Design, Implementation, and Performance Evaluation of Network Slicing in a Simulated 5G Core Network By using Python

A Project report submitted in partial fulfilment
Of the requirements for the degree of
B.Tech in Electronics and communication Engineering

By

S.Jagadeeswar (2200049148)

Under supervision of

Dr. GSK Santhosh
Dr. Aravind Kilaru

Department of Electronics and Communication Engineering
K L E F, Green Fields,
Vaddeswaram- 522502, Guntur (Dist.), Andhra Pradesh, India.
April, 2025

Declaration

The Project Report entitled "Design, Implementation & Performance Evaluation of Network Slicing in a Simulated 5G Core Network using Python" is a record of bonafide work of **S.Jagadeeswar - 2200049148** submitted in partial fulfillment for the award of B.Tech in Electronics and Communication engineering to the K L University. The results embodied in this report have not been copied from any other departments/University/Institute.

Signature of the students:

S Jagadeeswar (2200049148)

Department of Electronics and Communication Engineering

KONERU LAKSHMAIAH EDUCATION FOUNDATION

VADDESWARAM- 522502, GUNTUR (DIST.), ANDHRA PRADESH

CERTIFICATE

To whom it may concern

This is to certify that the project work entitled "Design, Implementation & Performance Evaluation of Network Slicing in a Simulated 5G Core Network using Python " is the bonafide work carried out by **S Jagadeeswar - 2200049148**, student of B.Tech in the Dept. of Electronics and communication Engineering, KL University, during the academic year 2024-25, in partial fulfillment of the requirements for the degree of Bachelor of Technology and this project has not submitted previously for the award of any other degree, diploma and fellowship.

Table of contents:

Abstract

This project focuses on implementing and evaluating network slicing within a simulated 5G core network environment using Python and VirtualBox. The objective was to explore how effectively network slicing can support diverse service requirements by analyzing key performance metrics such as latency, data rates, and coverage range. To achieve this, two virtual machines were set up—one running Open5GS to emulate the 5G core network, and another configured with UERANSIM to simulate the user equipment (UE) and radio access network (RAN). The virtualized setup provided a controlled environment for replicating real-world 5G conditions, facilitating in-depth experimentation and observation.

The core network components deployed in this simulation included the Access and Mobility Management Function (AMF), Session Management Function (SMF), User Plane Function (UPF), Authentication Server Function (AUSF), Network Repository Function (NRF), and Network Slice Selection Function (NSSF). These elements were meticulously configured to support six distinct network slices, each defined by unique Slice/Service Type (SST) and Slice Differentiator (SD) values. The slices were designed to reflect a broad range of 5G service categories: enhanced Mobile Broadband (eMBB), High Definition Low Latency Communication (HDLLC), Ultra-Reliable Low-Latency Communication (URLLC), Massive Internet of Things (MIoT), massive Machine Type Communication (mMTC), and Vehicle-to-Vehicle communication (V2V). This diversity allowed for a comprehensive assessment of how slicing can accommodate varied application demands within a single infrastructure.

To evaluate performance, each slice was tested under simulated traffic loads tailored to its specific use case. Metrics such as low latency for URLLC, high throughput for eMBB, and device density for mMTC were used to measure the system’s effectiveness. The results demonstrated the core network's ability to maintain service isolation and optimize resource allocation across slices. This confirmed the practical advantages of network slicing in delivering differentiated Quality of Service (QoS), supporting the dynamic and heterogeneous needs of modern 5G applications while highlighting its potential for scalable and efficient future deployments.

Chapter 1 – Introduction

The 5th generation of mobile networks (5G) represents a major leap forward in telecommunications, enabling a wide array of advanced applications through enhanced performance in terms of data rates, latency, and device connectivity. Unlike previous generations, 5G is designed to support diverse service types such as **enhanced Mobile Broadband (eMBB), Ultra-Reliable Low-Latency Communications (URLLC), and massive Machine-Type Communications (mMTC).**

A central enabler of this flexibility is **network slicing**, which allows operators to create multiple virtual networks "slices" on top of a shared physical infrastructure. Each slice is configured to meet the specific Quality of Service (QoS) needs of an application or service type, ensuring isolation and efficient use of resources. The concept and architecture of network slicing are formally defined in **3GPP standards**, particularly in **TS 23.501, TS 23.502, and TS 28.530–28.538**, which outline the service-based architecture (SBA), slicing lifecycle management, and slice-specific configurations.

This project is based on the 3GPP-defined architecture and aims to design, implement, and evaluate network slicing in a simulated 5G core network. The implementation was carried out using **VirtualBox**, with two virtual machines—one running **Open5GS** (a 5G core network implementation) and the other running **UERANSIM** (for user equipment and RAN simulation). Core functions such as AMF, SMF, UPF, AUSF, NRF, and NSSF were deployed and configured to support six network slices, each defined using unique **SST (Slice/Service Type)** and **SD (Slice Differentiator)** values: "eMBB, hDLLC, URLLC, MIoT, mMTC, and V2V".

The project primarily focuses on performance evaluation of these slices under varying traffic conditions, with particular emphasis on **latency**, **data rate**, and **throughput**. By adhering to 3GPP slicing procedures and analyzing real traffic behavior, this work demonstrates the effectiveness of network slicing in achieving performance isolation and service differentiation in 5G networks.

Chapter 2 – Literature Survey

Network slicing is a pivotal concept in 5G architecture that enables the creation of multiple virtual networks over a shared physical infrastructure. Over the past decade, several studies have explored its feasibility, design, and implementation.

Zhang et al. (2017) presented a comprehensive review of network slicing for 5G networks, emphasizing its role in supporting diverse use cases such as enhanced Mobile Broadband (eMBB), massive Machine-Type Communications (mMTC), and Ultra-Reliable Low-Latency Communications (URLLC). Their work outlined the requirements for orchestration and management of network slices to meet stringent Quality of Service (QoS) metrics.

Foukas et al. (2017) introduced Orion, a slicing architecture that decouples the control and data planes, enabling flexible slice creation and resource allocation. Their research demonstrated how software-defined networking (SDN) and network function virtualization (NFV) are critical enablers of network slicing.

A study by Rost et al. (2016) proposed a framework for dynamic network slicing, which focuses on the elasticity and scalability of slices. They highlighted the importance of slice isolation, customization, and life-cycle management to cater to the demands of vertical industries.

Li et al. (2018) explored machine learning approaches for predictive slice orchestration. Their research showed that intelligent algorithms can significantly enhance resource efficiency by adapting slice configurations based on traffic prediction.

The 3rd Generation Partnership Project (3GPP) has also contributed significantly by standardizing the architecture for 5G network slicing. Their technical specifications (e.g., TS 23.501 and TS 28.530) define the reference models, slice types, and network functions necessary for end-to-end slice deployment.

These studies collectively establish that network slicing is not just a conceptual advancement but a technically and economically viable solution for delivering differentiated services in future wireless networks.

Chapter 3 – Network Slicing

Network slicing

- “Network slicing enables the creation of isolated, end-to-end networks that can run concurrently over a common 5G infrastructure”. These slices can be customized with different performance characteristics such as latency, bandwidth, and reliability, depending on the use case.

