Multi-UE Emulation using srsRAN + ZMQ + GNU Radio

Complete Installation & Execution Guide (Single Ubuntu System)

Reference Links Used

1. GitHub Demo (VM readiness, UE build, workflow)
   `https://github.com/devopsjourney23/my-srsproject-demo?tab=readme-ov-file#02-vm-machine-readine`
2. Official srsRAN Multi-UE Documentation
   `https://docs.srsran.com/projects/project/en/latest/tutorials/source/srsUE/source/index.html#multi-ue-emulation`

These references are followed exactly, with corrections based on real execution and debugging.

Project Overview (Read Once)

This project builds a fully virtual 5G network on a single Ubuntu machine using software-defined radio concepts.

There is no RF hardware:

- no USRP, SIM, Antenna

Everything runs in software.

Components Implemented

- 5G Core (5GC) → Open5GS (Docker-based)
- gNB (5G Base Station) → srsRAN Project
- UEs (Phones) → srsUE (from srsRAN_4G)
- RF Medium → ZeroMQ (ZMQ)
- Multi-UE RF fabric → GNU Radio

This setup validates:

- UE registration
- RRC & NAS procedures
- PRACH contention
- Scheduling of multiple UEs
- Paging capability
- End-to-end 5G signaling

System Assumptions

Item	Value
OS	Ubuntu 22.04.1 LTS (64-bit)
User	r-309
Home	/home/r-309
Disk	$\geq$ 50 GB
RAM	$\geq$ 8 GB
CPU	$\geq$ 4 cores

Directory Layout (Do NOT change)

```
/home/r-309/
 srsRAN_Project/  srsRAN Project (5G gNB)
  build/
  docker/  Open5GS 5GC

 srsRAN_4G/  srsRAN 4G (UE source)
  srsue/
  build/
  lib/

 srsRAN_config/  Runtime configs
  gnb_zmq.yaml
  ue1_zmq.conf
  ue2_zmq.conf
  ue3_zmq.conf
  multi_ue_scenario.grc

 /usr/bin/
    srsue  Installed UE binary
```

Why this separation matters:

- clean builds, no config pollution, Docker and RF stacks stay isolated

Phase 1 — VM / System Readiness

```
sudo apt update && sudo apt upgrade -y
```

Phase 2 — Install Required Packages

```
sudo apt install -y \
git cmake build-essential pkg-config \
libfftw3-dev libmbedtls-dev \
libboost-program-options-dev \
libconfig++-dev libsctp-dev \
libzmq3-dev \
gnuradio gnuradio-dev \
python3 python3-pip \
net-tools iproute2
```

Phase 3 — Docker Installation

Remove conflicting packages

```
for pkg in docker.io docker-doc docker-compose docker-compose-v2 podman-docker containerd runc; do
  sudo apt-get remove -y $pkg
done
```

Add Docker GPG key

```
sudo apt-get update
sudo apt-get install -y ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg \
-o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc
```

Add Docker repository

```
echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] \
https://download.docker.com/linux/ubuntu \
$(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
```

Install and enable Docker

```
sudo apt-get install -y docker-ce docker-ce-cli \
containerd.io docker-buildx-plugin docker-compose-plugin
sudo systemctl enable docker
sudo systemctl start docker
sudo usermod -aG docker r-309
```

Log out and log back in once.

Verify:

```
---------Signal-----------|-----------------DL-----------------|-----------UL-----------
rat pci rsrp pl cfo | mcs snr iter brate bler ta_us | mcs buff brate bler
 nr 1 43 0 -4.5u | 0 65 0.0 0.0 0% 0.0 | 0 0.0 0.0 0%
 nr 1 8 0 -12u | 0 n/a 0.0 0.0 0% 0.0 | 0 0.0 0.0 0%
```

5

docker run hello-world

Phase 4 — Build srsRAN Project (gNB)

```
cd ~/srsRAN_Project
git clone https://github.com/srsran/srsRAN_Project.git
cd srsRAN_Project
mkdir build && cd build
cmake .. -DENABLE_ZMQ=ON
make -j$(nproc)
sudo make install
sudo ldconfig
```

Verify:

```
gnb --help
```

Phase 4A — UE Installation (srsRAN_4G)

```
sudo apt-get install -y build-essential cmake \
libfftw3-dev libmbedtls-dev \
libboost-program-options-dev \
libconfig++-dev libsctp-dev \
git curl jq
```

```
cd ~
git clone https://github.com/srsRAN/srsRAN_4G.git
cd srsRAN_4G
mkdir build
cd build
cmake ../ -DENABLE_EXPORT=ON -DENABLE_ZEROMQ=ON
make -j$(nproc)
sudo cp ~/srsRAN_4G/build/srsue/src/srsue /usr/bin/srsue
sudo chmod +x /usr/bin/srsue
```

Verify:

```
srsue --help
```

Phase 4B — UE Network Namespaces (Optional)

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

Used for traffic isolation and testing.

