Realizing 5G Network Slicing Provisioning with Open Source Software

Kuan-Lin Lee†, Chung-Nan Lee* and Ming-Feng Lee+
National Sun Yat-sen University, Taiwan
†E-mail: lee850220@gmail.comTel: + 886-7-5254335
*E-mail: cnlee@mail.cse.nsysu.edu.tw Tel: + 886-7-5252000 ext. 4313
+ E-mail: mflee@mail.nsysu.edu.tw Tel: + 886-7-5254335

***Abstract*— 5G has gradually been commercialized in countries around the world, but for most telecom companies, network slicing is still in the development stage and has not been applied to appropriate scenarios. Automated provisioning of network slicing is even more challenging for development. The purpose of this research is to implement a complete, open source, and automatically deployable 5G network slicing architecture. We use OpenStack as the platform required to realize virtualization, Tacker module for slicing environment deployment, free5GC as the core network of the 5G system, and UERANSIM as the role of simulating UE and gNB. Through the proposed architecture, an automatic slicing service with specific functions can be created, the slice can be registered into the 5G network with the support of the core network, and the simulated UE can be used to connect the related slice. Experimental results show that the proposed open-source-based architecture is feasible and the QoS is guaranteed for each slice.**

I. INTRODUCTION

The concept of network slicing is to use slicing technology to create multiple logical networks on a physical network, and each logical network can have its own network configuration. Logically, each slice can be dedicated to a certain type of application or meet the dedicated network needs of specific users. The provider of network slicing adjusts resources according to the dynamic needs of the business and users to improve the flexibility of the network, and at the same time, it can also reduce the construction cost of hardware resources and network construction to achieve hardware resource sharing. Network slicing provides different Quality of Service (QoS) services to meet various application requirements. When this technology enters the commercialization, each telecom company can tailor the exclusive transmission network service for the enterprise according to different service requirements or application scenarios.

Before the advent of 4G and 5G, traditional networks could already support network slicing. Service providers can implement part of the network slicing functionality through network resource management. However, in the past, most of the operations of network slicing were completed by hardware, which required a relatively large cost. Until Beyond 4G (B4G), virtualization technology was introduced to provide support for software slicing, that is, through Software-Defined Networking (SDN) and Network Function Virtualization (NFV) technologies, new methods of network slicing were realized

Although network slicing technology has brought breakthroughs in the development of 5G communication network architecture and services, according to the Global Mobile Suppliers Association (GSA) [1] in 2021, only five countries have operated the 5G Standalone (SA) model, and the rest of the countries still use the Non-Standalone (NSA) mode that coexists with 4G, and there is no region where network slicing is commercially available. On the one hand, it is because each manufacturer develops independently and lacks a unified implementation standard and the implementation methods are also complicated. On the other hand, it faces greater challenges in conducting research without a suitable environment. Automated provisioning of network slicing is even more challenging for development.

This paper aims to build a 5G mobile communication system with open source software that allows users to quickly deploy 5G network slices. Implementing 5G network slicing through open source software allows research to be highly freely used and quickly modified. We use OpenStack as the virtualization platform, equipped with Tacker modules act as VNF Manager (VNFM) and NFV Orchestrator (NFVO), combined with free5GC as the core network of the 5G mobile communication system, and UERANSIM as the role of simulating UE and gNB. Working with Tacker through scripts we develop, the network environment required by users can be automatically deployed, and QoS configuration scripts can be used to achieve Service Level Agreement (SLA), which is also more conducive to related work and research. We use 5G Core Network (CN) and Radio Access Network (RAN) that comply with 3GPP R15 for implementation, and design several experiments to verify the network slicing standards proposed by 3GPP to ensure that the system can fully comply with the published 5G network slicing standards.

The remainder of this paper is as follows. In the second section, we review the related works about 5G network slicing. The third section presents the proposed mechanism realizing network slicing provisioning. The fourth section conducts some experiments and presents their results. Conclusions are drawn in the last section

II. RELATED WORK

As early as 2016, 5G PPP organization first proposed the overall model in the 5G mobile communication system architecture white paper [2] can be divided into three layers. The bottom layer is physical resources, including computing resources, storage resources, and network resources. These resources are effectively configured and distributed through the management unit to facilitate splitting and cutting into different network services. In 2017, 3GPP proposed the standard of 5G core network architecture in TS 23.501 [3] and Choi et al. [4] carried out detailed analysis of various functions of the core network according to the 3GPP standard. Same year, Ordonez-Lucena et al. [5] began to conduct a detailed analysis of the relationship between network slicing, SDN and NFV, and they proposed a number of technical implementation concepts and possible challenges for the 5G network slicing architecture. Each function in the 5G core network has its own task, and the connection between each function represents a different protocol. These functions are User Plane Function (UPF), Session Management Function (SMF), Access and Mobility Management Function (AMF), Authentication Server Function (AUSF), Network Slice Selection Function (NSSF), Network Exposure Function (NEF), Network Repository Function (NRF), Policy Control Function (PCF), Unified Data Management (UDM) and Application Function (AF).

