EasyChair Preprint

№ 15688

Open Source 5G Network Deployment

Andrew Fox

EasyChair preprints are intended for rapid dissemination of research results and are integrated with the rest of EasyChair.

January 8, 2025

Open Source 5G Network Deployment

Andrew Fox
Network and Computer Security
SUNY Polytechnic Institute
Utica, NY
foxaj3@sunypoly.edu

***Abstract*— In the networking industry, there is a lack of tools and software that provide a testing platform to experiment with mobile networks. Without these platforms to experiment with, researchers can only present concepts for changes to mobile networks and cannot prove their work is practical. Open AI Cellular (OAIC) provides the software and tools to install, deploy, and configure a 5G network. OAIC is a 5G network research and experimentation platform where researchers can learn more about mobile networks with simulated equipment that matches today's networking standards. You will learn more about the core components needed to create a 5G network and experiment with a wireless mobile network. The experimental results highlight OAIC's ability to handle Denial of Service (DOS) attacks with little impact on network performance. This project also highlights implementing network slices within OAIC to reduce the effects of a DOS attack. This paper provides a new perspective on mobile network experiments and developments and offers valuable insight into the future of 5G and 6G networks.**

I. Introduction

A. Motivation

5G networks are discussed briefly in some of my classes, but we didn't get to go into further details on it. I learned about 5G and what goes into a mobile network, but with a hands-on project. I began researching 5G networks one year ago on campus as a research job and found that I loved learning more about this topic. I was trying to develop a low-budget project to experiment with a small-scale 5G network but could not do it. While researching, I found open-source software allowing users to set up their own 5G network from their computers. I wanted to use this tool to learn more about 5G but also to experiment with the tool and see what features it has to offer.

The main goal of this project is to share information about this tool and to encourage students to experiment with it to learn more about 5G networks. Another goal of this project is to fill the gap in hands-on learning with 5G. For many in the IT industry wanting to experiment more with 5G, finding ways to apply what you learn about mobile networks takes time and effort. Few resources are available for testing or experimenting with 5G networks, especially open-source ones. This project provides zero-cost, open-source, well-documented software to help fill the gap in hands-on learning with 5G.

B. Roadmap

- Objectives – The Open AI Cellular tool aims to research with simulated equipment that matches what is used in the industry. Within the industry, there is a lack of tools that allow researchers and developers to experiment with 5G and test out proposals to see if their ideas are possible. This paper highlights the importance of OAIC and how it can help further develop the future of mobile networks.
- Methodology – I will go in-depth about the equipment and installation requirements for this project. The installation process for OAIC will be explained in detail, along with the challenges I encountered while installing it. I will explain the importance of each tool used to make OAIC run. I will talk about the critical components of the 5G network that OAIC will run. I will also discuss the various experiments I ran with OAIC. These experiments include a DOS attack and implementing network slicing on my network.
- Contributions – I will highlight the results of my experiment, showing how OAIC can be useful when developing new ideas for future generations of mobile networks. I will also share potential applications for solving a problem based on the output of my experiments.

II. Background

A. 5G

5G is the fifth generation of mobile networks, intending to provide more availability, reliability, and lower latency than previous generations. 5G can support a 100x traffic increase compared to 4G, data rates of over 100 Megabits-per-second (Mbps), and utilizes Millimeter waves for better spectrum use [1]. With the demand for fast connections growing, 5G was a massive milestone for wireless communications.

B. Open AI Cellular (OAIC)

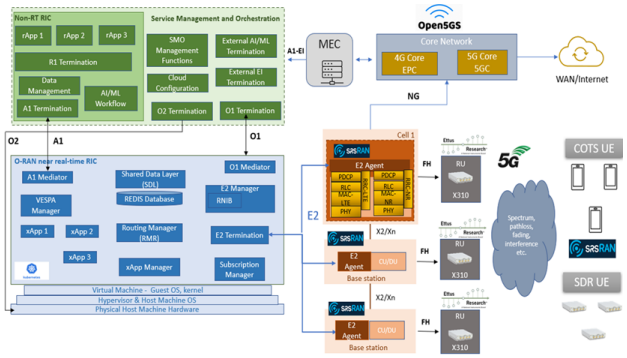


Fig. 1. Open AI Cellular (OAIC) Framework [2].

Open AI Cellular (OAIC) is an open-source effort that provides libraries and toolsets that contain AI controllers and an AI testing framework [2]. This effort helps the development and research of AI-enabled cellular radio networks. OAIC provides a framework, as shown in Figure 1, that acknowledges various open-source 5G software, which allows users to implement radio access network intelligent controllers (RICs) and O-RAN interfaces for testbed and production real-time experiments. OAIC will enable you to install, configure, and run your own 5G network from a single device for research and experimentation. Each component used to simulate a 5G network matches the equipment used in the real world for mobile networking.

C. Ubuntu

Ubuntu is an open-source operating system used throughout this project and is where OAIC will run [3]. Ubuntu is a flavor of Linux that is stable and secure. Although there are other Linux flavors, Ubuntu is the only version of Linux that is used to install and configure OAIC.

D. Docker

Docker is a platform that can utilize containers to run applications in a small, separate environment [4]. Containers are images that have the required software to run packaged together. They are like virtual machines (VMs) but are lighter and less resource-intensive. These containers are essential for the OAIC installation because they require containers for specific components.

E. Kubernetes

Kubernetes is an open-source platform for automating deployment, scaling, and operations of application containers across clusters of hosts [5]. Placement, restarts, replication, and autoscaling of containers are all automated using Kubernetes. In each of the pods being run by Kubernetes, there could be any number of containers being used. It is an essential tool for OAIC because the containers and pods are running separate components and software, and Kubernetes manages it all for us.

F. Helm

Helm is a package manager for Kubernetes, which is the equivalent of apt installing packages [6]. It deploys packaged applications, which are called Helm Charts, on Kubernetes clusters. Helm Charts contain configuration and package information that will be essential for OAIC to run. The Helm Charts are used in this project for package and configuration management.

G. srsRAN

srsRAN is a 5G software radio suite developed by SRS (Software Radio Systems). It was designed to create open-source, high-performance software radio solutions for 4G and 5G [7]. The srsRAN suite includes srsUE, srsENB, and srsEPC. Everything in the suite represents the different components that go into a mobile network. This is important software that is needed for OAIC to run.