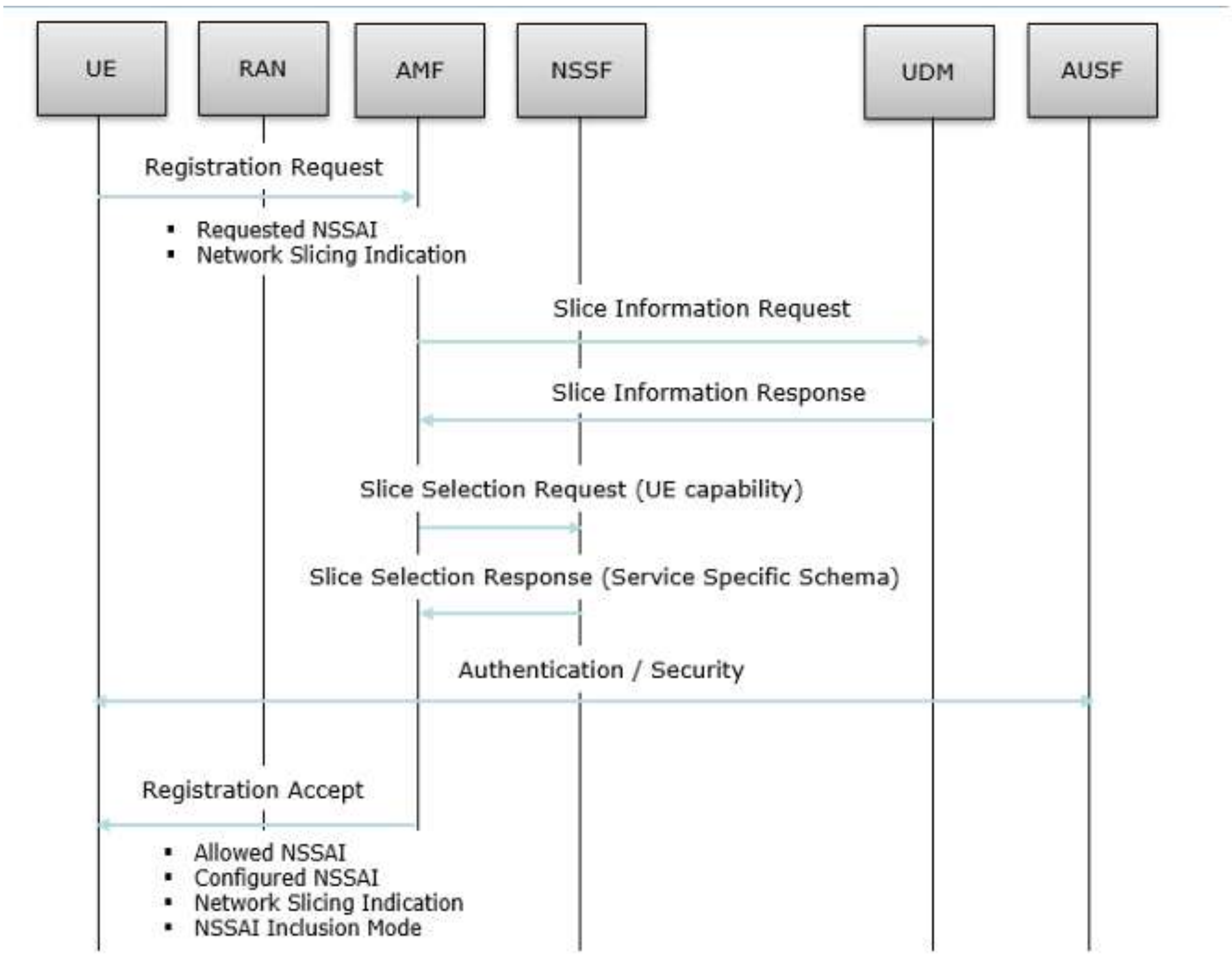
- It is a key feature in 5G networks that allows multiple virtual networks to be created on a shared physical infrastructure. Each “slice” is tailored to meet the specific needs of an application, service, or customer.

- **Importance of Network Slicing:**
  - **Customization**: Different services have different requirements.
  - **Efficiency**: Optimal use of resources by allocating only what's needed**.**
  - **Scalability**: Easy to scale or deploy new slices for emerging use cases.
  - **Security**: Slices are isolated from each other, reducing security risks.

- **Challenges in Network Slicing**
  - Complex orchestration and management.
  - Interoperability between vendors.
  - Security and isolation between slices.
  - Dynamic resource allocation.

- **Applications of Network Slicing**
  - Smart cities
  - Autonomous vehicles
  - Telemedicine
  - Virtual Reality (VR) and Augmented Reality (AR)
  - Industrial automation

Signalling for Network Slicing


UE
RAN
AMF
NSSF
UDM
AUSF
Registration Request
Requested NSSAI
Network Slicing Indication
Slice Information Request
Slice Information Response
Slice Selection Request (UE capability)
Slice Selection Response (Service Specific Schema)
Authentication / Security
Registration Accept
Allowed NSSAI
Configured NSSAI
Network Slicing Indication
NSSAI Inclusion Mode

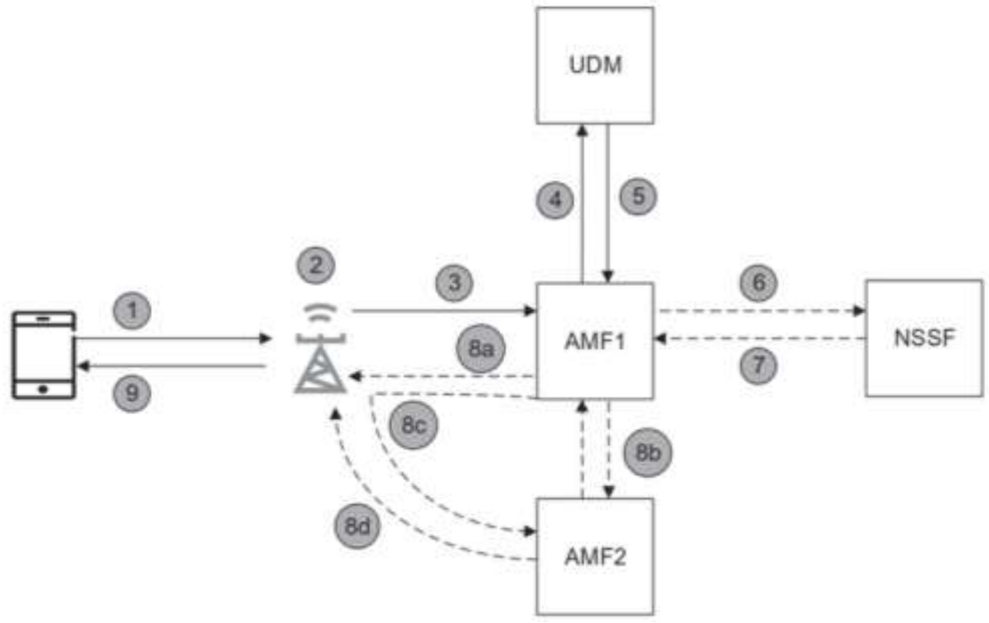
Keywords:

- **UE**: User Equipment
- **RAN**: Radio Access Network
- **AMF**: Access and Mobility Management Function
- **NSSF**: Network Slice Selection Function
- **UDM**: Unified Data Management
- **AUSF**: Authentication Server Function

Signal Route:

1. The UE sends a registration request with its slice preferences (NSSAI).
2. The AMF checks available slices by contacting the NSSF.
3. NSSF provides slice options based on UE capability and network policy.
4. AMF performs authentication and selects the appropriate network slice.
5. The UE receives confirmation and connects using the selected slice.

Registration procedure



1. **UE initiates connection request** to the network (e.g., Registration Request).
2. The **gNB forwards the request** to an AMF (initially AMF1).
3. AMF1 processes the initial request.
4. **AMF1 queries UDM** to retrieve user subscription and slicing information.
5. **UDM provides slice-related data** to AMF1.
6. AMF1 consults **NSSF** to determine the appropriate network slice based on policies and user data.
7. **NSSF responds** with slice selection information (e.g., which slice(s) to use, or if another AMF is better suited).
8. If needed, **AMF1 transfers the session** to a different AMF (AMF2) based on NSSF guidance:
   - **8a**: gNB receives instructions.
   - **8b**: AMF1 transfers context to AMF2.
   - **8c & 8d**: gNB re-establishes the connection with AMF2.
9. UE is now **connected via the correct slice** through the appropriate AMF.

