

Virtual Machine (VM) and Container-based Deployment of 5G using Ansible with Security Implementation



Participants:

**Abdul Aziz Abdul Ghaffar
Hussaini Zubairu
Ezekiel Ndubisi**

Teacher:

Prof. Jun (Steed) Huang

TA:

Mohammed Abuibaid

Date:

09/April/2022

Table of Contents

Abstract	1
1. Introduction.....	2
5G Service Based Architecture (SBA)	3
Ansible Workflow.....	6
2. Literature Review	9
3. Proposed Architecture	13
4. Deployment Scenarios	15
Docker Deployment with Security Implementation	15
IPsec Security Implementation.....	15
Virtual Machine Deployment with Free5GC and UERANSIM	16
5. Results.....	32
Docker Deployment with Security Implementation	32
IPsec Security Implementation.....	34
Virtual Machine Deployment with Free5GC and UERANSIM	36
6. Conclusion	40
7. Appendices.....	42
Appendix A: IPsec Implementation Source Code	42
Appendix B: VM deployment Source Code	43
References	47

List of Figures

Figure 1: 5G Service Based Architecture [6]	3
Figure 2: Ansible Workflow	7
Figure 3: Network slice lifecycle and resource monitoring [9].....	9
Figure 4: 5G-Aware Evaluation Testbed Architecture Proposed in [13]	10
Figure 5: Proposed Solution.....	14
Figure 6: Network architecture for Docker deployment	15
Figure 7: Unencrypted transmission vs encrypted transmission using IPsec	16
Figure 8: Free5GC Deployment with 5G SA gNB and UE.....	17
Figure 9: Running ifconfig command to show IP address	18
Figure 10: Using SSH to connect to Ubuntu for Installing Free5GC.....	19
Figure 11: Update and Upgrade of Ubuntu	19
Figure 12: Running Ifconfig command to show updated IP settings	22
Figure 13: Showing network route using the route -n command	22
Figure 14: Running Free5GC	25
Figure 15: Building UERANSIM	27
Figure 16: Starting Free5GC web console	28
Figure 17: Configuring UERANSIM and Free5GC.....	29
Figure 18: Setting up UERANSIM	30
Figure 19: Ansible playbook running and configuring the Core network functions.....	32
Figure 20: Ansible playbook running the containers of the network entities.....	33
Figure 21: List of containers running inside the system.	33
Figure 22: Testing UE connectivity to the internet.	34
Figure 23: Transmission of a packet from the UE - Unencrypted packet	35
Figure 24: Transmission of a packet from the UE - Encrypted packet	35
Figure 25: Reception of packets on UPF	36
Figure 27: The result of a TestGUIRegistration procedure.....	37
Figure 28: The result of a TestServiceRequest procedure.....	38
Figure 29: UERANSIM connecting with Free5GC and connecting to the outside network	38
Figure 30: Configuration file of one of our UEs	39

Abstract

In response to the demand for an exponential increase in mobile data traffic and the provision of new-generation services, applications, or scenarios. The fifth-generation network (5G) was developed. To provide such services with high flexibility, new technologies and major adjustments to existing mobile networks are required. Amongst these adjustments is in its implementation. Another important aspect is enabling automation to introduce scalability and flexibility in the network. This paper demonstrates two 5G deployment options. First, a container-based (Docker) implementation of 5G using Ansible and a Virtual machine (VM) based implementation using Free5GC and UERANSIM. This paper further implemented IPsec on the Ansible deployment between the UE and UPF, thereafter, examined the results. Furthermore, we ran some 5GC procedures on Free5GC and delays were examined. Lastly, we demonstrated communication between Free5GC (core network) and a UE with the help of UERANSIM. The proposed solution enables operators to reduce expenditures by utilizing automation and implementing network security to protect the confidentiality of users' data.

1. Introduction

5G denotes a fifth-generation cellular telecommunication network. Although 5G networks will incorporate elements from earlier generations' wireless networks, 5G networks will usher in a new era of complete transition and transformation in the role of networks in modern society. 5G network is anticipated to support enhanced Mobile Broadband (eMBB), provision of higher speeds for web browsing, streaming, high definition (HD) videos, virtual reality (VR), augmented reality (AR) and video conferencing, Ultra-reliable and Low-latency Communications (URLLC); allows mission-critical applications, industrial automation, innovative medical applications, and self-driving vehicles that require very fast network traversal times, massive Machine Type Communications (mMTC): massive machine-type communications to allow Internet-of-Things (IoT), industrial Internet, Vehicle-to-Anything (V2X), industrial automation, platooning among others [1], [2]. These use cases require low latency, efficient throughput, and reliability. To meet these requirements, research activities aiming to implement automation of the 5G network through softwarization are currently gaining attention.

Network automation is the process of automating the testing, management, operation, deployment, and configuration of virtual and physical network devices [3]. Network automation improves a network's functioning while also increasing its operating efficiency and simplifying complex tasks. Network automation is geared towards addressing the increased demand for effective network services from a varied spectrum of end-users. The automation of network functions and the adoption of the Service Based Architecture (SBA) for the 5G Core Network has enabled the actual network functions towards their efficient management and a single-click deployment, through cloud-native approaches.

Ansible [4] is a configuration management, device provisioning, application deployment, and orchestration automation tool that is an easy and extensively used IT task. It is an open-source network automation tool that automates cloud provisioning, intra-service orchestration, configuration management, application deployment, and different network deployment-related tasks [5]. Ansible is primarily designed for multi-tier

deployments and therefore aligns with the cloud's basic structure. It is a Linux-based tool and the system on which Ansible is installed is called the control node, the devices (servers) that are automated with Ansible are termed the managed nodes.

5G Service Based Architecture (SBA)

The current 4G network of P2P techniques produces many interfaces between function pieces, which leads to dependencies between functions and makes it difficult to update a deployed architecture, which is critical in 5G. In order to take full advantage of the cloud environment and Network Function Virtualization (NFV), 3GPP mobile network must evolve to allow flexibility and enable both functional and service agility. Therefore, Service-based 5G core network architecture is designed to take advantage of network function virtualization and software-defined networking. The service-based 5G core network splits mobile functions into two categories: "stateless" control functions and state management functions by decoupling computation and storage resources [6].

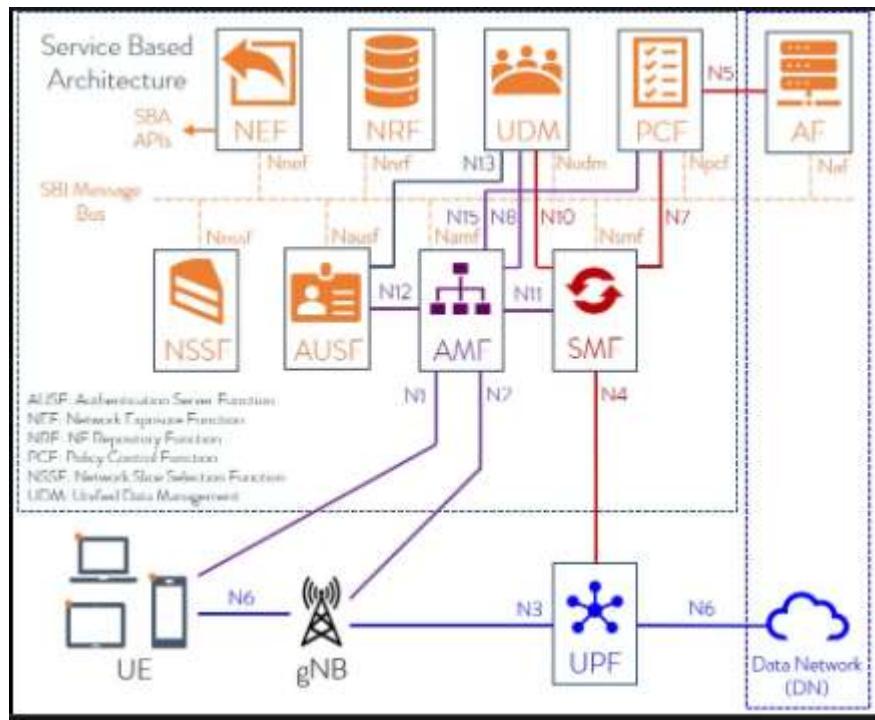


Figure 1: 5G Service Based Architecture [6]

The service-based 5G architecture [7] reorganizes mobile network functions into finer-grained network functions and micro-services that connect with others through

application-programmable interfaces (APIs) [8]. This innovative feature improves the reusability of mobile network services by allowing them to be orchestrated and shared across network slices [9]. Aside from control and data plane separation, the service-based 5G architecture adds a third degree of separation, decoupling computation and storage resources [6]. The service-based architecture of the 5G mobile core is shown in Figure 1.

The service-based architecture is designed around services that can register themselves and subscribe to other services. This makes it easier to construct new services because it's feasible to connect to other components without having to create new interfaces. This innovative system architecture is specified in the 3GPP technical specification [8]. The functional descriptions of the important Network functions are highlighted below:

Access and Mobility Management Function (AMF) includes Registration Management, Connection Management, Reachability Management, Mobility Management, and various function relating to security and access management and authorization. The AMF receives all connection and session-related information from the User Equipment (UE) (N1/N2) but is responsible only for handling connection and mobility management tasks.

Session Management Function (SMF) performs the following among others; Session Management, UE IP address allocation & management, DHCPv4 (server and client) and DHCPv6 (server and client) functions, respond to Address Resolution Protocol (ARP) requests, and or IPv6 Neighbour Solicitation requests based on local cache information for the Ethernet PDUs. The SMF responds to the ARP and or the IPv6 Neighbour Solicitation Request by providing the MAC address corresponding to the IP address sent in the request. The SMF does the Selection and control of the UP function, including controlling the UPF to proxy ARP or IPv6 Neighbour Discovery.

User Plane Functions (UPF) functionalities include anchor points for Intra-or Inter-RAT mobility (when applicable), external PDU Session point of interconnecting to Data Network, Packet routing, and forwarding, Packet inspection, user Plane part of policy rule enforcement, Lawful intercept (UP collection), traffic usage reporting, quality of

service handling for the user plane, uplink traffic verification, transport level packet marking in the uplink and downlink, downlink packet buffering and downlink data notification triggering a sending and forwarding of one or more "end marker" to the source NG-RAN node.

Policy Control Function (PCF) supports a unified policy framework, within the 5G infrastructure, for governing network behavior. It accesses the subscription information, required to make policy decisions, from the UDM and then provides the appropriate policy rules to the control plane functions so that they can enforce them. The PCF is similar to the PCRF in EPC architectures.

Unified Data Management (UDM) provides services to other service-based architectures functions, such as the Access and Mobility Management Function (AMF), Session management function (SMF), and Network Exposure Function (NEF). The UDM is commonly referred to as a stateful message store because it stores data in local memory. The UDM, on the other hand, can be stateless, storing data externally in a Unified Data Repository (UDR). The UDM supports user Identification Handling, it provides support for the de-concealment of privacy-protected subscription identifier (SUCI), and Access authorization based on subscription data. To provide these functionalities, the UDM uses subscription data that may be stored in UDR, in which case a UDM implements the application logic and does not require internal user data storage and then several different

Unified Data Repository (UDR): Stores and retrieves UDM subscription data, PCF policy data, application data, and other information.

Unstructured Data Storage Function (UDSF): The NF can store and retrieve unstructured data that is not defined in the 3GPP specification in the UDSF.

Network Slicing Selection Function (NSSF): Select the list of network slice instances serving the UE using the NSSF (Network Slice Selection Function).

NEF: It is responsible for opening the capabilities of the 5G network to external network elements as an API gateway accessible by external users. NEF allows network nodes to

provide functions and events to other nodes, and it completes the collection, analysis, and re-organization of network capabilities.

Non-3GPP Inter Working Function (N3IWF) oversees connecting the 5G core network to the untrusted non-3GPP access network (e.g., Wi-Fi). Create a tunnel between the UE and the core network's control plane and user plane via the interface.

UE radio Capability Management Function (UCMF) The UE radio function ID is saved here. The AMF can receive fresh values of the assigned UE Radio Function IDs from the UCMF and cache them locally.

Ansible Workflow

Ansible connects to the nodes and sends short software known as Ansible modules to them. After that, Ansible ran these modules and removed them when they were done. There are no daemons, servers, or databases required for the library of modules to run on any machine. As shown in Figure 2, the Management Node is the controlling node that oversees the playbook's full execution. The inventory file contains a list of hosts on which the Ansible modules must be installed. The Management Node establishes an SSH connection to the host system and executes the tiny modules, as well as installs the software.

Once the modules are perfectly installed, Ansible removes the modules. It connects to the host machine, runs the instructions, and then removes the code that was copied on the host machine if it was successful.

The major contributions of this project are the following:

- We first propose a solution to deploy the 5G network using an automation tool called Ansible. This will allow us to ease the deployment and management of the deployed 5G network. Additionally, this will enable network operators to save capital (CAPEX) and operational (OPEX) expenditures of the network.

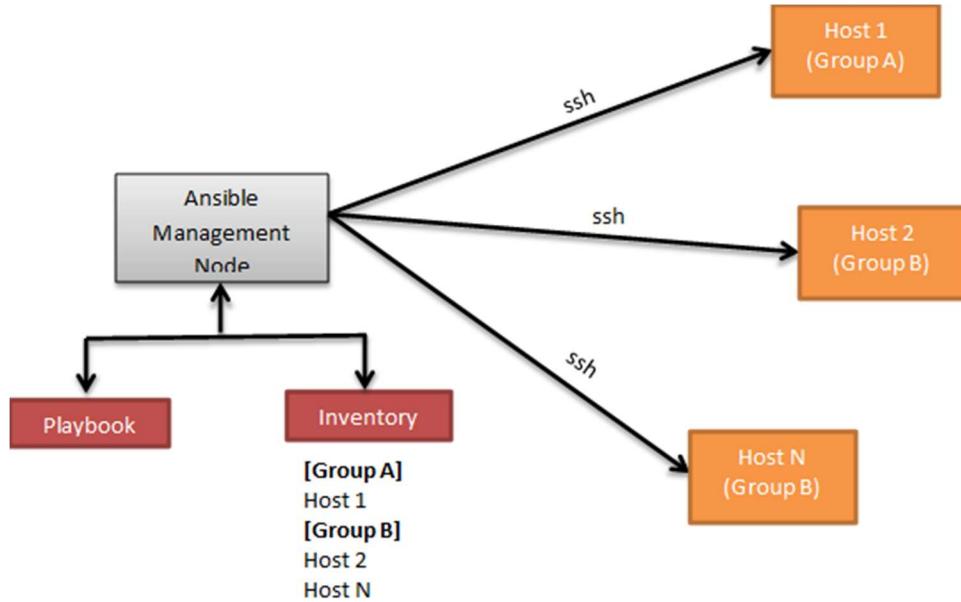


Figure 2: Ansible Workflow

- We then propose two different scenarios to deploy the 5G network. In the first scenario, we utilize the Docker tool to deploy the 5G network functions in a containerized environment. This virtualized deployment will enable network operators to add/remove network functions flexibly with minimal overhead and avoid the complexities of physical network function deployment. The 5G network components are deployed using Free5GC software.
- Furthermore, we also implement a 5G network using Virtual Machines. Particularly with the Free5GC and UERANSIM. We demonstrated this implementation by running the popular 5GC procedures with the help of the test.sh script [see appendix B]. We examined the delays of these procedures. Moreso, we demonstrated the implementation and communication between Free5GC and UERANSIM and lastly established a connection to the network (Internet) from a UE.
- Finally, we implement an IPsec security protocol to enable secure transmission of packets between UE and the UPF. We demonstrated with the help of an experiment that the use of this security protocol will provide confidentiality and integrity to the transmitted messages and protect the user from a range of malicious attacks.