H. Iperf3

```
root@OAIC:~# iperf3 -s -i 1
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 172.16.0.2, port 35928
[  5] local 172.16.0.1 port 5201 connected to 172.16.0.2 port 49694
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-1.00   sec   245 KBytes  2.00 Mbits/sec
[  5]   1.00-2.00   sec   252 KBytes  2.06 Mbits/sec
[  5]   2.00-3.00   sec   124 KBytes  1.02 Mbits/sec
[  5]   3.00-4.00   sec   168 KBytes  1.38 Mbits/sec
[  5]   4.00-5.00   sec   195 KBytes  1.60 Mbits/sec
```

Fig. 2. The Iperf3 command was entered while running OAIC.

Iperf is an open-source tool that performs various measurements of network bandwidth and packet loss on IP

networks [8]. In this project, we will be using the third version of this software, Iperf3. This tool tracks the OAIC component's network performance, which is critical for ensuring the components are running. Figure 2 shows Iperf3 running with OAIC.

I. Denial of Service (DOS) Attack

A Denial of Service (DOS) attack occurs when a malicious actor prevents users from accessing devices and network resources [9]. DOS attacks frequently work by flooding or overwhelming a targeted device with requests until the device can no longer process normal traffic. In this project, we will perform two DOS attacks against OAIC components. The first DOS attack will be against the network base station, and the second DOS attack will also be on the base station, but with network slicing throughout the network.

J. Hping3

```
root@OAIC:~# sudo hping3 -S --flood -V -p 80 172.16.0.1
using lo, addr: 127.0.0.1, MTU: 1500
HPING 172.16.0.1 (lo 172.16.0.1): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
```

Fig. 3. Hping3 running a DOS against an OAIC component.

Hping is a tool used to create and send custom network packets (ICMP/UDP/TCP) and displays replies like ping would [10]. Although it is a more advanced ping, it can be used to run different types of network attacks. In this project, we will be using the third version of this tool to run a DOS attack against the OAIC components. One of the uses of the OAIC tool is to experiment with 5G components without any effect on an actual network. Figure 3 shows hping3 running a DOS attack against an OAIC component.

K. NexRAN

The NexRAN app implements RAN slicing by sending instructions to RAN nodes that perform custom resource allocation and bind UEs to those slices of allocated resources [11]. For this project, we will slice our 5G network into fast and slow slices. The number of slices in the network will affect the network's data rates, and we will test how well these slices run with different numbers of slices. We will also test how these slices perform while under a DOS attack.

L. Intended Audience

This project was developed for various purposes and audiences. Students and researchers are the target audience, intended to help both learn and experiment with 5G networks. OAIC allows users to configure, deploy, and experiment with a 5G network and its components. For students, OAIC provides a learning experience for students interested in topics related to wireless systems. You can learn about the various components and parts of a wireless network, like a 5G network. Students can also run multiple types of attacks against this network, allowing them to learn other topics in the cybersecurity field. For researchers, OAIC will enable you to run different experiments and attacks to see how various components perform. OAIC also provides a testing platform where you can run programs and scripts within the platform to produce data outputs. This project will attempt to run OAIC and experiment with it to test if it is a good fit for students and researchers.

III. PROCEDURE

A. Lab Setup

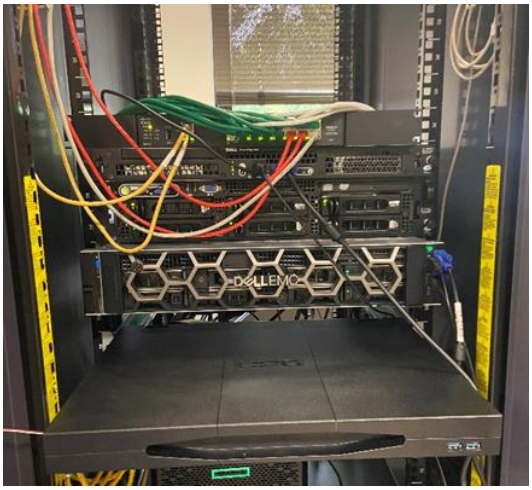


Fig. 4. The 5G Research Testbed. The server labeled “Dell EMC” is the server that will be used for this project.

For my capstone, I used a testbed designed for researching and experimenting with open-source 5G tools. This testbed consists of 3 servers: small, medium, and large. They are labeled based on the storage size of each server. For this project, I will use the large server, a Dell PowerEdge R7515 Rack Server, which has Ubuntu 24.04 as its operating system [12]. Figure 4 shows the testbed where OAIC will be running. OAIC’s components and additional software are fully installed and functional on this server. Although I am using a server for this project, this can also be installed on a laptop or PC at home.

OAIC provides instructions for allowing users to create their own 5G network. This can be used for testing 5G network components, learning about 5G, and developing tools and software related to mobile networks. When running a 5G network with OAIC, there are a few components used that are important to know. The first component is the EPC, which represents the core of the network. It is used for session management, mobility management, and authentication. The second component is the gNB/gNodeB, which functions like a base station, providing connectivity between the EPC and the user. The third component is the User Equipment (UE), which is the mobile device that will be connected to the 5G network.

B. OAIC Installation

The OAIC installation process has various stages that must be completed in the correct order. Each stage is important, as it configures and sets up various tools for OAIC to run properly.

- Hardware & Software Requirements – This section covers the requirements needed to install OAIC and its dependencies. We also discuss the different ways we can set up OAIC to make it easier to install.
- O-RAN Near-Real Time RIC & srsRAN Installation – In this section, we have four important steps that we follow to install the O-RAN Near-Real Time RIC. The first step is to install Ubuntu and have the operating system running. The second step is installing Docker, Kubernetes, and Helm, all critical programs needed to run OAIC. The third step is to build a modified docker image using OAIC's DockerHub images. The last step is to deploy the Near-Real Time RIC, which requires the Kubernetes clusters to be deployed and running. This section also has some installations to get srsRAN installed, which includes ZeroMQ and Ettus UHD binary.
- Running Your Own 5G Network – This section covers all of the components that OAIC will use to simulate a 5G network. I will explain the purpose and use of each component and highlight the communication that occurs between them. I will also share some challenges I encountered while running OAIC with the components for the first time.

