Project 1: free5GC performance testing

Due Date: @April 29, 2025

NYCU Computer Networks 113 Fall

Group Project: Patrick Nicholas, Pham Dinh Quy

Presentation Slide:
**https://docs.google.com/presentation/d/15nNOwvMZTg450UfK6vIj_QQaM710wdPdFDiOD9oFH
usp=sharing**

Introduction

In this Project, we are asked to replicate the results of the research paper "Evaluating the performance of open source software implementations of the 5G network core" by Lando et al., 2023. We will be attempting to reproduce the exact results of using my5G-RANTester to test free5GC v3.0.6 and v3.2.1's performance based on Session Establishment and Registration Time Tests and Data Plane Performance Tests.

The source code this project will work on is from https://github.com/PORVIR-5G-Project/my5G-RANTester-Scripts as mentioned in the research paper, as this repository is already filled with automated scripts to run the same tests as the paper.

Environment Setup

Attempting to replicate the results as closely as possible requires us to have the closest environment with the original testing as possible. Here are what the paper mentioned:

VM Configurations:

- Ubuntu Server 20.04.4 LTS, running Linux Kernel v5.4.90
- Intel Core i7 8700T @ 2.4 GHz (4 Cores, 6 Cores, 8 Cores, 12 Cores)
- DDR4 2666 MHz RAM (4GB, 8GB, 8GB, 8GB)
- SSD M.2 64GB NVMe

Due to the limited resources we have, here is what our VM configurations was for this project:

- Ubuntu Server 20.04.4 LTS, running Linux Kernel v5.4.90
- Intel Core i7 9750H @ 2.6 GHz (4 Cores, 6 Cores)
- DDR4 2666 MHz SODIMM RAM (4GB, 8GB)
- Virtual Hard Disk ~20GB

After successfully booting up the VM and setting up the Ubuntu Server, we first made sure to install Linux Kernel v5.4.90 by following this tutorial. Once finished, we continued to clone the github repository of the source code as our main working directory.

Methodology

1. Initial Setup and Run Tests

   Inside of the working directory, we can see some scripts in the form of ./run.sh and ./run_experiments.sh, but during working with these scripts we realize that there were some package dependencies that were outdated that made our runs not work, due to important packages not being installed. This issue was resolved by modifying the golang versions inside of the Dockerfiles from `golang:1.14.4-stretch` to `golang:1.17.13-buster`.

We also realized that in the original github repository, the author is asking us to run the scripts by using curl to run the script directly from the old repository. We found this to be troublesome because of the previously mentioned golang version problem we faced, so we resolved this issue by forking all of the repositores that needed its Dockerfile to be modified which were:

- https://github.com/PORVIR-5G-Project/free5gc-compose
- **free5gc-docker-v3.0.6**
- https://github.com/PORVIR-5G-Project/free5gc-my5G-RANTester-docker

We also modified the run.sh file so that when it runs with using either free5GC's, it's taking it from the local `utils/5g_core/free5gc_v3.0.6.sh` and `utils/5g_core/free5gc_v3.2.1.sh` shell script, which clones the modified repositories we already forked.

Following these changes we were able to run our first test:

```
./run.sh -c 2 -e 1 -g 5 -u 10 -t 60 -w 100 -v
./capture_and_parse_logs.sh
./stop_and_clear.sh
```

We followed the paper's configurations for testing, which includes using free5GC v3.2.1 for the connectivity test and free5GC v3.0.6 for the throughput test.

Another issue was identified when we tried running the throughput test:

```
./run.sh -c 1 -e 2 -g 1 -v
```

For some reason, my5grantester0 was not able to apt install iproute2 when building, this resulted in the container not being able to run commands such as `ip -4 addr show uetun1`. To resolve this issue, we modified the docker-compose.yaml file in the https://github.com/109550187/free5gc-my5G-RANTester-docker repository to have my5grantester's command be : `bash -c "apt update && apt install -y iproute2 && ./app ${TEST_PARAMETERS}"` , therefore whenever a my5grantester container is built, it ensures that iproute2 is also properly installed.

