Installation Manual for Online Watcher for LTE (OWL)

Sneihil Gopal

PREP Associate, National Institute of Standards and Technology (NIST), USA
Department of Physics, Georgetown University, USA
Email: sneihil.gopal@nist.gov

In this article, we provide a complete installation guide for the open-source Long Term Evolution (LTE) control channel decoder called Online Watcher of LTE (OWL), developed by IMDEA Networks Institute [1], [2]. Additionally, this article includes steps for the preferable installation of the Universal Software Radio Peripheral (USRP) open-source toolchain (USRP Hardware Driver (UHD) and GNU Radio) and the Nuand BladeRF software. The aim of this project is to collect raw LTE traces using either a USRP or BladeRF, decode the LTE control channel information using OWL, and utilize the resource usage information extracted from the LTE control channel to (a) synthetically generate radio resource demand data corresponding to New Radio (NR), and (b) demonstrate the effectiveness of the Proactive Spectrum Adaptation Scheme (ProSAS) for spectrum sharing between LTE and NR by using the LTE and NR data.

I. Building and Installing the USRP Open-Source Toolchain (UHD and GNU Radio) on Ubuntu 20.04

We start by describing in detail the steps required to build and install the open-source toolchain for the Ettus USRP B210 (UHD and GNU Radio) on Ubuntu 20.04 [3]. We discuss the installation of UHD first, followed by that of GNU Radio.

- **Update and Install dependencies:**
  Before building UHD and GNU Radio, ensure that all the dependencies are installed. However, before installing any dependencies, make sure that all the packages that are already installed on the system are up-to-date.
  Update the system by running the following command:

```
sudo apt-get update
```

  Once the system has been updated, install the required dependencies for UHD and GNU Radio.

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

sudo apt-get -y install autoconf automake build-essential ccache
   ↪ cmake cpufrequtils doxygen ethtool fort77 g++ gir1.2-gtk
   ↪ -3.0 git gobject-introspection gpsd gpsd-clients inetutils-
   ↪ tools libasound2-dev libboost-all-dev libcomedi-dev
   ↪ libcppunit-dev libfftw3-bin libfftw3-dev libfftw3-doc
   ↪ libfontconfig1-dev libgmp-dev libgps-dev libgsl-dev
   ↪ liblog4cpp5-dev libncurses5 libncurses5-dev libpulse-dev
   ↪ libqt5opengl5-dev libqwt-qt5-dev libsdl1.2-dev libtool
   ↪ libudev-dev libusb-1.0-0 libusb-1.0-0-dev libusb-dev libxi-
   ↪ dev libxrender-dev libzmq3-dev libzmq5 ncurses-bin python3-
```

```
    ↪ cheetah python3-click python3-click-plugins python3-click-
    ↪ threading python3-dev python3-docutils python3-gi python3-
    ↪ gi-cairo python3-gps python3-lxml python3-mako python3-
    ↪ numpy python3-numpy-dbg python3-opengl python3-pyqt5
    ↪ python3-requests python3-scipy python3-setuptools python3-
    ↪ six python3-sphinx python3-yaml python3-zmq python3-ruamel.
    ↪ yaml swig wget
```

After installing the dependencies, reboot the system. If the installation of the dependencies completes without any errors, proceed to build and install UHD and GNU Radio.

- **Building and installing UHD from source code:**
  To build UHD from source code, clone the GitHub repository, check out the compatible release of the repository, and build and install using the following steps. Note that while building and installing UHD, no USRP device should be connected to the system.
  Begin by creating a folder to hold the repository.

```
cd $HOME
mkdir workarea
cd workarea
```

Next, clone the repository and change into the cloned directory.

```
git clone https://github.com/EttusResearch/uhd
cd uhd
```

After changing into the cloned directory, checkout UHD version 3.9[1].

```
git checkout UHD-3.9.LTS
```

Now create a build folder within the repository and invoke cmake.

```
cd host
mkdir build
cd build
cmake ..
```

Once the cmake command succeeds without errors, proceed to build UHD using `make`.

```
make
```

Run some basic tests to verify that the build process completed properly.

```
make test
```

Next, install UHD, using the default install prefix, which will install UHD under the `/usr/local/lib` folder. To do so, run this as root due to the permissions on that folder.

```
sudo make install
```

After installing UHD, update the system's shared library cache.