The report is organized as follows: Section 2 provides a summary of the literature review and the research works that deal with introducing automation in 5G networks. Section 3 describes the proposed solution and the tools we utilize in our solution. Section 4 deals with the details of the deployment scenarios and implementation details. Next, Section 5 provides the detailed results of our implementations and experiments. Finally, Section 6 provides the conclusion of our report.

2. Literature Review

Previous studies have shown that deploying a 5G network in a cloud environment and introducing automation reduces the overall expenditure of the network. Arouk et al. [9] propose a 5G cloud-based network management and automation solution. The proposed solution implements the 5G core in a docker container. The authors introduce Kubernetes [10] and OpenShift operator [11] to enable flexible management of 5G services. The main 5G network components were implemented using OpenAirInterface (OAI) [12] nodes. Figure 3 shows the deployed architecture of this study. The authors implemented two different scenarios. In the first scenario, a monolithic RAN is deployed as a single node. Whereas, in the second scenario the RAN is divided into Distributed Unit (DU), and the Centralized Unit (CU), this is known as disaggregated RAN. The Moisac5G operator is the OpenShift operator, and it can do three main tasks; 1. performing network configurations, 2. upgrading network entities, 3. switching between monolithic and disaggregated RAN. The authors claim that utilizing such cloud-based network architecture will help operators to reduce network costs.

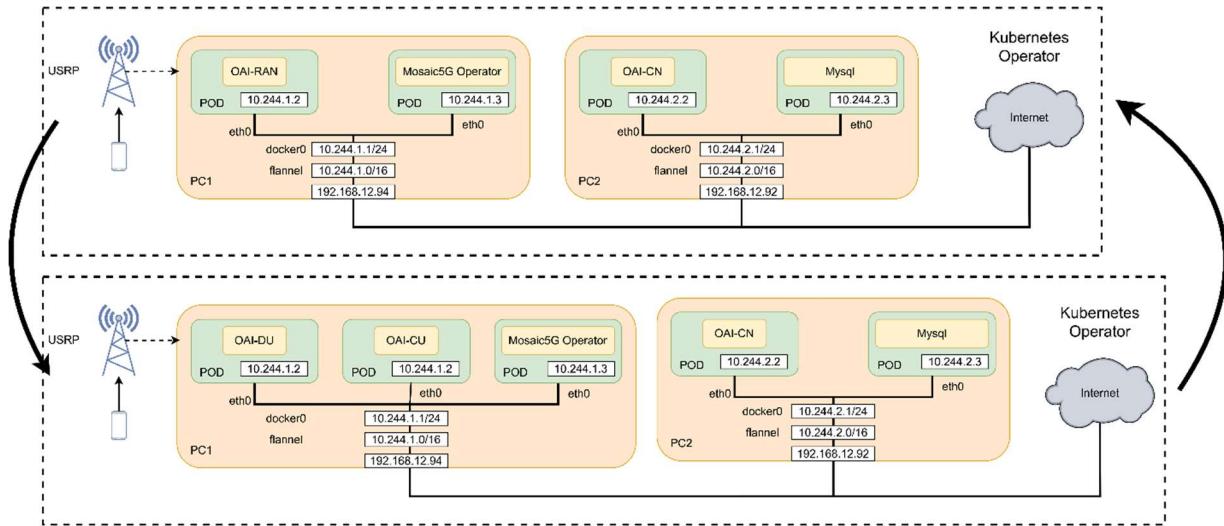


Figure 3: Network slice lifecycle and resource monitoring [9]

Bolivar et al. [13] propose a 5G testbed based on open-source software components. The proposed solution contains a combination of virtual machines as well as containers that operate simultaneously. The authors mainly use OpenStack components in their design.

Kuryr [14] was used to manage both the VMs and the containers and allow interoperability. The orchestration of the network was done using ManageIQ [15] along with Ansible [4]. The deployment scenario is shown in Figure 4. There are 4 different layers in this architecture: NFV Orchestrator (NFVO), the VNF Manager (VNFM), and the Virtualized Infrastructure Manager (VIM). For virtual machine (VM) deployment, the authors use OpenStack [16] based VMs. While for container deployment, Kubernetes [10] based containers are selected. The authors also mentioned that OpenShift [11] is another alternative for container deployment. Kuryr [14] is used as an integration bridge between the VMs and the containers and it allows interoperability between them. The authors utilize OpenStack components in order to manage the VMs and containers. Heat and Tacker are used to manage VM-based Virtual Network Functions (VNFs). While Magnum is used for container-based VNFs. Furthermore, ManageIQ is used as an orchestrator because it supports both VMs and containers, whereas Ansible is used to automate the network configurations.

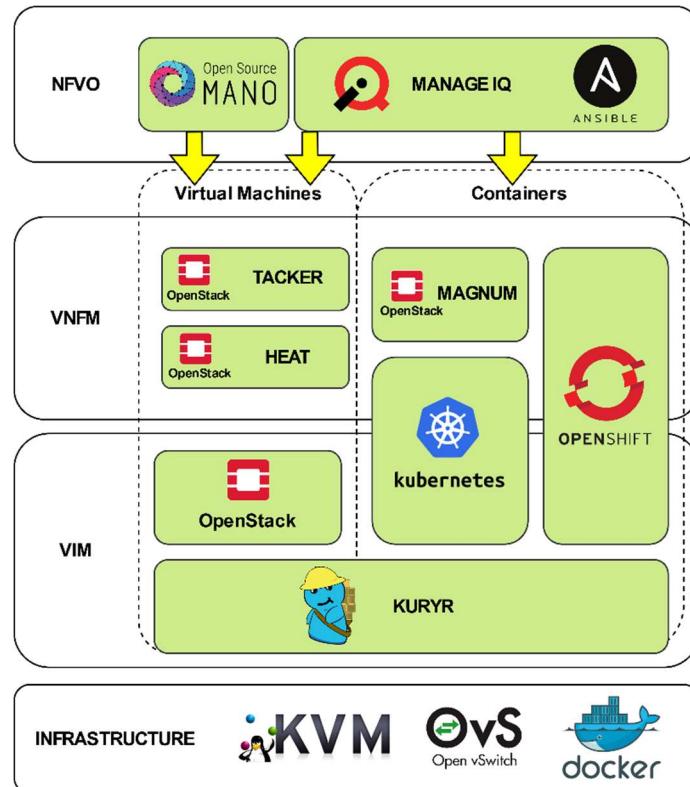


Figure 4: 5G-Aware Evaluation Testbed Architecture Proposed in [13]

Another study done by Afolabi et al. [17] proposes a framework to orchestrate and configure network slicing enabled 5G networks. The proposed solution consists of four different layers namely, Resources Layer (RL), Orchestration Layer (OL), Orchestration and Configuration Convergence Abstraction Layer (OCCAL), and Distributed Orchestration Abstraction Layer (DOAL). The first layer is DOAL, and it handles the cloud orchestration requests and performs different operations to select and sort VNFs of the same type and forward the request to the second layer which is OCCAL. OCCAL, on the other hand, is responsible to manage and register multiple NFVOs that are located in the Orchestration layer (OL). Once OL receives the instructions from OCCAL, it will instantiate and configure each network based on the instructions. Finally, the Resource layer (RL) contains the actual resources (physical or virtual) that are being used by the VIMs to deploy the VNFs. This solution also includes the Ansible tool [4] for automation. To evaluate the performance of the solution, the authors implemented two use cases including, network service orchestration and streaming service orchestration. For network service, the author deployed a 4G network using OpenAirInterface (OAI)[12]. While the streaming service is based on Nginx [18] implementation. Network service instantiation and network service configuration are selected as performance metrics. From the results, the author concluded that each service should be handled separately as the results for the two selected services were opposite to each other.

Huang et al. [19] integrate M-CORD and OAI to deploy and evaluate the network. The control and data plane functions of SGW and PGW are separated. The control plane functions are implemented on top of the control platform. The network functions are managed by OpenStack [16] while the automation is implemented by using Ansible [4]. The authors evaluate the deployment and conclude that the deployment overhead of this approach is 10 ms compared to the legacy OAI deployment.

In [20] authors covered almost everything in this paper, including the history and background of 5G companies and manufacturers, legacy RAN solutions and equipment, and various methods and approaches for building a 5G network, such as RAN, CORE, and EDGE frameworks, open-source software, and generic hardware. After that, this paper walks through the steps of building and configuring a 5G network with srsRAN, LimeSDR,

and a Raspberry Pi 4. Finally, the authors provided a list of the optimum PC and software-defined radio (SDR) combinations that can be particularly useful in the development of 5G networks. However, more closely to our work, Liu et al. [21] analyze and compares the Stand Alone (SA) NR and Non-Standalone (NSA) NR deployment modes in terms of coverage, network capability, interworking between 4G and 5G, complexity and cost of network deployment, and the latest industry progress. NSA NR performs better in interworking performance in the initial phase, while SA NR performs better in network capabilities, device performance, simple network deployment, and cost-efficiency. 5G SA NR is recommended for operators who have the ambition to explore new opportunities in the vertical and enterprise markets

3. Proposed Architecture

We utilize a combination of tools and software to propose multiple deployment scenarios for the 5G network. Figure 5 shows our proposed solution. Our proposed solution contains three main components. The first component of our solution is the automation platform. In our deployment, we employ the Ansible automation tool [4] to introduce automation in our network and allow ease of management and orchestration of the deployed network. Regarding the deployment scenarios, we consider two platforms, VirtualBox [22] and Docker [23], to deploy and run 5G network functions and components. In Docker deployment, each 5G network function and network entity, like UE, is deployed and run in a separate container. The major advantage of this approach is that this will enable network operators to reduce CAPEX and OPEX of the network since the deployed network is in a virtualized environment. Additionally, it will further ease the deployment of the network as automation reduces the complexities of installing and configuring network services and functions. Compared to the traditional deployment approach where the network administrator is required to install and configure each and every network function/service manually.

For a second deployment scenario, we consider Virtual Machine (VM) based 5G deployment using the VirtualBox platform. We implemented Free5GC and UERANSIM for managing the UEs and gNbs. We demonstrated this implementation first, by running three procedures and noting the delays. Again, we further illustrated the communication between the UERANSIM and Free5GC. Our result shows connectivity to the network (Internet) from a UE.

The third component of our proposed solution is the implementation of the IP security (IPsec) protocol. IPsec protocol is used to provide secure communication between different entities across the internet. In our implementation, we deploy Encapsulating Security Payload (ESP) function of IPsec. ESP provides confidentiality and encryption functions. The confidentiality function prevents unauthorized users to penetrate the network. While encryption prevents adversaries from eavesdropping on the communication and reading the messages or carrying out a man-in-the-middle attack.

We use the Scapy Python library [24] to implement IPsec security functionality in our network. More implementation details and results of the experiments are provided in subsequent sections.

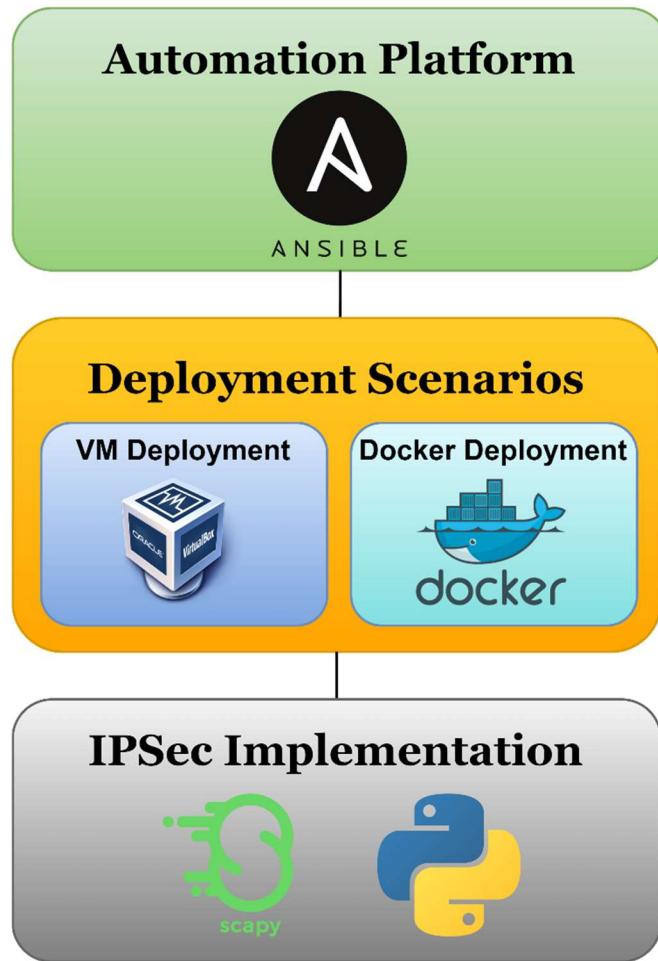


Figure 5: Proposed Solution

4. Deployment Scenarios

In this section, we explain both Docker-based and VM-based deployment scenarios and explain the IPsec implementation.

Docker Deployment with Security Implementation

We first deploy the proposed architecture using Docker and Ansible tools. The goal of this experiment is to analyze the benefits we obtain using such containerized deployment and the benefits of using automation through Ansible. For this deployment, we first create an Ansible playbook that handles the deployment and instantiation of various containers in the network. Each entity in the network is running in a separate isolated Docker container. As shown in Figure 6, the UE and eNB are deployed separately in different containers. Similarly, each of the 5G core network functions is also deployed in distinct containers. UE containers can communicate and exchange packets with eNB and 5GC containers through the use of tunnels. Each container has a network interface which is used to create a tunnel with other network containers.

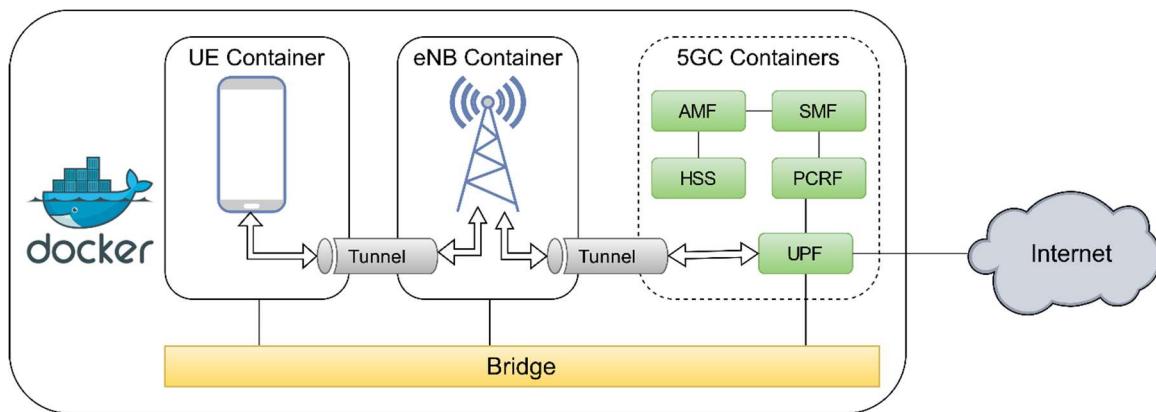


Figure 6: Network architecture for Docker deployment

IPsec Security Implementation

In order to secure and encrypt the communication between the UE and the internet, we implement IPsec security protocol in our deployed network. Figure 7 provides a visual comparison between an unencrypted deployment and an IPsec-enabled deployment. In the case of unencrypted deployment, any packet transmitted between the UE and the internet can be captured by an adversary, which can examine the content of that packet

and perform several attacks. To protect against the compromise of user data, we deploy the IPsec protocol. With IPsec, any packet sent by the UE will be encrypted and the content of the packet will not be visible to an attacker overseeing the network with malicious intent. To perform the experiment, we create a UDP packet in the UE with the destination IP address of the UPF. It is important to mention that we only use the IP address of UPF for the sake of this experiment as we will receive and observe the successful transmission of an encrypted packet on the UPF. Further details and results of this experiment are provided in the Results section. However, this can be expanded to a public IP address of any server on the internet. We utilize Scapy [24] Python library for packet manipulation including, creating the packet, encrypting the packet with IPsec protocol, transmitting the packet to the UPF, and finally capturing the packet at the UPF. This deployment will ensure confidentiality, integrity, and authentication of the transmitted data, and protect against replay attacks, man-in-the-middle attacks, unauthorized data tampering, etc.