Chapter 4 - Installation of open5gs

Links for open5gs ,UERANSIM provided in References.

Building Open5GS from Sources

Getting MongoDB

Step1:

```
$ sudo apt update
```

Updates the package list on your system.

Step2:

```
$ sudo apt install gnupg
```

Installs GNU Privacy Guard (GPG) for managing keys.

Step3:

```
$ curl -fsSL https://pgp.mongodb.com/server-8.0.asc | sudo gpg -o
/usr/share/keyrings/mongodb-server-8.0.gpg --dearmor
```

Downloads and stores MongoDB's GPG key in a secure format for verifying packages.

Step4:

```
$ echo "deb [ arch=amd64,arm64 signed-by=/usr/share/keyrings/mongodb-server8.0.gpg]
https://repo.mongodb.org/apt/ubuntu jammy/mongodb-org/8.0 multiverse"
|sudo tee /etc/apt/sources.list.d/mongodb-org-8.0.list
```

Adds the MongoDB 8.0 APT repository to the system's sources list with its GPG key for package authentication.

Install the MongoDB packages

Step1:

```
$ sudo apt update
```

→ Refreshes the package index to include the newly added MongoDB repository.

Step2:

```
$ sudo apt install -y mongodb-org
```

→ Installs MongoDB 8.0 and its related packages without prompting for confirmation.

Step3:

```
$ sudo systemctl start mongod
```

Starts the MongoDB server if it’s not already running.

Step4:

```
$ sudo systemctl enable mongod
```

Configures MongoDB to automatically start on system boot.

Ubuntu

Step1:

```
$ sudo add-apt-repository ppa:open5gs/latest
```

Adds the Open5GS software source to your system.

Step2:

```
$ sudo apt update
$ sudo apt install open5gs
```

Installs the Open5GS core network software.

Debain:

```
$ sudo apt update
```

Updates the local package index before adding new sources.

Step2:

```
$ wget -qO -
https://download.opensuse.org/repositories/home:/acetcom:/open5gs:/latest/Debian_10/
Release.key | sudo apt-key add -
```

Adds the repository's GPG key to authenticate packages.

Step3:

```
$ sudo sh -c "echo 'deb
  http://download.opensuse.org/repositories/home:/acetcom:/open5gs:/latest/Debian
  _10/./'>/etc/apt/sources.list.d/open5gs.list"
```

Adds the Open5GS repository to your system's source list.

Step4:

$ sudo apt update

$ sudo apt install open5gs

Installs the Open5GS core software from the new repository.

NightlyBuild

Step1:

$ sudo apt update

Updates the package list before installing anything

Step2:

$ sudo apt install wget gnupg

Installs tools needed to download and verify repository keys.

Step3:

```
wget-qO
https://downloads.osmocom.org/packages/osmocom:/nightly/xUbuntu_20.04
/Release.key | sudo apt-key add -
```

Adds the Osmocom repo's GPG key to trust its packages.

Step4:

```
$ sudo sh -c "echo 'deb
 https://downloads.osmocom.org/packages/osmocom:/nightly/xUbuntu_20.04/    ./'
 > /etc/apt/sources.list.d/open5gs.list"
```

Adds the Osmocom nightly Open5GS repo to your sources.

Step5:

$ sudo apt update

Refreshes the package index to include Osmocom repo packages.

Step6:

$ sudo apt install open5gs

→ Installs Open5GS from the Osmocom nightly repository.

openSUSE

Here we want to install ZYPPER so, use the command as **“sudo apt install zypper”,**then continue to steps of openSUSE.

step1:

```
$ sudo zypper addrepo -f obs://home:mnhauke:open5gs home:mnhauke:open5gs
```

Adds the Open5GS repository for openSUSE systems.

Step2:

```
$ sudo zypper install mongodb-server mongodb-shell
```

Installs MongoDB, which Open5GS uses as its database.

Step3:

```
$ sudo zypper install open5gs
```

Installs the Open5GS core network components.

Install the WebUI of Open5GS

Download and import the Nodesource GPG key

```
$ sudo apt update
$ sudo apt install -y ca-certificates curl gnupg
$ sudo mkdir -p /etc/apt/keyrings
$ curl -fsSL https://deb.nodesource.com/gpgkey/nodesource-repo.gpg.key | sudo gpg --dearmor -o /etc/apt/keyrings/nodesource.gpg
```

Create deb repository

```
$ NODE_MAJOR=20
$echo"deb[signed-by=/etc/apt/keyrings/nodesource.gpg] https://deb.nodesource.com/node_$NODE_MAJOR.x nodistro main" | sudo tee /etc/apt/sources.list.d/nodesource.list
```

Run Update and Install

```
$ sudo apt update
$ sudo apt install nodejs -y
```

Step1:

$ sudo apt update

Refreshes the package list before making changes.

Step2:

$ sudo apt install -y ca-certificates curl gnupg → Installs tools needed for secure downloads and key management.

Step3:

$ sudo mkdir -p /etc/apt/keyrings → Creates a directory to securely store APT keyrings.

Step4:

$ curl -fsSL https://... | sudo gpg --dearmor -o /etc/apt/keyrings/nodesource.gpg
→ Downloads and saves the NodeSource GPG key for verifying packages.

Step5:

$ NODE_MAJOR=20 → Sets the Node.js major version to install (v20 in this case).

Step6:

$ echo "deb [...]" | sudo tee /etc/apt/sources.list.d/nodesource.list→ Adds the NodeSource repo for the specified Node.js version.

Step7:

$ sudo apt update → Updates the package list to include the NodeSource repo.

Step8:

$ sudo apt install nodejs -y → Installs Node.js from the NodeSource repository.

To install Node.js on *openSUSE*, run the following:

$ sudo zipper install nodejs8

Installs Node.js version 8 on an openSUSE system using Zypper.

You can now install WebUI of Open5GS.

$ curl -fsSL https://open5gs.org/open5gs/assets/webui/install | sudo -E bash -

Downloads and runs the official Open5GS WebUI installation script with root privileges.

After changing config files, please restart Open5GS daemons.

Step1:

$ sudo systemctl restart open5gs-mmed

$ sudo systemctl restart open5gs-sgwud

Step2:

Restarts the MME (Mobility Management Entity) service in Open5GS.

→ Restarts the SGWU (Serving Gateway User Plane) service in Open5GS.

After changing config files, please restart Open5GS daemons.

Step3:

$ sudo systemctl restart open5gs-nrfd

$ sudo systemctl restart open5gs-amfd

$ sudo systemctl restart open5gs-upfd

Restarts the NRF (Network Repository Function) service in Open5GS.

Restarts the AMF (Access and Mobility Management Function) service.

Restarts the UPF (User Plane Function) service.

Register Subscriber Information

Connect to http://localhost:9999 and login with **admin** account.

Username : admin
Password : 1423

Tip: You can change the password in *Account* Menu.

To add subscriber information, you can do WebUI operations in the following order:

1. Go to Subscriber Menu.
2. Click + Button to add a new subscriber.
3. Fill the IMSI, security context(K, OPc, AMF), and APN of the subscriber.
4. Click SAVE Button

Enter the subscriber details of your SIM cards using this tool, to save the subscriber profile in the HSS and UDR MongoDB database backend. If you are using test SIMs, the details are normally printed on the card.

