0
```

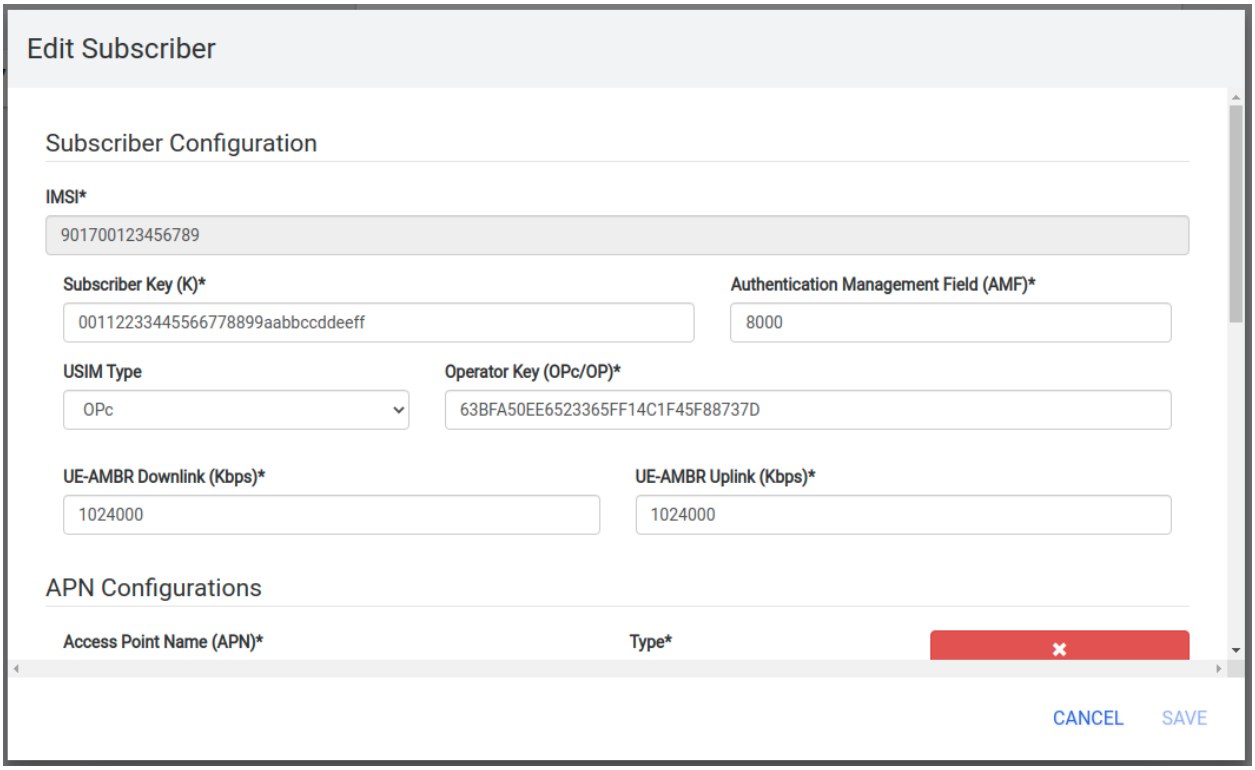
For reference, this configuration can be found from line 204 to 226.

Subscriber List:

Adding subscribers to the network is done via the web-UI provided by open5GS. Their documentation outlines how this is done here, under the section *Register Subscriber Information*.

First open the UI, found at http://localhost:3000, and enter the credentials found in the UE configuration file (ue.conf). The following credentials are used:

Note, the first five digits (PLMN) in the IMSI to 90170, and OPc (Milenage Authentication) is being used. This differs from the USIM configuration found in ue.conf, the changes made here will later be reflected in the ue.conf file. The IMSI is edited to reflect the values used for the MCC and MNC. Milenage is used here to show how the sim credentials can be changed to suit certain use-cases.



srsRAN Set-Up

To ensure srsRAN is correctly configured to implement S1 Handover, changes must be made to the UE and eNB configurations.

UE:

As previously outlined, the USIM credentials in the configuration file must be modified. The following sections taken from the config file show the sections that need to be modified:

```
[rf]
dl_earfcn = 2850
freq_offset = 0
tx_gain = 80
#rx_gain = 40

[usim]
mode = soft
algo = milenage
opc  = 63BFA50EE6523365FF14C1F45F88737D
k    = 00112233445566778899aabbccddeeff
imsi = 901700123456789
imei = 353490069873319
#reader =
#pin  = 1234
```

The downlink EARFCN is set to 2850 for this application, this is matched across the rest of the network. This sets the LTE Band and carrier frequency for the UE and eNB(s), they must match so that a connection can be successfully established and held. The changes made when adding the UE to the subscriber list in the EPC are also shown here,

the IMSI now leads with the correct PLMN code, and the authentication algorithm is set to milenage; the opc is uncommented to enable this.

eNB:

For the eNB config the PLMN must be changed, the MME address must also be changed to that of the MME associated with the Open5GS EPC. The following are the changes made to the enb.conf file:

```
[enb]
enb_id = 0x19B
mcc = 901
mnc = 70
mme_addr = 127.0.0.2
gtp_bind_addr = 127.0.1.1
s1c_bind_addr = 127.0.1.1
n_prb = 50
#tm = 4
#nof_ports = 2
```

eNB RR:

The rr.conf file must also be edited to allow for S1 Handover. To do this, two new rr.conf files are created, named rr1.conf and rr2.conf. As there will be two eNBs, there is an rr.conf associated with each. It is recommend that the existing rr.conf is simply copied into two new files, and only the *cell_list* changed for each of the new filles. This should help to avoid misconfiguration.

rr1.conf:

After the rr.conf has been copied to a new file (in the same location as the existing configuration files), the cell list must be edited. The following snippet shows this:

```
cell_list =
(
  {
    // rf_port = 0;
    cell_id = 0x01;
    tac = 0x0007;
    pci = 1;
    root_seq_idx = 204;
    dl_earfcn = 2850;
    // ul_earfcn = 474;
    ho_active = true;

    // CA cells
    scell_list = (
      // {cell_id = 0x02; cross_carrier_scheduling = false; scheduling_cell_id = 0x02;␣
↪ul_allowed = true}
    )