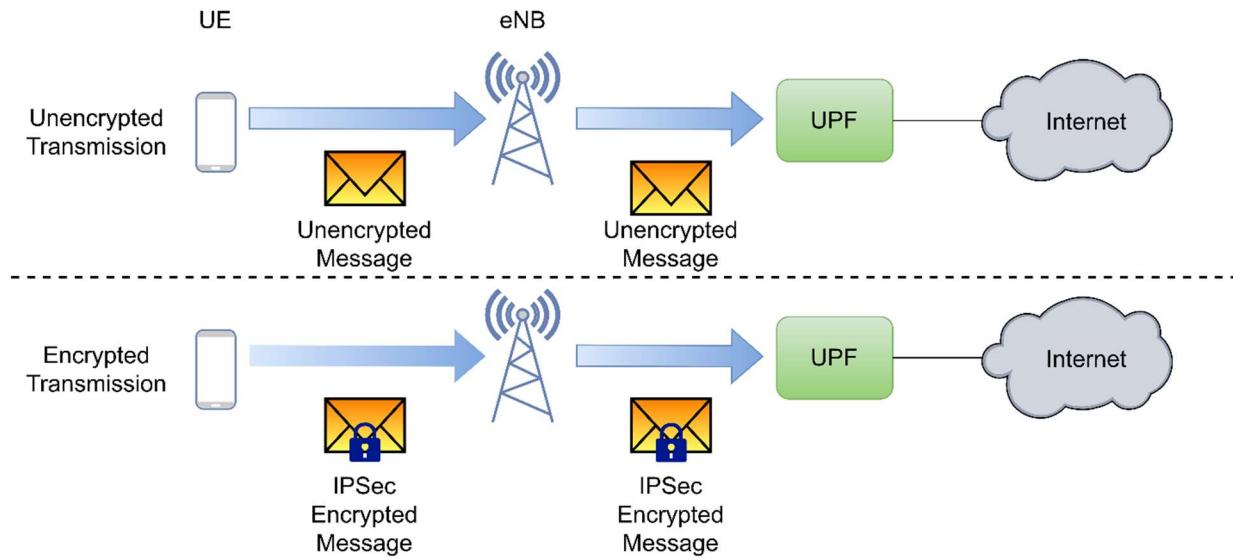


Figure 7: Unencrypted transmission vs encrypted transmission using IPsec

Virtual Machine Deployment with Free5GC and UERANSIM

The Free5GC [25] is an open-source project for mobile core networks of the fifth-generation (5G). This project's ultimate purpose is to implement the 5G core network

(5GC) as defined in 3GPP Release 15 (R15) and beyond. The overall architecture of the code refers to the SBA 5GC framework, as shown above in Figure 8, Service Based Architecture. The network elements involved are AMF, SMF, UPF, AUSF, N3IWF, NRF, NSSF, PCF, and UDM, the functions of each network element are explained in the introductory aspect of this work. For this deployment, our environment requires the creation of two virtual machines: one named UERANSIM, which acts as a (R)AN and UE, and the other named Free5GC, which acts as a 5GC Core network and is used to deploy and run and test the connectivity of our 5G network.

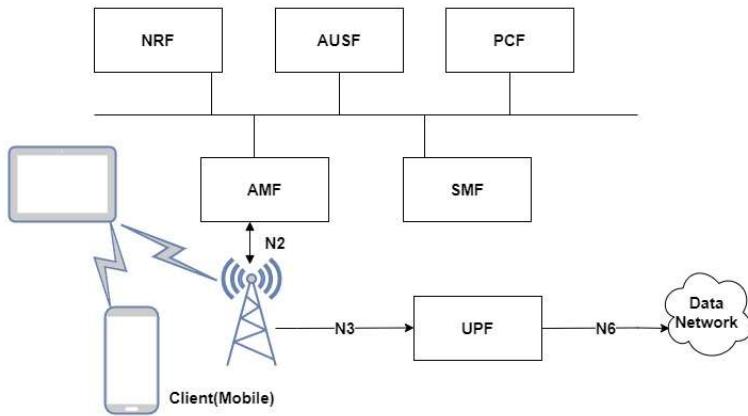


Figure 8: Free5GC Deployment with 5G SA gNB and UE

Environment Setup

Free5GC was tested against the following environment:

Software OS: Ubuntu 20.04LTS, gcc 7.3.0, Go 1.14. Linux/amd64, kernel version 5.0.0-23-generic. The listed kernel version is required for the UPF element.

Hardware Configuration: CPU: Intel i5 processor, RAM: 16GB, Hard drive: 256GB

NIC: Supports 1-10Gbps Ethernet card in the Linux kernel

UERANSIM was deployed for managing UEs and gNodeB

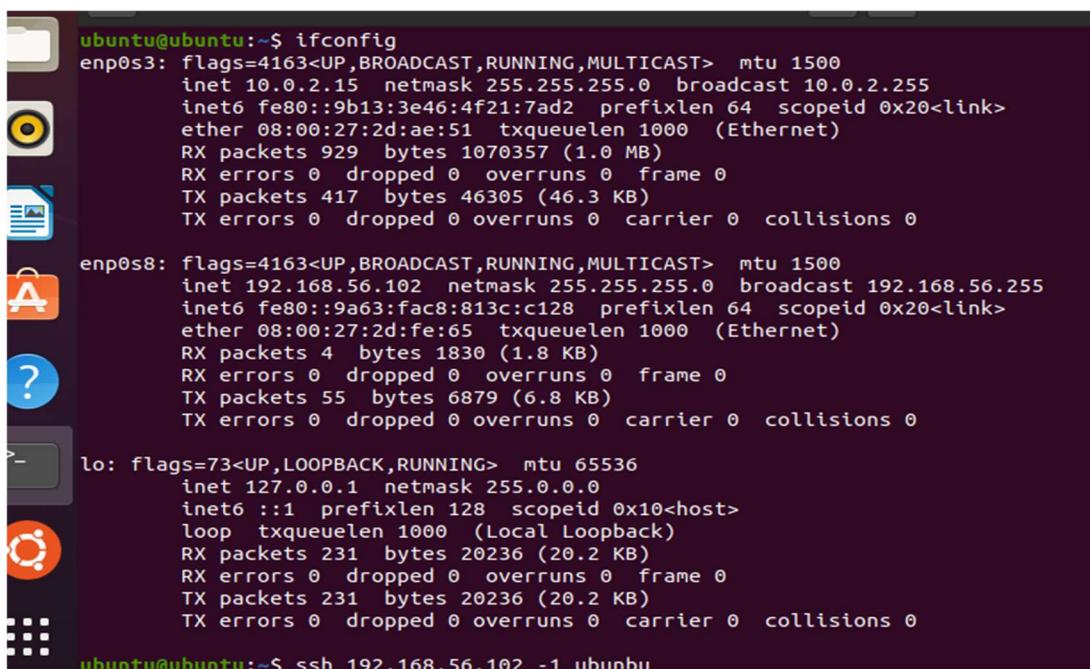
Preliminary Setup

1. Install VirtualBox: Search “VirtualBox download”, or visit virtualbox.org to download and install VirtualBox
2. Create an Ubuntu Server VM using VirtualBox: First download an Ubuntu .iso image file, for this project, we used [Ubuntu-20.04.1-live-server-amd64.iso](http://ubuntu-20.04.1-live-server-amd64.iso). Launch VirtualBox and use the downloaded .iso image file to make your first Ubuntu virtual machine. Install Ubuntu and log in.

First try the ifconfig command : `ubuntu@ubuntu:~$ ifconfig`

If it errors “Command 'ifconfig' not found, but can be installed with: `sudo apt install net-tools`” Then use the command >> `Sudo apt install net-tools`

After these our setup looked like this:



```
ubuntu@ubuntu:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 10.0.2.15 netmask 255.255.255.0 broadcast 10.0.2.255
              inet6 fe80::9b13:3e46:4f21:7ad2 prefixlen 64 scopeid 0x20<link>
                ether 08:00:27:2d:ae:51 txqueuelen 1000 (Ethernet)
                  RX packets 929 bytes 1070357 (1.0 MB)
                  RX errors 0 dropped 0 overruns 0 frame 0
                  TX packets 417 bytes 46305 (46.3 KB)
                  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 192.168.56.102 netmask 255.255.255.0 broadcast 192.168.56.255
              inet6 fe80::9a63:fac8:813c:c128 prefixlen 64 scopeid 0x20<link>
                ether 08:00:27:2d:fe:65 txqueuelen 1000 (Ethernet)
                  RX packets 4 bytes 1830 (1.8 KB)
                  RX errors 0 dropped 0 overruns 0 frame 0
                  TX packets 55 bytes 6879 (6.8 KB)
                  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
        inet 127.0.0.1 netmask 255.0.0.0
              inet6 ::1 prefixlen 128 scopeid 0x10<host>
                loop txqueuelen 1000 (Local Loopback)
                  RX packets 231 bytes 20236 (20.2 KB)
                  RX errors 0 dropped 0 overruns 0 frame 0
                  TX packets 231 bytes 20236 (20.2 KB)
                  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

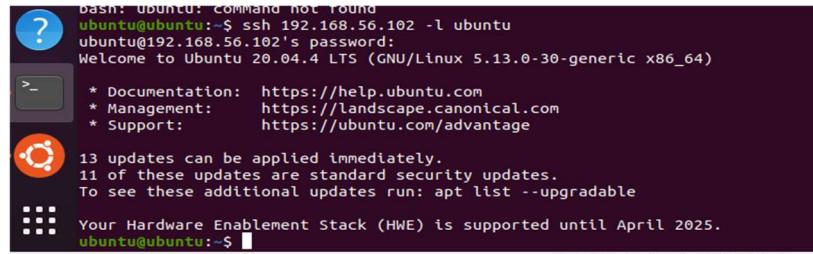
ubuntu@ubuntu:~$ ssh 192.168.56.102 -l ubuntu
```

Figure 9: Running ifconfig command to show IP address

We took notes about the IP address of the Host-only interface card. The screenshot above shows our IP **192.168.56.101**. We SSH from our host machine into this Ubuntu VM using

the IP later. Finally, we confirmed that our VM has internet access by using the following command>> ping google.com

3. Use SSH to connect to the Ubuntu VM to install Free5GC: Ubuntu has an SSH client pre-installed and using the command >> ssh 192.168.56.101 -l ubuntu



```
bash: ubuntu: command not found
ubuntu@ubuntu:~$ ssh 192.168.56.102 -l ubuntu
ubuntu@192.168.56.102's password:
Welcome to Ubuntu 20.04.4 LTS (GNU/Linux 5.13.0-30-generic x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/advantage

13 updates can be applied immediately.
11 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

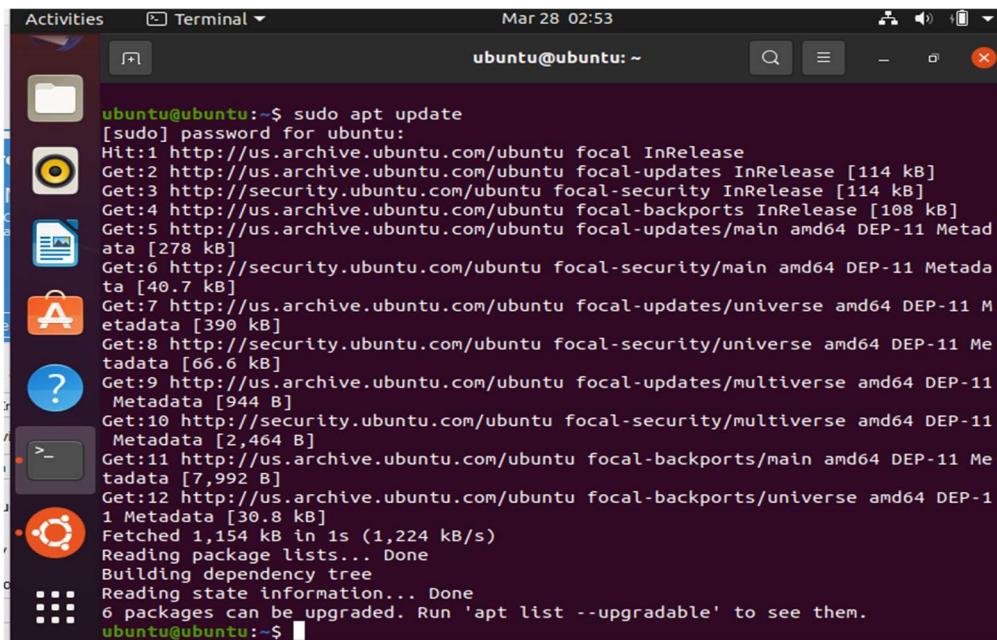
Your Hardware Enablement Stack (HWE) is supported until April 2025.

ubuntu@ubuntu:~$
```

Figure 10: Using SSH to connect to Ubuntu for Installing Free5GC

4. Update and upgrade Ubuntu: Finally for preliminary setup, we updated and upgraded our Ubuntu using the following command:

```
>>>sudo apt update
>>>sudo apt upgrade
```



```
Activities Terminal Mar 28 02:53
ubuntu@ubuntu:~$ sudo apt update
[sudo] password for ubuntu:
Hit:1 http://us.archive.ubuntu.com/ubuntu focal InRelease
Get:2 http://us.archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:3 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Get:4 http://us.archive.ubuntu.com/ubuntu focal-backports InRelease [108 kB]
Get:5 http://us.archive.ubuntu.com/ubuntu focal-updates/main amd64 DEP-11 Metadata [278 kB]
Get:6 http://security.ubuntu.com/ubuntu focal-security/main amd64 DEP-11 Metadata [40.7 kB]
Get:7 http://us.archive.ubuntu.com/ubuntu focal-updates/universe amd64 DEP-11 Metadata [390 kB]
Get:8 http://security.ubuntu.com/ubuntu focal-security/universe amd64 DEP-11 Metadata [66.6 kB]
Get:9 http://us.archive.ubuntu.com/ubuntu focal-updates/multiverse amd64 DEP-11 Metadata [944 B]
Get:10 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 DEP-11 Metadata [2,464 B]
Get:11 http://us.archive.ubuntu.com/ubuntu focal-backports/main amd64 DEP-11 Metadata [7,992 B]
Get:12 http://us.archive.ubuntu.com/ubuntu focal-backports/universe amd64 DEP-11 Metadata [30.8 kB]
Fetched 1,154 kB in 1s (1,224 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
6 packages can be upgraded. Run 'apt list --upgradable' to see them.

ubuntu@ubuntu:~$
```

Figure 11: Update and Upgrade of Ubuntu

Creating the Free5GC VM and setting up the Network: To achieve this, we cloned our existing VM and installed Free5GC on it. Next set up the networking for the Free5GC VM.