Adding a route for the UE to have WAN connectivity

In order to bridge between the PGWU/UPF and WAN (Internet), you must enable IP forwarding and add a NAT rule to your IP Tables.

To enable forwarding and add the NAT rule, enter

Enable IPv4/IPv6 Forwarding

$ sudo sysctl -w net.ipv4.ip_forward=1

$ sudo sysctl -w net.ipv6.conf.all.forwarding=1

Add NAT Rule

$ sudo iptables -t nat -A POSTROUTING -s 10.45.0.0/16 ! -o ogstun -j MASQUERADE

$ sudo ip6tables -t nat -A POSTROUTING -s 2001:db8:cafe::/48 ! -o ogstun -j MASQUERADE

Step1:

sudo sysctl -w net.ipv4.ip_forward=1
Enables IPv4 packet forwarding for network traffic routing.

Step2:

sudo sysctl -w net.ipv6.conf.all.forwarding=1
Enables IPv6 packet forwarding for network traffic routing.

Step3:

sudo iptables -t nat -A POSTROUTING -s 10.45.0.0/16 ! -o ogstun -j MASQUERADE
Adds a NAT rule to masquerade (hide) the source IP address for outbound IPv4 traffic not going out the ogstun interface.

Step4:

sudo ip6tables -t nat -A POSTROUTING -s 2001:db8:cafe::/48 ! -o ogstun -j MASQUERADE
Adds a similar NAT rule for outbound IPv6 traffic, excluding ogstun.

Configure the firewall correctly. Some operating systems (Ubuntu) by default enable firewall rules to block traffic.

```
$ sudo ufw status
Status: active
$ sudo ufw disable
Firewall stopped and disabled on system startup
$ sudo ufw status
Status: inactive
```

Step1:

sudo ufw status
Displays the current status of the UFW (Uncomplicated Firewall), showing whether it's active or inactive.

Step2:

sudo ufw disable
Stops and disables the UFW firewall, preventing it from starting on system boot.

Step3:

sudo ufw status
Confirms that the firewall is now inactive after being disabled.

Starting and Stopping Open5GS

```
$ sudo systemctl stop open5gs-mmed
$ sudo systemctl stop open5gs-sgwcd
$ sudo systemctl stop open5gs-smfd
$ sudo systemctl stop open5gs-amfd
$ sudo systemctl stop open5gs-sgwud
$ sudo systemctl stop open5gs-upfd
$ sudo systemctl stop open5gs-hssd
$ sudo systemctl stop open5gs-pcrfd
$ sudo systemctl stop open5gs-nrfd
$ sudo systemctl stop open5gs-scpd
$ sudo systemctl stop open5gs-seppd
$ sudo systemctl stop open5gs-ausfd
$ sudo systemctl stop open5gs-udmd
$ sudo systemctl stop open5gs-pcfd
$ sudo systemctl stop open5gs-nssfd
$ sudo systemctl stop open5gs-bsfd
$ sudo systemctl stop open5gs-udrd
$ sudo systemctl stop open5gs-webui
```

- Stops the MME (Mobility Management Entity) service for 4G core.
- Stops the SGWC (Serving Gateway Control) service.
- Stops the SMF (Session Management Function) service.
- Stops the AMF (Access and Mobility Management Function) service for 5G core.
- Stops the SGWU (Serving Gateway User Plane) service.
- Stops the UPF (User Plane Function) service.
- Stops the HSS (Home Subscriber Server) service.
- Stops the PCRF (Policy and Charging Rules Function) service.
- Stops the NRF (Network Repository Function) service.
- Stops the SCP (Service Communication Proxy) service.
- Stops the SEPP (Security Edge Protection Proxy) service.
- Stops the AUSF (Authentication Server Function) service.
- Stops the UDM (Unified Data Management) service.
- Stops the PCF (Policy Control Function) service.
- Stops the NSSF (Network Slice Selection Function) service.
- Stops the BSF (Bootstrapping Server Function) service.
- Stops the UDR (Unified Data Repository) service.
- Stops the Open5GS WebUI interface for network management.

```
$ sudo systemctl restart open5gs-mmed
$ sudo systemctl restart open5gs-sgwcd
$ sudo systemctl restart open5gs-smfd
$ sudo systemctl restart open5gs-amfd
$ sudo systemctl restart open5gs-sgwud
$ sudo systemctl restart open5gs-upfd
$ sudo systemctl restart open5gs-hssd
$ sudo systemctl restart open5gs-pcrfd
$ sudo systemctl restart open5gs-nrfd
$ sudo systemctl restart open5gs-scpd
$ sudo systemctl restart open5gs-seppd
$ sudo systemctl restart open5gs-ausfd
$ sudo systemctl restart open5gs-udmd
$ sudo systemctl restart open5gs-pcfd
$ sudo systemctl restart open5gs-nssfd
$ sudo systemctl restart open5gs-bsfd
$ sudo systemctl restart open5gs-udrd
$ sudo systemctl restart open5gs-webui
```

→ Restarts the MME service for 4G mobility and session management.
→ Restarts the SGW-C (Serving Gateway Control Plane) service.
→ Restarts the SMF (Session Management Function) service for 5G.
→ Restarts the AMF (Access and Mobility Management Function) service for 5G.
→ Restarts the SGW-U (Serving Gateway User Plane) service.
→ Restarts the UPF (User Plane Function) service for data forwarding.
→ Restarts the HSS (Home Subscriber Server) for 4G subscriber management.
→ Restarts the PCRF (Policy and Charging Rules Function) service.
→ Restarts the NRF (Network Repository Function) for service discovery.
→ Restarts the SCP (Service Communication Proxy) service.
→ Restarts the SEPP (Security Edge Protection Proxy) service for secure inter-PLMN.
→ Restarts the AUSF (Authentication Server Function) service.
→ Restarts the UDM (Unified Data Management) service.
→ Restarts the PCF (Policy Control Function) service.
→ Restarts the NSSF (Network Slice Selection Function) service.
→ Restarts the BSF (Bootstrapping Server Function) service.
→ Restarts the UDR (Unified Data Repository) service.
→ Restarts the WebUI for managing Open5GS components via a browser interface.

Chapter 5 - Installation steps for ueransim

Step1:

$ Sudo apt install git

It is used to install the git library file

Step2:

$ cd ~ git clone https://github.com/aligungr/UERANSIM

Get the source code for 5G core network (Open5GS) and UE simulation (UERANSIM).

step3:

$ sudo apt update

$ sudo apt upgrade

step4:

$ sudo apt install make gcc g++ libsctp-dev lksctp-tools iproute2 cmake –classic

Installs tools and libraries required for SCTP (Stream Control Transmission Protocol) and network emulation.

step5:

$ sudo snap install cmake –classic

Installs CMake, a build system tool used to compile projects like Open5GS or UERANSIM, using Snap

step6:

$ cd UERANSIM/

Changes your current directory to the cloned UERANSIM folder.

then type make

Runs the Makefile that's included in the UERANSIM repo. This compiles the code to generate the binaries like:

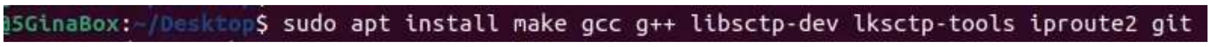
- nr-ue → simulated UE (User Equipment)
- nr-gnb → simulated gNB (base station)

ex:~UERANSIM/$:make

"below mentioned the typped commands follow it "

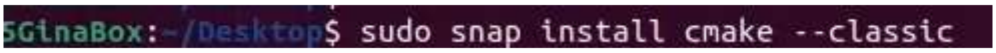
Typped commands for reference

Step1:



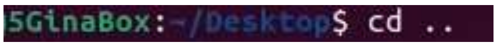
```
5GinaBox:~/Desktop$ sudo apt install make gcc g++ libsctp-dev lksctp-tools iproute2 git
```