Yoo [6] proposed a variety of possible model architectures for 5G network slicing, subdivided the control plane and data plane, and separated the related functions and the reference points between each other. There are different plans for network slices in different scenarios, such as a single UE connected to multiple slices, a single slice connected to multiple UEs, etc., and the signal transmission process of different functions between UE registration and session establishment under these plans is proposed. Kotulski et al. [7] discussed the problem of slice isolation in point-to-point (P2P) connections in 5G networks and related challenges that need to be overcome and solved. They also analyzed the security issues of sharing functions or data exchange between slices. Kotulski et al. [8] also analyzed the slicing isolation for the RAN and CN of the end-to-end (E2E) connection, and quantitatively analyzed the characteristics of the corresponding technology, and the obtained parameters can be used for the realization of slicing.

Mechtri et al. [9] discussed the architecture of implementing Service Function Chaining (SFC) from the perspective of NFV scheduling, and compared the current software's support for SFC, implementation methods and effectiveness, and finally discussed the problems that may be faced in planning SFC. Li et al. [10] measured and analyzed the deployment location and resource utilization of network functions in the cloud SFC, and found the best resource utilization through algorithms. The same function may affect the service quality of the entire slice depending on the location of the data center or edge cloud. To achieve the low-latency requirements of ultra-Reliable and Low Latency Communications (uRLLC), it is necessary to carefully deploy important functions in appropriate locations.

Raza et al. [11] used mixed integer linear programming (MILP) to solve the problem of VNF deployment and reconfiguration of computing resources, which can make the use of VNF resources highly flexible. This way of dynamically deploying resources allows resources to be used effectively. Troia et al. [12] also used MILP to solve the VNF resource problem. The MILP formula evaluates the VNF reconfiguration for power consumption, which can solve the VNF deployment and routing and wavelength allocation problems.

In terms of implementing network slicing with open source software, Minami et al. [13] used an automated slicing system to implement microservices, and provided a visual user interface to drag each VNF for serial connection. The back-end system can automatically deploy slicing according to the content configured by the user, but the system does not connect the core network of mobile communication. Costanzo et al. [14] used a 4G core network to build a dynamic network slicing system, and simultaneously simulated the coexistence of eMBB and IoT slices and sharing the same base station.

Garcia-Aviles et al. [15] used open source software to implement the slicing of RAN, CN, and Virtualized Network Function (VNF) respectively, and combined with OpenStack Management and Orchestration (MANO) to manage the slicing network. The core network is implemented using srsLTE and implemented for enhanced Mobile Broadband (eMBB) and uRLLC scenarios. In addition, they implemented a Local Breakout (LB) mechanism according to the delay requirements of network services, so that packets with special requirements can be processed at edge nodes as much as possible. Chen et al. [16] also used open source software to implement network slicing. They use OpenAirInterface (OAI) Evolved Packet Core (EPC) as the core network and use Universal Software Radio Peripheral (USRP) to enable physical UE access. In their architecture, the deployment of slices also uses OpenStack and Tacker for VNF deployment. Moreover, they implement QoS services and use Open vSwitch (OVS) Queue commands for traffic control.

The main research direction of this paper is similar to the literatures [15] and [16]. We also use OpenStack as the development platform to implement VNF, and combine RAN and CN to implement related 5G scenarios. However, in literatures [15] and [16], The core network used is the open source 4G core network, and the architecture of this study is implemented with the open source 5G core network. Literature [15] focused on RAN slicing and packet delay control while this paper focused on packet transmission control between VNFs in slicing and theoretical verification under the co-existence of multiple slices. Although a slice contains multiple VNFs in the architecture, there is only one VNF in the experimental verification in [16]. In addition, in the management of QoS, due to the limitation of Queue command, downlink cannot be restricted, so this paper adopts the method of Meter and OpenFlow to control downlink.

The main difference between 4G and 5G is that the change of the core network architecture, 5G in the control plane using the HTTP/2 protocol, safety promotion also increased the

difficulty of debugging, communication between functions are encrypted, and add many network functions, which also means that synergy and configuration between the function appears especially important, also increased the difficulty of deployment.

III. Proposed Method

A. System architecture

The system of 5G mobile communication network equipped with network slicing functionality proposed in this paper is shown in Fig. 1. According to the number of UEs *N*, a total of *N*+1 Virtual Machines (VMs) equipped with UERANSIM are created. Each VM has an independent private IP, *N* UEs use *N* UERANSIM VMs for simulation, and the remaining VM is for simulation of gNB. The 5G core network of this research is built according to the official process of free5GC. After the relevant network functions are compiled, all the functions except UPF run on the same VM. One SMF controls all UPFs and corresponds to different network slices. Finally, regarding network services, we use OpenStack installed on a separate host to generate related VNFs, and use Tacker to concatenate the corresponding VNFs into a SFC for subsequent network slicing. The first VNF of SFC is used to carry free5GC, but this VNF only runs UPF. All hosts are deployed on the same public area network, and there are multiple private network segments inside OpenStack, and each SFC uses one network segment. In addition, UPF has two interfaces, one is connected to the public area network, and the other is connected to one of the internal OpenStack networks. The purpose is to enable other hosts in the public area network to identify and connect to the UPF so as to enable SFC can be connected to the core network.