- Log in to the VM using SSH from the host machine, and check if the VM has internet access >> ssh 192.168.56.101 -l ubuntu
- Confirmed sudo apt update and sudo apt upgrade again
- Shutdown the VM: use the command sudo shutdown -P now
- For reboot, we use enter sudo shutdown -r now

Creating the Free5GC VM: Using these basic steps: Select an existing VM (ubuntu) and click the buttons on the right: / Snapshots / Clone >>> Name the new VM Free5GC

- The MAC address rule: Create new MAC addresses for all network cards. Choose the Link cloning option (or you can also choose to complete clone the VM if you like)

After the new VM is created: We started the new Free5GC VM, and use the same username and password to log in

- In the Ubuntu terminal, we issued the ping and ifconfig command again to make sure it has internet access, and also make note of the IP address of the Host-only network interface
- 192.168.56.101, and the interface name is enp0s8
- Log in into Free5GC VM using SSH, and we made sure all things worked properly

Change the hostname: The cloned Free5GC VM still has the hostname ubuntu. We renamed the VM to Free5GC. We did this by editing the file /etc/hostname >>> sudo nano /etc/hostname and changed “Ubuntu” to Free5GC

- Again, use the /etc/hosts command to add

```
127.0.0.1 localhost  
127.0.1.1 Free5GC
```

Setting Static IP Addressing and Network Configurations: By default, the IP address for the Host-only network interface is obtained via DHCP. The cloned Free5GC VM appears to be having difficulty getting a new IP address. We updated the host-only interface to use a static IP address.

```
>>>cd /etc/netplan
```

```
>>> nano 00-installer-config.yaml >> We observed the VM has two network interfaces. Using ifconfig we know that enpos8 is the name of the Host-only network interface. We edited the file to show the network settings below:
```

network:

ethernets:

enpos3:

 dhcp4: true

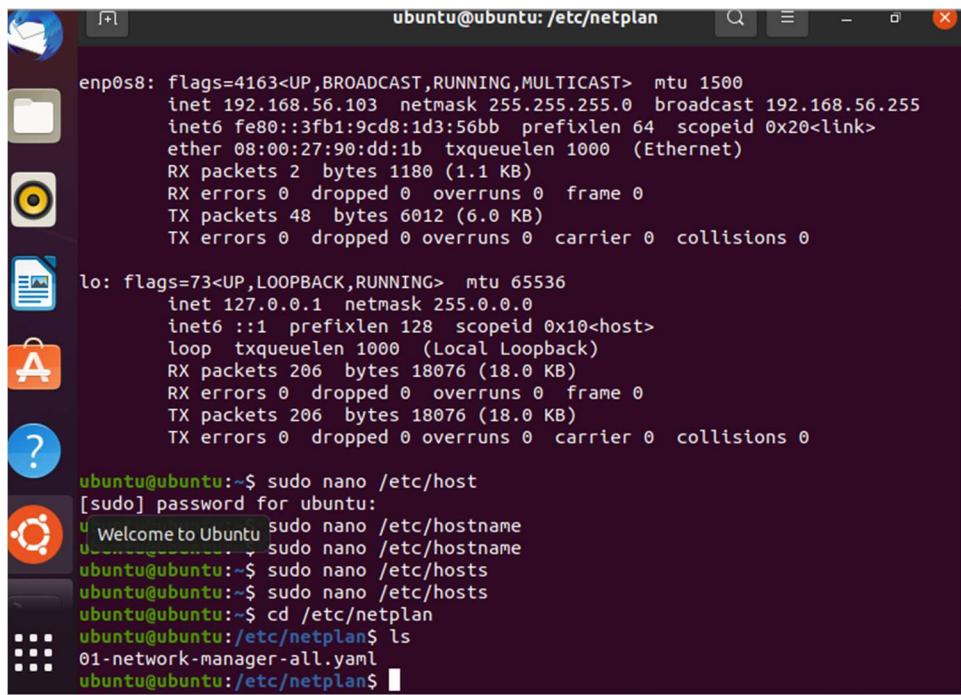
enpos8:

 dhcp4: false

 addresses: [192.168.56.101/24]

version: 2

- Saved with the ctrl O command and Ctrl X for exit. Run the command >>> sudo netplan try then >>> sudo netplan apply
- Running the Ifconfig command showed us the updated changes.



```

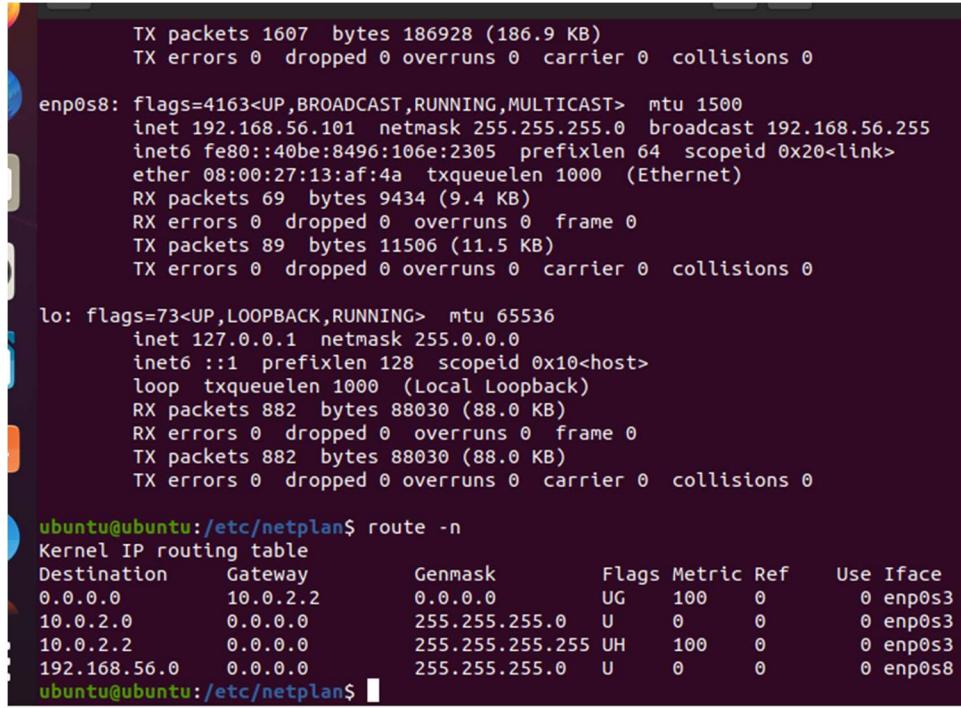
ubuntu@ubuntu: /etc/netplan
enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 192.168.56.103 netmask 255.255.255.0 broadcast 192.168.56.255
          inet6 fe80::3fb1:9cd8:1d3:56bb prefixlen 64 scopeid 0x20<link>
            ether 08:00:27:90:dd:1b txqueuelen 1000 (Ethernet)
              RX packets 2 bytes 1180 (1.1 KB)
              RX errors 0 dropped 0 overruns 0 frame 0
              TX packets 48 bytes 6012 (6.0 KB)
              TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
        inet 127.0.0.1 netmask 255.0.0.0
          inet6 ::1 prefixlen 128 scopeid 0x10<host>
            loop txqueuelen 1000 (Local Loopback)
              RX packets 206 bytes 18076 (18.0 KB)
              RX errors 0 dropped 0 overruns 0 frame 0
              TX packets 206 bytes 18076 (18.0 KB)
              TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ubuntu@ubuntu:~$ sudo nano /etc/host
[sudo] password for ubuntu:
[...]
ubuntu@ubuntu:~$ sudo nano /etc/hostname
[...]
ubuntu@ubuntu:~$ sudo nano /etc/hosts
[...]
ubuntu@ubuntu:~$ sudo nano /etc/hosts
[...]
ubuntu@ubuntu:~$ cd /etc/netplan
ubuntu@ubuntu:/etc/netplan$ ls
01-network-manager-all.yaml
ubuntu@ubuntu:/etc/netplan$ 

```

Figure 12: Running Ifconfig command to show updated IP settings



```

TX packets 1607 bytes 186928 (186.9 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 192.168.56.101 netmask 255.255.255.0 broadcast 192.168.56.255
          inet6 fe80::40be:8496:106e:2305 prefixlen 64 scopeid 0x20<link>
            ether 08:00:27:13:af:4a txqueuelen 1000 (Ethernet)
              RX packets 69 bytes 9434 (9.4 KB)
              RX errors 0 dropped 0 overruns 0 frame 0
              TX packets 89 bytes 11506 (11.5 KB)
              TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
        inet 127.0.0.1 netmask 255.0.0.0
          inet6 ::1 prefixlen 128 scopeid 0x10<host>
            loop txqueuelen 1000 (Local Loopback)
              RX packets 882 bytes 88030 (88.0 KB)
              RX errors 0 dropped 0 overruns 0 frame 0
              TX packets 882 bytes 88030 (88.0 KB)
              TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ubuntu@ubuntu:/etc/netplan$ route -n
Kernel IP routing table
Destination     Gateway         Genmask        Flags Metric Ref  Use Iface
0.0.0.0         10.0.2.2       0.0.0.0       UG    100    0      0 enp0s3
10.0.2.0        0.0.0.0        255.255.255.0  U      0      0      0 enp0s3
10.0.2.2        0.0.0.0        255.255.255.255 UH    100    0      0 enp0s3
192.168.56.0    0.0.0.0        255.255.255.0  U      0      0      0 enp0s8
ubuntu@ubuntu:/etc/netplan$ 

```

Figure 13: Showing network route using the route -n command

Now we SSH into Free5GC VM using the command >>> ssh 192.168.56.101 -l ubuntu

Installing Free5GC: For us to be able to run Free5GC, we installed the following tools:
Golang: First, we made sure “go” was not installed by checking the version using the following command >>> go version. To install the latest go, we searched “Ubuntu golang install” on the web, then followed the instructions thereafter. Installed using the command below

```
>>>cd ~  
>>>wget https://golang.org/dl/go1.15.8.linux-amd64.tar.gz  
>>>sudo tar -C /usr/local -xzf go1.15.8.linux-amd64.tar.gz
```

Then executing the following commands below:

```
>>>mkdir -p ~/go/{bin,pkg,src}  
>>>echo 'export GOPATH=$HOME/go' >> ~/.bashrc  
>>>echo 'export GOROOT=/usr/local/go' >> ~/.bashrc  
>>>echo 'export PATH=$PATH:$GOPATH/bin:$GOROOT/bin' >> ~/.bashrc  
>>>echo 'export GO111MODULE=auto' >> ~/.bashrc  
>>>source ~/.bashrc
```

To confirm that go is installed, we use the command >>> go version

Mongo Db: We installed MongoDB using the commands below:

```
>>>sudo apt -y update  
>>>sudo apt -y install mongodb  
>>>sudo systemctl start mongodb
```

To check if Mongo Db was installed, we used the command >>> mongo

Other development tools: Next, we installed other development tools using this command below.

```
>>>sudo apt -y install git gcc g++ cmake autoconf libtool pkg-config libmnl-dev libyaml-dev go get -u github.com/sirupsen/logrus
```

Set up Networking Rules: To set up our network rules, we used the following commands one after the other:

```
>>>sudo sysctl -w net.ipv4.ip_forward=1  
>>>sudo iptables -t nat -A POSTROUTING -o enp0s3 -j MASQUERADE  
>>>sudo systemctl stop ufw
```

It is important to note that the name `enpos3` is the network interface Free5GC uses to connect to Data Network and also because these settings would reset to default after each reboot, we were curious not to reboot our machine for the period we worked on the implementation.

Install Free5GC Core Network: We run the following commands:

```
>>>cd ~  
>>>git clone --recursive https://github.com/Free5GC/Free5GC.git  
>>>To build Free5GC, we did >>> cd ~/Free5GC  
>>>make
```

We again installed kernel module `gtp5g`:

```
>>>cd ~  
>>>git clone https://github.com/Free5GC/gtp5g.git  
>>>cd gtp5g  
>>>make  
>>>sudo make install
```

Lastly, we confirmed that Free5GC was installed properly by using `>>> lsmod | grep gtp`

```

Activities Terminal Apr 6 03:48
ubuntu@free5gc: ~/free5gc
2022-04-06T03:18:44-04:00 [INFO][NRF][GIN] | 201 | 127.0.0.1 | PUT | /nrf-nfm/v1/nf-instances/86e9cb45-2700-46a8-b55a-ea63882f91f1 |
2022-04-06T03:18:44-04:00 [INFO][AUSF][CFG] config version [1.0.2]
2022-04-06T03:18:44-04:00 [INFO][AUSF][Init] AUSF Log level is set to [info] level
2022-04-06T03:18:44-04:00 [INFO][AUSF][App] ausf
2022-04-06T03:18:44-04:00 [INFO][AUSF][App] AUSF version:
    free5GC version: v3.1.0
    build time: 2022-04-04T04:05:59Z
    commit hash: 114bd26e
    commit time: 2021-11-09T11:48:40Z
    go version: go1.18 linux/amd64
2022-04-06T03:18:44-04:00 [INFO][AUSF][Init] Server started
2022-04-06T03:18:44-04:00 [INFO][AUSF][Init] ausfconfig Info: Version[1.0.2] Description[AUSF initial local configuration]
ausf context = &{{[0 0] {<nil>} map[] 0} {[0 0] {<nil>} map[] 0} 41a3e0fc-5333-4c89-b0a0-346f6fdb1c89 ausfGroup001 8000 127.0.0.9 127.0.0.9 http://127.0.0.9:8000 http http://127.0.0.10:8000 map[nausf-auth:{41a3e0fc-5333-4c89-b0a0-346f6fdb1c89 nausf-auth 0xc000398f00 http REGISTERED 0xc000398ee8 [] <nil> [] [] <nil> 0 0 0 <nil> <nil> ]] [[208 93] {123 45}] 0xc0003ecc80 false}
2022-04-06T03:18:44-04:00 [INFO][NRF][MGMT] Handle NFRegisterRequest
2022-04-06T03:18:44-04:00 [INFO][NRF][MGMT] urilist update
2022-04-06T03:18:44-04:00 [INFO][NRF][MGMT] Create NF Profile
2022-04-06T03:18:44-04:00 [INFO][NRF][MGMT] Location header: http://127.0.0.10:8000/nrf-nfm/v1/nf-instances/41a3e0fc-5333-4c89-b0a0-346f6fdb1c89
2022-04-06T03:18:44-04:00 [INFO][GIN] | 201 | 127.0.0.1 | PUT | /nrf-nfm/v1/nf-instances/41a3e0fc-5333-4c89-b0a0-346f6fdb1c89 |
2022-04-06T03:18:53-04:00 [INFO][LIB][PFCP] Remove Request Transaction [1]

```

Figure 14: Running Free5GC

Installing UERANSIM VM: We Repeated the steps of cloning the Free5GC VM from the base VM, create a new VM for the UERANSIM simulator:

- Name the VM UERANSIM, and create new MAC addresses for all network cards
- Make sure the VM has internet access and can log in using SSH
- Change the hostname to UERANSIM
- Make the Host-only network interface have a static IP address 192.168.56.102
- Reboot the UERANSIM VM, as well as the Free5GC VM
- Ensured we can ping 192.168.56.101 from the UERANSIM VM, and also ping 192.168.56.102 from the Free5GC VM

Install UERANSIM: UERANSIM is an open-source application for managing UEs and gNbs. We downloaded the required tools and install them using the commands below:

```

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

```

```
>>>cd UERANSIM  
>>>git checkout v3.1.0
```

Then update and upgrade UERANSIM VM using the commands>>> sudo apt update

Then >>> sudo apt upgrade

Installing the required tools: We used the following commands:

```
>>>sudo apt install make  
>>>sudo apt install g++  
>>>sudo apt install libsctp-dev lksctp-tools  
>>>sudo apt install iproute2  
>>>sudo snap install cmake -classic
```

Again, we use the following commands to successfully build UERANSIM. Screenshot below

```
>>>cd /UERANSIM  
>>>make
```