Step2:



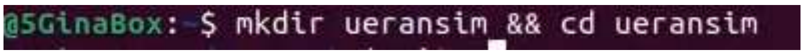
```
5GinaBox:~/Desktop$ sudo snap install cmake --classic
```

Step3:



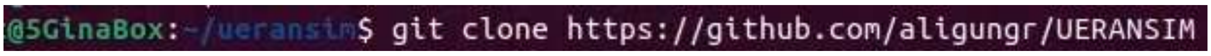
```
5GinaBox:~/Desktop$ cd ..
```

Step4:



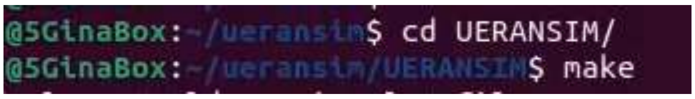
```
@5GinaBox:~$ mkdir ueransim && cd ueransim
```

Step5:



```
@5GinaBox:~/ueransim$ git clone https://github.com/aligungr/UERANSIM
```

Step6:

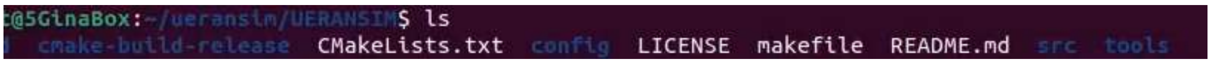


```
@5GinaBox:~/ueransim$ cd UERANSIM/
@5GinaBox:~/ueransim/UERANSIM$ make
```

Step7:

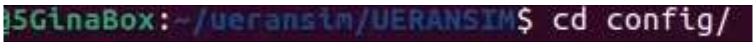
Here open a new terminal without any changes of file path as

cd ~/ueransim/UERANSIM



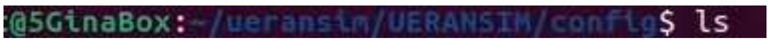
```
@5GinaBox:~/ueransim/UERANSIM$ ls
  cmake-build-release  CMakeLists.txt  config  LICENSE  makefile  README.md  src  tools
```

Step8:

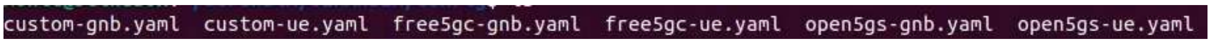


```
5GinaBox:~/ueransim/UERANSIM$ cd config/
```

Step9:

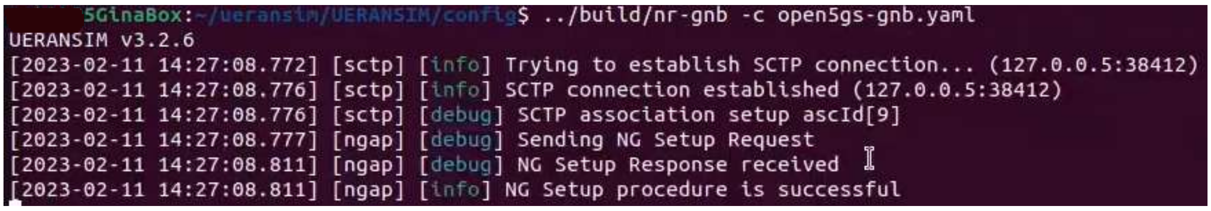


```
@5GinaBox:~/ueransim/UERANSIM/config$ ls
```



```
custom-gnb.yaml  custom-ue.yaml  free5gc-gnb.yaml  free5gc-ue.yaml  open5gs-gnb.yaml  open5gs-ue.yaml
```

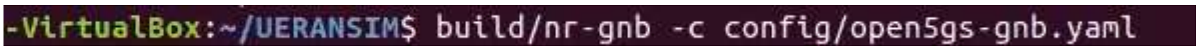
Step10:



```
[redacted]5GinaBox:~/ueransim/UERANSIM/config$ ../build/nr-gnb -c open5gs-gnb.yaml
UERANSIM v3.2.6
[2023-02-11 14:27:08.772] [sctp] [info] Trying to establish SCTP connection... (127.0.0.5:38412)
[2023-02-11 14:27:08.776] [sctp] [info] SCTP connection established (127.0.0.5:38412)
[2023-02-11 14:27:08.776] [sctp] [debug] SCTP association setup ascId[9]
[2023-02-11 14:27:08.777] [ngap] [debug] Sending NG Setup Request
[2023-02-11 14:27:08.811] [ngap] [debug] NG Setup Response received
[2023-02-11 14:27:08.811] [ngap] [info] NG Setup procedure is successful
```

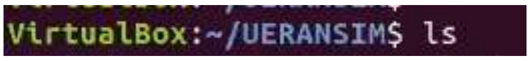
Next in a new command prompt in a another virtual machine or take it on same system also for understanding do all the operations in a same machine.

step1:

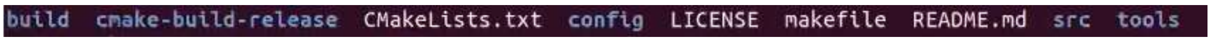


```
-VirtualBox:~/UERANSIM$ build/nr-gnb -c config/open5gs-gnb.yaml
```

Step2:

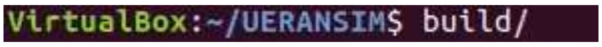


```
VirtualBox:~/UERANSIM$ ls
```

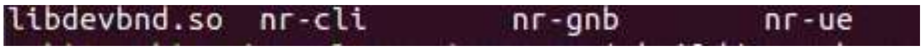


```
build  cmake-build-release  CMakeLists.txt  config  LICENSE  makefile  README.md  src  tools
```

Step3:

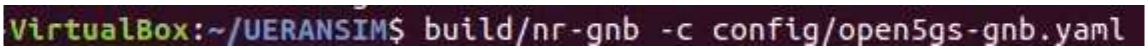


```
VirtualBox:~/UERANSIM$ build/
```

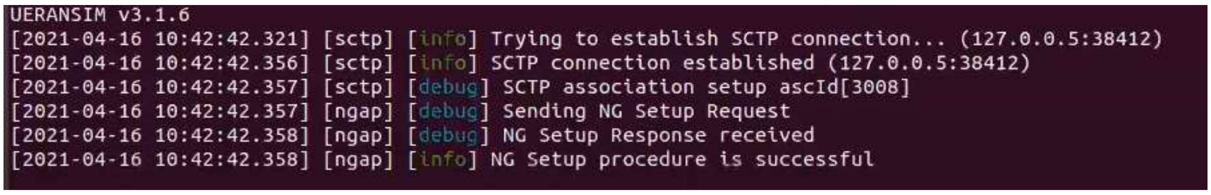


```
libdevbnd.so  nr-cli        nr-gnb        nr-ue
```

Step4:



```
VirtualBox:~/UERANSIM$ build/nr-gnb -c config/open5gs-gnb.yaml
```



```
UERANSIM v3.1.6
[2021-04-16 10:42:42.321] [sctp] [info] Trying to establish SCTP connection... (127.0.0.5:38412)
[2021-04-16 10:42:42.356] [sctp] [info] SCTP connection established (127.0.0.5:38412)
[2021-04-16 10:42:42.357] [sctp] [debug] SCTP association setup ascId[3008]
[2021-04-16 10:42:42.357] [ngap] [debug] Sending NG Setup Request
[2021-04-16 10:42:42.358] [ngap] [debug] NG Setup Response received
[2021-04-16 10:42:42.358] [ngap] [info] NG Setup procedure is successful
```

Typed commands:

$ build/nr-gnb -c config/open5gs-gnb.yaml

$ ls

$ build

$ build/nr-gnb -c config/open5gs-gnb.yaml

Chapter 6 - Configuring SCTP & NGAP with UERANSIM & OPEN5gs

Step1:

$ sudo service open5gs-amfd status

It tells the status of AMF block

Step2:

$ sudo vi /etc/open5gs/amf.yaml

Here we can observe data of AMF in command prompt itself.