    // Cells available for handover
    meas_cell_list =
    (
      {
        eci = 0x19B01;
        dl_earfcn = 2850;
        pci = 1;
```

(continues on next page)

(continued from previous page)

```
      },
      {
        eci = 0x19C01;
        dl_earfcn = 2850;
        pci = 6;
      }
    );

    // ReportCfg (only A3 supported)
    meas_report_desc = {
      a3_report_type = "RSRP";
      a3_offset = 6;
      a3_hysteresis = 0;
      a3_time_to_trigger = 480;
      rsrq_config = 4;
    };
  }
);
```

Here the TAC is set to 7, and the DL EARFCN is set to 2850. To ensure S1 Handover is successful the cell(s) associated with the second eNB must be added to the *meas_cell_list*. This can be seen here where a cell with *eci = 0x19C01* is included, this is the cell associated with the second eNB. The cell with *eci = 0x19B01* is the cell active on the current eNB. The DL EARFCN is the same across both.

rr2.conf:

Similarly to rr1.conf, a file rr2.conf must be created where the other configuration files are found and the *cell_list* updated:

```
cell_list =
(
  {
    // rf_port = 0;
    cell_id = 0x01;
    tac = 0x0007;
    pci = 6;
    root_seq_idx = 268;
    dl_earfcn = 2850;
    // ul_earfcn = 474;
    ho_active = true;

    // CA cells
    scell_list = (
      // {cell_id = 0x02; cross_carrier_scheduling = false; scheduling_cell_id = 0x02;␣
↪ul_allowed = true}
    )

    // Cells available for handover
    meas_cell_list =
    (
      {
        eci = 0x19B01;
        dl_earfcn = 2850;
```

(continues on next page)

(continued from previous page)

```
        pci = 1;
      },
      {
        eci = 0x19C01;
        dl_earfcn = 2850;
        pci = 6;
      }
    );

    // ReportCfg (only A3 supported)
    meas_report_desc = {
      a3_report_type = "RSRP";
      a3_offset = 6;
      a3_hysteresis = 0;
      a3_time_to_trigger = 480;
      rsrq_config = 4;
    };
  }
);
```

It is possible to enable both intra-eNB and S1 handover at the same time by combining the rr configuration used for intra-enb HO with those shown above. Although, that will not be covered in this application note.

Using Scripts

To efficiently instantiate and run the network for S1 HO, Bash scripts will be employed. Scripts will be used to run the two eNBs and the UE. The scripts should be created in the same folder as the other configuration files to avoid any errors when passing file names and when running them.

eNB 1:

The first eNB will need to have ZMQ set as the RF device, and the ports assigned. As well as this, the new rr1.conf file must be set as the radio resource configuration to be used:

```
#!/bin/bash

LOG_ARGS="--log.all_level=debug"

PORT_ARGS="tx_port=tcp://*:2101,rx_port=tcp://localhost:2100"
ZMQ_ARGS="--rf.device_name=zmq --rf.device_args=\"${PORT_ARGS},id=enb,base_srate=23.04e6\
↪""

OTHER_ARGS="--enb_files.rr_config=rr1.conf"

sudo srsenb enb.conf ${LOG_ARGS} ${ZMQ_ARGS} ${OTHER_ARGS} $@
```

Note how the logging level is also set here using the script. Every argument in the configuration file can be changed via the command line when the eNB is instantiated, this shows how it is done when using a script with the logging as the example.

eNB 2:

For the second eNB we will need to set the ZMQ device, with the correct ports as above. The rr2.conf file must also be given as the rr configuration file to be used. Additional steps must be taken with this eNB so as to allow it to be

instantiated correctly. The eNB ID must be changed, and the GTP and S1C bind addresses must be modified. This is done with the following script:

```
#!/bin/bash

LOG_ARGS="--log.all_level=info"

PORT_ARGS="tx_port=tcp://*:2201,rx_port=tcp://localhost:2200"
ZMQ_ARGS="--rf.device_name=zmq --rf.device_args=\"${PORT_ARGS},id=enb,base_srate=23.04e6\
↪""

OTHER_ARGS="--enb_files.rr_config=rr2.conf --enb.enb_id=0x19C --enb.gtp_bind_addr=127.0.
↪1.2 --enb.s1c_bind_addr=127.0.1.2"