Figure 15 is a screenshot of a successful building of UERANSIM.

```

make[3]: Leaving directory '/home/ubuntu/UERANSIM/cmake-build-release'
make[3]: Entering directory '/home/ubuntu/UERANSIM/cmake-build-release'
[ 99%] Building CXX object CMakeFiles/nr-ue.dir/src/ue.cpp.o
[ 99%] Linking CXX executable nr-ue
make[3]: Leaving directory '/home/ubuntu/UERANSIM/cmake-build-release'
[ 99%] Built target nr-ue
make[3]: Entering directory '/home/ubuntu/UERANSIM/cmake-build-release'
make[3]: Leaving directory '/home/ubuntu/UERANSIM/cmake-build-release'
make[3]: Entering directory '/home/ubuntu/UERANSIM/cmake-build-release'
[100%] Building CXX object CMakeFiles/devbnd.dir/src/binder.cpp.o
[100%] Linking CXX shared library libdevbnd.so
make[3]: Leaving directory '/home/ubuntu/UERANSIM/cmake-build-release'
[100%] Built target devbnd
make[3]: Entering directory '/home/ubuntu/UERANSIM/cmake-build-release'
make[3]: Leaving directory '/home/ubuntu/UERANSIM/cmake-build-release'
make[3]: Entering directory '/home/ubuntu/UERANSIM/cmake-build-release'
[100%] Building CXX object CMakeFiles/nr-cli.dir/src/cli.cpp.o
[100%] Linking CXX executable nr-cli
make[3]: Leaving directory '/home/ubuntu/UERANSIM/cmake-build-release'
[100%] Built target nr-cli
make[2]: Leaving directory '/home/ubuntu/UERANSIM/cmake-build-release'
make[1]: Leaving directory '/home/ubuntu/UERANSIM/cmake-build-release'
cp cmake-build-release/nr-gnb build/
cp cmake-build-release/nr-ue build/
cp cmake-build-release/nr-cli build/
cp cmake-build-release/libdevbnd.so build/
cp tools/nr-binder build/
UERANSIM successfully built.
ubuntu@ueransim:~/UERANSIM$
```

Figure 15: Building UERANSIM

Installing Free5GC WebConsole: WebConsole is a basic web interface provided by Free5GC to assist in the creation and management of UE registrations for usage by various 5G network operations (NF). We installed Node.js and Yarn to make WebConsole. First, we SSH into Free5GC (192.168.56.101), and remove obsolete tools that exist using:

```
>>> sudo apt remove cmdtest
>>> sudo apt remove yarn
```

Installing Node.js and Yarn

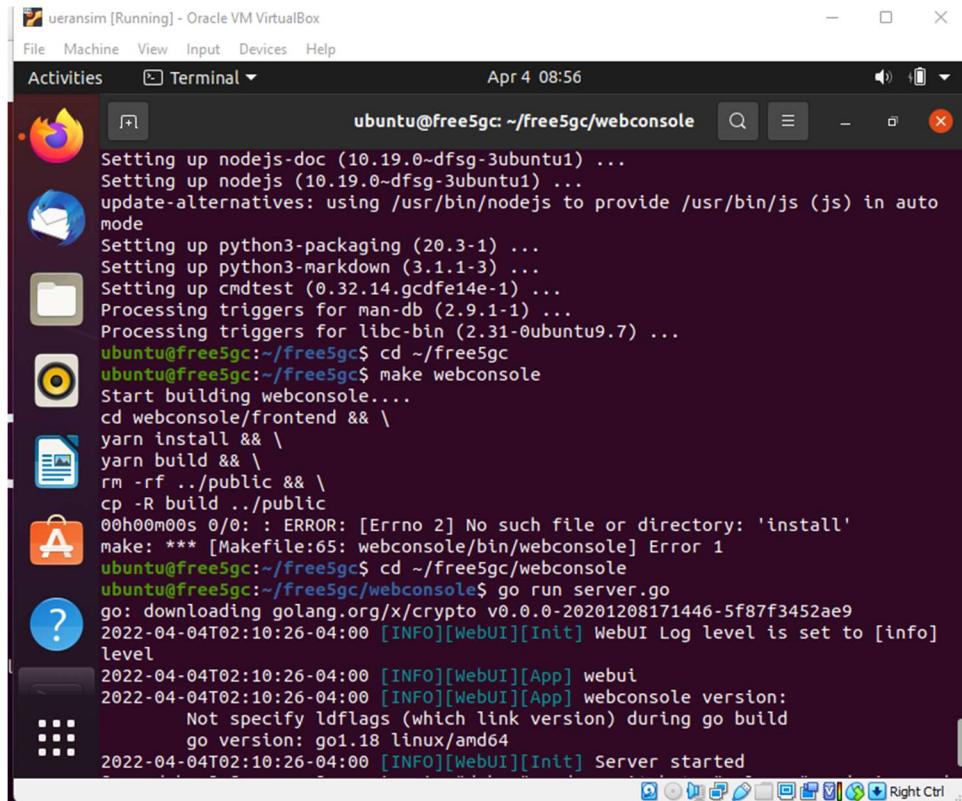
```
>>>curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg | sudo apt-key add -
>>>echo "deb https://dl.yarnpkg.com/debian/ stable main" | sudo tee /etc/apt/sources.list.d/yarn.list
>>>sudo apt-get update
>>>sudo apt-get install -y nodejs yarn
```

Building WebConsole, we then run this command.

```
>>>cd ~/Free5GC  
>>>make webconsole
```

Use WebConsole to Add a UE: We created the UERANSIM using the following command:

```
>>>cd ~/Free5GC/webconsole  
>>>go run server.go
```



The screenshot shows a terminal window with the following text output:

```
Setting up nodejs-doc (10.19.0~dfsg-3ubuntu1) ...  
Setting up nodejs (10.19.0~dfsg-3ubuntu1) ...  
update-alternatives: using /usr/bin/nodejs to provide /usr/bin/js (js) in auto mode  
Setting up python3-packaging (20.3-1) ...  
Setting up python3-markdown (3.1.1-3) ...  
Setting up cmdtest (0.32.14.gcdfe14e-1) ...  
Processing triggers for man-db (2.9.1-1) ...  
Processing triggers for libc-bin (2.31-0ubuntu9.7) ...  
ubuntu@free5gc:~/free5gc$ cd ~/free5gc  
ubuntu@free5gc:~/free5gc$ make webconsole  
Start building webconsole....  
cd webconsole/frontend && \  
yarn install && \  
yarn build && \  
rm -rf ../public && \  
cp -R build ../public  
00h0m00s 0/0: : ERROR: [Errno 2] No such file or directory: 'install'  
make: *** [Makefile:65: webconsole/bin/webconsole] Error 1  
ubuntu@free5gc:~/free5gc$ cd ~/free5gc/webconsole  
ubuntu@free5gc:~/free5gc/webconsole$ go run server.go  
go: downloading golang.org/x/crypto v0.0.0-20201208171446-5f87f3452ae9  
2022-04-04T02:10:26-04:00 [INFO][WebUI][Init] WebUI Log level is set to [info]  
level  
2022-04-04T02:10:26-04:00 [INFO][WebUI][App] webui  
2022-04-04T02:10:26-04:00 [INFO][WebUI][App] webconsole version:  
    Not specify ldflags (which link version) during go build  
    go version: go1.18 linux/amd64  
2022-04-04T02:10:26-04:00 [INFO][WebUI][Init] Server started
```

Figure 16: Starting Free5GC web console

Setting up Free5GC and UERANSIM: We did set a couple of parameters, edited the following configuration files:

- /Free5GC/config/amfcfg.yaml :
- ~/Free5GC/config/smfcfg.yaml

- `~/Free5GC/NFs/upf/build/config/upfcfg.yaml`

First SSH into Free5GC VM, and change `~/Free5GC/config/amfcfg.yaml`

```
>>>cd ~/Free5GC
>>>nano config/amfcfg.yaml
```

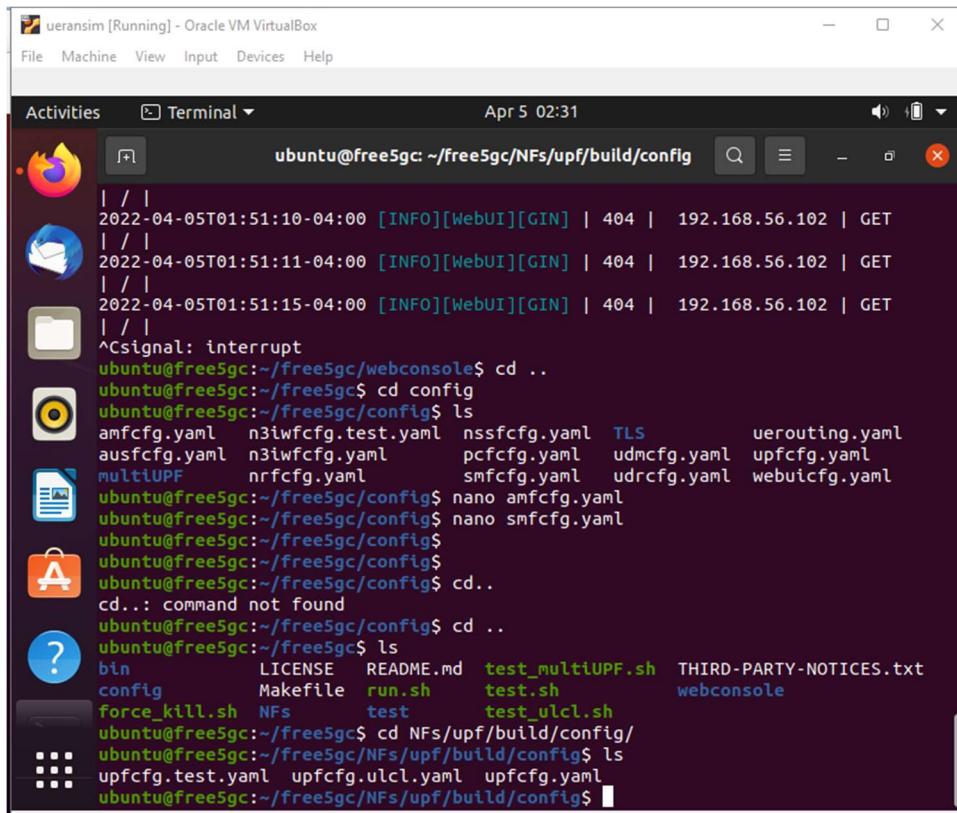
We replaced ngapIpList IP from 127.0.0.1 to 192.168.56.101

Next edited `/Free5GC/config/smfcfg.yaml` using the same command above.

```
>>> nano /config/smfcfg.yaml
```

Also, and in the entry inside userplane_information / up_nodes / UPF / interfaces / endpoints, we changed the IP from 127.0.0.8 to 192.168.56.101

Lastly, edited `~/Free5GC/NFs/upf/build/config/upfcfg.yaml`, and changed gtpu IP from 127.0.0.8 into 192.168.56.101



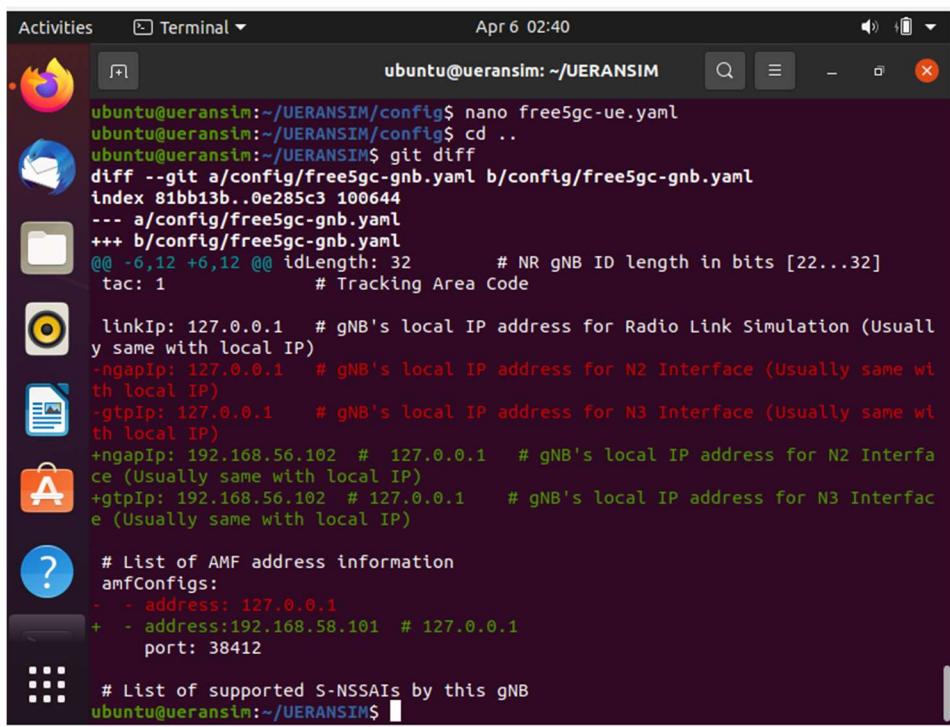
```
ueransim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal ▾ Apr 5 02:31
ubuntu@free5gc:~/free5gc/NFs/upf/build/config
| / |
2022-04-05T01:51:10-04:00 [INFO][WebUI][GIN] | 404 | 192.168.56.102 | GET
| / |
2022-04-05T01:51:11-04:00 [INFO][WebUI][GIN] | 404 | 192.168.56.102 | GET
| / |
2022-04-05T01:51:15-04:00 [INFO][WebUI][GIN] | 404 | 192.168.56.102 | GET
| / |
^Csignal: interrupt
ubuntu@free5gc:~/free5gc/webconsole$ cd ..
ubuntu@free5gc:~/free5gc$ cd config
ubuntu@free5gc:~/free5gc/config$ ls
amfcfg.yaml n3iwfcfg.test.yaml nssfcfg.yaml TLS uerouting.yaml
ausfcfg.yaml n3iwfcfg.yaml pcfcfg.yaml udmcfg.yaml upfcfg.yaml
multiUPF nrpcfgr.yaml smfcfg.yaml udrcfg.yaml webuicfg.yaml
ubuntu@free5gc:~/free5gc/config$ nano amfcfg.yaml
ubuntu@free5gc:~/free5gc/config$ nano smfcfg.yaml
ubuntu@free5gc:~/free5gc/config$ 
ubuntu@free5gc:~/free5gc/config$ 
ubuntu@free5gc:~/free5gc/config$ cd..
cd..: command not found
ubuntu@free5gc:~/free5gc/config$ cd ..
ubuntu@free5gc:~/free5gc$ ls
bin LICENSE README.md test_multIUPF.sh THIRD-PARTY-NOTICES.txt
config Makefile run.sh test.sh webconsole
force_kill.sh NfS test test_ulcl.sh
ubuntu@free5gc:~/free5gc$ cd NfS/upf/build/config/
ubuntu@free5gc:~/free5gc/NfS/upf/build/config$ ls
upfcfg.test.yaml upfcfg.ulcl.yaml upfcfg.yaml
ubuntu@free5gc:~/free5gc/NfS/upf/build/config$
```

Figure 17: Configuring UERANSIM and Free5GC

Setting UERANSIM: For setting up UERANSIM, we edited two files related to Free5GC:

- `~/UERANSIM/config/Free5GC-gnb.yaml`
- `~/UERANSIM/config/Free5GC-ue.yaml`

First, we SSH into UERANSIM, edit the file `~/UERANSIM/config/Free5GC-gnb.yaml`, and change the ngapIp IP, as well as the gtpIp IP, from `127.0.0.1` to `192.168.56.102`, and change the IP in amfConfigs into `192.168.56.101`



```
Activities Terminal ▾ Apr 6 02:40
ubuntu@ueransim:~/UERANSIM$ nano free5gc-ue.yaml
ubuntu@ueransim:~/UERANSIM$ cd ..
ubuntu@ueransim:~/UERANSIM$ git diff
diff --git a/config/free5gc-gnb.yaml b/config/free5gc-gnb.yaml
index 81bb13b..0e285c3 100644
--- a/config/free5gc-gnb.yaml
+++ b/config/free5gc-gnb.yaml
@@ -6,12 +6,12 @@ idLength: 32      # NR gNB ID length in bits [22...32]
 tac: 1            # Tracking Area Code
 linkIp: 127.0.0.1 # gNB's local IP address for Radio Link Simulation (Usually same with local IP)
 -ngapIp: 127.0.0.1 # gNB's local IP address for N2 Interface (Usually same with local IP)
 -gtpIp: 127.0.0.1 # gNB's local IP address for N3 Interface (Usually same with local IP)
 +ngapIp: 192.168.56.102 # 127.0.0.1 # gNB's local IP address for N2 Interface (Usually same with local IP)
 +gtpIp: 192.168.56.102 # 127.0.0.1 # gNB's local IP address for N3 Interface (Usually same with local IP)
 # List of AMF address information
 amfConfigs:
 - - address: 127.0.0.1
 + - address:192.168.56.101 # 127.0.0.1
   port: 38412
 # List of supported S-NSSAIs by this gNB
ubuntu@ueransim:~/UERANSIM$
```

Figure 18: Setting up UERANSIM

Testing UERANSIM against Free5GC: We SSH into Free5GC. Kindly note that if each case we had a reboot, we would need to run the following command again:

```
>>>sudo sysctl -w net.ipv4.ip_forward=1
>>>sudo iptables -t nat -A POSTROUTING -o enp0s3 -j MASQUERADE
>>>sudo systemctl stop ufw
>>>sudo iptables -I FORWARD 1 -j ACCEPT
```

Then run the script >>> cd ~/Free5GC
>>> ./run.sh

At this time Free5GC has been started. Next, we prepared three additional SSH terminals from our host machine.