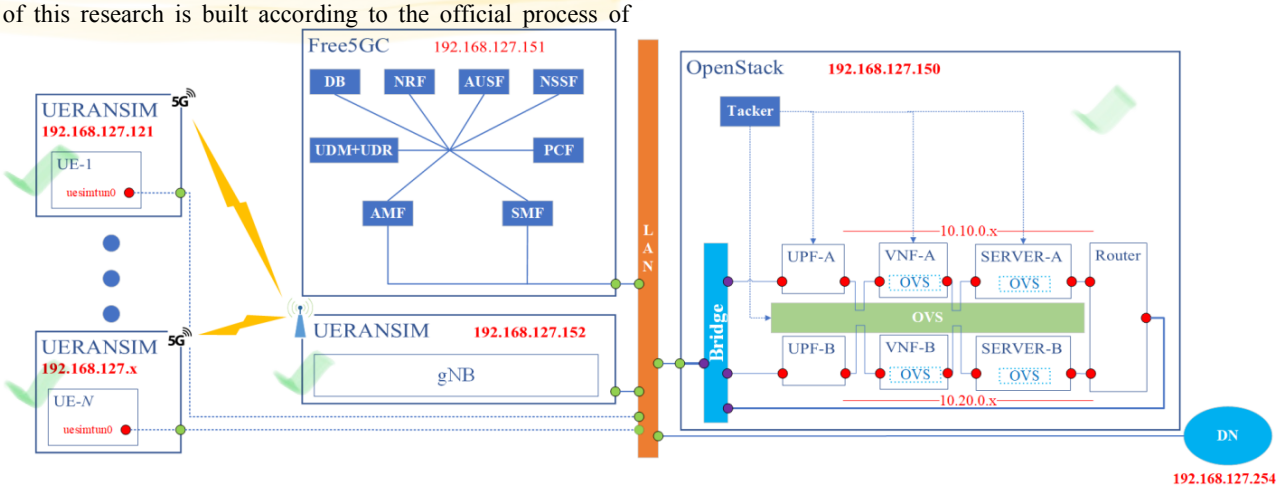

Fig. 1 Context model of the proposed system

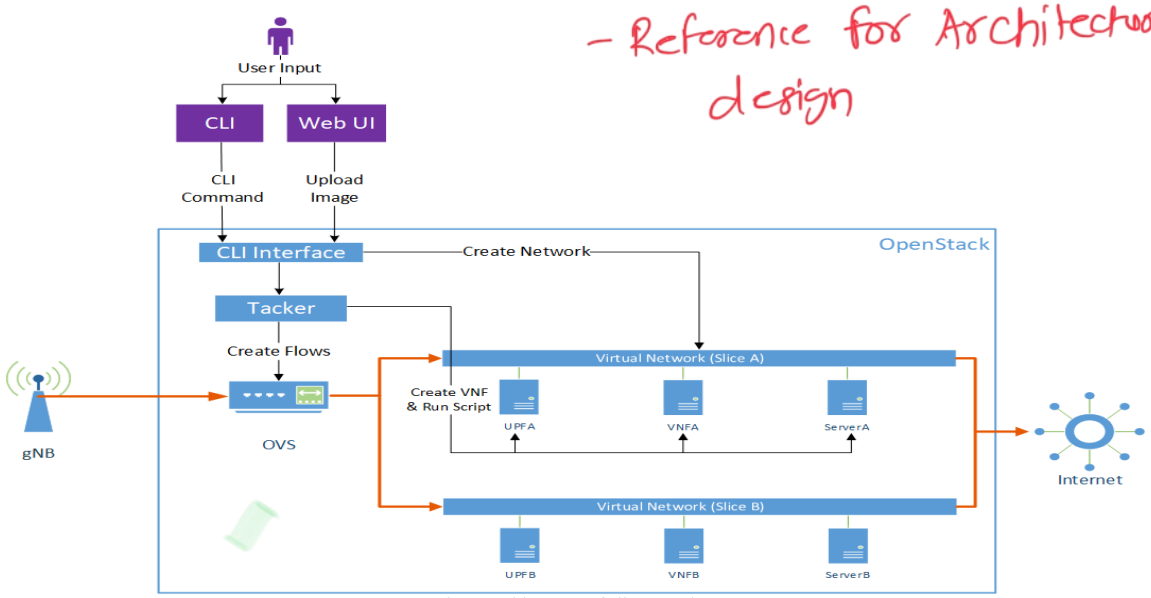

Fig.2 Architecture of slice creation

B. *Slice creation*

Fig. 2 shows the architecture of slice creation proposed in this paper. The orange arrows indicate the flow of user data packets. This architecture diagram simulates any slicing environment that can be created, and the source of packets can come from anywhere, including any device on the Internet and private networks. At this time, this system plays the role of creating cloud services to deploy customized services, such as private websites or File Transfer Protocol (FTP) servers, and manage user authentication and other VNFs to form a set of service chains, which can also be regarded as a slice service. If the packet comes from a mobile client such as a mobile phone, then the VNFs may deploy with RAN and CN functions. The service provider deploys network functions through this system, and specify the correct packet routing through VNF Forwarding Graph (VNFFG) to form an E2E slice service.

