Realization of 5G Network Slicing Using Open Source Softwares

Sheng Chen†, Chung-Nan Lee* and Ming-Feng Lee+
National Sun Yat-sen University, Taiwan
†E-mail: patrick021588@gmail.com Tel: + 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: + 886-7-5252000 ext. 4335

***Abstract*—With the advent of 5G, billions of devices use the 5G radio access network. Network slicing is a fundamental architecture component of the 5G. End-to-end network slicing leverages the attributes of central virtualization technology in 5G to flexibly address. In this paper we realize the network slicing using OpenStack as a cloud platform, Tacker to manage the required virtual network functions for each service and OpenFlow switch for slice. We also apply OpenFlow Queue command to schedule the priority and proportion of each service. Finally, the actual implementation is carried out via OpenAirInterface and nextEPC. Experimental results show that the network slicing is feasible using open sources and the bandwidth can be assigned using queue. As a result, the QoS is guaranteed for each slice.**

I. INTRODUCTION

The fifth generation (5G) mobile communication system has the potential to support the communication of billions of devices at ultra-high speed, and it may change people's lives. At present, the construction of 5G in countries around the world is also in full swing. Unlike the fourth generation (4G) mobile communication system, which has only one service type for all customers, 5G can provide different service needs, such as enhanced Mobile Broadband (eMBB), Ultra Reliable Low Latency Communications (URLLC), massive Machine Type Communications (mMTC), for different customers. For each type of business customer, the most logical way is to build a dedicated network to provide services, but this is very uneconomical. The more reasonable and economical way to support different customers and different services in the same mobile communication system is network slicing technology. Network slicing is to run multiple logical networks on the communication architecture in an efficient and economical way as an embodiment of almost independent service operation concepts. Compared with the communication network system, this is a fundamental change.

The concept of network slicing has been proposed in the traditional network era, but before 5G, there has not been a network slicing standard suitable for the new generation of mobile communications. Many studies have proposed architectures and suggestions about realization of 5G network slicing, and the architectures proposed by each research are different. It was only in recent years that the standard of 3rd Generation Partnership Project (3GPP) [1] began to make some clear definitions and preliminary implementation methods for the 5G network slicing.

The purpose of this study is to implement and verify the Proof of Concept (PoC) of 5G network slicing using existing open source softwares. We use OpenStack as a cloud platform, use Tacker to realize the Virtualized Network Functions (VNFs) required by each service, and then combine with OpenFlow switch, and use OpenFlow Queue command to ensure the priority and proportion of each service slice. The SDN controller OpenDaylight (ODL) is used to control the physical/virtual SDN switches, and the OpenFlow commands is used to implement 5G network slicing through the controller. In the proposed implementation, the framework of the mobile communication network is realized through OpenAirInterface (OAI) and nextEPC. Finally, some experiments are designed to verify that the proposed implementation in this paper can comply with the architecture and principles proposed by 3GPP.

The main characteristics of this study are as follows. We propose an open source network architecture based on OAI, nextEPC, and OpenStack. On this architecture, we implement and verify the technical concept of 5G network slicing and adjust the OpenFlow Queue command at the service end to ensure the priority and usage ratio of each service. This effectively improve the utilization of network resources, and make the architecture more in line with low latency and higher transmission guarantee. The propoased open source network architecture makes it easy to modify or adjust the system through Software Defined Networking (SDN) and Network Function Virtualization (NFV), which makes the overall system architecture more versatile and convenient for management.

The rest of this paper is organized as follows. In Section II, we review the related works about 5G network slicing. Section III describes the proposed mechanism realizing network slicing. Section IV gives experimental results. Conclusions are drawn in Section V.

II. Related Work

Li et al. [2] mentioned that the 5G network architecture has higher bandwidth requirements, and can integrate the Cloud Radio Access Network (C-RAN) architecture on the same transmission route. They also mentioned that the future network needs to cope with different applications of multi-tenants at the same time. They then proposed a network architecture, and explained the required technology from the control plane and the data plane, and what conditions need to be met at each plane. For example, the four conditions that need to be met at the data plane are as follows.