In terminal 1: SSH into UERANSIM, made sure UERANSIM is built, and configuration files have been changed correctly, then execute nr-gnb

```
>>>cd ~/UERANSIM  
>>>build/nr-gnb -c config/Free5GC-gnb.yaml
```

In terminal 2, SSH into UERANSIM, and executed nr-gnb with admin right:

```
>>>cd ~/UERANSIM  
  
>>>sudo build/nr-ue -c config/Free5GC-ue.yaml
```

Finally, in terminal 3, we SSH into UERANSIM, and ping 192.168.56.101 to see Free5GC is alive. Then, use ifconfig to see that the tunnel uesimtuno has been created.

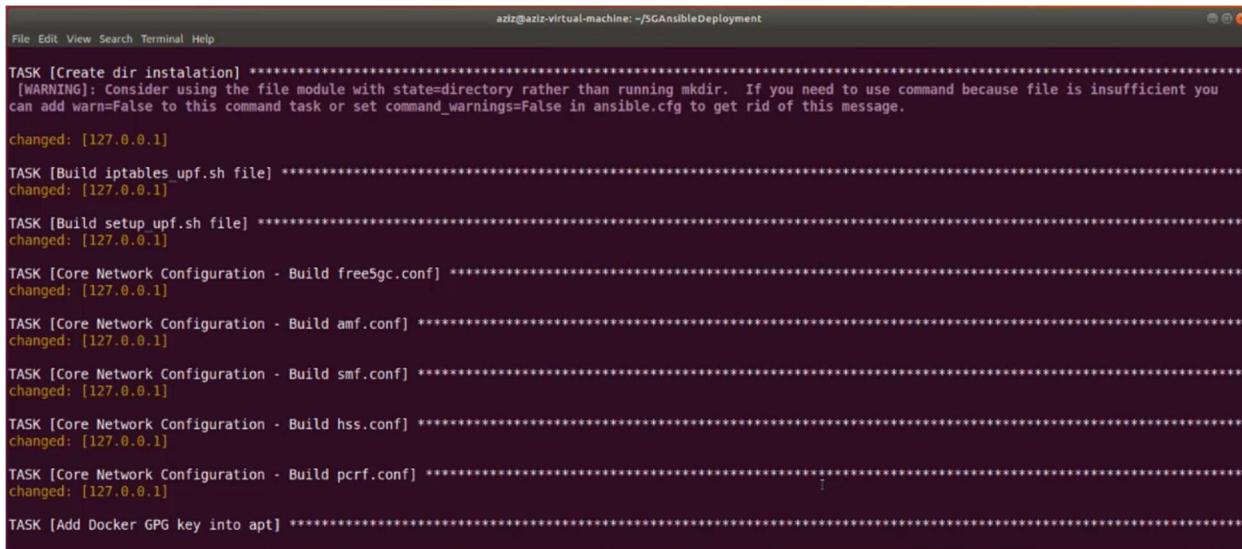
5. Results

In this section, we explain the results of both Docker-based deployment and IPsec implementation. Furthermore, we also present the VM-based deployment with Free5GC and UERANSIM.

Docker Deployment with Security Implementation

We deploy the container-based network using the Ansible playbook which allows the deployment of a complete network in an automated fashion. Using a single command, a network admin may deploy new network functions within a few minutes. However, before running the command to deploy the network, we must install the prerequisites including Python and Ansible. The following commands can be used to install the prerequisites:

- sudo apt-get update
- sudo apt-get install python-minimal
- sudo apt -y install ansible



```
aziz@aziz-virtual-machine: ~/SGAnsibleDeployment
File Edit View Search Terminal Help
TASK [Create dir instalation] *****
[WARNING]: Consider using the file module with state=directory rather than running mkdir. If you need to use command because file is insufficient you can add warn=False to this command task or set command_warnings=False in ansible.cfg to get rid of this message.
changed: [127.0.0.1]
TASK [Build iptables_upf.sh file] *****
changed: [127.0.0.1]
TASK [Build setup_upf.sh file] *****
changed: [127.0.0.1]
TASK [Core Network Configuration - Build free5gc.conf] *****
changed: [127.0.0.1]
TASK [Core Network Configuration - Build amf.conf] *****
changed: [127.0.0.1]
TASK [Core Network Configuration - Build smf.conf] *****
changed: [127.0.0.1]
TASK [Core Network Configuration - Build hss.conf] *****
changed: [127.0.0.1]
TASK [Core Network Configuration - Build pcrf.conf] *****
changed: [127.0.0.1]
TASK [Add Docker GPG key into apt] *****
```

Figure 19: Ansible playbook running and configuring the Core network functions.

After this we deploy the network using the command “ansible-playbook -K [Name of file].yml -e “internet_network_interface=[network interface name]”. Once the command runs, it starts to set up the network and configures the core

network functions from Free5GC [25] including, AMF, SMF, HSS, and PCRF. This process of configuration is shown in Figure 19.

After the network configuration is completed, Ansible runs all the network entities in separate containers. This includes all core network functions, OpenAirSIM enB, OpenAirSIM UE, and Mongo database container. This process is shown in Figure 20. Once the network setup is completed, we can verify the running containers using the command “`sudo docker ps`”. This command shows all the running containers in the system and this is shown in Figure 21.

```

aziz@aziz-virtual-machine: ~/5GAnsibleDeployment
File Edit View Search Terminal Help
TASK [Run free5gc MONGO-DB container] *****
changed: [127.0.0.1]
TASK [Run free5gc AMF container] *****
changed: [127.0.0.1]
TASK [Run free5gc UPF container] *****
changed: [127.0.0.1]
TASK [Run free5gc SMF container] *****
changed: [127.0.0.1]
TASK [Run free5gc HSS container] *****
changed: [127.0.0.1]
TASK [Run free5gc PCRF container] *****
changed: [127.0.0.1]
TASK [Run free5gc WEBUI container] *****
changed: [127.0.0.1]
TASK [Run OpenAirSIM-enB container] *****
changed: [127.0.0.1]
TASK [Run OpenAirSIM-UE container] *****
changed: [127.0.0.1]
TASK [Build enB rcc.band7.tml.nfapi.conf file 1/3] *****

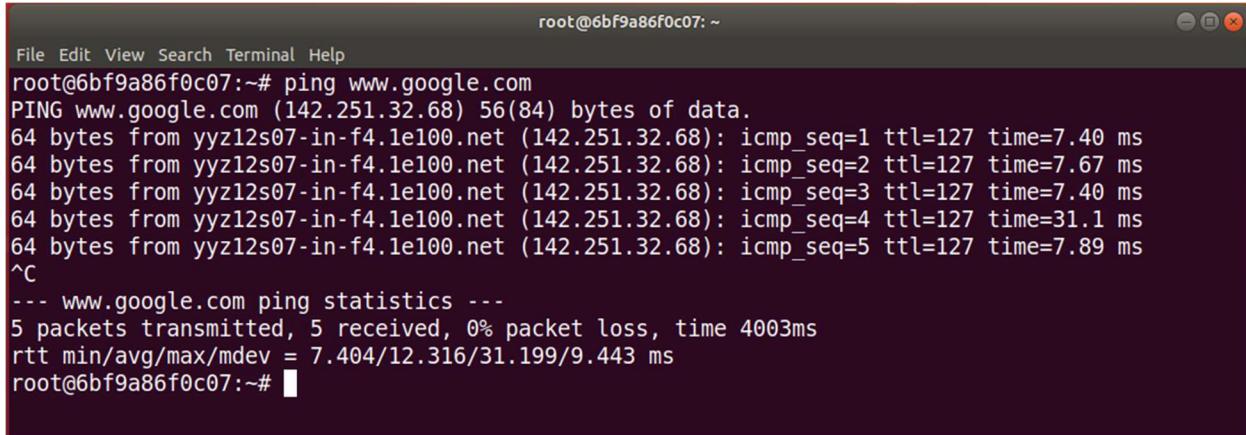
```

Figure 20: Ansible playbook running the containers of the network entities.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
6bf9a86f0c07	laboraufg/ue-openairsim	"bash"	8 minutes ago	Up 8 minutes		ue
cde1ccb49ee	laboraufg/enb-openairsim	"bash"	8 minutes ago	Up 8 minutes		enb
18e25a65204d	laboraufg/webui-free5gc	"bash"	8 minutes ago	Up 8 minutes	0.0.0.0:3000->3000/tcp, :::3000->3000/tcp	webui
e12249051956	laboraufg/free5gc-st1	"bash"	8 minutes ago	Up 8 minutes		pcrf
41c84ab39651	laboraufg/free5gc-st1	"bash"	8 minutes ago	Up 8 minutes		hss
fb6817509536	laboraufg/free5gc-st1	"bash"	8 minutes ago	Up 8 minutes		smf
8132e043ad8f	laboraufg/free5gc-st1	"bash"	8 minutes ago	Up 8 minutes		upf
0a0d0d778dddb8	laboraufg/free5gc-st1	"bash"	8 minutes ago	Up 8 minutes		amf
4ee9cf145d0b	laboraufg/mongodb-free5gc	"bash"	8 minutes ago	Up 8 minutes		mongodb-svc

Figure 21: List of containers running inside the system.

We test the internet connectivity of the UE once the network deployment is complete. To perform this test, we access the UE container via the terminal using the command “`docker exec -ti ue bash`”. This command will provide access to the UE terminal, and we simply test the network connection using “`ping www.google.com`”. The result of this test is provided in Figure 22.



```
root@6bf9a86f0c07: ~
File Edit View Search Terminal Help
root@6bf9a86f0c07:# ping www.google.com
PING www.google.com (142.251.32.68) 56(84) bytes of data.
64 bytes from yyz12s07-in-f4.1e100.net (142.251.32.68): icmp_seq=1 ttl=127 time=7.40 ms
64 bytes from yyz12s07-in-f4.1e100.net (142.251.32.68): icmp_seq=2 ttl=127 time=7.67 ms
64 bytes from yyz12s07-in-f4.1e100.net (142.251.32.68): icmp_seq=3 ttl=127 time=7.40 ms
64 bytes from yyz12s07-in-f4.1e100.net (142.251.32.68): icmp_seq=4 ttl=127 time=31.1 ms
64 bytes from yyz12s07-in-f4.1e100.net (142.251.32.68): icmp_seq=5 ttl=127 time=7.89 ms
^C
--- www.google.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4003ms
rtt min/avg/max/mdev = 7.404/12.316/31.199/9.443 ms
root@6bf9a86f0c07:~#
```

Figure 22: Testing UE connectivity to the internet.

IPsec Security Implementation

To implement IPsec in the network, we create a Python script [see appendix A] to create a packet with a destination address of the UPF. We use Scapy [24] for this purpose. After creating the packet, we also add a payload to the packet as “username_password” to represent some sensitive information. We then encrypt this packet using the ESP mode of IPsec. This will encrypt the payload plaintext to create a ciphertext which is not readable by an adversary who is observing the channel. After creating the unencrypted and encrypted packets, we send both unencrypted and encrypted packets to the UPF using the interface of the UE. Figure 23 shows the transmission of a packet from the UE. The terminal on the right is used to run the python script that will generate, encrypt and send the packet. The terminal on the left is a UE terminal as well and is used for packet capturing to ensure that our UE is able to send the packets to the UPF. The terminal on the right is also printing the content of the packet and other information. The packet includes the source IP address of the UE, destination IP address of the UPF, a payload ‘username_password’, and along with other information. On the left terminal, we observe

the packets sent and received by the UE. We utilize Scapy's 'sniff' function with a filter to only capture UDP and ESP packets. The unencrypted and encrypted packets are highlighted with red and green arrows, respectively. The UE also continuously communicates with the eNB which is why most of the packets captured are sent from the UE (192.187.3.254) to the eNB (192.187.3.253). Figure 24 shows the encrypted version of the packet on the right terminal. We can see that IPsec encrypted the payload of the UDP packet and the new payload (data) is encrypted and unreadable.

```

root@bbf9a86fc07:~# python IPsec_end.py
*** Sending Username and Password using Plaintext ***
Packets:
###[ IP ]##
version = 4
ihl = 5
tos = 0x0
len = 45
id = 1
flags =
frag = 0
ttl = 64
proto = udp
chksum = 0x7244
src = 192.187.3.254
dst = 192.187.3.6
###[ UDP ]##
sport = 45012
dport = 80
len = 25
chksum = 0x8a2
###[ Raw ]##[ load = 'username_password' ]
Sending Packet from UE...
Packet sent successfully.

Unencrypted Packet
sent to UPF
(payload is visible)

```

Figure 23: Transmission of a packet from the UE - Unencrypted packet

```

root@bbf9a86fc07:~# python IPsec_end.py
*** Sending Username and Password using IPsec ***
Packets:
###[ IP ]##
version = 4
ihl = 5
tos = 0x0
len = 56
id = 1
flags =
frag = 8
ttl = 64
proto = esp
chksum = 0x7218
src = 192.187.3.254
dst = 192.187.3.6
###[ Options ]##
spi = 0x222
seq = 1
data = afd40500001908a2757365726e16d655f70617373776f7264010111
Sending Packet from UE...
Packet sent successfully.
root@bbf9a86fc07:~#
```