Fig. 3 shows the process of slice creation. The service provider creates OpenStack's internal private network segment. Then the service provider uploads the pre-prepared system image files. These image files become the operating matrix of various network functions, similar to the operating mode of Docker. Then the service provider writes the VNF description file required by Tacker for the required network functions, and imports the description file into OpenStack for VNF creation. After VNF is launched, it will run a series of process to setup its environment, including system, network, and OpenFlow rules. Then the service provider writes the VNFFG description file required by Tacker for the required forwarding graph, and imports the description file into OpenStack for VNFFG creation. Finally, the service provider runs the script the setup QoS service for slices, and a full function slice is created.

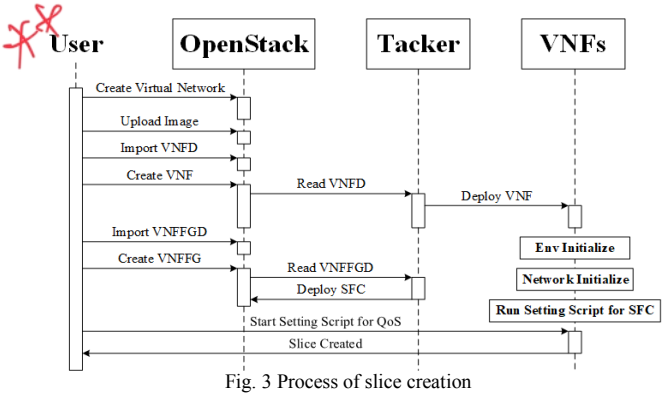

Fig. 3 Process of slice creation

C. *UE packet transmission*

Fig. 4 shows the packet transmission situation in the proposed system, where the slice is located in the SFC section marked in this figure. The packet is transmitted with a Multi-Protocol Label Switching (MPLS) header. The location of the packet encapsulation is in the OVS in OpenStack, and the packet decapsulation is in the OVS of the VNF. The reason is that the Media Access Control (MAC) address of the destination needs to be modified during decapsulation, so that the packet can be sent to the correct destination. To ensure the stable operation of OVS in OpenStack, we delegate the work of modifying the packet to the VNF.

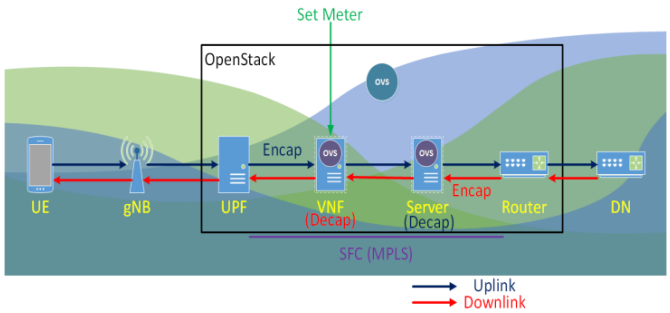

Fig. 4 Transmission route of UE packets

D. *QoS management*

When deploying VNF, the image file of OVS must be prepared as boot media. We install OVS as a kernel module in the Ubuntu system. In addition to higher performance than user mode, kernel mode also uses hardware support to implement OpenFlow Meter. We write part of the flow table into the VNF Descriptor (VNFD) boot script, so that the required OpenFlow flow table is built at the same time as the VNF is created.

We design OpenFlow rules to separate uplink and downlink traffic as shown in Fig. 5. In default, only table 0 will be matched. There are three flows in it, and one NORMAL flow is created by OVS by default, in order to allow packets to pass through normally. The other two match the labels 13567 and 13566 of the MPLS packet respectively. We apply the rules by matching 13567, which means uplink traffic, send to table 5. Packets matching 13566, which means downlink traffic, send to table 6, to separate uplink and downlink. Note that the label is assigned by Tacker, and the value is captured by an automated script provided by us and written into the VNF. In addition, in the MPLS protocol, the label value will be reduced by 1 for each transmission to the next node. Therefore, for uplink, VNF is the first stop, and for downlink, VNF is the second stop. Therefore, the label value can be used as a basis for distinguishing between unlink and downlink.

```
[18:36] root@vn-9dca-c2da-4698-bd2d-50c6a5a25564-vdu1-hkzii46gaulc:~ # shflow
 cookie=0x0, duration=501416.293s, table=0, n_packets=6415181, n_bytes=2092833113, priority=11111,mpls,mpls_label=13567 actions=resubmit(,5)
 cookie=0x0, duration=501416.185s, table=0, n_packets=10055479, n_bytes=13860476464, priority=11111,mpls,mpls_label=13566 actions=resubmit(,6)
 cookie=0x0, duration=573512.396s, table=0, n_packets=10216368, n_bytes=13907409044, priority=0 actions=NORMAL
 cookie=0x0, duration=573500.060s, table=5, n_packets=6415181, n_bytes=2092833113, priority=10 actions=resubmit(,10)
 cookie=0x0, duration=573500.042s, table=6, n_packets=10055479, n_bytes=13860476464, priority=10 actions=resubmit(,11)
 cookie=0x0, duration=501416.168s, table=10, n_packets=6415181, n_bytes=2092833113, priority=11111,mpls actions=set_field:13566->mpls_label,IN_PORT
 cookie=0x0, duration=573500.025s, table=11, n_packets=10055479, n_bytes=13860476464, priority=11111,mpls actions=pop_mpls:0x0800,set_field:ce:04:3f:92:a3:4f->eth_dst,NORMAL
```