2. Automated Scripts and Running Actual Tests

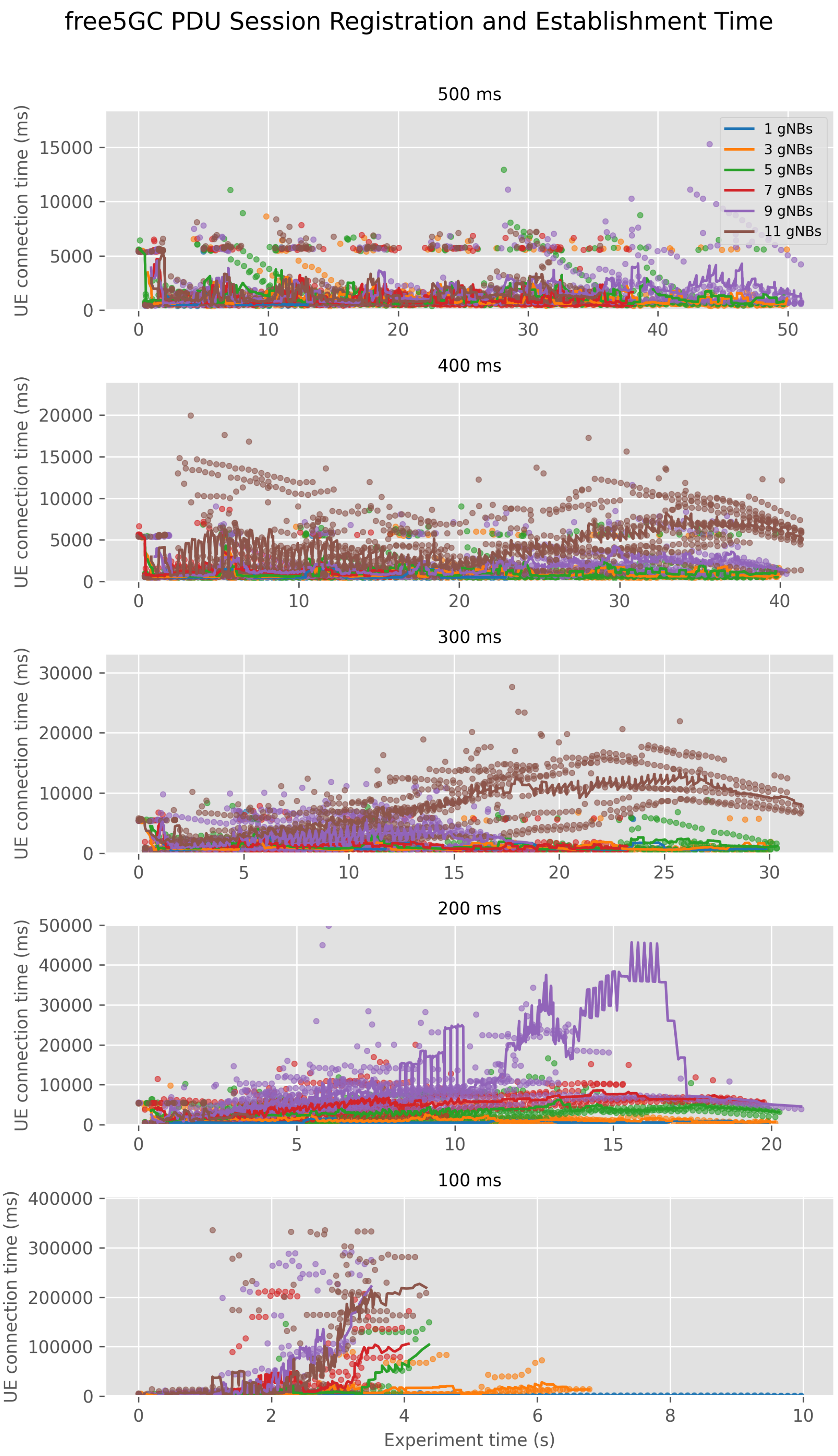
   After resolving the previous issues, we continued to build our own automated script to run both the connectivity test and throughput test with the script collecting the output csv files for every run, configured exactly just like the research paper. Here are the parameters for each tests:

   - Connectivity Test:
     - 6 Virtual Cores - 8GB RAM
     - free5GC v3.2.1
     - gNB : 1, 3, 5, 7, 9, 11
     - Interval (ms): 100ms, 200ms, 300ms, 400ms, 500ms
     - Performed 2 times
   - Throughput Test:
     - 6 Virtual Cores - 8GB RAM, 4 Virtual Cores - 4GB RAM
     - free5GC v3.0.6
     - gNB: 1, 2, 4, 6, 8, 10
     - Performed 2 times