Figure 24: Transmission of a packet from the UE - Encrypted packet

On the UPF side, we also run Scapy's sniff function to capture the packets sent by the UE. We can observe from Figure 25 that the UPF is able to receive both the UDP and IPsec ESP packets. We further examine the content of the captured packets to ensure the captured packets are indeed the ones sent from the UE. We can see that both the packets are the same as those sent by the UE and UDP packet is showing the payload in cleartext while the IPsec ESP packet contains the encrypted data. This implementation of IPsec will ensure the security of user data and prevent numerous types of attacks.

```

Activities Terminal
root@8132e043ad8f:~/scapy
File Edit View Search Terminal Help
sc4cccp///pSP///p      p//Y | day on earth.
sY/////////y caa        S/P | 
cayCayP//Ya             pV/Ya |
sY/PsY///YYcc          aC//Yp |
sc sccaCY//PCyapaapY//Yss
spCPY//YYPSps
ccaaacs

>>> capture = sniff(filter='udp or esp')
^C>>> capture.summary()
Ether / IP / UDP 192.187.3.254:45012 > 192.187.3.6:80 / Raw
Ether / 192.187.3.254 > 192.187.3.6 esp / ESP
>>> capture.summary()
Ether / IP / UDP 192.187.3.254:45012 > 192.187.3.6:80 / Raw
Ether / 192.187.3.254 > 192.187.3.6 esp / ESP
>>> capture[0]
<Ether dst=02:42:c0:bb:03:06 src=02:42:c0:bb:03:fe type=IPv4 |<IP version=4 i
hl=5 tos=0x0 len=45 id=1 flags= frag=0 ttl=64 proto=udp checksum=0xf244 src=192.1
87.3.254 dst=192.187.3.6 |<UDP sport=45012 dport=80 len=25 checksum=0x8a2 |Raw
load='username password' |>>> 
>>> capture[1]
<Ether dst=02:42:c0:bb:03:06 src=02:42:c0:bb:03:fe type=IPv4 |<IP version=4 i
hl=5 tos=0x0 len=56 id=1 flags= frag=0 ttl=64 proto=esp checksum=0xf218 src=192.1
87.3.254 dst=192.187.3.6 |<ESP spi=0x222 seq=1 data=af40050001908a2757365726e
616d655f70617373776f7264010111 |>>>
>>>

```

UPF terminal to capture packets from UE

Unencrypted Packet

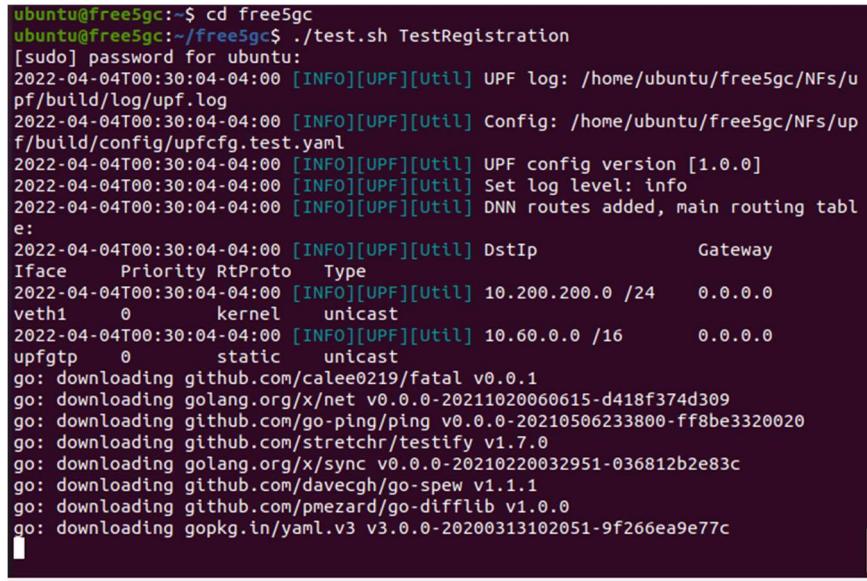
Encrypted Packet

Figure 25: Reception of packets on UPF

Virtual Machine Deployment with Free5GC and UERANSIM

Free5GC provides some procedures to make sure it works properly. In this report, we have shown below the result, testing some common procedures in 5GC. The ./test.sh script, source code attributed to Free5GC. This script helps to run several procedures within the Free5GC. We examined the time of completion for these procedures, and also demonstrated the connectivity between the UERANSIM and Free5GC. For the RAN, each interface IP address of the NE is UERANSIM/config/Free5GC-gnb.yaml being configured. Again, for UPF, each interface IP address of the NE is configured in Free5GC/NFs/upf/build/config/upfcfg.yaml. For SMF, each interface IP address of the NE is configured in Free5GC/config/smfcfg.yaml

Lastly, for AMF, the address of each port of the NE IP is configured in is Free5GC/config/amfcfg.yaml. By default, other network element configuration files are not modified. The Figures 26-30 below show the results for Registration procedures, connectivity between UERANSIM and Free5GC, and also to the internet.

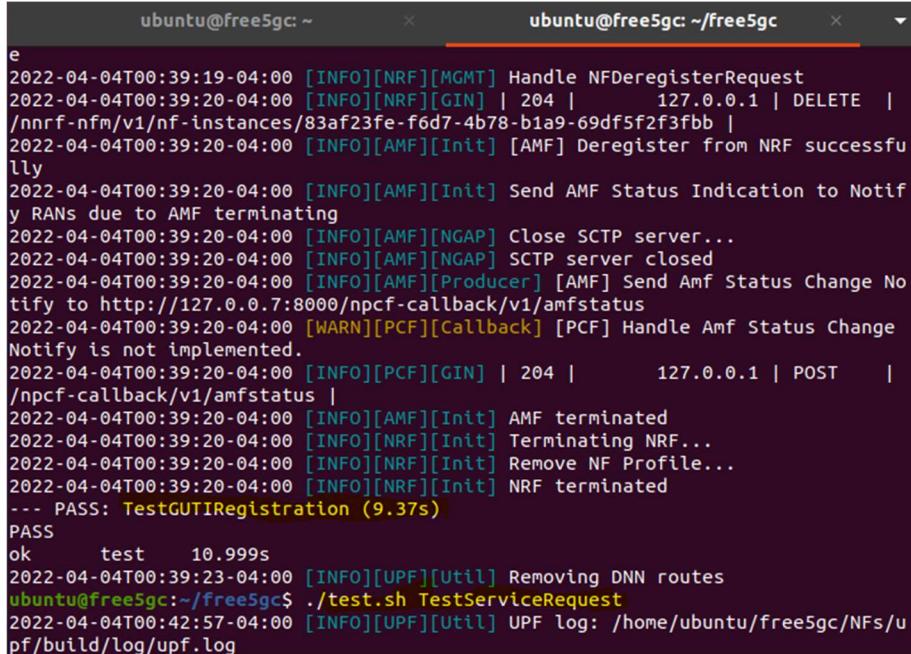


```

ubuntu@free5gc:~$ cd free5gc
ubuntu@free5gc:~/free5gc$ ./test.sh TestRegistration
[sudo] password for ubuntu:
2022-04-04T00:30:04-04:00 [INFO][UPF][Util] UPF log: /home/ubuntu/free5gc/NFs/upf/build/log/upf.log
2022-04-04T00:30:04-04:00 [INFO][UPF][Util] Config: /home/ubuntu/free5gc/NFs/upf/build/config/upfcfg.test.yaml
2022-04-04T00:30:04-04:00 [INFO][UPF][Util] UPF config version [1.0.0]
2022-04-04T00:30:04-04:00 [INFO][UPF][Util] Set log level: info
2022-04-04T00:30:04-04:00 [INFO][UPF][Util] DNN routes added, main routing table:
2022-04-04T00:30:04-04:00 [INFO][UPF][Util] DstIp          Gateway
Iface  Priority RtProto  Type
2022-04-04T00:30:04-04:00 [INFO][UPF][Util] 10.200.200.0 /24   0.0.0.0
veth1    0      kernel  unicast
2022-04-04T00:30:04-04:00 [INFO][UPF][Util] 10.60.0.0 /16   0.0.0.0
upfgtp   0      static  unicast
go: downloading github.com/cailee0219/fatal v0.0.1
go: downloading golang.org/x/net v0.0.0-20211020060615-d418f374d309
go: downloading github.com/go-ping/ping v0.0.0-20210506233800-ff8be3320020
go: downloading github.com/stretchr/testify v1.7.0
go: downloading golang.org/x/sync v0.0.0-20210220032951-036812b2e83c
go: downloading github.com/davecgh/go-spew v1.1.1
go: downloading github.com/pmezard/go-difflib v1.0.0
go: downloading gopkg.in/yaml.v3 v3.0.0-20200313102051-9f266ea9e77c

```

Figure 26: Screenshot shows the initiation of a Test Registration Procedure



```

e
2022-04-04T00:39:19-04:00 [INFO][NRF][MGMT] Handle NFDeregisterRequest
2022-04-04T00:39:20-04:00 [INFO][NRF][GIN] | 204 |      127.0.0.1 | DELETE |
/nnrf-nfm/v1/nf-instances/83af23fe-f6d7-4b78-b1a9-69df5f2f3fb9 |
2022-04-04T00:39:20-04:00 [INFO][AMF][Init] [AMF] Deregister from NRF successfully
2022-04-04T00:39:20-04:00 [INFO][AMF][Init] Send AMF Status Indication to Notify RANs due to AMF terminating
2022-04-04T00:39:20-04:00 [INFO][AMF][NGAP] Close SCTP server...
2022-04-04T00:39:20-04:00 [INFO][AMF][NGAP] SCTP server closed
2022-04-04T00:39:20-04:00 [INFO][AMF][Producer] [AMF] Send Amf Status Change Notify to http://127.0.0.7:8000/npcf-callback/v1/amfstatus
2022-04-04T00:39:20-04:00 [WARN][PCF][Callback] [PCF] Handle Amf Status Change Notify is not implemented.
2022-04-04T00:39:20-04:00 [INFO][PCF][GIN] | 204 |      127.0.0.1 | POST |
/npcf-callback/v1/amfstatus |
2022-04-04T00:39:20-04:00 [INFO][AMF][Init] AMF terminated
2022-04-04T00:39:20-04:00 [INFO][NRF][Init] Terminating NRF...
2022-04-04T00:39:20-04:00 [INFO][NRF][Init] Remove NF Profile...
2022-04-04T00:39:20-04:00 [INFO][NRF][Init] NRF terminated
--- PASS: TestGUTIRegistration (9.37s)
PASS
ok    test    10.999s
2022-04-04T00:39:23-04:00 [INFO][UPF][Util] Removing DNN routes
ubuntu@free5gc:~/free5gc$ ./test.sh TestServiceRequest
2022-04-04T00:42:57-04:00 [INFO][UPF][Util] UPF log: /home/ubuntu/free5gc/NFs/upf/build/log/upf.log

```

Figure 27: The result of a TestGUIGraphicalRegistration procedure

```

ubuntu@free5gc: ~
ubuntu@free5gc: ~
2022-04-04T00:43:09-04:00 [INFO][SMF][Consumer] Send Deregister NFIstance
2022-04-04T00:43:09-04:00 [INFO][NRF][MGMT] Handle NFDeRegisterRequest
2022-04-04T00:43:10-04:00 [INFO][NRF][GIN] | 204 | 127.0.0.1 | DELETE | /nrf-nfm/v1/nf-instances/c5323bb2-3581-4fb4-af01-a7637590563c |
2022-04-04T00:43:10-04:00 [INFO][SMF][Init] Deregister from NRF successfully
2022-04-04T00:43:10-04:00 [INFO][AMF][Init] Terminating AMF...
2022-04-04T00:43:10-04:00 [INFO][AMF][Consumer] [AMF] Send Deregister NFIstance
2022-04-04T00:43:10-04:00 [INFO][NRF][MGMT] Handle NFDeRegisterRequest
2022-04-04T00:43:11-04:00 [INFO][NRF][GIN] | 204 | 127.0.0.1 | DELETE | /nrf-nfm/v1/nf-instances/221ac292-bf07-47e6-b172-e13e9882766e |
2022-04-04T00:43:11-04:00 [INFO][AMF][Init] [AMF] Deregister from NRF successfully
2022-04-04T00:43:11-04:00 [INFO][AMF][Init] Send AMF Status Indication to Notif y RANS due to AMF terminating
2022-04-04T00:43:11-04:00 [INFO][AMF][NGAP] Close SCTP server...
2022-04-04T00:43:11-04:00 [INFO][AMF][NGAP] SCTP server closed
2022-04-04T00:43:11-04:00 [INFO][AMF][Init] AMF terminated
2022-04-04T00:43:11-04:00 [INFO][NRF][Init] Terminating NRF...
2022-04-04T00:43:11-04:00 [INFO][NRF][Init] Remove NF Profile...
2022-04-04T00:43:11-04:00 [INFO][NRF][Init] NRF terminated
--- PASS: TestServiceRequest (9.57s)
PASS
ok    test    11.205s
2022-04-04T00:43:15-04:00 [INFO][UPF][Util] Removing DNN routes
ubuntu@free5gc:~/free5gc$ 

```

Figure 28: The result of a TestServiceRequest procedure

The screenshot below shows UERANSIM connecting with Free5GC and connecting to the outside network.

```

ubuntu@ueransim: ~
ubuntu@ueransim: ~
* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage

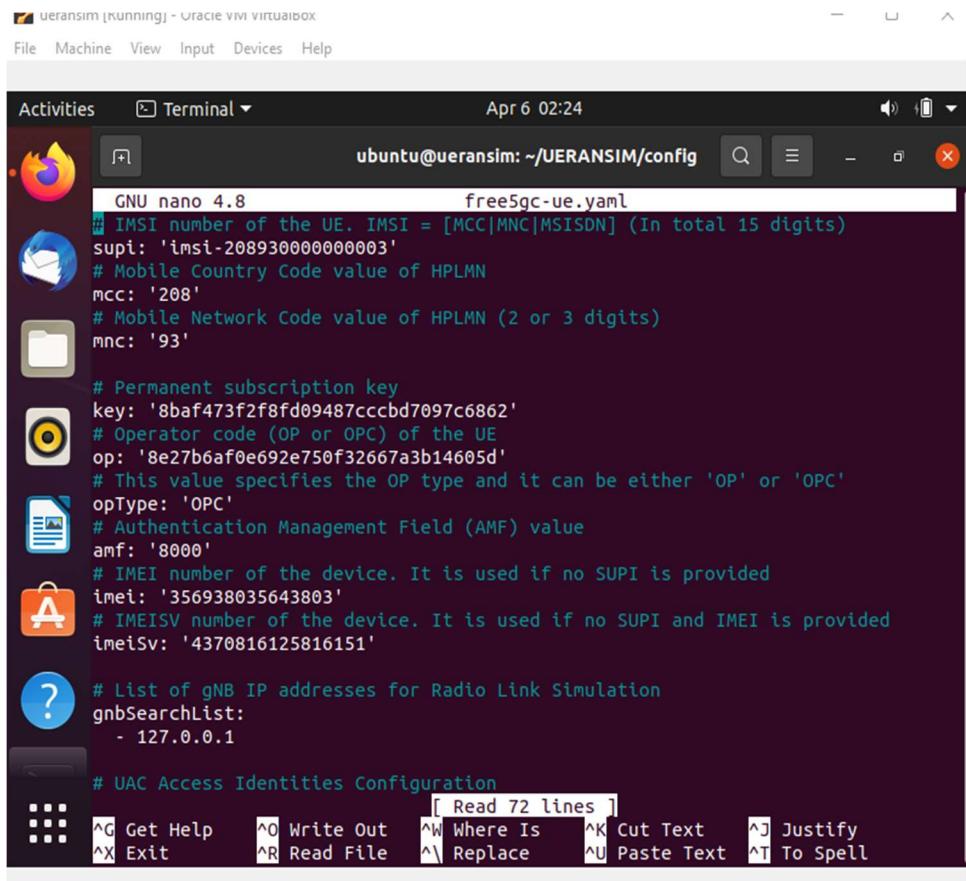
7 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Your Hardware Enablement Stack (HWE) is supported until April 2025.
Last login: Mon Mar 28 02:47:31 2022 from 192.168.56.102
ubuntu@ueransim:~$ ping google.com
PING google.com (172.217.13.110) 56(84) bytes of data.
64 bytes from yul02s04-in-f14.1e100.net (172.217.13.110): icmp_seq=1 ttl=116 time=23.7 ms
^C
--- google.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 23.722/23.722/23.722/0.000 ms
ubuntu@ueransim:~$ ping 192.168.56.101
PING 192.168.56.101 (192.168.56.101) 56(84) bytes of data.
64 bytes from 192.168.56.101: icmp_seq=1 ttl=64 time=0.899 ms
64 bytes from 192.168.56.101: icmp_seq=2 ttl=64 time=0.459 ms
64 bytes from 192.168.56.101: icmp_seq=3 ttl=64 time=0.493 ms
64 bytes from 192.168.56.101: icmp_seq=4 ttl=64 time=0.514 ms
64 bytes from 192.168.56.101: icmp_seq=5 ttl=64 time=0.460 ms
^C
--- 192.168.56.101 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 407ms
rtt min/avg/max/mdev = 0.459/0.565/0.899/0.168 ms
ubuntu@ueransim:~$ 

```

Figure 29: UERANSIM connecting with Free5GC and connecting to the outside network

The screenshot below shows the configuration file of one of our UE.