Figure 5. OpenFlow rules without meter in VNF (middle forwarder)

After the packet is sent to tables 5 and 6, QoS control is implemented here. When no action is taken by default, the packet is directly forwarded to tables 10 and 11. If Meter is added, it is necessary to add the flow table to which Meter is applied, and to force the rule to be overwritten by priority. Finally, in table 10, the uplink packet needs to be reduced by 1 for the label and forwarded to the next node to complete the work performed by a VNF. In table 11, since the downlink packet has reached the end of the SFC in the VNF, the MPLS header will be disassembled, and the packet will be written into the MAC address of VNF to its destination. Therefore, the routing function in the system can send the packet to the destination correctly.

IV. Experimental Results

This session conducts experimental verification of the proposed system, and the system components include OpenStack, free5GC, UERANSIM, Tacker, and OVS.

A. *Experimental environment*

The environment of the network slicing experiment is shown in Fig. 6 and the software specifications of each component are shown in Table 1, Table 2, Table 3 and Table 4. The figure shows the situation of creating two slices. Note that the number of slices can be increased or decreased as needed. The two slices belong to net1 (orange network segment) and net2 (green network segment) within OpenStack, and their corresponding IP segments are 10.10.0.0/16 and 10.20.0.0/16. Each slice contains UPF, VNF, and Server. UPF has two interface cards, one is connected to 192.168.127.0/24 as an external network to receive packets from the core network and UE, and the other is connected to the OpenStack intranet as the first node of the slice, which is the starting stop of SFC.

Since there is currently no 5G radio hardware available to test, we used software simulation to implement UE and gNB operations. RAN communication processes are in compliance with the 3GPP specification, and each node is wired to ensure a stable network environment for optimal slicing results.

Table 1 Host requirement of UE

UE
● Machine Type: VM ● CPU: 2 cores (Intel® Core™ i7-9700 CPU @ 3.00 GHz) ● RAM: 4 GB ● Disk: 10 GB
UERANSIM v3.2.2
Linux Kernel 5.8.0-59
Lubuntu 20.04

Table 2 Host requirement of gNB

gNB
● Machine Type: VM ● CPU: 8 cores (Intel® Core™ i7-9700 CPU @ 3.00 GHz) ● RAM: 4 GB ● Disk: 10 GB
UERANSIM v3.2.2
Linux Kernel 5.4.0-77
Ubuntu Server 20.04

Table 3 Host requirement of Core Network

Core Network
● Machine Type: Physical Host ● CPU: Intel® Core™ i7-4790 CPU @ 3.60 GHz ● RAM: 16 GB ● Disk: 1 TB
free5GC v3.0.5
Linux Kernel 5.8.0-59
Ubuntu Server 20.04

Table 4 Host requirement of NFV platform

NFV Platform
● Machine Type: Physical Host ● CPU: Intel® Core™ i7-5960X CPU @ 3.00 GHz ● RAM: 64 GB ● Disk: 1.2 TB
OpenStack Train
Linux Kernel 3.10.0-1160.24.1.el7
Centos 7 2009

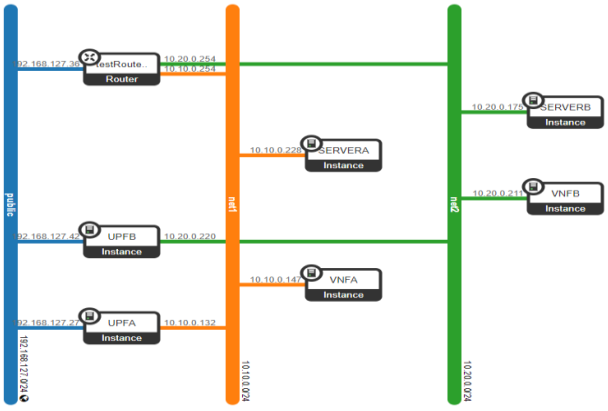

Fig. 6 Intranet of OpenStack in the proposed system

B. *Experimental results*

Experiment 1

The purpose of the first experiment is to verify whether the packet passes through each node in sequence when passing through the SFC. The test method is to observe by capturing packets at each node with the packet analysis tool tcpdump.

In Fig. 7, we captured the network card packet of GPRS Tunnelling Protocol (GTP) in the UPF, and caught the UE packet from 60.60.0.1, which is assigned by UPF, and the destination is the IP address of Google.com. At the same time, we also caught the Internet Control Message Protocol (ICMP) response from Google.com. As in Figure 8, we captured the packet from 10.10.0.101, which is assigned by UPF with NAT transition, means the packet sent by UPF. At the same time, the packet has been encapsulated into MPLS, so it needs to be captured by tcpdump on OVS to verify that UPF is the starting point of SFC, and the destination is also the IP of the Google.com. In addition, we can observe that the labels of MPLS are 13567 for uplink and 13566 for downlink. The first node of the first SFC is preset to use 13567 as its label by default. In Figure 9, we can observe that the packets captured by server are also from 10.10.0.101.