1) Hardware & Software Requirements

OAIC will be installed on any device that meets the specifications provided in OAIC's installation documentation. The operating system required for OAIC is Ubuntu version 20.04 or later. The hardware required includes a CPU with 2-4 cores, 16GB or more of RAM, and a minimum storage capacity of 80GB. To allow non-Linux users to install and configure OAIC, users can even install the software on a virtual machine (VM), which has the benefit of running multiple installations of OAIC on different VMs. Students can even install OAIC for personal use on their computers if they meet the software's requirements. As previously mentioned, we will be installing OAIC on one of the servers in our testbed to utilize equipment used in the IT industry.

2) O-RAN Near-Real Time RIC & srsRAN Installation

After ensuring our system meets the software and hardware requirements, we can install the software needed to get OAIC up and running. The first thing we need to get running is the O-RAN RIC, which is responsible for controlling and optimizing Radio Access Network (RAN) functions. OAIC documentation provides us with a script, as shown in Figure 5, that will install Kubernetes, Docker, and Helm. The script will also install pods that help with cluster, service creation, and internetworking between services.

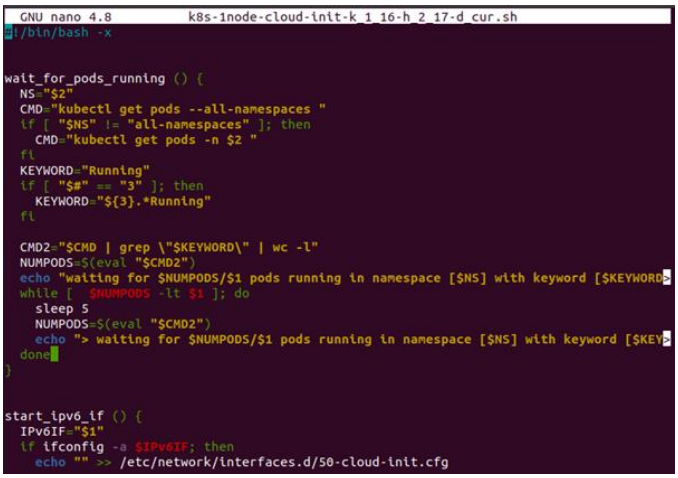


Fig. 5. OAIC created a small portion of the script to install Kubernetes, Docker, and Helm.

While running the script, I ran into some problems with the script finishing successfully. The script got stuck in a loop while configuring and running the pods. Figure 6 shows a message saying, "No resources found in kube-system

namespace." This issue with the script not being able to find the resources resulted in a loop where the script was waiting for the eight pods to run. After spending hours researching this issue, I realized that the various solutions provided for solving this problem that I had attempted to do were not working. I then decided to post about this issue on OAIC's GitHub page, hoping a developer would assist with this problem. Unfortunately, there was no response after waiting for multiple weeks, so I had to troubleshoot this issue independently. Eventually, I solved this issue by running the script a second time, and the eight pods started to run. Figure 7 shows a list of all the pods created after the script is finished running.

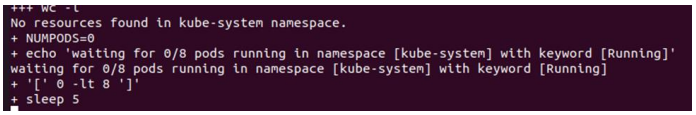


Fig. 6. The pod configuration issue.

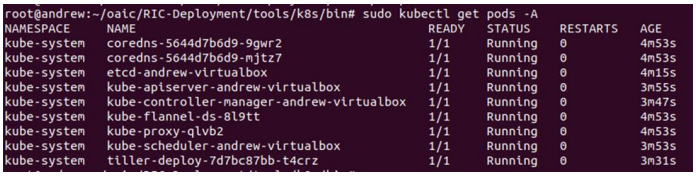


Fig. 7. The Kubernetes list of running pods.

The next step is to build a modified docker image. First, we must create a local docker registry to host the docker images. A local docker registry is a registry you host and manage your own infrastructure. Once the docker registry is created, we need to pull a docker image from OAIC's DockerHub and then push it to the local registry, as shown in Figure 8. With the docker image pulled and pushed, we can then deploy the near-real-time RIC. The specific command we need to use to deploy the RIC uses something called a recipe. A recipe provides customized requirements for the components of a deployment group for a specific deployment site. With the specific recipe designed for OAIC, it is deployed with the near-real-time RIC.

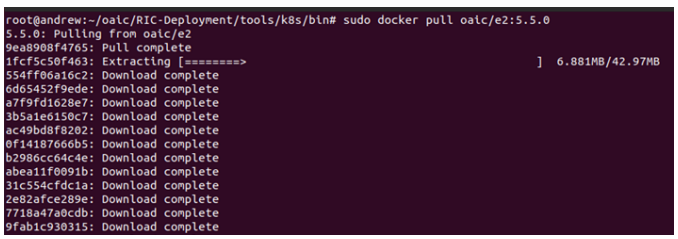


Fig. 8. Building a modified docker image.

With the O-RAN Near-Real Time RIC installed, we can install the srsRAN. We need to have ZeroMQ installed, in which srsRAN uses its networking library to transfer radio samples between applications. We also need to have Ettus UHD binary and the asn1c compiler installed. The installation process for srsRAN is simple and requires running a script created by OAIC to get it installed and running. Figure 9 shows srsRAN installing on the server.

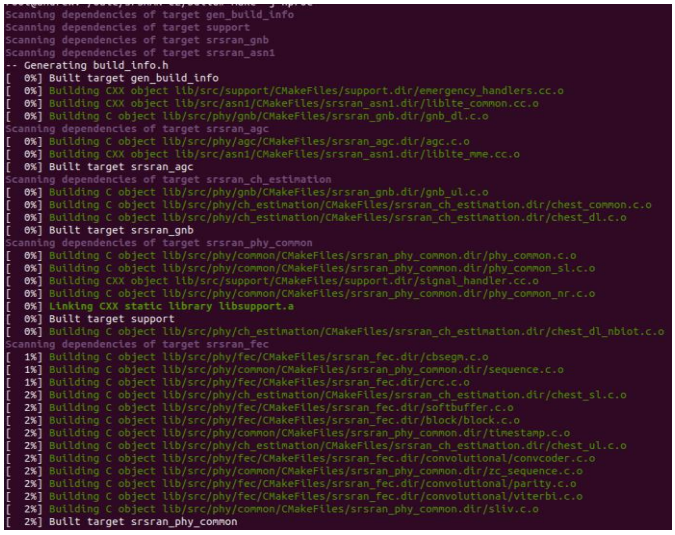


Fig. 9. The srsRAN installation process.