```
sudo ldconfig
```

Lastly, make sure that the `LD_LIBRARY_PATH` environment variable is defined and includes the folder under which UHD was installed. To do so, add the line below to the end of the `$HOME/.bashrc` file:

[1]Version 3.9 is chosen because it is compatible with OWL, which is installed later to decode LTE control channel information.

```
export LD_LIBRARY_PATH=/usr/local/lib
```

For this change to take effect, close the current terminal window, and open a new terminal.
At this point, UHD should be installed and ready to use. To check the installation, with no USRP device attached, run `uhd_find_devices`, which should give the following output:

```
linux; GNU C++ version 9.4.0; Boost_107100; UHD_003.009.007-6-
   ↪ g9ebbb8eb

No UHD Devices Found
```

- **Downloading the UHD FPGA Images:**
Run the following command to download the UHD FPGA Images for the installation.

```
sudo uhd_images_downloader
```

Note: Since this installation is being installed to a system level directory (e.g. `/usr/local`), the `uhd_images_downloader` command requires sudo privileges.

- **Installing GNU Radio:**
Installing gnuradio is simple and easy. With no USRP device connected to the machine, run the following command:

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

After installing GNU Radio, test the installation (again with no USRP device attached) by running the following commands.

```
gnuradio-config-info --version
gnuradio-config-info --prefix
gnuradio-config-info --enabled-components
```

Try launching the GNU Radio Companion (GRC) tool, a visual tool for building and running GNU Radio flowgraphs.

```
gnuradio-companion
```

- **Configuring USB:** On Linux, udev handles USB plug and unplug events. The following commands will install a udev rule so that non-root users may access the device as well. This setting should take effect immediately and does not require a reboot or logout/login. Again, be sure that no USRP device is connected via USB when running these commands.

```
cd $HOME/workarea/uhd/host/utils
sudo cp uhd-usrp.rules /etc/udev/rules.d/
sudo udevadm control --reload-rules
sudo udevadm trigger
```

- **Connect the USRP:** The installation of UHD and GNU Radio should now be complete. At this point, connect the USRP to the host computer. Try running `uhd_find_devices` and `uhd_usrp_probe`. Example output for `uhd_find_devices`:

```
linux; GNU C++ version 9.4.0; Boost_107100; UHD_003.009.007-6-
   ↪ g9ebbb8eb
```

```
--------------------------------------------------
-- UHD Device 0
--------------------------------------------------
Device Address:
   type: b200
   name: MyB210
   serial: 3188967
   product: B210
```

Example output for `uhd_usrp_probe`:

```
linux; GNU C++ version 9.4.0; Boost_107100; UHD_003.009.007-6-
   ↪ g9ebbb8eb

-- Detected Device: B210
-- Operating over USB 3.
-- Detecting internal GPSDO.... Found an internal GPSDO: GPSTCXO
   ↪ , Firmware Rev 0.929a
-- Initialize CODEC control...
-- Initialize Radio control...
-- Performing register loopback test... pass
-- Performing register loopback test... pass
-- Performing CODEC loopback test... pass
-- Performing CODEC loopback test... pass
-- Asking for clock rate 16.000000 MHz...
-- Actually got clock rate 16.000000 MHz.
-- Performing timer loopback test... pass
-- Performing timer loopback test... pass
-- Setting master clock rate selection to 'automatic'.
  _____________________________________________________
 /
| Device: B-Series Device
| _____________________________________________________
| /
| | Mboard: B210
| | revision: 4
| | product: 2
| | serial: 3188967
| | name: MyB210
| | FW Version: 8.0
| | FPGA Version: 13.0
| |
| | Time sources: none, internal, external, gpsdo
| | Clock sources: internal, external, gpsdo
| | Sensors: gps_gpgga, gps_gprmc, gps_time, gps_locked,
   ↪ gps_servo, ref_locked
| | _____________________________________________________
| | /
| | | RX DSP: 0
| | | Freq range: -8.000 to 8.000 MHz
```