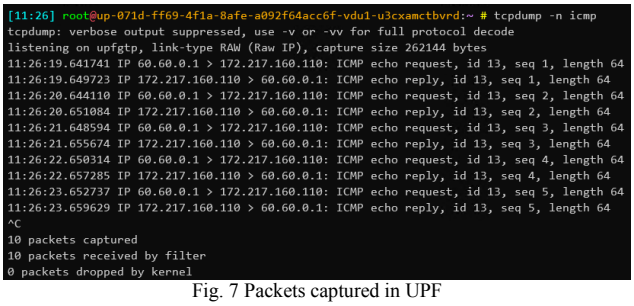

Fig. 7 Packets captured in UPF

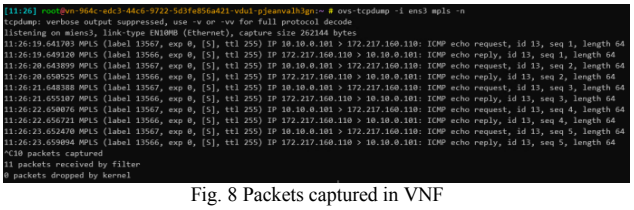

Fig. 8 Packets captured in VNF

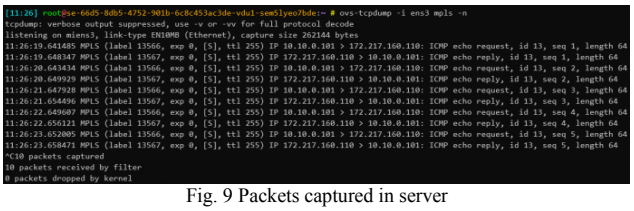

Fig. 9 Packets captured in server

Experiment 2

The purpose of the second experiment is to test the performance of network transmission in the Virtual eXtensible Local Area Network (VxLAN) environment and observe the difference between the network quality controlled by Meter and the theoretical value. The test method is to use the speed test software iperf3 and Speedtest to verify the service performance.

In order to ensure the accuracy of the experimental results, we adopt three packet routes and conduct network performance tests through iperf3 and Speedtest. The iperf3 server is located in the local area network, so the next node after the packet reaches the gateway is the iperf3 server, whereas Speedtest automatically selects the neighboring server for speed measurement.

The speed measurement results of the three routes are shown in Table 1. The speed measurement result with iperf3 is an average of 60 seconds, and the result with Speedtest is the average after 10 runs. It can be observed that the results of the two speed measurement methods for downlink are very close, while for uplink, when the packet enters the Internet, the speed is slightly lower than that in the local area network. From the perspective of routing, the direct connection rate of the VM has almost reached the limit (1Gbps) that the physical line can carry, indicating that the VM is running well. However, after connecting to the gNB and core network, the rate is obviously left half, and after slicing through the complete 5G network, the rate is slightly lower than the result without slicing, indicating that the traffic bottleneck lies in the gNB and 5G core network.

Table 1 Speed test for 3 different routes

Route	iperf3	Speedtest
	Uplink / Downlink Speed (Mb/s)	
Direct	922 / 939	870 / 921
Via UPF	556 / 635	448 / 634
Via SFC	513 / 558	430 / 572

Then we tested the impact of OVS Meter on service performance. Measure method same as testing traffic bottleneck, also we set the traffic limit implemented by OVS meter and OpenFlow rules. We tested it separately from low speed to high speed, and compared the speed measurement results of the downlink with the theoretical values. As shown in Figure 10, under the action of meter, the testing results are close to expectations. Also, iperf3 is more accurate cause its server is on local area network.

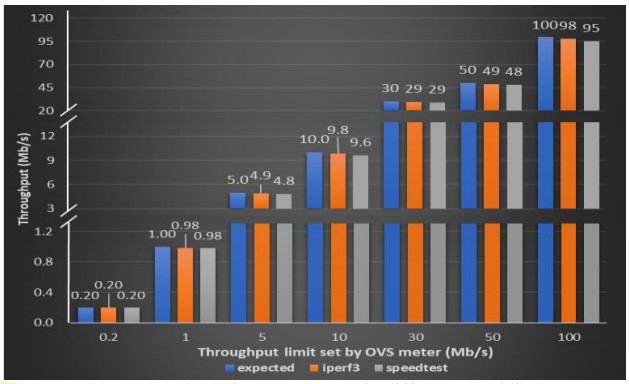

Fig. 10 Speed test for UE to server in different speed limit

Experiment 3

The purpose of the third experiment is to test the slicing operation in the eMBB scenario. The test method is that the UE connects to the slice to perform high-traffic services, and observes the traffic in individual devices and slices.

We created two VMs on separate hosts and equipped with Lubuntu 20.04, a lightweight branch of the Ubuntu operating system, to run UERANSIM. Each VM run a UE. Both UEs were connected to the same slice network, and each UE obtained an IP of the 60.60.0.0/16 network segment. In addition, this slice did not impose any restrictions, so the traffic could reach the theoretical upper limit to meet the definition of eMBB slice. We placed a huge file on the gateway (192.168.127.254), and UEs used the SFTP protocol to download the file on the server to observe the bandwidth usage. In addition, in order to avoid other factors affecting network performance, such as hard disk write bottlenecks, we set the write destination of the UE to NULL, and all write actions were automatically discarded.