3) Running Your Own 5G Network

With everything installed, we can start up all the components needed to run a 5G network. Before we start each component, we need to remember that each component must be started in the correct order. If they are not opened in the correct order, OAIC will not run properly. The components are all running on separate terminal windows, as they are each running different things. The first component we will start is the EPC, which represents the core of the network [13]. Figure 10 shows the command to start the EPC and how it looks while running.

```
root@OAIC:~# sudo srsepc

Built in RelWithDebInfo mode using commit 384d343 on branch HEAD.


---  Software Radio Systems EPC  ---

Couldn't open , trying /root/.config/srsran/epc.conf
Reading configuration file /root/.config/srsran/epc.conf...
Couldn't open user_db.csv, trying /root/.config/srsran/user_db.csv
HSS Initialized.
MME S11 Initialized
MME GTP-C Initialized
MME Initialized. MCC: 0xf001, MNC: 0xff01
SPGW GTP-U Initialized.
SPGW S11 Initialized.
SP-GW Initialized.
Received S1 Setup Request.
S1 Setup Request - eNB Name: enb1, eNB id: 0x19b
S1 Setup Request - MCC:001, MNC:01
S1 Setup Request - TAC 7, B-PLMN 0xf110
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: 001010123456789
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
Downlink NAS: Sending Authentication Request
UL NAS: Received Authentication Response
Authentication Response -- IMSI 001010123456789
UE Authentication Accepted.
```

Fig. 10. OAIC's 5G network's EPC running.

The next component to start is the gNB/gNodeB, which represents the base station or cell tower [14]. Before the command to start the gNodeB, we need to note some critical IP information. We need to find the current machine's IP address and the IP address of the E2 Termination service at the near-RT RIC. Once we have these IP addresses, we can then run our gNodeB. Figure 11 shows the IP addresses being collected and the gNodeB running. The last component we need to start is the UE, which is the user equipment or the device connecting to the cell tower [15]. Figure 12 shows the UE running.

```
RACH:  tti=1301, cc=0, preamble=48, offset=0, temp_crnti=0x46
User 0x46 connected
User 0x46 connected
User 0x46 connected
RACH:  slot=2011, cc=0, preamble=0, offset=0, temp_crnti=0x4602
Disconnecting rnti=0x4602.
Disconnecting rnti=0x46.
Disconnecting rnti=0x4601.
RACH:  tti=9201, cc=0, preamble=44, offset=0, temp_crnti=0x47
User 0x47 connected
RACH:  slot=9451, cc=0, preamble=0, offset=0, temp_crnti=0x4604
Disconnecting rnti=0x4604.
User 0x47 connected
User 0x47 connected
Disconnecting rnti=0x47.
Disconnecting rnti=0x4603.
RACH:  tti=9941, cc=0, preamble=30, offset=0, temp_crnti=0x48
User 0x48 connected
User 0x48 connected
User 0x48 connected
RACH:  slot=10211, cc=0, preamble=0, offset=0, temp_crnti=0x4606
Disconnecting rnti=0x4606.
Disconnecting rnti=0x48.
Disconnecting rnti=0x4605.
RACH:  tti=8931, cc=0, preamble=42, offset=0, temp_crnti=0x49
User 0x49 connected
User 0x49 connected
User 0x49 connected
RACH:  slot=9211, cc=0, preamble=0, offset=0, temp_crnti=0x4608
Disconnecting rnti=0x4608.
```

Fig. 11. OAIC's 5G network's gNB/gNodeB running.

```
Available RF device list: UHD  zmq
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
CH1 rx_port=tcp://localhost:2100
CH1 tx_port=tcp://*:2101
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, CP=Normal, CFO=-0.3 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=00101, TAC=7
Random Access Transmission: seq=48, tti=1301, ra-rnti=0x2
RRC Connected
Random Access Complete.     c-rnti=0x46, ta=0
Network attach successful. IP: 172.16.0.2
Software Radio Systems RAN (srsRAN) 5/10/2023 23:56:5 TZ:0
RRC NR reconfiguration successful.
Random Access Transmission: prach_occasion=0, preamble_index=0, ra-rnti=0xf, tti=2011
Random Access Complete.     c-rnti=0x4601, ta=0
```

Fig. 12. OAIC's 5G network's UE running.

To confirm that every component is connected to each other, we need to make sure that an IP address for the UE is shown within the terminal, like in Figure 12. With every component running in our 5G network and an IP address provided for the UE, we must exchange traffic to confirm that our network works correctly. This will confirm whether the UE and EPC can communicate with each other. We will use iperf3 to check that there is incoming traffic from both the UE and EPC. Figure 13 shows iperf running from the EPC side, and Figure 14 shows iperf running from the UE side.

```
root@andrew:~# iperf3 -s -i 1
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 172.16.0.2, port 33290
[  5] local 172.16.0.1 port 5201 connected to 172.16.0.2 port 33298
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-1.00   sec   204 KBytes  1.67 Mbits/sec
[  5]   1.00-2.00   sec   143 KBytes  1.17 Mbits/sec
[  5]   2.00-3.00   sec   205 KBytes  1.68 Mbits/sec
```

Fig. 13. Iperf running from the EPC side of the network.

```
root@andrew:~# sudo ip netns exec ue1 iperf3 -c 172.16.0.1 -b 10M -i 1 -t 60
Connecting to host 172.16.0.1, port 5201
[  5] local 172.16.0.2 port 33298 connected to 172.16.0.1 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec   424 KBytes  3.47 Mbits/sec   19   25.5 KBytes
[  5]   1.00-2.00   sec   128 KBytes  1.05 Mbits/sec    2   24.0 KBytes
[  5]   2.00-3.00   sec   256 KBytes  2.10 Mbits/sec    0   25.5 KBytes
[  5]   3.00-4.00   sec   128 KBytes  1.05 Mbits/sec    1   24.0 KBytes
```

Fig. 14. Iperf running from the EU side of the network.

While starting each of these components for the first time, I ran into various issues that prevented me from running them all together. The first issue was with gNodeB, where it would get stuck in a loop while trying to establish a socket connection to a specific IP address. While looking into this IP address, I found that it is the IP address for a specific pod, which was the service-ricplt-e2term-sctp-alpha pod. After hours of attempting to fix this issue, I went to OAIC's GitHub page to see if anyone else had experienced this issue. One user ran into the same issue and claimed the solution was to "fix the Kubernetes IP address when installing it." This solution needed clarification, and I needed help figuring out how to do that. I

restarted the OAIC installation from scratch, and the issue was fixed. I still do not know other solutions for this issue, but it was resolved for me. Figure 15 shows this issue and highlights the pod that was involved.