sudo srsenb enb.conf ${LOG_ARGS} ${ZMQ_ARGS} ${OTHER_ARGS} $@
```

UE:

The script for the UE will be used to set the ZMQ device and ports, while also being used to set-up the network namespace used for the UE:

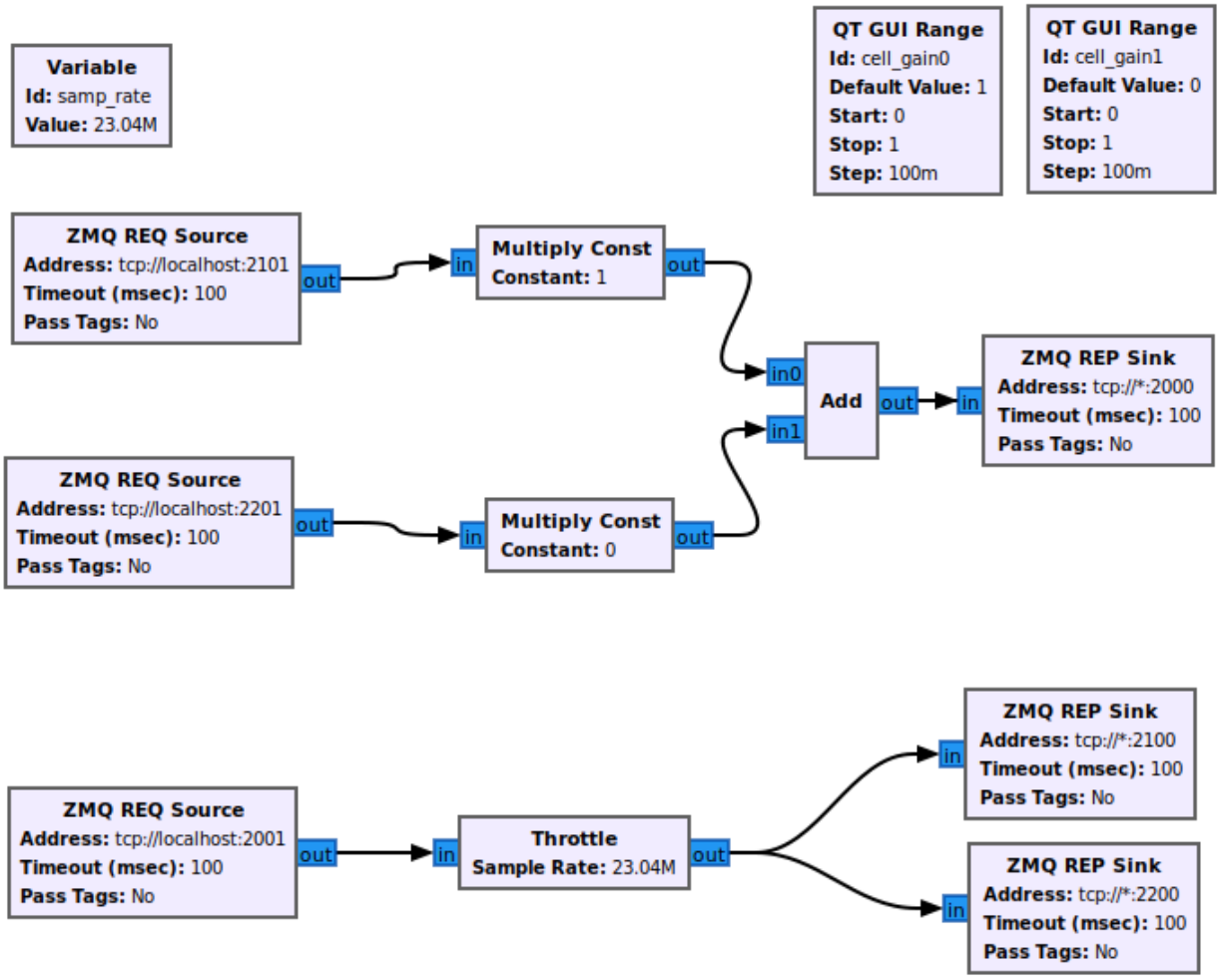
```
#!/bin/bash

LOG_PARAMS="--log.all_level=debug"

PORT_ARGS="tx_port=tcp://*:2001,rx_port=tcp://localhost:2000"
ZMQ_ARGS="--rf.device_name=zmq --rf.device_args=\"${PORT_ARGS},id=ue,base_srate=23.04e6\
↪" --gw.netns=ue1"


## Create netns for UE
ip netns list | grep "ue1" > /dev/null
if [ $? -eq 1 ]; then
  echo creating netspace ue1...
  sudo ip netns add ue1
  if [ $? -ne 0 ]; then
   echo failed to create netns ue1
   exit 1
  fi
fi

sudo srsue ue.conf ${LOG_PARAMS} ${ZMQ_ARGS} "$@"
```

The UE does not require any other parameters to be passed when it is instantiated.

GNU-Radio

The GRC file can be downloaded `here`. Download and/ or save the file as a *.grc* file. Run with GNU-Radio Companion when needed.

The GRC Broker used here is the same as that used for intra-eNB HO. The following figure shows the flowgraph used:



The following outlines which ports belong to which network element:

Table 6: Ports Used

Port Direction	eNB 1 Port #	eNB 2 Port #	UE Port #
Rx	2100	2200	2000
Tx	2101	2201	2001

Running the Network

To run the network the following steps must be taken:

1. Run the scripts to start each of the network elements
2. Run the GRC Broker to connect the UE to the eNB(s)

The eNB that the UE connects to first is known as the Source eNB, in this case it will be eNB 1. The Target eNB will be eNB 2, i.e. the eNB that the UE will be transferred to.

Confirming Connection

To confirm the initial connection has been successful look for the following readouts on the relevant consoles.

Source eNB:

```
---  Software Radio Systems LTE eNodeB  ---

Reading configuration file enb.conf...

Built in Release mode using commit 7e60d8aae on branch next.

Opening 1 channels in RF device=zmq with args="tx_port=tcp://*:2101,rx_port=tcp://
↪localhost:2100,id=enb,base_srate=23.04e6"
CHx base_srate=23.04e6"
CHx id=enb
Current sample rate is 1.92 MHz with a base rate of 23.04 MHz (x12 decimation)
CH0 rx_port=tcp://localhost:2100
CH0 tx_port=tcp://*:2101
Current sample rate is 11.52 MHz with a base rate of 23.04 MHz (x2 decimation)
Current sample rate is 11.52 MHz with a base rate of 23.04 MHz (x2 decimation)
Setting frequency: DL=2630.0 Mhz, UL=2510.0 MHz for cc_idx=0

==== eNodeB started ===
Type <t> to view trace
RACH:  tti=341, cc=0, preamble=38, offset=0, temp_crnti=0x46
User 0x46 connected
```

Target eNB:

```
---  Software Radio Systems LTE eNodeB  ---

Reading configuration file enb.conf...

Built in Release mode using commit 7e60d8aae on branch next.

Opening 1 channels in RF device=zmq with args="tx_port=tcp://*:2201,rx_port=tcp://
↪localhost:2200,id=enb,base_srate=23.04e6"
CHx base_srate=23.04e6"
CHx id=enb
Current sample rate is 1.92 MHz with a base rate of 23.04 MHz (x12 decimation)
CH0 rx_port=tcp://localhost:2200
CH0 tx_port=tcp://*:2201
```

(continues on next page)

(continued from previous page)

```
Current sample rate is 11.52 MHz with a base rate of 23.04 MHz (x2 decimation)
Current sample rate is 11.52 MHz with a base rate of 23.04 MHz (x2 decimation)
Setting frequency: DL=2630.0 Mhz, UL=2510.0 MHz for cc_idx=0

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

Note, you wont see anything on this eNB console until handover has successfully been made between the eNBs.