Step3:

$ cd /etc/open5gs/

$ ls

Here we can observe 5g core element YAML files which is present

Step4:

$ls-lrt

Roots of 5gcore

Step5:

$more *.yaml

Step6:

$ grep -I “addr” *.yaml

Here we can observe the ip addresses of each network function

Step6:

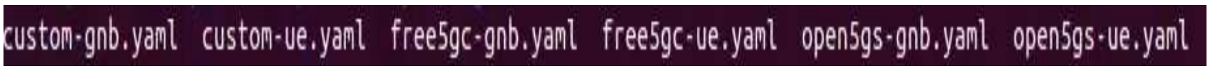
$ cd ~/ueransim/UERANSIM/

$ ls

Step7:

$ cd config/

$ ls



```
custom-gnb.yaml  custom-ue.yaml  free5gc-gnb.yaml  free5gc-ue.yaml  open5gs-gnb.yaml  open5gs-ue.yaml
```

Step8:

$ cp open5gs-gnb.yaml gnb1.yaml

$ cp open5gs-ue.yaml ue1.yaml
Here we can see the gnb & ue yaml files

Step9:

$ ls-lrt

$ gedit gnb1.yaml &

It is used to open the gnb yaml file

$ gedit ue1.yaml &

It is used to open the ue yaml file

Chapter 7 - RRC setup

Step1:

Take a new command window then perform this operations

$ cd ueransim

$ ls

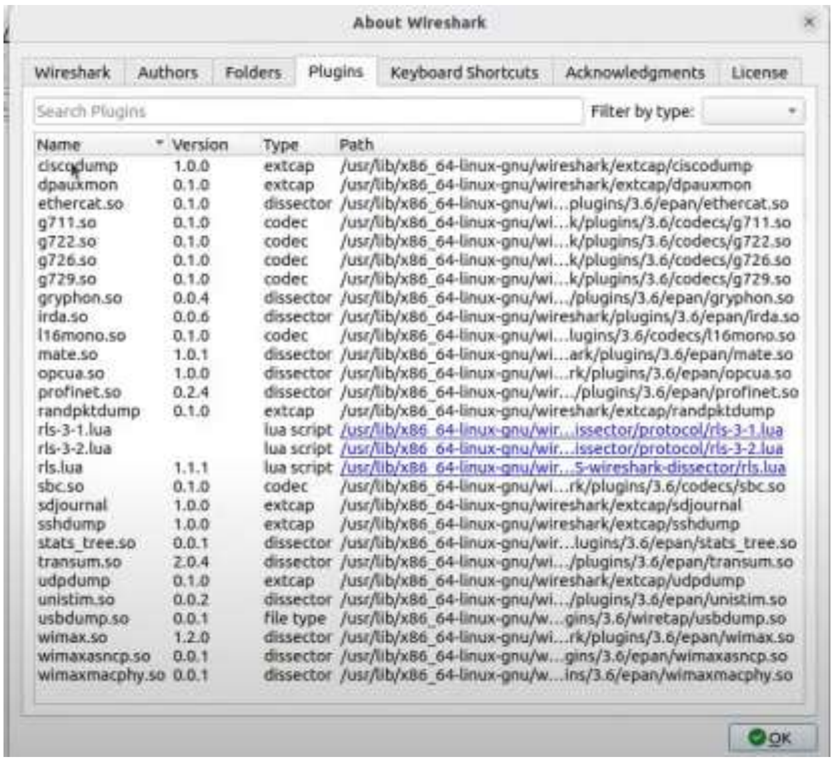
Step2:

$git clone https://github.com/louisroyer/RLS-wireshark-dissector.git

Step3:

Then open the wireshark in ubuntu

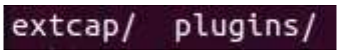
Goto HELP→about wireshark→goto plugins there we can observe like this below mentioned



Name	Version	Type	Path
ciscodump	1.0.0	extcap	/usr/lib/x86_64-linux-gnu/wireshark/extcap/ciscodump
dpauxmon	0.1.0	extcap	/usr/lib/x86_64-linux-gnu/wireshark/extcap/dpauxmon
ethercat.so	0.1.0	dissector	/usr/lib/x86_64-linux-gnu/wi...plugins/3.6/epan/ethercat.so
g711.so	0.1.0	codec	/usr/lib/x86_64-linux-gnu/wi...k/plugins/3.6/codecs/g711.so
g722.so	0.1.0	codec	/usr/lib/x86_64-linux-gnu/wi...k/plugins/3.6/codecs/g722.so
g726.so	0.1.0	codec	/usr/lib/x86_64-linux-gnu/wi...k/plugins/3.6/codecs/g726.so
g729.so	0.1.0	codec	/usr/lib/x86_64-linux-gnu/wi...k/plugins/3.6/codecs/g729.so
gryphon.so	0.0.4	dissector	/usr/lib/x86_64-linux-gnu/wi.../plugins/3.6/epan/gryphon.so
irda.so	0.0.6	dissector	/usr/lib/x86_64-linux-gnu/wireshark/plugins/3.6/epan/irda.so
l16mono.so	0.1.0	codec	/usr/lib/x86_64-linux-gnu/wi...lugins/3.6/codecs/l16mono.so
mate.so	1.0.1	dissector	/usr/lib/x86_64-linux-gnu/wi...ark/plugins/3.6/epan/mate.so
opcua.so	1.0.0	dissector	/usr/lib/x86_64-linux-gnu/wi...rk/plugins/3.6/epan/opcua.so
profinet.so	0.2.4	dissector	/usr/lib/x86_64-linux-gnu/wir.../plugins/3.6/epan/profinet.so
randpktdump	0.1.0	extcap	/usr/lib/x86_64-linux-gnu/wireshark/extcap/randpktdump
rls-3-1.lua		lua script	/usr/lib/x86_64-linux-gnu/wir...issector/protocol/rls-3-1.lua
rls-3-2.lua		lua script	/usr/lib/x86_64-linux-gnu/wir...issector/protocol/rls-3-2.lua
rls.lua	1.1.1	lua script	/usr/lib/x86_64-linux-gnu/wir...S-wireshark-dissector/rls.lua
sbc.so	0.1.0	codec	/usr/lib/x86_64-linux-gnu/wi...rk/plugins/3.6/codecs/sbc.so
sdjournal	1.0.0	extcap	/usr/lib/x86_64-linux-gnu/wireshark/extcap/sdjournal
sshdump	1.0.0	extcap	/usr/lib/x86_64-linux-gnu/wireshark/extcap/sshdump
stats_tree.so	0.0.1	dissector	/usr/lib/x86_64-linux-gnu/wir...lugins/3.6/epan/stats_tree.so
transum.so	2.0.4	dissector	/usr/lib/x86_64-linux-gnu/wi.../plugins/3.6/epan/transum.so
udpdump	0.1.0	extcap	/usr/lib/x86_64-linux-gnu/wireshark/extcap/udpdump
unistim.so	0.0.2	dissector	/usr/lib/x86_64-linux-gnu/wi.../plugins/3.6/epan/unistim.so
usbdump.so	0.0.1	file type	/usr/lib/x86_64-linux-gnu/w...gins/3.6/wiretap/usbdump.so
wimax.so	1.2.0	dissector	/usr/lib/x86_64-linux-gnu/wi...rk/plugins/3.6/epan/wimax.so
wimaxasncp.so	0.0.1	dissector	/usr/lib/x86_64-linux-gnu/w...gins/3.6/epan/wimaxasncp.so
wimaxmacphy.so	0.0.1	dissector	/usr/lib/x86_64-linux-gnu/w...ins/3.6/epan/wimaxmacphy.so