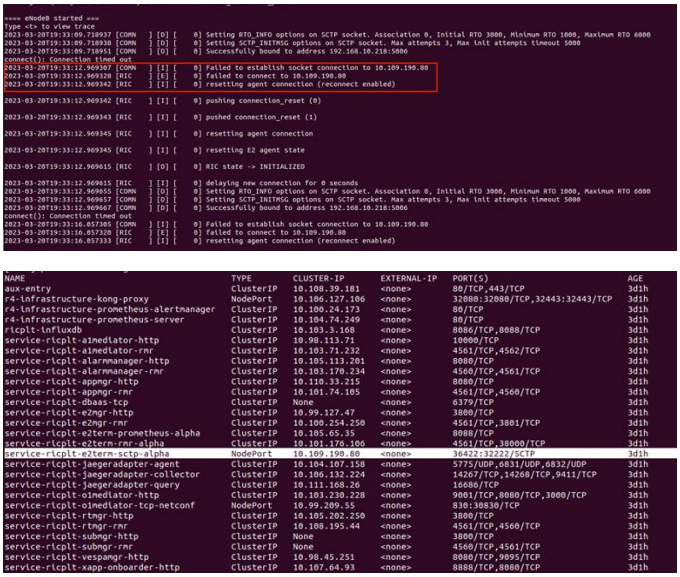


Fig. 15. Socket connection to pod failure.

The second issue I ran into while starting OAIC's components was that leaving components alone for more than 10 seconds caused them to go idle. I found this out when I did not use iperf3 to send traffic from the EPC side of the network to the UE side. When I did not send this traffic, the UE would prompt a message in the terminal saying “RRC IDLE,” as shown in Figure 16. The RRC (Radio Resource Control) is the process by which the UE and gNodeB establish a connection. I eventually realized that I needed to send traffic between components for the network to continue running. Without the traffic running between the two, the RRC process cannot finish, resulting in the components not working with each other.

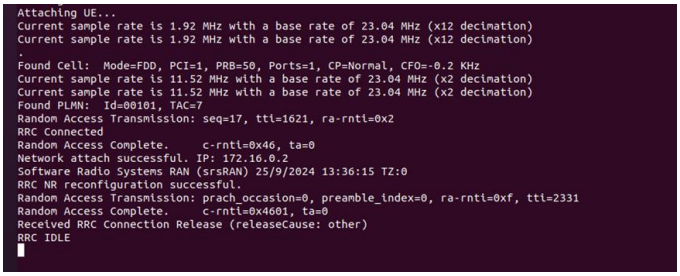


Fig. 16. RRC idle problem.

C. Experiments

1) Denial-of-Service (DOS)

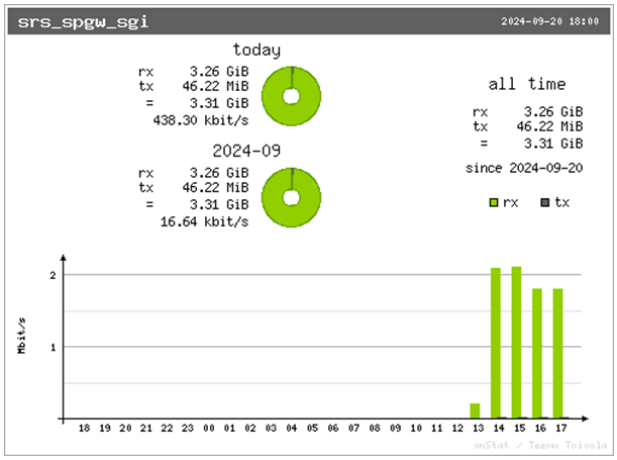


Fig. 17. The bit rate of the 5G network per hour. Hours 14 and 15 are with normal traffic, and hours 16 and 17 are with DOS attack traffic.

Our first experiment will test how well components within the configured OAIC installation can handle Denial-of-service (DOS) traffic. We ran normal traffic (pings) on the simulated 5G network for 2 hours to get a baseline of data rates, amount of traffic sent, and component performance. After running normal traffic, we ran a DOS attack on the gNodeB for 2 hours to compare component performance with the normal traffic performance.

For the normal traffic, we used vnStat to track network and component performance and ping to send traffic within the network [16]. vnStat is a tool to track data involving specific network adapters and creates graphs with the data collected. The ping traffic will be used as a baseline to see the network performance without any large traffic. We would ping for 2 hours and then transition into running a DOS attack on the gNodeB. We used hping3 to flood the network with requests for our DOS attack to see how it would impact overall performance. While monitoring normal and DOS traffic, vnStat created a graph showing data rates per hour, logged ping time to see any delays in responses, and logged iperf bitrates from both the user equipment and network side. My prediction for this experiment is that there will be a slight decrease in overall network performance.

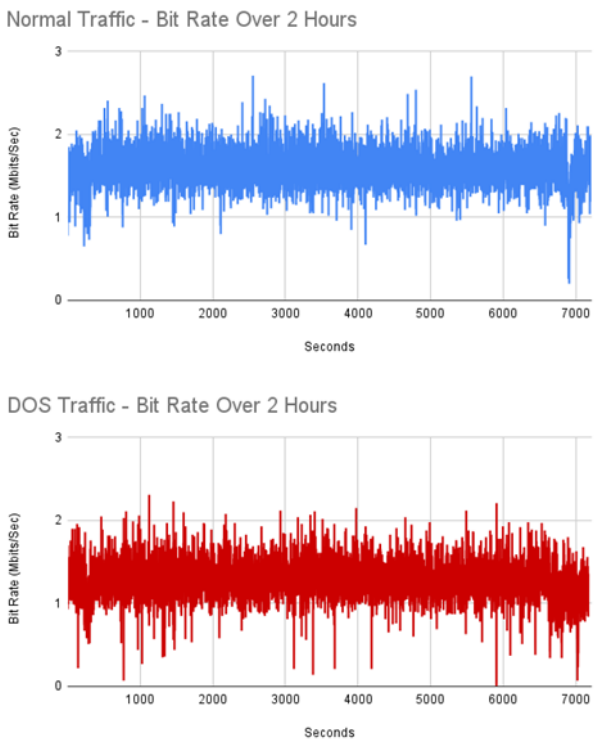


Fig. 18. A comparison of bit rates during normal traffic and DOS traffic.