UE:

```
Reading configuration file ue.conf...

Built in Release mode using commit 7e60d8aae on branch next.

Opening 1 channels in RF device=zmq with args="tx_port=tcp://*:2001,rx_port=tcp://
↪localhost:2000,id=ue,base_srate=23.04e6"
CHx base_srate=23.04e6"
CHx id=ue
Current sample rate is 1.92 MHz with a base rate of 23.04 MHz (x12 decimation)
CH0 rx_port=tcp://localhost:2000
CH0 tx_port=tcp://*:2001
Waiting PHY to initialize ... done!
Attaching UE...
Current sample rate is 1.92 MHz with a base rate of 23.04 MHz (x12 decimation)
Current sample rate is 1.92 MHz with a base rate of 23.04 MHz (x12 decimation)
.
Found Cell:  Mode=FDD, PCI=1, PRB=50, Ports=1, CFO=-0.2 KHz
Current sample rate is 11.52 MHz with a base rate of 23.04 MHz (x2 decimation)
Current sample rate is 11.52 MHz with a base rate of 23.04 MHz (x2 decimation)
Found PLMN:  Id=90170, TAC=7
Random Access Transmission: seq=38, ra-rnti=0x2
Random Access Complete.     c-rnti=0x46, ta=0
RRC Connected

Network attach successful. IP: 10.45.0.7
nTp) 6/11/2020 15:36:1 TZ:0
```

You should now start to send traffic between the UE and the EPC, this is done via the following command:

```
sudo ip netns exec ue1 ping 10.45.0.1
```

This will stop the UE from timing out and keep the connection to the core open.

Forcing Handover

The network is now ready for handover to be forced, this is done in the same way as before using the GRC Broker:

1. Set the gain of the *Source eNB* from 1 to 0.5
2. Slowly increase the gain of the *Target eNB* from 0, to above 0.5, and on to 1.
3. Wait for handover to be acknowledged.
4. Move the gain of the *Source eNB* to 0.

If HO is successful the following will be seen on the relevant console outputs:

Source eNB:

```
Starting S1 Handover of rnti=0x46 to cellid=0x19c01.
Encoded varShortMAC: cellId=0x19b01, PCI=1, rnti=0x46 (7 bytes)
Disconnecting rnti=0x46.
```

Target eNB:

```
Received S1 HO Request
Received S1 MMEStatusTransfer
RACH:  tti=3421, cc=0, preamble=20, offset=0, temp_crnti=0x47
Disconnecting rnti=0x47.
User 0x46 connected
```

UE:

```
Received HO command to target PCell=6, NCC=2
Random Access Transmission: seq=20, ra-rnti=0x2
Random Access Complete.     c-rnti=0x46, ta=0
HO successful
```

This can be repeated as many times as needed by following the above steps.

Troubleshooting

Intra-eNB Handover

- If the gains of the cells are changed too abruptly the handover messages will not have enough time to be exchanged successfully. Gradually moving the sliders between values is best practice when changing the gain values.

S1 Handover

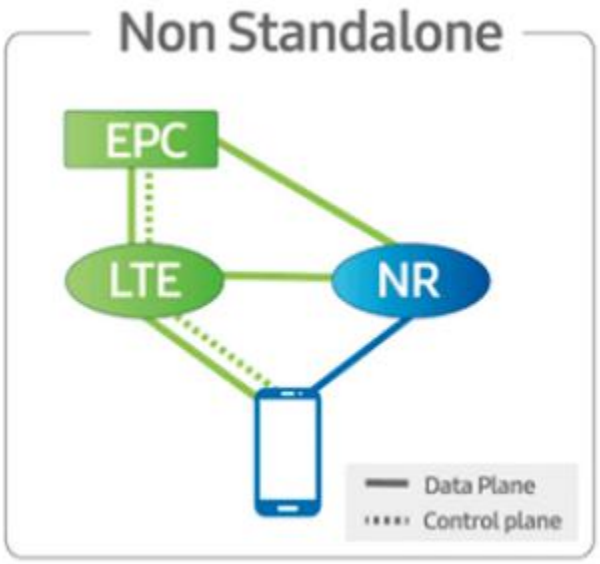
- Open5GS can also be installed from source, but it is easier to install from packages for this use-case.
- Ensure the PLMN, TAC and EARFCN are correct across all relevant network elements, as this can cause the connection to fail or stop an attach occuring.

2.3.9 5G NSA UE Application Note

Introduction

The 21.04 release of srsRAN brings 5G NSA (Non-Standalone) support to srsUE. This application note shows how the UE can be used with a third-party 5G NSA network. In this example, we use the Amari Callbox Classic from Amarisoft to provide the network.

5G NSA: What you need to know



5G Non-Standalone mode provides 5G support by building upon and using pre-existing 4G infrastructure. A secondary 5G carrier is provided in addition to the primary 4G carrier. A 5G NSA UE connects first to the 4G carrier before also connecting to the secondary 5G carrier. The 4G anchor carrier is used for control plane signaling while the 5G carrier is used for high-speed data plane traffic.

This approach has been used for the majority of commercial 5G network deployments to date. It provides improved data rates while leveraging existing 4G infrastructure. UEs must support 5G NSA to take advantage of 5G NSA services, but existing 4G devices are not disrupted.

Limitations

The current 5G NSA UE application has a few feature limitations that require certain configuration settings at both the gNB and the core network. The key feature limitations are as follows:

- 4G and NR carrier need to use the same subcarrier-spacing (i.e. 15 kHz) and bandwidth (we've tested 10 and 20 MHz)
- Support for NR in TDD mode for sub-6Ghz (FR1) in unpaired spectrum
- Only DCI format 0_0 (for Uplink) and 1_0 (for Downlink) supported
- No cell search and reference signal measurements (NR carrier PCI needs to be known)
- NR carrier needs to use RLC UM (NR RLC AM not yet supported)

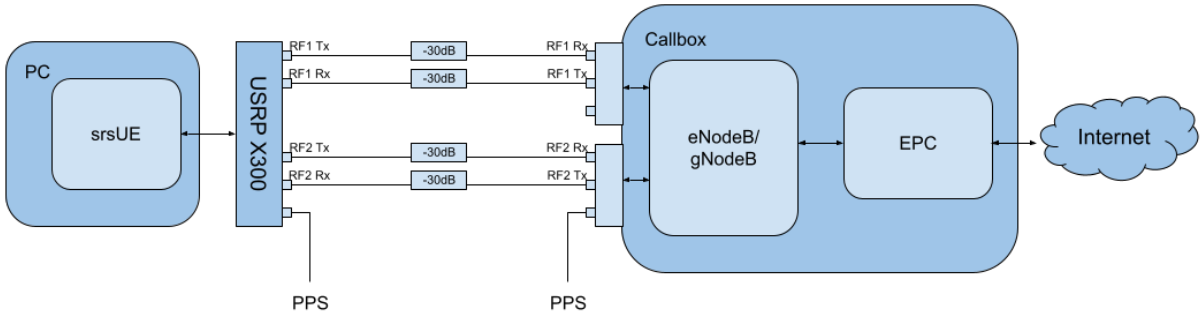
Hardware Requirements

For this application note, the following components are used:

- Amari Callbox with 5G NSA support as eNB/gNB and core
- AMD Ryzen5 3600X Linux PC as UE compute platform
- Ettus Research USRP X310 connected over 10GigE as UE RF front-end

The Amari Callbox is an LTE/NR SDR-based UE test solution from Amarisoft. It contains an EPC/5GC, an eNodeB, a gNodeB, an IMS server, an eMBMS server and an Intel i7 Linux PC with PCIe SDR cards. The gNodeB is release 15 compliant and supports both NSA and SA modes. A further outline of the specifications can be found in the data sheet. This test solution was chosen as it's widely available, easily configurable, and user-friendly.

Hardware Setup



Tests may be carried out over-the-air or using a cabled setup. For this example, we use a cabled setup between the UE and the eNB/gNB (i.e from the X310 to the PCIe SDR cards on the Callbox). These connections run through 30dB attenuators as shown in the figure above. The PPS inputs for the accurate clocking of both the UE and Callbox are also shown. Both UE and Callbox require accurate clocks - in our testing we provide PPS inputs to both.

Configuration

To set-up and run the 5G NSA network and UE, the configuration files for both the Callbox and srsUE must be changed.

All of the modified configuration files have been included as attachments to this App Note. It is recommended you use these files to avoid errors while changing configs manually. Any configuration files not included here do not require modification from the default settings.

UE files:

- `UE config example`

Callbox files:

- `MME config`
- `gNB NSA config`

srsUE

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

Firstly the following parameters need to be changed under the **[rf]** options so that the X310 is configured optimally:

```
[rf]
tx_gain = 10
nof_antennas = 1
device_name = uhd
device_args = type=x300,clock=external,sampling_rate=11.52e6,lo_freq_offset_hz=11.52e6
srate = 11.52e6
```

The next set of changes need to be made to the **[rat.eutra]** options. This make sure the anchor cell is found by the UE:

```
[rat.eutra]
dl_earfcn = 300
```

Finally the **[rat.nr]** options need to be configured for 5G NSA mode operation:

```
[rat.nr]
#enable 5G data link
nof_carriers = 1
```

Callbox

To correctly configure the Callbox changes must be made to the following files: *mme.cfg* and *gnb_nsa.cfg*.

MME Configuration

The *mme.cfg* file must be changed to reflect the QoS Class Identifier (QCI) which will be used across the network. We use QCI 7 as NR RLC UM is supported by the UE. The following change must be made to the *erabs:* configurations:

```
qci: 7,
```

gNB NSA Configuration

gnb_nsa.cfg is responsible for the configuration of both the LTE and NR cells needed for NSA mode. The LTE cell will mainly be used for the control plane, while the NR cell will be used for the data plane.

The number of Resource Blocks (RBs) and number of antennae used in the DL must first be modified:

```
#define N_RB_DL             50  // Values: 6 (1.4MHz), 25 (5MHz), 50 (10MHz), 75 (15MHz),
↪ 100 (20MHz)
#define N_ANTENNA_DL        1    // Values: 1 (SISO), 2 (MIMO 2x2), 4 (MIMO 4x4)
```

The NR cell bandwidth should also be set:

```
#define NR_BANDWIDTH        10   // NR cell bandwidth. With the PCIe SDR50 board, up to␣
↪50 MHz is supported.
```

The TX gain, sampling rates for each cell and the UL & DL frequencies for the NR cell must be set. The tx_gain is set for the *rf_driver:*:

```
tx_gain: 70.0, /* TX gain (in dB) */
```

The sample rate is set for the LTE cell in the *rf_ports:* configuration:

```
/* RF port for the LTE cell */
sample_rate: 11.52,
```

The sample rate and DL/UL frequencies are set for the NR cell in the *rf_ports:* configuration:

```
/* RF port for the NR cell */
sample_rate: 23.04,
dl_freq: 3507.84, // Moves NR DL LO frequency -5.76 MHz
ul_freq: 3507.84, // Moves NR UL LO frequency -5.76 MHz
```

The NR absolute radio-frequency channel number (ARFCN) for the DL needs to be changed to match the new DL frequency that has been set:

```
dl_nr_arfcn: 634240,  /* 3507.84 MHz */
```

Next, the default settings of the NR cell must be adjusted. The subcarrier spacing(s) should be changed in the *nr_cell_default:* configuration:

```
subcarrier_spacing: 15, /* kHz *
ssb_subcarrier_spacing: 30,
```

The timing offset should be set to 0:

```
n_timing_advance_offset: 0,
```

The TDD config options now need to be adjusted:

```
period: 10,
dl_slots: 6,
dl_symbols: 0,
ul_slots: 3,
ul_symbols: 0,
```

After this the PRACH configuration needs to be adjusted:

```
#if NR_TDD == 1
prach_config_index: 0,

msg1_frequency_start: 1,
zero_correlation_zone_config: 0,

ra_response_window: 10, /* in slots */
```

For the PDCCH configuration (starting at line 411), the following changes must be made:

```
pdcch: {
  common_coreset: {
    rb_start: -1, /* -1 to have the maximum bandwidth */
    l_crb: -1, /* -1 means all the bandwidth */
    duration: 1,
    precoder_granularity: "sameAsREG_bundle",
    //dmrs_scid: 0,
  },
```

(continues on next page)

(continued from previous page)

```
  dedicated_coreset: {
    rb_start: -1, /* -1 to have the maximum bandwidth */
    l_crb: -1, /* -1 means all the bandwidth */
    duration: 1,
    precoder_granularity: "sameAsREG_bundle",
    //dmrs_scid: 0,
  },

  css: {
    n_candidates: [ 1, 1, 1, 0, 0 ],
  },
  rar_al_index: 2,

  uss: {
    n_candidates: [ 0, 2, 1, 0, 0 ],
    dci_0_1_and_1_1: false,
    force_dci_0_0: true, // Forces DCI format 0_0 for Uplink
    force_dci_1_0: true, // Forces DCI format 1_0 for Downlink
  },
  al_index: 1,
},
```