- Traffic separation: any tenant cannot hear or obtain other tenants' traffic or find their usage trends in any way.
- Traffic isolation: the network must be able to guarantee the quality of service (QoS) of each tenant. Any tenant in the network will not be affected by the amount of traffic of other tenants.
- Traffic differentiation: even if different tenants enter the same network connection point, tenants' packets can be forwarded in different ways.
- Statistical multiplexing: the network can simultaneously handle network traffic of different tenants through different channels.

Yuki et al. [3] proposed an automated network slicing system composed of microservices. Since network slices are used in various scenarios, optimizing the slices becomes a rather cumbersome part for designers. The main application of this automated system is that as long as the user gives some basic restrictions for each slice, it can optimize the slice in a short time according to the conditions given by the user. Farrel [10] proposed that NFV, SDN and Service Function Chaining (SFC) can be used to implement network slicing in 5G networks, and also proposed several problems that may be encountered in the integration of these technologies.

Barakabitze et al. [11] collated possible application scenarios of 5G networks and compared multiple network slicing architectures and strategies implemented through SDN and NFV. They finally further explained the problems and challenge that 5G networks may face. Kwak et al. [5] proposed an algorithm to implement dynamic network slicing. The simulation results show that their dynamic slicing algorithm is superior to the static slicing algorithm in terms of average total cost and average total delay. Vassilaras et al. [6] pointed out that the ability to quickly deploy network slicing through SDN and NFV will be the future development trend of mobile communications, so how to effectively manage resources and quickly allocate resources will be the focus of future 5G networks. They then analyzed network slicing technology from some algorithmic aspects.

Based on SDN and NFV, Ordonez-Lucena et al. [7] proposed an architecture that supports multi-tenancy. This architecture enables the network slicing provider to deploy network slicing instances for multiple tenants in real time and provide them with isolation guarantees. Following the proposed model of network slicing as a service, tenants can choose the slicing that best suits their needs.

Song et al. [8] implemented the concept of service-oriented network slicing. They also proposed several calculation formulas for evaluating slice performance and latency. Through the simulation, they tested the impact of different traffic load on the latency of each scenario in the three major application scenarios of 5G. Through this study, a reliable numerical method for evaluating the performance of slice can be learned. Guan et al. [9] used mathematical models to construct network slice requests and map them to the infrastructure network. They presented the mathematical model of deploying end-to-end slices and proposed a network slice request implementation algorithm. The results show that the implementation algorithm performed well in terms of resource efficiency and acceptance ratio.

Afaq et al. [12] used XOS to process the TOSCA file configuration and the required service configuration, coordinate the Virtual Machine (VM) and virtual Network (VN) required by OpenStack configuration through XOS, and define the QOS of the service configured by XOS through SDN controller ONOS. In addition, they introduced the concept of visiting Network Slice Selection Function (vNSSF) to enhance the concept of slicing, which is used to allow 5G networks to allow UEs to access more services at the same time.

Salvatore et al. [4] implemented network slicing in the open source network using the open source software OAI. They adopted the OAI C-RAN version, used FlexRAN as their controller to control the switches and achieve network slicing by issuing OpenFlow commands. Their verification method is to download packets through two user equipments (UE) at the same time. One UE increases the download traffic to test whether the other UE will be affected by it. The result of the experiment confirm that their architecture has met the concept of network slicing.

Chien et al. [13] proposed a service-based approach to provide network slicing, aiming at allocating a slice to each service. Their architecture mainly integrates OpenStack with Tacker and other related open source softwares. This research also introduced a value R to define slice efficiency. By comparing the resources required by the computing service with the resources that can actually be given, R can be used to verify whether the method meets the needs of the three major application scenarios of 5G, including URLLC and mMTC and eMBB.

The main research direction of our work is similar to the studies of [4] and [13]. Compared with [4] and [13], we design more experiments for verification according to the standard proposed by 3GPP, so as to ensure that the system can meet the network slicing criteria. More specifically, the literature [4] does not provide appropriate experimental scenarios and experimental verification, nor does it express how to verify the independence between slices, and the overall architecture is relatively incomplete. In the literature [13], although a good architecture is proposed to implement network slicing, it also

lacks verification of the independence between slices. This paper not only provides more complete simulation and practical tests, but also verifies the independence between slices and the security under UDP DDoS attacks.