The results from this experiment show that there was a decrease in bit rate and longer response times while running the DOS attack. Figure 17 shows the bit rate from the network interface srs_spgw_sgi, where this AI-enabled 5G network is running. The figure shows the bit rate increasing from hours 14 to 15, which is when the normal traffic was sent. It also indicates that the bit rate decreased from hours 15 to 16 when the DOS attack began to run. Figure 18 shows the second-by-second bit rate of the base station, tracked with iperf, from both the normal and DOS traffic over 2 hours. The normal traffic bit rate is higher overall, while the DOS traffic is lower and has multiple spikes where the bit rate drops close to 0. We can see this when comparing the averages of bit rates under normal and DOS conditions. The bit rate during normal traffic averaged 1.59 Mbits/sec, while the bit rate averaged 1.28 Mbits/sec during DOS traffic. This proves that the DOS had an impact on the network's bit rate.

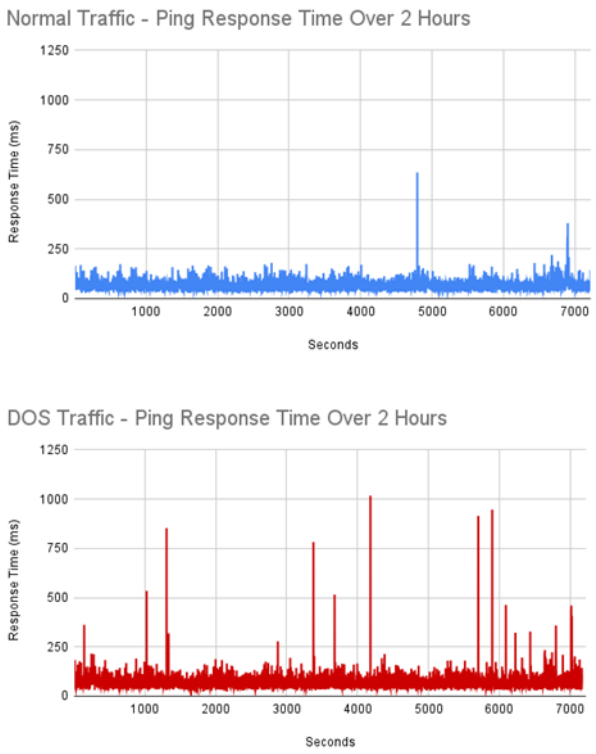


Fig. 19. A comparison of ping response time during normal traffic and DOS traffic.

Figure 19 shows the second-by-second ping response time when pinging the base station during normal and DOS traffic over 2 hours. The response time was quicker during normal traffic, while there were moments during the DOS traffic when it would take more than 500ms for a response. The attack has impacted the network response time, with many spikes in response time. This is shown when we compare the average ping response time in both types of traffic. The average ping response time under normal traffic was 66.38ms, while the average response time under DOS traffic was 71.43ms.

2. Denial-of-Service (DOS) on Network Slices

```
root@andrew-VirtualBox:~/oaic/nexran# curl -i -X PUT -H "Content-type: application/json" -d '
{"allocation_policy":{"type":"proportional","share":1024}}' http://${NEXRAN_XAPP}:8000/v1/sli
ces/slow ; echo ; echo
HTTP/1.1 200 OK
Connection: Close
Content-Length: 0
```

```
root@andrew-VirtualBox:~/oaic/nexran# curl -i -X PUT -H "Content-type: application/json" -d '
{"allocation_policy":{"type":"proportional","share":256}}' http://${NEXRAN_XAPP}:8000/v1/slic
es/fast ; echo ; echo
HTTP/1.1 200 OK
Connection: Close
Content-Length: 0
```

Fig. 20. Implementing slow slicing (top) and fast slicing (bottom).

Our second experiment will test how well components within the configured OAIC installation with RAN slicing can handle Denial-of-service (DOS) traffic. The NexRAN tool is

used to implement these slices on the network. OAIC offers two different ways of implementing RAN slicing, which includes slow and fast slices. Slow slicing is where there are fewer slices, meaning higher data rates for each slice. Fast slicing is where there are more slices, meaning lower data rates. For each type of slicing, we collected ping and iperf information in normal traffic for 2 hours, then collected the same information while under a DOS attack for 2 hours. Figure 20 shows the implementation of slow and fast slices on the simulated 5G network.

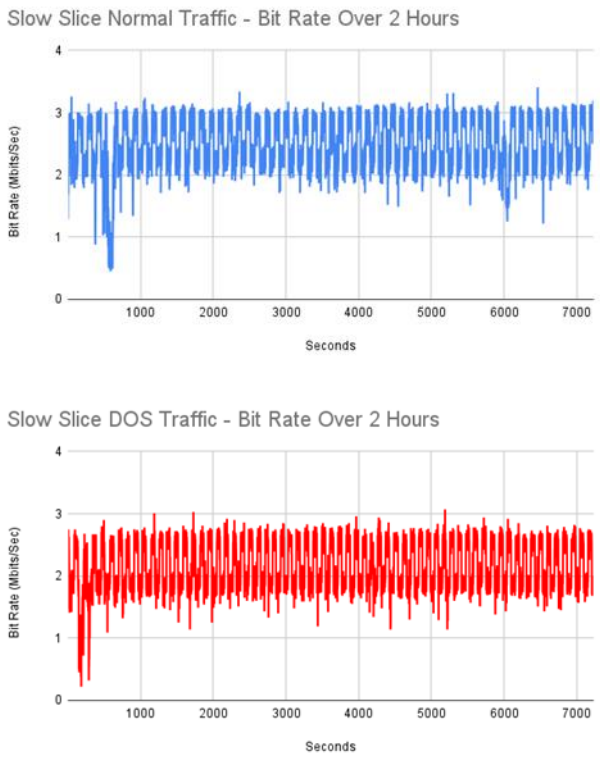


Fig. 21. A comparison of bit rates during slow-sliced normal traffic and DOS traffic.

This experiment's results show a noticeable decrease in bit rate and slightly longer response times for the slow slices. Figure 21 shows the second-by-second bit rate of the base station with slow slices, tracked with iperf, from both the normal and DOS traffic over 2 hours. The normal traffic bit rate is higher overall, while the DOS traffic is lower, and both had a significant spike towards the beginning, where their bit rate reached below 1 Mbit/sec. We can confirm that DOS traffic's average bit rate was lower by looking at the average bit rate in normal traffic. The average bit rate for the slow sliced network in normal traffic was 2.51 Mbits/sec, while the average bit rate in DOS traffic was 2.16 Mbits/sec. When comparing these results to the non-sliced network, the slow sliced network had jumps in data rates from high to low, which is something to pay attention to. There are also fewer spikes where the bit rate reaches below 1 Mbit/sec.