Step4:

Then come to command prompt again type the command
$ cd /usr/lib/x86_64-linux-gnu/wireshark/
There we can observe like this



```
extcap/  plugins/
```

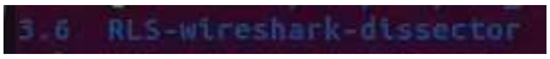
Step5:

Add file as a mentioned below
cd /usr/lib/x86_64-linux-gnu/wireshark/plugins/

Step6:

$ ls

We can get like this



```
3.6  RLS-wireshark-dissector
```

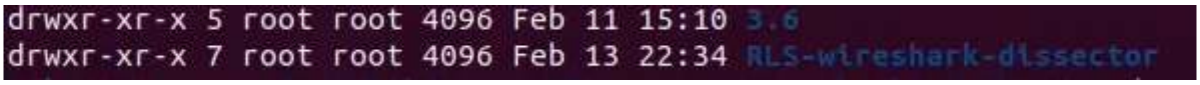
Step7:

Type this commands which is mentioned below

$sudo cp -R ~/ueransim/RLS-wireshark-dissector/ RLS-wireshark-dissector/

$ls-lrt

We can observe ouput like this



```
drwxr-xr-x 5 root root 4096 Feb 11 15:10 3.6
drwxr-xr-x 7 root root 4096 Feb 13 22:34 RLS-wireshark-dissector
```

Step8:

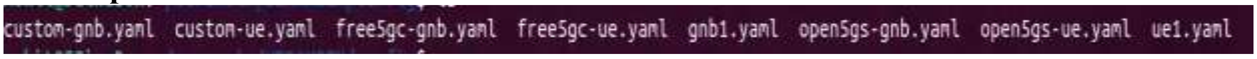
$sudo cp -R ~/ueransim/ RLS-wireshark-dissector/ RLS-wireshark-dissector/

Step9:

$ sudo wireshark

Here the wireshark will be opened then
Type some commands which are mention below:
1>cd
2>cd ueransim/UERANSIM/config

Then use the commands as shown in the figure below
$ ls
The output will be like this



```
custom-gnb.yaml  custom-ue.yaml  free5gc-gnb.yaml  free5gc-ue.yaml  gnb1.yaml  open5gs-gnb.yaml  open5gs-ue.yaml  ue1.yaml
```

Step10:

$gedit ue1.yaml &

This command used to open the ue1 Yaml file in text editor

$gedit gnb1.yaml &

This command used to open the gnb1 Yaml file in text editor

Now we want to follow some steps here

step1:

Open the browser enter the **127.0.0.1:9999 or localhost:9999**
Based on your system configuration the local host address will be changed.

Enter the username as a **“admin”**

Password as a **“1423”**

Step2:
there we can observe the open5gs page & add subscriber option click on it now,

See the Ue.Yaml file
enter the IMSI number as per ue yaml file
check the subscriber key & operator key from Ue.yaml file
Check the slice type from ue.yaml file

Step3:

Now observe the both Ue & gnb Yaml files for configuration
now ip add:127.0.0.101
this ip address should be same for gnbSearchlist in Ue.yaml file
and same for linkIp in gb.Yaml file.

Then save both yaml files & close it .

Step4:

Open the wireshark .

Click on any

Use the command in searchspace **nr-rrc** in a wireshark.