According to the results in Fig. 11, it can be observed that when the UEs use the SFTP protocol to download files, the bandwidth can almost be fully used. Note that the traffic in VNF (green line) is always bigger than in gateway (blue line) because packet has been encapsulated with MPLS header, the total length is larger than normal packet. In the figure, the brown line is the average maximum rate of slices measured in experiment 1, and each red line means the process of which UE is downloading files, when two UEs download files at the same time, the bandwidth is equally allocated to the two users. The total bandwidth used is also close to the upper limit, showing that the eMBB operation results are very satisfactory. According to the results, we can also see that in the case of multiple users, it is extremely important to ensure the QoS of individual users to prevent users from over-using or unable to obtain due resources.

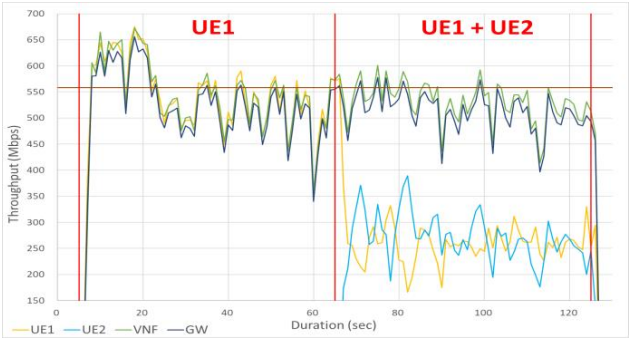

Fig. 11 Bandwidth testing result of eMBB slice

Experiment 4

The purpose of the fourth experiment is to test the slicing operation in the massive Machine Type Communication (mMTC) scene. The test method is to create a large number of IoT clients in batches to connect mMTC slices through a script, and compare the connection speed and delay time of different numbers of UEs in the slices.

This experiment uses the Message Queuing Telemetry Transport (MQTT) protocol as a method to implement mMTC slicing, and uses Mosquitto software to implement MQTT broker and client. We set up an MQTT broker on the Server and use four UEs to run clients, which are publishers and subscribers. The first UE is responsible for sending messages to the broker as a publisher, and the second UE is for receiving messages as a single independent subscriber for efficiency measurement. The remaining two UEs run a specified number of subscribers on average. Each publisher publishes 10 messages with an interval of 0.1 second between each message. The first and second UEs synchronize time to ensure that the delay time can be accurately calculated. The publisher entrains the timestamp when sending in the MQTT message, and subscriber adds a timestamp to the output after the receives the message and redirect to file. The time accuracy is microseconds to facilitate subsequent calculation delays. The server monitors the CPU usage rate of the broker and outputs it to the file. The time precision is 0.1 second. The usage rate is multiplied by the execution time, and finally the actual usage time of the CPU is used for performance comparison.

As can be seen in Table 2, as the number of subscribers and publishers increases, the load on the server also increases. When the amount of published messages exceeds 1 million (1000 publishers each publish 10 messages, and copy 100 copies to 100 subscribers, referring to the 100/1000 data), packet loss begins. When the number of subscribers is *N,* after each message is published, *N* copies need to be copied and sent to *N* subscribers at the same time, which becomes a performance bottleneck. It can also be observed that the increase in the number of publishers causes greater pressure on the CPU. The reason is that as the number of publishers increases, the published messages need to be copied and sent on the broker, and subsequent messages must enter the queue before sending. Once the broker cannot handle it, the queue will overflow and cause packet drop.

Table 2 Performance of different subscribes/publishers (no. of subscriber is too small)

Subscribers / Publishers	Latency (ms)	Packet Loss (%)	Process's CPU Load (%)	Execution Time (sec)	CPU Time (sec)
100 / 100	19.259	0.00	14.22	4.0	0.56860
100 / 500	44.585	0.00	13.63	20.2	2.75360
100 / 1000	175.758	0.02	12.81	42.6	5.45680
500 / 100	39.654	0.00	37.28	5.0	1.86420
500 / 500	169.199	0.84	36.71	25.6	9.39840
500 / 1000	390.211	1.68	46.21	47.0	21.71760

Experiment 5

The purpose of the fifth experiment is to test the slicing operation in the uRLLC scene. The test method is to create SFCs of different lengths, to simulate the amount of service by increasing or decreasing the number of VNFs, and to test the performance through the delay of ping. We created two slices, as shown in Fig. 13. The first slice contains only UPF and a server, and the second slice contains UPF, a server and an additional VNF. In this experiment, except for UPF, the rest of the nodes only help the packet forwarding. We used two UEs

to connect to slice one and slice two respectively, and measured the delay of ping to the gateway.

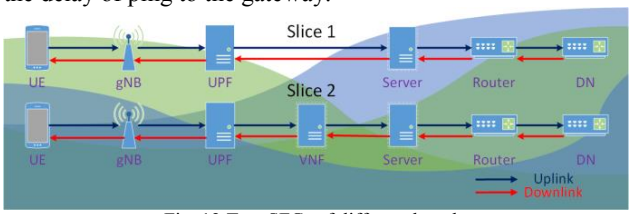

Fig. 12 Two SFCs of different lengths

We performed 20 ping instructions and recorded the results as shown in Table 3. As the number of sliced nodes increases, the delay also increases. Therefore, when the service of uRLLC is needed, reducing the routing path and reducing the number of computing nodes is a key task.