Fig. 22. A comparison of ping response time during slow-sliced normal traffic and DOS traffic.

Figure 22 shows the second-by-second ping response time when pinging the base station with slow slices during normal and DOS traffic over 2 hours. The ping response time was faster during normal traffic, but there were only a few spikes where the response time was almost 400ms. There were fewer spikes in response time than the non-sliced network, except for one massive spike in the slow sliced network towards the beginning. Looking at the ping response time averages in both traffic conditions, we see a higher ping response time under DOS conditions. The average ping response time in normal traffic conditions was 143.83ms, while the average ping response in DOS traffic conditions was 165.53ms.

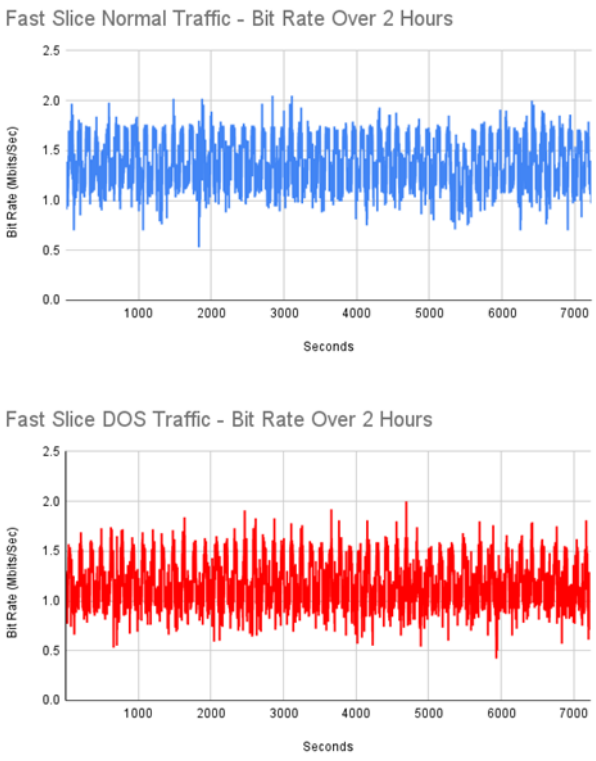
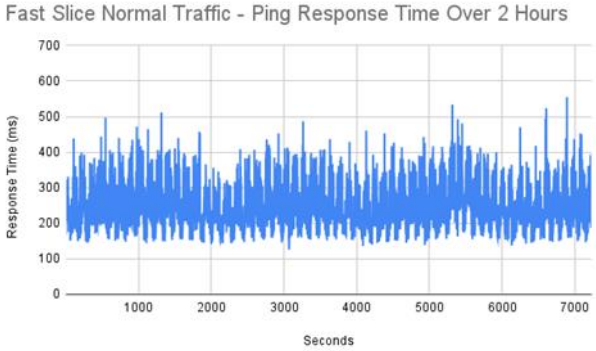


Fig. 23. A comparison of bit rates during fast-sliced normal traffic and DOS traffic.

The results from this experiment show that the fast slices' bit rate slightly decreases during a DOS attack. Figure 23 shows the second-by-second bit rate of the base station with fast slices, tracked with iperf, from both the normal and DOS traffic over 2 hours. Although the bit rate during the DOS is slightly lower, the attack did not significantly impact network performance. The fast slices managed this attack very well. We can prove this by looking at the bit rate averages in both types of traffic. The average bit rate in normal traffic is 1.37 Mbits/sec, while the average bit rate in DOS traffic is 1.16 Mbits/sec.



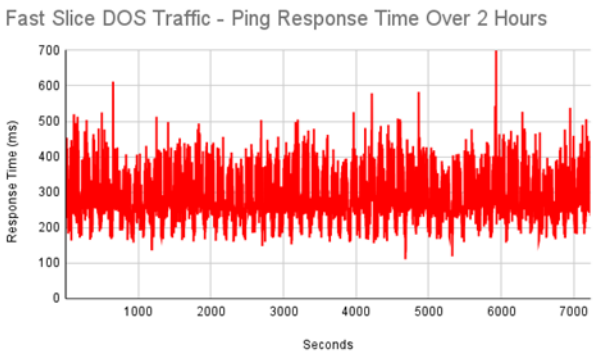


Fig. 24. A comparison of ping response time during fast-sliced normal traffic and DOS traffic.

Figure 24 shows the second-by-second ping response time when pinging the base station with fast slices during normal and DOS traffic over 2 hours. There is a noticeable difference in response time between the normal and DOS traffic. he DOS traffic had longer response times than normal traffic, which is the common trend with each experiment. The average ping response time in normal traffic was 245.57ms, while the average ping response time in DOS traffic was 288.97ms.

Figure 25 summarizes the key metrics for each network scenario covered in these experiments. If we compare each of these scenarios, the slow-sliced network had the best bit rate averages, while the normal network had the lowest ping response times. With these averages, we need to determine where our priorities are with the network. In the normal network, we have average bit rates with the lowest ping response time. We have higher bit rates in the slow-sliced network but slightly higher ping response times. We have lower bit rates and higher ping response times in the fast-sliced network. It all comes down to personal preferences.

NETWORK METRICS IN DIFFERENT SCENARIOS

	Bit Rate (Mbits/Sec	Ping Response Time (ms)
Normal Traffic	1.59	66.38
DOS Traffic	1.28	71.43
Slow-Sliced Normal Traffic	2.51	143.83
Slow-Sliced DOS Traffic	2.16	165.53
Fast-Sliced Normal Traffic	1.38	245.58
Fast-Sliced DOS Traffic	1.16	288.97

Fig. 25. A table that compares the average network bit rate and ping response times with normal and DOS traffic, sliced and non-sliced, and slow-sliced vs fast-sliced

IV. Discussion

A. Conclusions

Based on the results of these experiments, the sliced 5G network appeared to handle the DOS attack better than the regular 5G network. The sliced network had little to no drops in bit rate and fewer large spikes in ping response time compared to the regular network. However, with slicing involved, we see higher ping response times. Of all the network options available, a slow-sliced network would work best. The slow-sliced network has the highest bit rates and slightly higher ping response times, but better than fast-slicing. This experiment reveals that a DOS attack can impact OAIC's mobile network's performance, but slow-slicing could reduce the effects on network performance.