For the PDSCH configuration the following change needs to be made:

```
k1: [ 8, 7, 6, 6, 5, 4],
```

QAM 64 must be selected for the Modulation Coding Scheme (MCS) table:

```
mcs_table: “qam64”,
```

In the PUCCH set-up frequency hopping needs to be turned off:

```
freq_hopping: false,
```

For the *pucch2* entry, the following settings can be selected, while the entries for *pucch3* and *pucch4* can be removed fully:

```
pucch2: {
  n_symb: 2,
  n_prb: 1,
  freq_hopping: false,
  simultaneous_harq_ack_csi: false,
  max_code_rate: 0.25,
},
```

The final changes to the configuration file are made to pusch settings:

```
pusch: {
  mapping_type: "typeA",
  n_symb: 14,
  dmrs_add_pos: 1,
  dmrs_type: 1,
```

(continues on next page)

(continued from previous page)

```
  dmrs_max_len: 1,
  tf_precoding: false,
  mcs_table: "qam64", /* without transform precoding */
  mcs_table_tp: "qam64", /* with transform precoding */
  ldpc_max_its: 5,
  k2: 4, /* delay in slots from DCI to PUSCH */
  p0_nominal_with_grant: -90,
  msg3_k2: 5,
  msg3_mcs: 4,
  msg3_delta_power: 0, /* in dB */
  beta_offset_ack_index: 9,

  /* hardcoded scheduling parameters */
  n_dmrs_cdm_groups: 1,
  n_layer: 1,
  /* if defined, force the PUSCH MCS for all UEs. Otherwise it is
  computed from the last received PUSCH. */
  /* mcs: 16, */
},
```

The Callbox should now be correctly configured for 5G NSA testing with srsUE.

Usage

Following configuration, we can run the UE and Callbox. The following order should be used when running the network:

1. MME
2. eNB/ gNB
3. UE

MME

To run the MME the following command is used:

```
sudo ltemme mme.cfg
```

eNB/ gNB

Next the eNB/ gNB should be instantiated, using the following command:

```
sudo lteenb gnb-nsa.cfg
```

Console output should be similar to:

```
LTE Base Station version 2021-03-15, Copyright (C) 2012-2021 Amarisoft
This software is licensed to Software Radio Systems (SRS).
Support and software update available until 2021-10-29.
```

(continues on next page)

(continued from previous page)

```
RF0: sample_rate=11.520 MHz dl_freq=2140.000 MHz ul_freq=1950.000 MHz (band 1) dl_ant=1␣
↪ul_ant=1
RF1: sample_rate=23.040 MHz dl_freq=3507.840 MHz ul_freq=3507.840 MHz (band n78) dl_
↪ant=1 ul_ant=1
```

UE

To run the UE, use the following command:

```
sudo srsue ue.conf
```

Once the UE has been initialised you should see the following:

```
Opening 2 channels in RF device=uhd with args=type=x300,clock=external,sampling_rate=11.
↪52e6,lo_freq_offset_hz=11.52e6,None
```

This will be followed by some information regarding the USRP. Once the cell has been found successfully you should see the following:

```
Found Cell:  Mode=FDD, PCI=1, PRB=50, Ports=1, CFO=0.1 KHz
Found PLMN:  Id=00101, TAC=7
Random Access Transmission: seq=17, tti=8494, ra-rnti=0x5
RRC Connected
Random Access Complete.     c-rnti=0x3d, ta=3
Network attach successful. IP: 192.168.4.2
Amarisoft Network (Amarisoft) 20/4/2021 23:32:40 TZ:105
RRC NR reconfiguration successful.
Random Access Transmission: prach_occasion=0, preamble_index=0, ra-rnti=0x7f, tti=8979
Random Access Complete.     c-rnti=0x4601, ta=23
---------Signal----------|-----------------DL-----------------|-----------UL-----------
rat  pci  rsrp  pl   cfo | mcs  snr  iter  brate  bler  ta_us | mcs   buff  brate  bler
lte    1   -52  13    12 |  19   40   0.5    15k    0%    7.3 |  16    0.0    10k    4%
 nr  500     4   0  881m |   2   31   1.0    0.0    0%    0.0 |  17    0.0   6.0k    0%
lte    1   -49   7  -4.8 |  28   40   0.5   1.4k    0%    7.3 |   0    0.0    0.0    0%
 nr  500     3   0  -5.9 |  27   35   1.0   1.3k    0%    0.0 |  28    0.0   148k    0%
lte    1   -58  16  -3.7 |  28   40   0.5   1.4k    0%    7.3 |   0    0.0    0.0    0%
 nr  500     3   0  -7.7 |  27   35   1.0   1.3k    0%    0.0 |  28    0.0   148k    0%
lte    1   -61  19  428m |  28   40   0.5   1.4k    0%    7.3 |   0    0.0    0.0    0%
 nr  500     4   0   2.2 |  27   30   1.4    67k    0%    0.0 |  28     28   143k    0%
lte    1   -61  19 -507m |  28   40   0.5   1.4k    0%    7.3 |   0    0.0    0.0    0%
 nr  500     4   0  924m |  27   24   1.9    18M    0%    0.0 |  28    0.0   3.7k    0%
lte    1   -61  19   3.8 |  28   40   0.5   1.4k    0%    7.3 |   0    0.0    0.0    0%
 nr  500     4   0   3.5 |  27   24   1.9    18M    0%    0.0 |   0    0.0    0.0    0%
lte    1   -61  19   3.8 |  28   40   0.5   1.4k    0%    7.3 |   0    0.0    0.0    0%
 nr  500     4   0   3.1 |  27   24   1.9    18M    0%    0.0 |   0    0.0    0.0    0%
```

To confirm the UE successfully connected, you should see the following on the console output of the **eNB**:

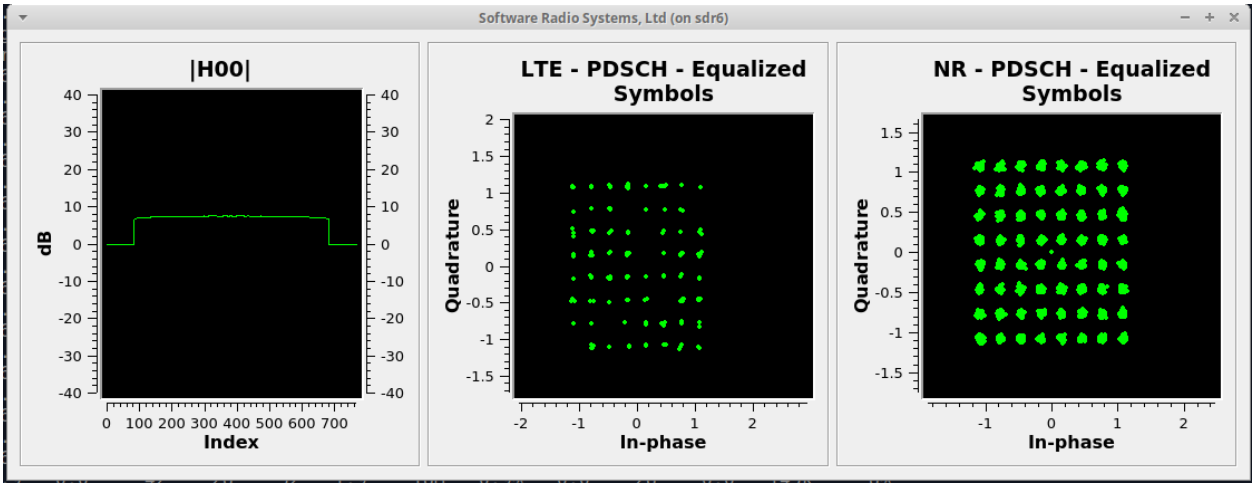
```
PRACH: cell=00 seq=17 ta=3 snr=28.3 dB
PRACH: cell=02 seq=0 ta=23 snr=28.3 dB
               ----DL----------------------- --UL-----------------------------------------
↪---------
```

(continues on next page)

(continued from previous page)

```
UE_ID  CL RNTI C cqi ri  mcs retx txok brate  snr puc1  mcs rxko rxok brate     #its phr␣
↪ pl   ta
   1 000 003d 1  15  1 15.0    0   16 5.58k 15.4 34.7 18.8    3   13 5.27k  1/3.7/6  31␣
↪ 38  0.0
   3 002 4601 1  15  1 27.0    0    1   320 36.2   -  27.7    0   87 64.0k  1/2.1/4   -␣
↪  - -0.3
   1 000 003d 1  15  1 28.0    0    4 1.42k 16.2 34.8 20.0    1    1   420  1/3.5/6  31␣
↪ 38  0.0
   3 002 4601 1  15  1 27.0    0    4 1.28k 28.1   -  28.0    0  200  148k  2/2.1/3   -␣
↪  - -0.3
   1 000 003d 1  15  1 28.0    0    4 1.42k 16.1 34.8    -    0    0     0        -  31␣
↪ 38  0.0
   3 002 4601 1  15  1 27.9    0 1037 16.8M 29.9   -  27.9    1   21 16.1k  1/2.3/5   -␣
↪  - -0.3
   1 000 003d 1  15  1 28.0    0    4 1.42k 16.3 35.2    -    0    0     0        -  31␣
↪ 38  0.0
   3 002 4601 1  15  1 27.9    5 1120 18.3M 29.9   -    -    0    0     0        -   -␣
↪  -    -
   1 000 003d 1  15  1 28.0    0    4 1.42k 16.0 34.8    -    0    0     0        -  31␣
↪ 38  0.0
   3 002 4601 1  15  1 27.9    0 1125 18.4M 29.9   -    -    0    0     0        -   -␣
↪  -    -
```

srsGUI Support



srsGUI is also supported for use with the UE in NSA mode. An example of the plots produced can be seen above.

To enable srsGUI, see here.

Note: If you have already built srsRAN without srsGUI support, you must re-do so after srsGUI has been built.

Understanding the console Trace

The console trace output from the UE contains useful metrics by which the state and performance of the UE can be measured. The traces can be activated by pressing t+Enter after UE has started. The following metrics are given in the console trace:

```
---------Signal----------|-----------------DL-----------------|-----------UL-----------
rat  pci  rsrp  pl   cfo | mcs  snr  iter  brate  bler  ta_us | mcs   buff  brate  bler
```

The following gives a brief description of which each column represents:

- **RAT:** This is a NSA specific column. It indicates the carrier for which the information is displayed.
- **PCI:** Physcial Cell ID
- **RSRP:** Reference Signal Receive Power (dBm)
- **PL:** Pathloss (dB)
- **CFO:** Carrier Frequency Offset (Hz)
- **MCS:** Modulation and coding scheme (0-28)
- **SNR:** Signal-to-Noise Ratio (dB)
- **ITER:** Average number of turbo decoder (LTE) or LDPC (NR) iterations
- **BRATE:** Bitrate (bits/sec)
- **BLER:** Block error rate
- **TA_US:** Timing advance (us)
- **BUFF:** Uplink buffer status - data waiting to be transmitted (bytes)

Troubleshooting

The UE currently doesn't support NR cell search and cell measurements. It therefore uses a pre-configured physical cell id (PCI) to send artificial NR cell measurements to the eNB. The reported PCI in those measurements is 500 by default (default value in Amarisoft configurations). If the selected PCI for the cell of interest is different, the value can we overwritten with:

```
$ ./srsue/src/srsue --rrc.nr_measurement_pci=140
```

Or by updating the **[rrc]** options in the config file:

```
[rrc]
nr_measurement_pci = 140
```

2.3.10 Suggested Hardware

This information is correct as of March 15th 2021

Introduction

This document aims to provide users with an overview of the suggested PC and SDR hardware combinations that can be used to best explore the functionality of srsRAN. There are 100's of possible combinations of PC, notebook, single board computer and SDR hardware that can demonstrate the uses of srsRAN. This list aims to provide three possible hardware packages that can help to guide users when choosing what to buy. These packages are grouped by price, with full set-ups cotsing >$400, >$3,000 and finally >$16,000. The three packages proposed here should provide any user with enough information to create their ideal set-up, which easily meets their needs.

Choosing Hardware

When choosing these packages we compared each hardware option under the same metrics. With one set for the computational hardware, and one for the SDR.