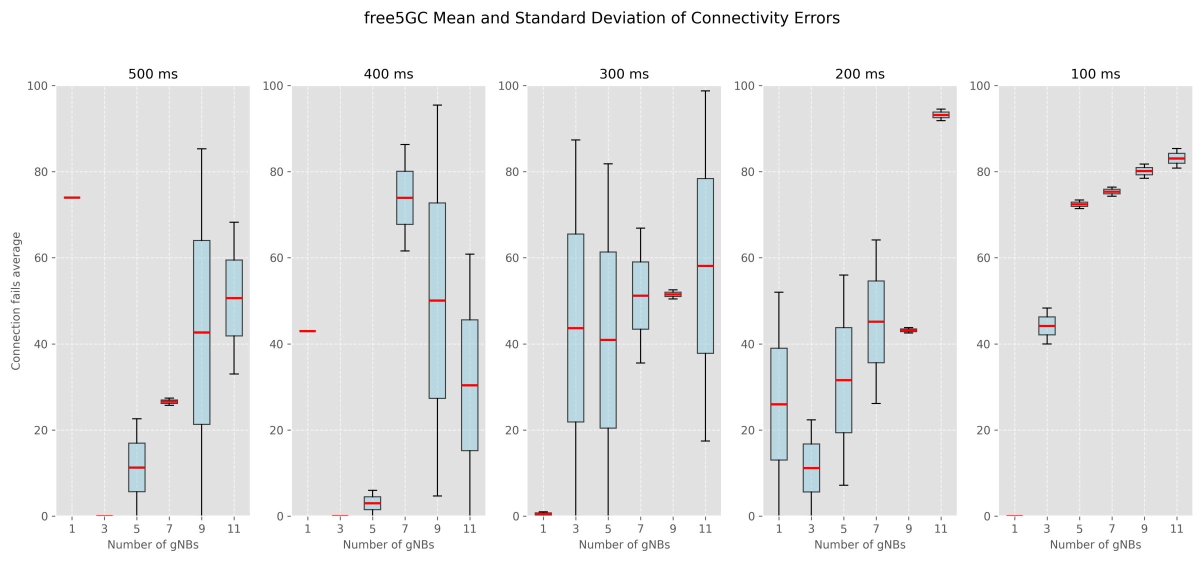
   Results are then collected to be used for analytics through using Python matplotlib.

Results

We found in our experiment that the two tests we run were quite similar with the original paper's results, getting similar pattern to the figures that were attached. There were quite some differences but we assume the differences in the test machine configuration contributed a lot. For example in the connectivity test, the original author was able to reach an average time of ~1000ms when setting delay at 500ms intervals, and even though we were also able to reach this number in our own testing, but the differences when the delay is set at the other intervals can be easily seen, with our testing results reaching around 4~5 times higher connection time than the original.