Phase 5 — Fix /tmp Permissions

```
sudo chmod 1777 /tmp
```

Phase 6 — Start 5G Core (Open5GS)

Terminal-1

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

Leave running.

Phase 7 — Configuration Files

All configs are placed in:

```
~/srsRAN_config/
```

Phase 8 — Mandatory gNB Configuration Fix

```
prach:
  prach_config_index: 1
  total_nof_ra_preambles: 64
```

Do NOT add:

```
nof_ssb_per_ro
nof_cb_preambles_per_ssb
```

Phase 9 — Runtime Execution (6 Terminals)

Terminal	Component
T1	Open5GS 5GC
T2	gNB
T3	GNU Radio
T4	UE-1
T5	UE-2
T6	UE-3

Terminal-2 — gNB

```
cd ~/srsRAN_config
sudo pkill -9 gnb
sudo gnb -c gnb_zmq.yaml
```

Terminal-3 — GNU Radio

```
sudo pkill -9 python3
sudo pkill -f multi_ue_scenario
sudo gnuradio-companion ~/srsRAN_config/multi_ue_scenario.grc
```

Click **Run**.

Terminal-4 / 5 / 6 — UEs

```
sudo pkill -9 srsue
sudo srsue ue1_zmq.conf
sudo srsue ue2_zmq.conf
sudo srsue ue3_zmq.conf
```

Phase 10 — Successful UE Attachment

When a UE connects successfully, the following (or similar) output appears in the `srsUE` console:

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

Interpretation:

- Random Access Complete → PRACH success

- RRC Connected → UE in RRC_CONNECTED
- IP assigned → Core + UPF working
- RRC reconfiguration → NR connection finalized

Once an IP is assigned, the UE is fully attached.

Phase 11 — GNU Radio Control Panel

Path-loss Control (GNU Radio Control Panel)

When the flow graph is running, a control panel pops up automatically.

This panel allows:

- Per-UE path loss control (slider based)
- Time Slow Down Ratio control

Observing path-loss effects in UE

In any `srsUE` terminal, enable trace logging:

```
t
```

Example output while changing path loss:

```
---------Signal-----------|-----------------DL-----------------|-----------UL-----------
rat pci rsrp pl cfo | mcs snr iter brate bler ta_us | mcs buff brate bler
 nr 1 43 0 -4.5u | 0 65 0.0 0.0 0% 0.0 | 0 0.0 0.0 0%
 nr 1 8 0 -12u | 0 n/a 0.0 0.0 0% 0.0 | 0 0.0 0.0 0%
```

This shows:

- RSRP variation
- Impact of simulated distance/path loss
- Real-time PHY behavior

Time Slow Down Ratio (Performance Tuning)

- Controls how fast IQ samples are exchanged
- Default:
  - Sample rate: 11.52 MHz
  - Slow down ratio: 4
  - Effective rate: 11.52 / 4 MHz

Effect:

- Higher ratio → slower sample transfer
- Lower CPU usage
- Higher RTT (ping delay)

Useful when:

- Running many UEs
- System has limited CPU

Monitoring tools:

```
htop
nload lo
```

Testing the Network (Traffic)

After all UEs are attached:

- Each UE has its own network namespace (ue1, ue2, ue3)
- Default route must be set per namespace

You can now test:

- ping
- iperf
- DL/UL throughput

(As described in the Testing the Network section of the ZMQ-based setup in srsRAN docs.)

Phase 12 — Cleanup

```
sudo pkill -9 gnb
sudo pkill -9 srsue
sudo pkill -9 python3
sudo pkill -f multi_ue_scenario
```

Final Mental Model

```
Open5GS
   ↑
  N2/N3
   ↑
gNB
   ↑
 ZMQ
   ↑
GNU Radio
   ↑
UE-1   UE-2   UE-3
```