A screenshot of a terminal window titled "ueransim [Running] - Oracle VM VirtualBox". The window shows a terminal session for the user "ubuntu" at the path "/UERANSIM/config". The terminal displays a YAML configuration file named "free5gc-ue.yaml". The file contains various parameters such as IMSI, MCC, MNC, key, op, opType, amf, imei, imeiSv, and gnbSearchList. The terminal interface includes icons for file operations, a search bar, and a status bar indicating "Read 72 lines".

```
GNU nano 4.8          free5gc-ue.yaml
# IMSI number of the UE. IMSI = [MCC|MNC|MSISDN] (In total 15 digits)
supi: 'imsi-208930000000003'
# Mobile Country Code value of HPLMN
mcc: '208'
# Mobile Network Code value of HPLMN (2 or 3 digits)
mnc: '93'

# Permanent subscription key
key: '8baf473f2f8fd09487cccbd7097c6862'
# Operator code (OP or OPC) of the UE
op: '8e27b6af0e692e750f32667a3b14605d'
# This value specifies the OP type and it can be either 'OP' or 'OPC'
opType: 'OPC'
# Authentication Management Field (AMF) value
amf: '8000'
# IMEI number of the device. It is used if no SUPI is provided
imei: '356938035643803'
# IMEISV number of the device. It is used if no SUPI and IMEI is provided
imeiSv: '4370816125816151'

# List of gNB IP addresses for Radio Link Simulation
gnbSearchList:
- 127.0.0.1

# UAC Access Identities Configuration
[ Read 72 lines ]
^G Get Help    ^O Write Out   ^W Where Is   ^K Cut Text   ^J Justify
^X Exit        ^R Read File   ^A Replace    ^U Paste Text  ^T To Spell
```

Figure 30: Configuration file of one of our UEs

6. Conclusion

In the research work, we study various 5G deployment options and examine the benefits operators can get with such approaches. The first deployment option we consider in our study is the automated deployment of a 5G network based on the containerized environment using Docker. We further provide the results of our implemented network by successfully testing the connectivity of the UE to the internet. Several benefits can be achieved by utilizing such an automated solution. The first benefit is the ease of deployment, which allow network operators to deploy and run networks with ease and avoid the overhead of manually configuring each device deployed in the network. This will enable operators to significantly save operational expenditures (OPEX). Another advantage is the reduction of capital expenditures (CAPEX), as network operators can install multiple virtual network functions on the same underlying physical node, rather than running a physical server for each network function.

We extend our deployed network by incorporating IPsec security protocol to protect the UE traffic from the adversary. We illustrated, with the help of an experiment, the importance of IPsec by comparing the transmission of an unencrypted packet and an encrypted packet. This approach will protect the network against various attacks including man-in-the-middle attacks, replay attacks, etc. This protocol is also important in providing secure communications and protecting the data in a radio network where anyone can capture the radio signal and obtain sensitive information.

More so, the 5G network components are deployed using Free5GC which is an open-source tool for the 5G core network. Furthermore, we also implement a 5G network using Virtual Machines. Particularly with the Free5GC and UERANSIM. We demonstrated this implementation by running the popular 5GC procedures with the help of the test.sh script[appendix]. We examined the delays of these procedures. Moreso, we demonstrated the implementation and communication between Free5GC and UERANSIM and lastly established a connection to the network (Internet) from a UE. One important benefit of this approach is isolation. Isolation is the bedrock of addressing fundamental security

concerns within the 5G space. Although, this feature is substantial, but for ease of use, this approach shows complexities, especially with individual VM set up and installing dependencies as against the Ansible deployment which is largely automated.

7. Appendices

Appendix A: IPsec Implementation Source Code

The source code of our implementation is available on [GitHub](#).

IPsec Code:

(Note: To run the code inside the UE container, we must create this python code inside the container (since it is isolated from other OS directories) after deploying the network. Also, the Scapy library must be installed inside the UE container to be able to manipulate the packet)

```
from scapy.all import *
from scapy.config import conf

sock = conf.L3socket(iface='eth0')

# Creating Packet
p = IP(src='192.187.3.254', dst='192.187.3.253')
p /= UDP(sport=45012, dport=80)
p /= Raw('username_password')
p = IP(raw(p))

print("**** Sending Username and Password using Plaintext ****")
print("Packet:")
print(p.show(dump=True))
print("Sending Packet from UE1...")
print("Packet sent successfully.")
sock.send(p) # Sending Packet

# Encrypting Packet with IPsec
sa = SecurityAssociation(ESP, spi=0x222,
                        crypt_algo='NULL', crypt_key=None,
                        auth_algo='NULL', auth_key=None)

e = sa.encrypt(p)

print("**** Sending Username and Password using IPsec ****")
print("Packet:")
print(e.show(dump=True))
print("Sending Packet from UE1...")
print("Packet sent successfully.")

sock.send(e) # Sending Packet
```

Appendix B: VM deployment Source Code

```
#!/usr/bin/env bash

# Check OS
if [ -f /etc/os-release ]; then
    # freedesktop.org and systemd
    . /etc/os-release
    OS=$NAME
    VER=$VERSION_ID
else
    # Fall back to uname, e.g. "Linux <version>", also works for BSD, etc.
    OS=$(uname -s)
    VER=$(uname -r)
    echo "This Linux version is too old: $OS:$VER, we don't support!"
    exit 1
fi

sudo -v
if [ $? == 1 ]
then
    echo "Without root permission, you cannot run the test due to our test is
using namespace"
    exit 1
fi

while getopts 'o' OPT;
do
    case $OPT in
        o) DUMP_NS=True;;
        esac
done
shift $((OPTIND - 1))

TEST_POOL="TestRegistration|TestGUTIRegistration|TestServiceRequest|TestXnHandover|TestN2Handover|TestDeregistration|TestPDUSessionReleaseRequest|TestPaging|TestNon3GPP|TestReSynchronization|TestDuplicateRegistration|TestEAPAKAPrimeAuthentication"
if [[ ! "$1" =~ $TEST_POOL ]]
then
    echo "Usage: $0 [ ${TEST_POOL//|/ | } ]"
    exit 1
fi

GOPATH=$HOME/go
if [ $OS == "Ubuntu" ]; then
    GOROOT=/usr/local/go
elif [ $OS == "Fedora" ]; then
    GOROOT=/usr/lib/golang
```

```

fi
PATH=$PATH:$GOPATH/bin:$GOROOT/bin

UPFNS="UPFn"
EXEC_UPFNS="sudo -E ip netns exec ${UPFNS}"

export GIN_MODE=release

function terminate()
{
    sleep 3
    sudo killall -15 free5gc-upfd
    sleep 1

    if [ ${DUMP_NS} ]
    then
        # kill all tcpdump processes in the default network namespace
        sudo killall tcpdump
        sleep 1
    fi

    sudo ip link del veth0
    sudo ip netns del ${UPFNS}
    sudo ip addr del 10.60.0.1/32 dev lo

    if [[ "$1" == "TestNon3GPP" ]]
    then
        if [ ${DUMP_NS} ]
        then
            cd .. && sudo ip xfrm state > ${PCAP_PATH}/NWu_SA_state.log
        fi
        sudo ip xfrm policy flush
        sudo ip xfrm state flush
        sudo ip link del veth2
        sudo ip link del IPsec0
        ${EXEC_UENS} ip link del IPsec0
        sudo ip netns del ${UENS}
        sudo killall n3iwf
        killall test.test
    fi

    sleep 5
}

function handleSIGINT()
{
    echo -e "\033[41;37m Terminating due to SIGINT ... \033[0m"
    terminate $1
}

trap handleSIGINT SIGINT

```

```

function setupN3ueEnv()
{
    UENS="UEns"
    EXEC_UENS="sudo -E ip netns exec ${UENS}"

    sudo ip netns add ${UENS}

    sudo ip link add veth2 type veth peer name veth3
    sudo ip addr add 192.168.127.1/24 dev veth2
    sudo ip link set veth2 up

    sudo ip link set veth3 netns ${UENS}
    ${EXEC_UENS} ip addr add 192.168.127.2/24 dev veth3
    ${EXEC_UENS} ip link set lo up
    ${EXEC_UENS} ip link set veth3 up
    ${EXEC_UENS} ip link add IPsec0 type vti local 192.168.127.2 remote
192.168.127.1 key 5
    ${EXEC_UENS} ip link set IPsec0 up

    sudo ip link add name IPsec0 type vti local 192.168.127.1 remote 0.0.0.0 key
5
    sudo ip addr add 10.0.0.1/24 dev IPsec0
    sudo ip link set IPsec0 up
}

function tcpdumpN3IWF()
{
    N3IWF_IPsec_iface_addr=192.168.127.1
    N3IWF_IPsec_inner_addr=10.0.0.1
    N3IWF_GTP_addr=10.200.200.2
    UE_DN_addr=10.60.0.1

    ${EXEC_UENS} tcpdump -U -i any -w $PCAP_PATH/${UENS}.pcap &
TCPDUMP_QUERY=" host $N3IWF_IPsec_iface_addr or \
                  host $N3IWF_IPsec_inner_addr or \
                  host $N3IWF_GTP_addr or \
                  host $UE_DN_addr"
    sudo -E tcpdump -U -i any $TCPDUMP_QUERY -w $PCAP_PATH/n3iwf.pcap &
}

# Setup network namespace
sudo ip netns add ${UPFNS}

sudo ip link add veth0 type veth peer name veth1
sudo ip link set veth0 up
sudo ip addr add 10.60.0.1 dev lo
sudo ip addr add 10.200.200.1/24 dev veth0
sudo ip addr add 10.200.200.2/24 dev veth0

sudo ip link set veth1 netns ${UPFNS}

```

```

${EXEC_UPFNS} ip link set lo up
${EXEC_UPFNS} ip link set veth1 up
${EXEC_UPFNS} ip addr add 10.60.0.101 dev lo
${EXEC_UPFNS} ip addr add 10.200.200.101/24 dev veth1
${EXEC_UPFNS} ip addr add 10.200.200.102/24 dev veth1

if [ ${DUMP_NS} ]
then
    PCAP_PATH=testpcap
    mkdir -p ${PCAP_PATH}
    ${EXEC_UPFNS} tcpdump -U -i any -w ${PCAP_PATH}/${UPFNS}.pcap &
    sudo -E tcpdump -U -i lo -w ${PCAP_PATH}/default_ns.pcap &
fi

cd NFs/upf/build && ${EXEC_UPFNS} ./bin/free5gc-upfd -c config/upfcfg.test.yaml &
sleep 2

if [[ "$1" == "TestNon3GPP" ]]
then
    # setup N3UE's namespace, interfaces for IPsec
    setupN3ueEnv
    if [ ${DUMP_NS} ]
    then
        tcpdumpN3IWF
    fi

    # Run CN
    cd test && $GOROOT/bin/go test -v -vet=off -timeout 0 -run TestCN &
    sleep 10

    # Run N3IWF
    sudo -E ./bin/n3iwf -c ./config/n3iwfcfg.test.yaml &
    sleep 5

    # Run Test UE
    cd test
    ${EXEC_UENS} $GOROOT/bin/go test -v -vet=off -timeout 0 -run TestNon3GPPUE -
args noinit
else
    cd test
    $GOROOT/bin/go test -v -vet=off -run $1
fi

terminate $1

```

References

- [1] S. Mwanje, G. Decarreau, C. Mannweiler, M. Naseer-ul-Islam, and L. C. Schmelz, "Network management automation in 5G: Challenges and opportunities," in *2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, Sep. 2016, pp. 1–6. doi: 10.1109/PIMRC.2016.7794614.
- [2] William Stallings, *5G Wireless: A Comprehensive Introduction*. Addison-Wesley, 2021.
- [3] S. T. Arzo, C. Naiga, F. Granelli, R. Bassoli, M. Devetsikiotis, and F. H. P. Fitzek, "A Theoretical Discussion and Survey of Network Automation for IoT: Challenges and Opportunity," *IEEE Internet of Things Journal*, vol. 8, no. 15, pp. 12021–12045, Aug. 2021, doi: 10.1109/JIOT.2021.3075901.
- [4] "Ansible is Simple IT Automation." <https://www.ansible.com/>
- [5] Anderson Patricio, "Ansible: Introduction to this open-source automation platform," May 26, 2020.
- [6] T.-X. Do and Y. Kim, "State Management Function Placement for Service-Based 5G Mobile Core Architecture," *Mobile Networks and Applications*, vol. 24, no. 2, pp. 504–516, Apr. 2019, doi: 10.1007/s11036-018-1153-5.
- [7] "Evolving the mobile core to being cloud native," 2017.
- [8] "3GPP TS 23.501 System architecture for the 5G System," 2017.
- [9] O. Arouk and N. Nikaein, "5G Cloud-Native: Network Management & Automation," 2020. doi: 10.1109/NOMS47738.2020.9110392.
- [10] "Production-Grade Container Orchestration." <https://kubernetes.io/>
- [11] "Red Hat OpenShift makes container orchestration easier." <https://www.redhat.com/en/technologies/cloud-computing/openshift>
- [12] "OpenAirInterface | 5G software alliance for democratising wireless innovation." <https://openairinterface.org/>
- [13] L. T. Bolivar, C. Tselios, D. M. Area, and G. Tsolis, "On the deployment of an open-source, 5G-aware evaluation testbed," in *2018 6th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, 2018, pp. 51–58.
- [14] "Kuryr." <https://wiki.openstack.org/wiki/Kuryr>
- [15] "ManageIQ." <https://www.manageiq.org/>
- [16] "OpenStack - Open Source Cloud Computing Infrastructure." <https://www.openstack.org/>
- [17] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2429–2453, 2018.
- [18] "Advanced Load Balancer, Web Server, & Reverse Proxy - NGINX." <https://www.nginx.com/>
- [19] C.-Y. Huang, C.-Y. Ho, N. Nikaein, and R.-G. Cheng, "Design and prototype of a virtualized 5G infrastructure supporting network slicing," in *2018 IEEE 23rd International Conference on Digital Signal Processing (DSP)*, 2018, pp. 1–5.
- [20] S. Iqbal and J. M. Hamamreh, "A Comprehensive Tutorial on How to Practically Build and Deploy 5G Networks Using Open-Source Software and General-Purpose, Off-the-Shelf Hardware," *RS Open Journal on Innovative Communication Technologies*, vol. 2, no. 6, Dec. 2021, doi: 10.46470/03d8ffbd.4ccb7950.
- [21] G. Liu, Y. Huang, Z. Chen, L. Liu, Q. Wang, and N. Li, "5G Deployment: Standalone vs. Non-Standalone from the Operator Perspective," *IEEE Communications Magazine*, vol. 58, no. 11, pp. 83–89, Nov. 2020, doi: 10.1109/MCOM.001.2000230.
- [22] "Oracle VM VirtualBox." <https://www.virtualbox.org/> (accessed Apr. 07, 2022).
- [23] "Docker," Apr. 08, 2022. <https://www.docker.com/> (accessed Apr. 07, 2022).
- [24] "Scapy." <https://scapy.net/>
- [25] "Free5GC." <https://www.Free5GC.org/>