With slicing reducing the effects of the DOS attack, we need to discuss reasons why this was possible. Slicing didn't completely stop the DOS attack, but it allowed the network to mitigate its impact to different slices. By having multiple slices in the network, we are isolating different parts of the network, allowing separate services to run without interrupting what's happening in the other slices. Based on this experiment, one of these slices took the majority of the DOS attack, while the rest was able to continue running as normal.

This capstone project was a success. OAIC was installed and deployed on the testbed, and we ran multiple experiments successfully, meaning anyone can use this tool to learn and experiment with 5G.

B. Challenges Faced

While I was beginning the process of preparing the lab for this capstone project, I ran into issues accessing the lab that had the testbed I was using. I was locked out of the lab for two weeks due to problems that were not in my control, resulting in a delayed start from when I wanted to begin this project. Along with the lab issues, I was also deciding whether to do a research-based capstone project or a hands-on capstone project. I decided to focus on a hands-on capstone since I would be using equipment, and it's a project that students can also try and set up.

While I was focused on installing OAIC, I previously described the various issues that held up the installation and deployment of the software. The first issue was the configuration of the pods during the installation. It caused me to stay in a loop because it could not find a specific namespace. The second issue was a socket connection issue with a particular pod set up in Kubernetes. It prevented me from running the gNodeB, one of the essential components. The final issue was an idle error, where I needed to run network traffic to keep the network functioning.

C. Lessons Learned

It took a lot of trial and error to get OAIC running and working correctly, but once it was working, its ability to test 5G components is helpful for those who want to experiment with 5G. This tool allowed me to learn more about what goes into making a 5G network and about industry-related tools, such as Docker and Kubernetes.

I also worked with a testbed, giving me the hands-on experience I sought. In the testbed, I learned how to manage and configure servers and how difficult it can be to run multiple programs and services simultaneously on one device.

If I had to restart this project, I would have tried installing OAIC on multiple devices to see how the installation process would have been. I successfully installed it on my laptop and the servers in the testbed, but I want to know if the installation process will run properly on other devices. I also would like to try different types of attacks on the components, such as a man-in-the-middle (MITM) attack, to capture traffic and interfere with its communications. Trying to interfere while the components were trying to establish a connection with each other would have been interesting to try out.

V. Conclusion

This project provides a detailed explanation of Open AI Cellular (OAIC), its features, the installation process, and deploying a 5G network on any device that it supports. It provides a great learning experience for students or for those interested in learning more about 5G networks. Not only can OAIC be used by students, but it can advance research in 5G. Projects using OAIC can be expanded to make changes to 5G and the future generations of mobile networks. All the software involved in making OAIC run is open source and available to utilize. Anyone can get this up and running within an hour or two.

If given more time, I would have also included OAIC's testing platform, OAIC-T, for more experiments and as an opportunity to learn more about programming [17]. OAIC-T would have allowed me to use test configuration files to configure a testing environment and to run various actors to test OAIC components. The tool captures statistics on component usage, provides a visualization of these performance metrics, and can provide any output as requested in the test configuration files. This would have been a great addition to this project, but the project still meets its goals of running a 5G network and experimenting with it.

BIBLIOGRAPHY

[1] "What Is 5g: Everything You Need to Know About 5G: 5G FAQ: Qualcomm." *Wireless Technology & Innovation*, www.qualcomm.com/5g/what-is-5g. Accessed 5 Sept. 2024.

[2] "Open AI Cellular (OAIC)." *Open AI Cellular (OAIC)*, www.openaicellular.org/. Accessed 3 Oct. 2024.

[3] "Enterprise Open Source and Linux." *Ubuntu*, ubuntu.com/. Accessed 5 Oct. 2024.

[4] "What Is Docker?" *Docker Documentation*, 10 Sept. 2024, docs.docker.com/get-started/docker-overview/.

[5] "Production-Grade Container Orchestration." *Kubernetes*, 1 Oct. 2024, kubernetes.io/.

[6] *Helm*, helm.sh/. Accessed 5 Oct. 2024.

[7] "SRSRAN Enterprise 5G - SRS: Software Radio Systems." *SRS*, 22 June 2023, srs.io/srsran-enterprise-5g/.

[8] "Iperf - the Ultimate Speed Test Tool for TCP, UDP and SCTPTEST the Limits of Your Network + Internet Neutrality Test." *iPerf.Fr*, iperf.fr/. Accessed 5 Oct. 2024.

[9] "Understanding Denial-of-Service Attacks: CISA." *Cybersecurity and Infrastructure Security Agency CISA*, 1 Feb. 2021, www.cisa.gov/news-events/news/understanding-denial-service-attacks.

[10] "HPING3: Kali Linux Tools." *Kali Linux*, 23 May 2024, www.kali.org/tools/hping3/.

[11] CCI-NextG-Testbed. "CCI-NextG-Testbed/NEXRAN: Modified Nexran Xapp from Powder That Works with E2ap v2.00+ and O-SC Ric F-Release and Above." *GitHub*, github.com/CCI-NextG-Testbed/nexran. Accessed 2 Oct. 2024.

[12] "Poweredge R7515 Rack Server: Dell USA." *Dell*, www.dell.com/en-us/shop/dell-poweredge-servers/poweredge-r7515-rack-server/spd/poweredge-r7515/pe_r7515_tm_vi_vp_sb. Accessed 5 Oct. 2024.

[13] "5G EPC (Evolved Packet Core)." *TELCOMA Training & Certifications*, 3 July 2024, telcomatraining.com/5g-epc-evolved-packet-core/.

[14] "What Is a GNB (gNodeB)?" Inseego, Inseego | United States, inseego.com/resources/5g-glossary/what-is-gnb/. Accessed 22 Oct. 2024.

[15] "What Is Ue?" Inseego, Inseego | Ireland, inseego.com/ie/resources/5g-glossary/what-is-ue/. Accessed 22 Oct. 2024.

[16] "vnStat." *VnStat - a Network Traffic Monitor for Linux and BSD*, humdi.net/vnstat/. Accessed 5 Oct. 2024.

[17] *OAIC-T*, openaicellular.github.io/oaic/oaic_t.html. Accessed 1 Oct. 2024.