```
| | _____________________________________________________________
| | /
| | | RX DSP: 1
| | | Freq range: -8.000 to 8.000 MHz
| | _____________________________________________________________
| | /
| | | RX Dboard: A
| | | _____________________________________________________________
| | | /
| | | | RX Frontend: A
| | | | Name: FE-RX2
| | | | Antennas: TX/RX, RX2
| | | | Sensors: temp, rssi, lo_locked
| | | | Freq range: 50.000 to 6000.000 MHz
| | | | Gain range PGA: 0.0 to 76.0 step 1.0 dB
| | | | Bandwidth range: 200000.0 to 56000000.0 step 0.0 Hz
| | | | Connection Type: IQ
| | | | Uses LO offset: No
| | | _____________________________________________________________
| | | /
| | | | RX Frontend: B
| | | | Name: FE-RX1
| | | | Antennas: TX/RX, RX2
| | | | Sensors: temp, rssi, lo_locked
| | | | Freq range: 50.000 to 6000.000 MHz
| | | | Gain range PGA: 0.0 to 76.0 step 1.0 dB
| | | | Bandwidth range: 200000.0 to 56000000.0 step 0.0 Hz
| | | | Connection Type: IQ
| | | | Uses LO offset: No
| | | _____________________________________________________________
| | | /
| | | | RX Codec: A
| | | | Name: B210 RX dual ADC
| | | | Gain Elements: None
| | _____________________________________________________________
| | /
| | | TX DSP: 0
| | | Freq range: -8.000 to 8.000 MHz
| | _____________________________________________________________
| | /
| | | TX DSP: 1
| | | Freq range: -8.000 to 8.000 MHz
| | _____________________________________________________________
| | /
| | | TX Dboard: A
| | | _____________________________________________________________
| | | /
| | | | TX Frontend: A
| | | | Name: FE-TX2
| | | | Antennas: TX/RX
```

```
| | | | Sensors: temp, lo_locked
| | | | Freq range: 50.000 to 6000.000 MHz
| | | | Gain range PGA: 0.0 to 89.8 step 0.2 dB
| | | | Bandwidth range: 200000.0 to 56000000.0 step 0.0 Hz
| | | | Connection Type: IQ
| | | | Uses LO offset: No
| | | ____________________________________________________
| | | /
| | | | TX Frontend: B
| | | | Name: FE-TX1
| | | | Antennas: TX/RX
| | | | Sensors: temp, lo_locked
| | | | Freq range: 50.000 to 6000.000 MHz
| | | | Gain range PGA: 0.0 to 89.8 step 0.2 dB
| | | | Bandwidth range: 200000.0 to 56000000.0 step 0.0 Hz
| | | | Connection Type: IQ
| | | | Uses LO offset: No
| | | ____________________________________________________
| | | /
| | | | TX Codec: A
| | | | Name: B210 TX dual DAC
| | | | Gain Elements: None
```

II. INSTALLING BLADERF ON UBUNTU 20.04

Next we describe the steps required to install the Nuand BladeRF software[2] on Ubuntu 20.04.

- **Update and Install dependencies:**
  Before installing BladeRF, ensure that all the dependencies are installed. However, before installing any dependencies, make sure that all the packages that are already installed on the system are up-to-date.
  Update the system by running the following command:

```
sudo apt-get update
```

- **Installing BladeRF:** Once the system has been updated, activate the BladeRF Personal Package Archive (PPA) as follows:

```
sudo add-apt-repository ppa:nuandllc/bladerf
sudo apt-get update
sudo apt-get install bladerf
```

  If you plan to build gnuradio, gr-osmosdr, etc, you will also need the header files:

```
sudo apt-get install libbladerf-dev
```

  Firmware and FPGA images can be installed from this PPA as well. Firmware should be manually updated using `bladeRF-cli --flash-firmware /usr/share/Nuand/bladeRF/bladeRF_fw.img`, but the FPGA image will be automatically loaded by libbladeRF when you open your device.

[2]For a complete installation guide see [4]

```
sudo apt-get install bladerf-firmware-fx3 # firmware for all
   ↪ models of bladeRF
sudo apt-get install bladerf-fpga-hostedx40 # for bladeRF x40
sudo apt-get install bladerf-fpga-hostedx115 # for bladeRF x115
sudo apt-get install bladerf-fpga-hostedxa4 # for bladeRF 2.0
   ↪ Micro A4
sudo apt-get install bladerf-fpga-hostedxa9 # for bladeRF 2.0
   ↪ Micro A9
```

- **Checking basic device operation:** Once the installation is complete see the guide [5] that shows how to exercise some very basic device operations using the `bladeRF-cli` program in order to verify that the device is functioning and that the required host software is installed.

III. Building and Installing IMDEA - OWL

Online Watcher of LTE (OWL) is a free and open-source LTE control channel decoder developed by IMDEA Networks Institute. It is based on `srsLTE` (now known as `srsRAN`), an LTE library for Software Defined Radio (SDR) User Equipment (UE) and Evolved Node B (eNodeB) developed by Software Radio Systems (SRS) [6]. OWL provides blind (oblivious of the terminal identity) decoding of Downlink Control Information (DCI) messages on LTE Physical Downlink Control Channel (PDCCH). It is built on srsLTE v1.3 (https://github.com/srsran/srsRAN) and inherits its modularity and main features. It is entirely written in C and, if available in the system, uses the acceleration library VOLK distributed in GNURadio.

- **Installing OWL:** Similar to the installation of UHD, GNU Radio, and Nuand BladeRF, start the installation of OWL by installing the dependencies.