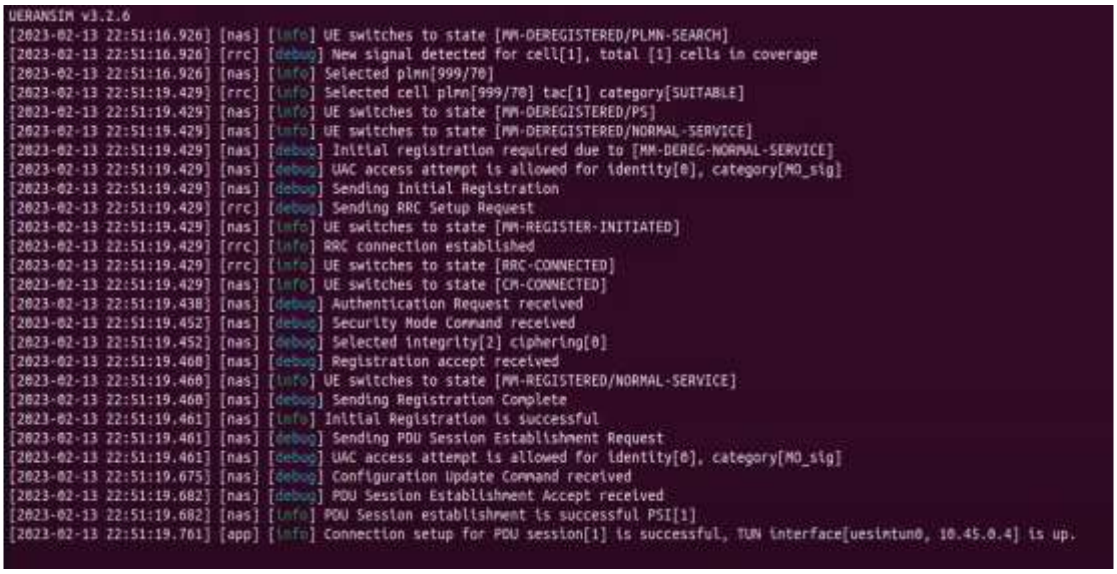
Then,

Open the command prompt again type this command shown below

$ cd ~/ueransim/UERANSIM/config

$ ../build/nr-ue -c ue.yaml

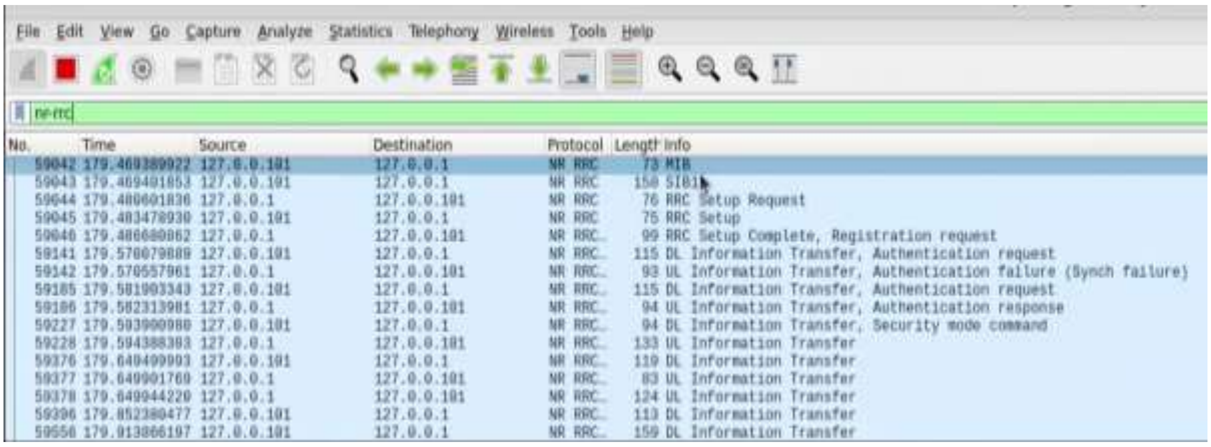
We can observe ouput like this



```
UERANSIM v3.2.6
[2023-02-13 22:51:16.926] [nas] [info] UE switches to state [MM-DEREGISTERED/PLMN-SEARCH]
[2023-02-13 22:51:16.926] [rrc] [debug] New signal detected for cell[1], total [1] cells in coverage
[2023-02-13 22:51:16.926] [nas] [info] Selected plmn[999/70]
[2023-02-13 22:51:19.429] [rrc] [info] Selected cell plmn[999/70] tac[1] category[SUITABLE]
[2023-02-13 22:51:19.429] [nas] [info] UE switches to state [MM-DEREGISTERED/PS]
[2023-02-13 22:51:19.429] [nas] [info] UE switches to state [MM-DEREGISTERED/NORMAL-SERVICE]
[2023-02-13 22:51:19.429] [nas] [debug] Initial registration required due to [MM-DEREG-NORMAL-SERVICE]
[2023-02-13 22:51:19.429] [nas] [debug] UAC access attempt is allowed for identity[0], category[MO_sig]
[2023-02-13 22:51:19.429] [nas] [debug] Sending Initial Registration
[2023-02-13 22:51:19.429] [rrc] [debug] Sending RRC Setup Request
[2023-02-13 22:51:19.429] [nas] [info] UE switches to state [MM-REGISTER-INITIATED]
[2023-02-13 22:51:19.429] [rrc] [info] RRC connection established
[2023-02-13 22:51:19.429] [rrc] [info] UE switches to state [RRC-CONNECTED]
[2023-02-13 22:51:19.429] [nas] [info] UE switches to state [CM-CONNECTED]
[2023-02-13 22:51:19.438] [nas] [debug] Authentication Request received
[2023-02-13 22:51:19.452] [nas] [debug] Security Mode Command received
[2023-02-13 22:51:19.452] [nas] [debug] Selected integrity[2] ciphering[0]
[2023-02-13 22:51:19.460] [nas] [debug] Registration accept received
[2023-02-13 22:51:19.460] [nas] [info] UE switches to state [MM-REGISTERED/NORMAL-SERVICE]
[2023-02-13 22:51:19.460] [nas] [debug] Sending Registration Complete
[2023-02-13 22:51:19.461] [nas] [info] Initial Registration is successful
[2023-02-13 22:51:19.461] [nas] [debug] Sending PDU Session Establishment Request
[2023-02-13 22:51:19.461] [nas] [debug] UAC access attempt is allowed for identity[0], category[MO_sig]
[2023-02-13 22:51:19.675] [nas] [debug] Configuration Update Command received
[2023-02-13 22:51:19.682] [nas] [debug] PDU Session Establishment Accept received
[2023-02-13 22:51:19.682] [nas] [info] PDU Session establishment is successful PSI[1]
[2023-02-13 22:51:19.761] [app] [info] Connection setup for PDU session[1] is successful, TUN interface[uesimtun0, 10.45.0.4] is up.
```

Then

Goto wireshark we can observe RRC packets.


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
nr-rrc

```
No.   Time           Source        Destination   Protocol Length Info
59042 179.469389922 127.0.0.101   127.0.0.1     NR RRC     73 MIB
59043 179.469401853 127.0.0.101   127.0.0.1     NR RRC    150 SIB1
59044 179.480601836 127.0.0.1     127.0.0.101   NR RRC     76 RRC Setup Request
59045 179.483478930 127.0.0.101   127.0.0.1     NR RRC     75 RRC Setup
59046 179.486680862 127.0.0.1     127.0.0.101   NR RRC…    99 RRC Setup Complete, Registration request
59141 179.570079889 127.0.0.101   127.0.0.1     NR RRC…   115 DL Information Transfer, Authentication request
59142 179.570557961 127.0.0.1     127.0.0.101   NR RRC…    93 UL Information Transfer, Authentication failure (Synch failure)
59185 179.581903343 127.0.0.101   127.0.0.1     NR RRC…   115 DL Information Transfer, Authentication request
59186 179.582313981 127.0.0.1     127.0.0.101   NR RRC…    94 UL Information Transfer, Authentication response
59227 179.593900980 127.0.0.101   127.0.0.1     NR RRC…    94 DL Information Transfer, Security mode command
59228 179.594388383 127.0.0.1     127.0.0.101   NR RRC…   133 UL Information Transfer
59376 179.649499993 127.0.0.101   127.0.0.1     NR RRC…   119 DL Information Transfer
59377 179.649901769 127.0.0.1     127.0.0.101   NR RRC…    83 UL Information Transfer
59378 179.649944220 127.0.0.1     127.0.0.101   NR RRC…   124 UL Information Transfer
59396 179.852380477 127.0.0.101   127.0.0.1     NR RRC…   113 DL Information Transfer
59556 179.913866197 127.0.0.101   127.0.0.1     NR RRC…   159 DL Information Transfer
```

Now main part starts

task is to add six slices
now we want to modify the yaml files
1>AMF

2>SMF

3>UDM

4>NSSF

5>UPF

6>NRF

After this configuration we want to check the outputs as per
SCTP & NGAP & RRC.

For that one we want to edit the UE1.Yaml file as well as gnb.Yaml file in UERANSIM setup.

Just we want to change the sst values and we want to check those slice is working or not.

By using ngap & rrc

- SCTP is used between the gNB (5G base station) and AMF (Access and Mobility Management Function) for signaling.
- NGAP Used for Control-plane signaling between gNB (5G base station) and AMF (core network node).
- RRC is used for Managing radio resources between UE (User Equipment) and gNB (or eNodeB in LTE).

References

[1] R. Palenik, “Setting Up Open5GS: A Step-by-Step Guide or How We Set Up Our Lab Environment,” *Medium - Networks @ FIIT STU*, [Online]. Available: https://medium.com/networkers-fiit-stu/setting-up-open5gs-a-step-by-step-guide-or-how-we-set-up-our-lab-environment-5da1c8db0439. [Accessed: Apr. 28, 2025].

[2] D. Hsieh, “Deploy a Free5GC Network Slice on OpenStack,” *free5gc.org*, Jul. 26, 2023. [Online]. Available: https://free5gc.org/blog/20230726/network_slice/. [Accessed: Apr. 28, 2025].

[3] K. Prakash, “Standalone 5G Network Simulation using Open5GS and UERANSIM,” *GitHub Repository*, [Online]. Available: https://github.com/kushalprakash6/Standalone-5G-netowrk-simulation. [Accessed: Apr. 28, 2025].

[4] OpenRAN Gym, “Deploying ORANSlice: A Comprehensive 5G Network Slicing Tutorial,” *openrangym.com*, [Online]. Available: https://openrangym.com/tutorials/oranslice-tutorial. [Accessed: Apr. 28, 2025].

[5] C. Y. A. Fadhlika, “Setting Up OpenCore and OpenRAN for 5G Simulation,” *Medium*, Jul. 2023. [Online]. Available: https://medium.com/@citrayaf/setting-up-opencore-and-openran-for-5g-simulation-b2ee1b11b089. [Accessed: Apr. 28, 2025].

[6] Open5GS Community, “Developer Setup Guide for Open5GS-UI and Network Slicing,” *Open5GS-UI Documentation*, [Online]. Available: https://open5gs-ui.readthedocs.io/en/latest/Appendix/dev_setup.html. [Accessed: Apr. 28, 2025].

[7] E. B. Ernie, “Step-by-Step Guide to Configuring a 5G SA Network with Open5GS and MEC on Virtual Machines,” *Personal Blog*, Mar. 2023. [Online]. Available: https://ernie55ernie.github.io/python/2023/03/02/step-by-step-guide-to-configuring-a-5g-sa-network-with-open5gs-and-mec-on-virtual-machines.html. [Accessed: Apr. 28, 2025].