Table 3 Latency of two slices of different lengths

Slice #	Avg Latency (ms)	Min Latency (ms)	Max Latency (ms)
1	0.449	0.228	0.545
2	0.495	0.391	0.726

V. CONCLUSIONS

This paper studies the realization of network slicing by controlling packet routing, using OpenStack to allocate cloud resources, carrying Tacker for VNF deployment and designing unique scripts to drive VNF, so that it can operate as scheduled. The main contribution of this paper is to use the serial connection of various open source software to realize the 5G network slicing environment. From UE connection, gNB configuration, core network construction, and service content in the slice, it can be completely connected into one E2E connection. Through Meter and OpenFlow, the SLA of slices and the separation of individual users can be achieved. We use a single physical host to create diversified network slices, use VxLAN to segment different slices to achieve slice isolation, and use Tacker to quickly deploy VNFs and use SFC for cascading to achieve highly flexible planning and configuration. In addition, the use of self-compiled scripts for simple settings to achieve automatic operation is extremely convenient and helpful for subsequent research and development. In the future, we are going to use the architecture provided in this paper to import and implement dynamic network slicing

REFERENCES

[1] GSA, "5G Standalone 2021 – Summary," [Online]. Available: https://gsacom.com/paper/5g-standalone-2021-summary/

[2] 5G PPP Architecture Working Group, "View on 5G Architecture," 2020. [Online]. Available: https://5g-ppp.eu/wp-content/uploads/2020/02/5G-PPP-5G-Architecture-White-Paper_final.pdf.

[3] ETSI, "System Architecture for the 5G System (5GS)," *3GPP TS 23.501 version 16.7.0,* 2021.

[4] Y. Choi and N. Park, "Slice Architecture for 5G Core Network," *Ninth International Conference on Ubiquitous and Future Networks (ICUFN 2017),* pp. 571-575, 2017.

[5] J. Ordonez-Lucena, P. Ameigeiras, D. Lopez, J. J. Ramos-Munoz, J. Lorca and J. Folgueira, "Network Slicing for 5G with SDN/NFV: Concepts, Architectures, and Challenges," *IEEE Communications Magazine,* vol. 55, issue 5, pp. 80-87, 2017.

[6] T. Yoo, "Network Slicing Architecture for 5G Network," *International Conference on Information and Communication Technology Convergence (ICTC 2016),* pp. 1010-1014, 2016.

[7] Z. Kotulski, T. W. Nowak, M. Sepczuk and M. A. Tunia, "On End-to-end Approach for Slice Isolation in 5G Networks. Fundamental Challenges," *2017 Federated Conference on Computer Science and Information Systems (FedCSIS 2017),* pp. 783-792, 2017.

[8] Z. Kotulski, T. W. Nowak, M. Sepczuk and M. A. Tunia, "5G Networks: Types of Isolation and Their Parameters in RAN and CN Slices," *Computer Networks,* vol. 171, pp. 1-15, 2020.

[9] M. Mechtri, C. Ghribi, O. Soualah and D. Zeghlache, "NFV Orchestration Framework Addressing SFC Challenges," *IEEE Communications Magazine,* vol. 55, issue 6, pp. 16-23, 2017.

[10] D. Li, P. Hong, K. Xue and j. Pei, "Virtual Network Function Placement Considering Resource Optimization and SFC Requests in Cloud Datacenter," *IEEE Transactions on Parallel and Distributed Systems,* vol. 29, issue 7, pp. 1664-1677, 2018.

[11] M. R. Raza, M. Fiorani, A. Rostami, P. Öhlen, L. Wosinska and P. Monti, "Dynamic Slicing Approach for Multi-Tenant 5G Transport Networks [Invited]," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 10, issue 1, pp. A77-A90, 2018.

[12] S. Troia, A. Cibari, R. Alvizu and G. Maier, "Dynamic Programming of Network Slices in Software-defined Metro-core Optical Networks," *Optical Switching and Networking*, vol. 36, pp. 1-13, 2020.

[13] Y. Minami, A. Taniguchi, T. Kawabata, N. Sakaida and K. Shimano, "An Architecture and Implementation of Automatic Network Slicing for Microservices," *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium,* pp. 1-4, 2018.

[14] S. Costanzo, I. Fajjari, N. Aitsaadi and R. Langar, "Dynamic Network Slicing for 5G IoT and eMBB services: A New Design with Prototype and Implementation Results," *2018 3rd Cloudification of the Internet of Things (CIoT),* pp. 1-7, 2018.

[15] G. Garcia-Aviles, M. Gramaglia, P. Serrano, F. Gringoli, S. Fuente-Pascual and I. L. Pavon, "Experimenting with Open Source Tools to Deploy a Multi-service and Multi-slice Mobile Network," *Computer Communications*, vol. 150, pp. 1-12, 2020.

[16] S. Chen, C. N. Lee and M. F. Lee, "Realization of 5G Network Slicing Using Open Source Softwares," *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference 2020 (APSIPA ASC 2020),* pp. 1549-1556, 2020.