```
sudo apt-get install build-essential git cmake libboost-system-
   ↪ dev libboost-test-dev libboost-thread-dev libqwt-dev libqt4
   ↪ -dev libfftw3-dev
```

Next, install `srsgui`. While this is not mandatory for OWL to work, but is a nice tool and it helps testing `srsLTE` and OWL:

```
git clone https://github.com/suttonpd/srsgui.git
cd srsgui
mkdir build
cd build
cmake ../
make
sudo make install
```

Lastly, clone and install OWL by running the below commands.

```
git clone https://github.com/cn0xroot/LTE.git
mv LTE imdeaowl
cd imdeaowl
mkdir build
cd build
cmake ../
make
```

If the installation succeeds, OWL's executables together with `srsLTE`'s examples can be found in the `srsLTE/build/srslte/examples` folder.

- **Working with the Executables:** OWL comprises of the following executable that can used to scan LTE bands, synchronize with base stations, emulate a UE, capture and decode LTE traces, etc.
  - `cell_search`:
    * This program is inherited from `srsRAN`. It scans an LTE band and tries to synchronize with the base station. To see the output of this program, run:

```
cd imdeaowl/build/srslte/examples
./cell_search
```

      The above command gives the following output:

```
linux; GNU C++ version 9.4.0; Boost_107100; UHD_003.009.007-6-
   ↪ g9ebbb8eb

Usage: ./cell_search [agsendtvb] -b band
   -a RF args [Default ]
   -g RF gain [Default 70.00 dB]
   -s earfcn_start [Default All]
   -e earfcn_end [Default All]
   -n nof_frames_total [Default 100]
   -v [set srslte_verbose to debug, default none]
```

      Refer to the website http://anisimoff.org/eng/lte_bands/usa.html for a list of frequency bands used by different operators in the US. On running `cell_search` on band 4, the following output is displayed:

```
Found 5 cells
Found CELL 2114.8 MHz, EARFCN=1998, PHYID=2, 15 PRB, 1 ports,
   ↪ PSS power=-19.3 dBm
Found CELL 2115.0 MHz, EARFCN=2000, PHYID=41, 50 PRB, 4 ports,
   ↪ PSS power=-13.7 dBm
Found CELL 2144.7 MHz, EARFCN=2297, PHYID=482, 50 PRB, 1 ports,
   ↪ PSS power=-25.1 dBm
Found CELL 2145.0 MHz, EARFCN=2300, PHYID=419, 100 PRB, 4 ports,
   ↪  PSS power=-20.4 dBm
Found CELL 2145.1 MHz, EARFCN=2301, PHYID=2, 75 PRB, 1 ports,
   ↪ PSS power=-21.8 dBm
```

  - `pdsch_ue`
    * Similar to `cell_search`, this program is inherited from `srsRAN`. It can be run on the frequencies found using `cell_search`. It emulates a UE trying to connect on a given frequency by first looking for synchronization and then decoding the control messages related to broadcast transmissions. In addition, it provides some useful statistics about synchronization and decoding success rate. To see the output of this program, run:

```
cd imdeaowl/build/srslte/examples
./pdsch_ue
```

      The above command gives the following output:

```
linux; GNU C++ version 9.4.0; Boost_107100; UHD_003.009.007-6-
   ↪ g9ebbb8eb
```