Compute Criteria

The following are the main specifications taken into account when selecting the compute platform for each of these packages:

- **Cost** - Overall cost of the machine
- **Number of cores** - This will affect overall performance
- **Processor frequency** - CPUs running at lower frequencies may struggle under heavy computational loads
- **Cache size** - A good indication of speed. More cache memory means certain computations will be faster.
- **Number of threads** - More threads will enable a processor to execute processes faster.

This is not an exhaustive list of criteria to look at when selecting a compute platform for SDR experimentation and development. Intended use-case will dictate choice the most here, as well as other external factors which can be subjective to either the user or overall use conditions.

Other useful things to take into account when choosing a compute platform for SDR research and experimentation are:

- **Processor Cinebench score** - This gives a good indication of a processor's ability to deal with high computational load. Find out more here.
- **Cooling ability** - More cooling ability will ensure CPU performance does not drop off significantly under heavy load
- **Portability** - Some use-cases may benefit from a PC that is portable

SDR Criteria

When selecting the SDR options to highlight we took the following into account:

- **Cost** - Cost per unit of the SDR
- **Driver** - Which driver the SDR uses (Soapy, UHD, etc)
- **Frequency range** - The frequency range(s) the SDR operates in
- **Bandwidth** - Maximum possible bandwidth available
- **Clock** - Clock rate
- **Channels** - The number of channels available (SISO, MIMO, etc)
- **FPGA** - The specifications of the onboard FPGA

Much like when choosing compute hardware, the metrics you may look at when choosing an SDR will vary depending on use-case and other factors. This list is in no way exhaustive, but provides a good platform by which to compare options.

Package Overview

Each package will contain a recommended SDR and compute hardware bundle. With some appropriate use-cases for each. A full end-to-end system will require at least two SDRs and two Compute platforms. As previously mentioned, these packages represent possible combinations, and are by no means a gold standard of the types of hardware needed for SDR experimentation.

Package 1

SDR	PC
Lime SDR mini	Raspberry Pi 4
Price: $159	**Price:** $75
Driver: SoapySDR	**# Cores:** 4
Frequency Range: 10 Mhz – 3.5 GHz	**Frequency:** 1.5 Ghz
RF Bandwidth: 30.72 Mhz	**Cache SIze:** 1 MiB
Clock: 30.72 MHz onboard VCTCXO	**# Threads:** 4
# Channels: 1x1	
FPGA: Intel Altera MAX 10	

This package is inspired by our *R. pi4 app note*.

Such a set-up would allow users to create a cheap end-to-end network, for under $400 without the need for a main PC. To run a full end-to-end system using the above equipment a user would need 3 Raspberry Pi4 units and 2 LimeSDR minis. A Pi4 is needed for the EPC, eNB and UE, and a front-end is needed for both the eNB and UE. Due to the small size and portability of the system this setup is ideal for on-the-fly demos and testing of networks and applications that don't require high-powered compute hardware or frontends.

Advantages

- Low cost
- Highly portable

Limitations

- Limited cell bandwidth (currently 5 MHz)
- Limited max bitrate in the UL

Package 2

SDR	PC
BladeRF micro 2.0 xA4	HP Omen 15 Intel i5-10300H
Price: $480	**Price:** $999.99
Driver: SoapySDR	**# Cores:** 4
Frequency Range: 47 Mhz – 6 Ghz	**Frequency:** 2.4 – 4.5 GHz
RF Bandwidth: 56 Mhz	**Cache SIze:** 8 MB
Clock: 38.4 MHz onboard VCTCXO	**# Threads:** 8
# Channels: 2x2	
FPGA: Altera Cyclone V (49 kLE)	

This offers a step up from the previous package; in price and performance. The BladeRF micro 2.0 xA4 offers users a 2X2 MIMO configuration, higher max bandwidth, a larger frequency range, and a larger FPGA. The HP Omen 15 is a gaming notebook, meaning it is built for high performance and high CPU load for a sustained period of time. The intel i5 10300H is the main draw here, having scored highly in the cinebench r20 benchmarking test. This set-up is considerably more expensive and would cost roughly $3000 for a full set up of 2 PCs and 2 frontends.

Advantages

- Easily portable, with improved performance
- Suits nearly any use-case

Limitations

- Single cell configuration but up to 20 MHz 2x2 MIMO
- Non-expandable Bandwidth and operating frequencies

Package 3

SDR	PC
Ettus x310	Dell Precision 3340 Workstation Intel i7-10700
Price: $6560	**Price:** $1349.00
Driver: UHD	**# Cores:** 8
Frequency Range: DC - 6GHz (w/ Daughter Cards)	**Frequency:** 2.9 - 4.8 GHz
RF Bandwidth: 160 MHz (w/ Daughter Cards)	**Cache SIze:** 16 MB
Clock: Configurable	**# Threads:** 16
# Channels: 2x2	
FPGA: KINTEX7-410T	

This system offers users the most potential in terms of RF-frontend capabilities on PC performance. The Ettus x310 offers users the largest frequency range, from DC to 6 GHz with the use of the appropriate daughter cards, a potential bandwidth of 160 MHz (requires the correct daughter cards), a multi-cell configuration and a powerful Kintex7 FPGA. The 3340 workstation offers an intel i7-10700 which is capable of high intensity computations without a significant drop off in performance over sustained periods of time. The workstation offers 10 Gbps ethernet connection, which allows users full utilization of the 10 Gbps connection available on the x310. A full E2E system would cost a total of roughly $15800.

Advantages

- Carrier Aggregation
- Multi-cell configuration

Limitations

- Not all PCs will be able to interface via 10Gb ethernet. May have to use adapters.

ZMQ

srsRAN has been designed with support for Zero-MQ. This is a “fake RF” driver, which allows users to set-up a virtual end-to-end network without the use of physical RF-hardware. This is a powerful tool for experimentations and development for users that do not have access to hardware, or for those who cannot purchase it.

ZMQ does not require large amounts of computational resources to run, meaning most PCs and notebooks (including the R. Pi4) can run it without sacrificing performance. ZMQ replaces the radio link between the eNB and UE, by creating a transmit and receive pipe for exchanging IQ samples TCP or IPC.

Our *ZMQ app note* clearly demonstrates how srsRAN can be used with ZMQ.