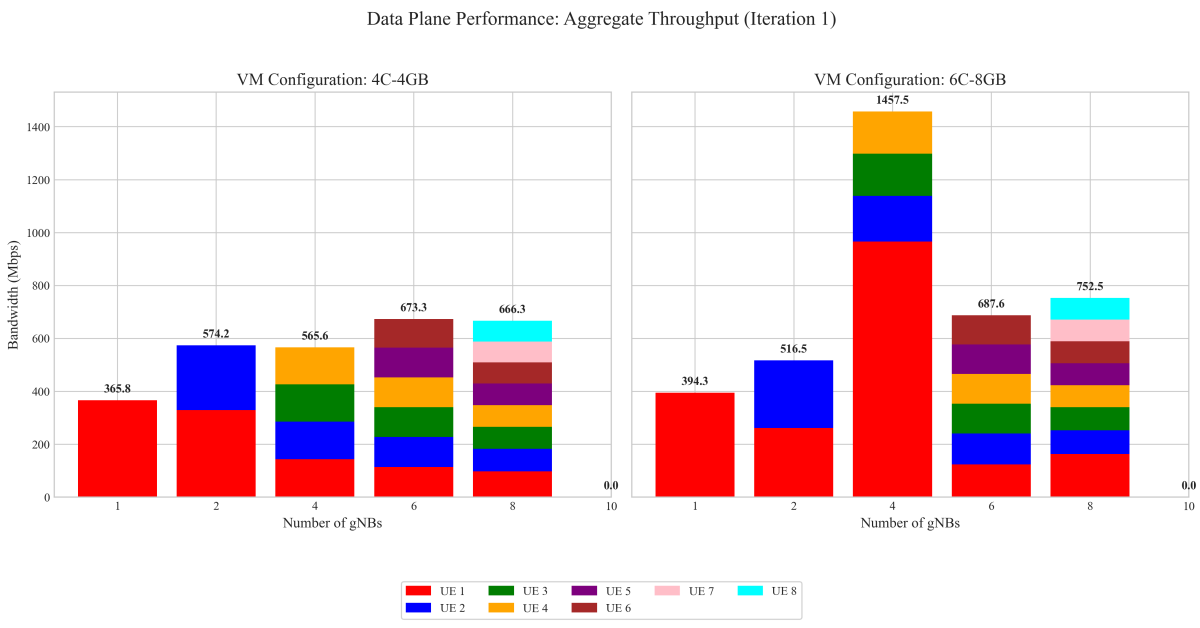

free5GC PDU Session Registration and Establishment Time
500 ms
400 ms
300 ms
200 ms
100 ms
UE connection time (ms)
Experiment time (s)
1 gNBs
3 gNBs
5 gNBs
7 gNBs
9 gNBs
11 gNBs

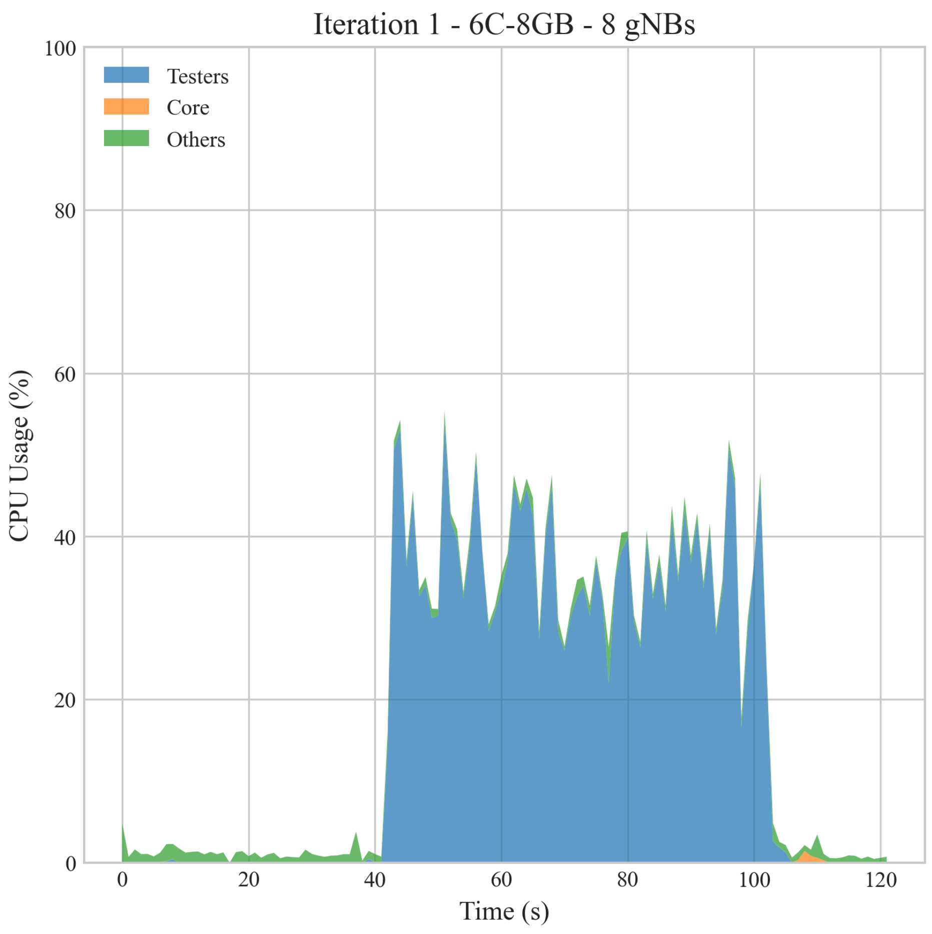
We also find that the boxplots for the mean and standard deviation of connectivity errors were quite similar to the ones presented in the paper. We assume the results might be more variative than the original paper because of only running 2 iterations instead of the 10 iterations the original authors tested, therefore allowing for a much bigger error window.



The results we got for the throughput test were also quite similar somehow with the paper's results, but the throughput test might face more absolute differences due to the original paper testing on all the different configurations and running 16 iterations, while we were only able to run on 2 of the lowest test machine configuration and for 2 iterations only.

The CPU usage figure we got from making use of data from using influxdb also gave us quite a similar result to the actual paper, where the Tester group took most of the CPU usage and the Core group using very low CPU. The tester group being active around the 40 second mark was also pretty accurate to the paper's results.




Iteration 1 - 6C-8GB - 8 gNBs
Testers
Core
Others
CPU Usage (%)
Time (s)
0
20
40
60
80
100
120