```
Usage: ./pdsch_ue [agpPoOcildDnruv] -f rx_frequency (in Hz) | -i
   ↪  input_file
   -a RF args [Default ]
   -g RF fix RX gain [Default AGC]
   -i input_file [Default use RF board]
   -o offset frequency correction (in Hz) for input file [
      ↪ Default 0.0 Hz]
   -O offset samples for input file [Default 0]
   -p nof_prb for input file [Default 25]
   -P nof_ports for input file [Default 1]
   -c cell_id for input file [Default 0]
   -r RNTI in Hex [Default 0xffff]
   -l Force N_id_2 [Default best]
   -C Disable CFO correction [Default Enabled]
   -t Add time offset [Default 0]
   -d disable plots [Default enabled]
   -D disable all but constellation plots [Default enabled]
   -n nof_subframes [Default -1]
   -s remote UDP port to send input signal (-1 does nothing with
      ↪  it) [Default -1]
   -S remote UDP address to send input signal [Default
      ↪ 127.0.0.1]
   -u remote TCP port to send data (-1 does nothing with it) [
      ↪ Default -1]
   -U remote TCP address to send data [Default 127.0.0.1]
   -v [set srslte_verbose to debug, default none]
```

On the found frequency, run the following command:

```
./pdsch_ue -f <freq>
```

where <freq> is the base station central frequency in Hertz, i.e. $2145e6$ or $2.134e9$ for band 4 found using cell_search. If the synchronization is successful, pdsch_ue will plot the constellations of the control channel and the shared downlink channel (only broadcast messages). If the signal is clean, a QPSK constellation in seen on both diagrams. In addition, the amplitude and phase channel responses are plotted together with the Primary Synchronization Signal (PSS) synchronization. The last one is okay if it looks like a Gaussian.

- imdea_capture_sync
  - This program captures a raw trace of the LTE channel synchronized on the beginning of the first subframe 0 detected. Run the command:

```
./imdea_capture_sync -f <freq> -l <cell_num> -n <subframe_num> -
   ↪ o <output_filename>
```

where <freq> is the base station central frequency in Hertz, i.e. 2.145 GHz, and can be given as $2145e6$ or $2.145e9$, <cell_num> is in $\{0,1,2\}$ and can be obtained from cell_search or pdsch_ue, <subframe_num> is the number of subframes to be recorded in the trace (1 subframe = 1 millisecond), and <output_filename> is the file where the trace is recorded. The trace can be then used with the other programs for offline processing. Note that putting -o /dev/null creates no output, but allows to test the signal synchronization without risking any buffer overrun. The output of the program is one line per frame (10 ms) and can be:

`Decoded MIB` (indicating good channel quality) `MIB not decoded` (indicating noise on the channel) and lastly, `sync loss` (indicating the channel is bad).

- `imdea_cc_decoder`
  - This program decodes the control channel and is the main part of OWL. It works both online and offline on pre-recorded traces.
    For online usage, run:

```
./imdea_cc_decoder -f <freq> -n <subframe_num> 1> <
  ↪ cc_out_filename> 2> /dev/null
```

    where, `<cc_out_filename>` specifies the location where the decoded control channel messages are saved. If omitted, the messages are printed to the `stdout`. Note that `2> /dev/null` redirects the `stderr`, which is used to produce the list of location to be checked by the fine-tuner. The output of the decoder is a tab separated list where each line represents a decoded message. The columns of the output are as follows:

    · System Frame Number (SFN): internal timing of LTE (1 every frame = 10 ms)
    · Subframe Index from 0 to 9 (1 subframe = 1 ms)
    · Radio Network Temporary Identifier (RNTI) in decimal
    · Direction: 1 = downlink; 0 = uplink
    · Modulation and Coding Scheme (MCS) in 0 - 31
    · Number of allocated resource blocks in 0 - 110
    · Transport block size in bits
    · Transport block size in bits (code word 0), -1 if n/a
    · Transport block size in bits (code word 1), -1 if n/a
    · Downlink Control Message (DCI) message type. This version only scans for 0 (format 0), 2 (format 1a), 6 (format 2a)
    · New data indicator toggle for codeword 0
    · New data indicator toggle for codeword 1
    · Hybrid Automatic Repeat Request (HARQ) process id
    · Narrowband Control Channel Element (NCCE) location of the DCI message
    · Aggregation level of the DCI message
    · Control Frame Indicator (CFI)
    · DCI correctness check

    Example `imdea_cc_decoder` is:

```
./imdea_cc_decoder -f 2145e6 -n 1000 1> output.tsv
```

    The `.tsv` (tab separated values) file generated by `imdea_cc_decoder` can be converted to `.csv` (comma separated values) file by using the following command:

```
sed 's/ /,/g' input_file.tsv > output_file.csv
```

    In case the output file generated on the Ubuntu machine needs to be transferred to a local Windows machine, follow the given instructions:

    · Setup the Ubuntu machine for SSH access.
    · Install PuTTY on Windows machine.
    · The PuTTY-GUI can be used to SSH-connect to the Ubuntu machine, but for file-transfer, one of the PuTTY tools called PSCP is required.
    · With PuTTY installed, set PuTTy's path so that PuTTY Secure Copy (PSCP) can be called from DOS command line.
    · PuTTY is installed with default settings (in C-drive). If PuTTY is installed in some other DIR, modify the below commands accordingly.

- On Windows DOS command prompt: a) set the path from Windows DOS command line(windows): type this command: `set PATH=C:\Program Files\PuTTY` b) check-/verify if PSCP is working from DOS command prompt: type this command: `pscp`.
- Lastly, run the following copy command:

```
pscp.exe linux-username@linux-server-ipaddress: [source-DIR-
   ↪ inLinux] [destination-DIR-inWin]
```

For offline usage of `imdea_cc_decoder`, run:

```
./imdea_cc_decoder -i <input_trace_filename> -l <cell_num> -c <
   ↪ pci> -P <ports> -p <prb> -z <rnti_out_filename> -Z <
   ↪ rnti_in_filename> 1> <cc_out_filename> 2> <cc_fix_filename
   ↪ >
```

where, `<input_trace_filename>` is the file containing the trace saved using the command `imdea_capture_sync`, `<pci>` is the physical cell id, `<ports>` is the antenna ports, and `<prb>` corresponds to the number of physical resource blocks of the LTE channel (all of these can be obtained using `cell_search`, `pdsch_ue`, and `imdea_capture_sync`). Lastly, we have `<rnti_in_filename>` and `<rnti_out_filename>` which are the rnti lists. Note that these lists are optional and if not provided the tool generates a new list. If available it starts with the information given. The format is a vector of 65355 elements, which can be 0 (not used), 1 (used, but not used in the last 10 seconds), 2 (used in the last 10 seconds). `<cc_fix_filename>` output file for the `fine_tuner` program. It specify one location to be checked per line. The columns are as follows:

- System Frame Number (SFN)
- Subframe index
- NCCE
- L
- CFI

– `imdea_fine_tuner`

* This is an offline tool that can be used to post-process the recorded trace to decode DCI messages in locations where it cannot be decoded by `imdea_cc_decoder`. To use this program run:

```
./imdea_fine_tuner -i <input_trace_filename> -l <cell_num> -c <
   ↪ pci> -P <ports> -p <prb> -z <cc_fix_filename> -Z <
   ↪ rnti_in_filename> 1> <cc_fixed_filename> 2> /dev/null
```

This program can only be used after `imdea_cc_decoder` on the output produced. It generates `<cc_fixed_filename>` with the same format of `<cc_out_filename>` (see above).

– `imdea_cc_decoder_graph`

* This program is identical to `imdea_cc_decoder`, but in addition it displays five graphs:
  - a spectrograph of the received power
  - average downlink frame resource usage
  - average downlink frame data rate
  - average uplink frame resource usage
  - average uplink frame data rate

For online use, run:

```
./imdea_cc_decoder_graph -f <freq> &> /dev/null
```

Both `stderr` and `stdout` are redirected to `/dev/null`, because they will produce the same output as `imdea_cc_decoder` does.

IV. Troubleshooting

- **Resetting the USRP**: In case you get the following error on running OWL:

```
Error opening RX stream: 30.
No compatible RF frontend found.
Error opening rf
```

reset the USRP by either unplugging the USB cable or run the following command:

```
/usr/local/lib/uhd/utils/b2xx_fx3_utils --reset-device
```

References

[1] N. Bui and J. Widmer, "OWL: a Reliable Online Watcher for LTE Control Channel Measurements," in *ACM All Things Cellular (MobiCom Workshop)*, Nov. 2016.

[2] Sep 2016. [Online]. Available: https://github.com/cn0xroot/LTE

[3] Feb 2022. [Online]. Available: https://kb.ettus.com/Building_and_Installing_the_USRP_Open-Source_Toolchain_(UHD_and_GNU_Radio)_on_Linux

[4] April 2023. [Online]. Available: https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Linux#user-content-Easy_installation_for_Ubuntu_The_bladeRF_PPA

[5] April 2022. [Online]. Available: https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Verifying-Basic-Device-Operation

[6] [Online]. Available: https://www.srsran.com/