III. Proposed Mechanism Realizing Network Slicing

A. System architecture

The proposed system architecture is combined with nextEPC and OpenStack under OAI's open network architecture. If there is a complete 5G open source core network in the future, it can also be used to verify the concept of network slicing under this architecture. Under the proposed architecture, users with different needs can be separated by slicing and allocated to network resources according to service needs and behavior.

The architecture of the proposed implementation mechanism is shown in Fig. 1. In this architecture, the three types of service (eMBB, mMTC, URLLC) are planned for three different slices according to the 5G network specifications. The streaming server is mainly used to simulate the eMBB scenario that requires a large bandwidth to transmit images; the IoT server is mainly used to simulate the mMTC scenario of multiple IoT devices; finally, the V2X server is used to simulate URLLC scenario that transmits important signals. Then we use SDN controller and OpenStack to allocate resources reasonably to achieve the effective utility of resources.

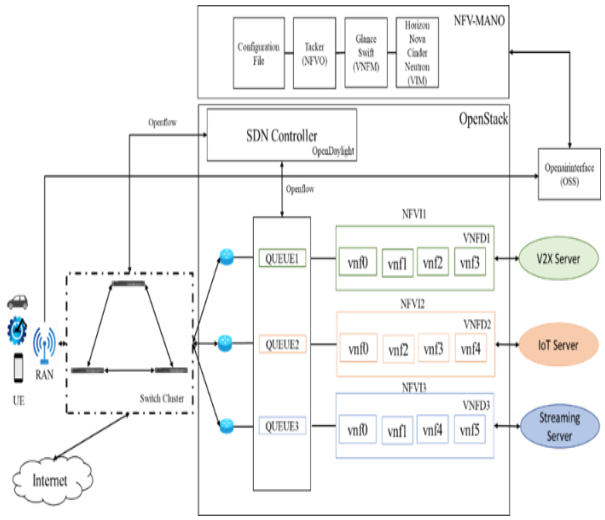

Fig. 1 Architecture of the proposed implementation mechanism

Fig. 2 gives an example of network topology that may be used in three types of service applications. Through Fig. 2, it can be seen that different services cannot communicate with each other, and service slices can be deployed more according to demand, not necessarily just the three slices shown in the figure. Through the SDN controller, we can correctly forward the packets of each service to the correct router to deploy the slice.

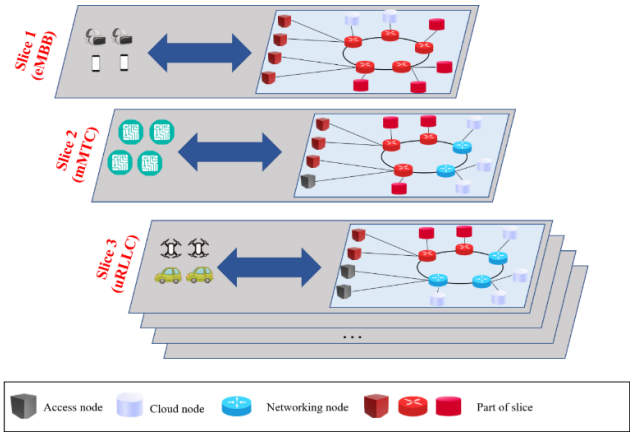

Fig. 2 Schematic diagram of network slicing in the proposed implementation mechanism

Fig. 3 illustrates how to implement the virtual network functions (VNFs) used in the Network Function Virtualization Infrastructure (NFVI) in 5G core network architecture. There is mainly NSSF (Network Slice Selection Functions, NSSF) responsible for slice management; Access and Mobility Management Function, (AMF) and Session Management Function (SMF) are used for session authentication; Authentication Server Function (AUSF) is responsible for the authentication part with Unified Data Management (UDM) database, and finally some of the functions needed for this service such as Policy and Charge Function (PCF) and Application Function (AF).

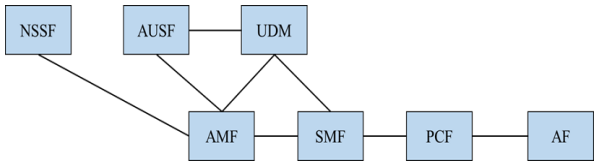

Fig.3. Schematic diagram of implementing NFVI through VNFs in 5G core network architecture

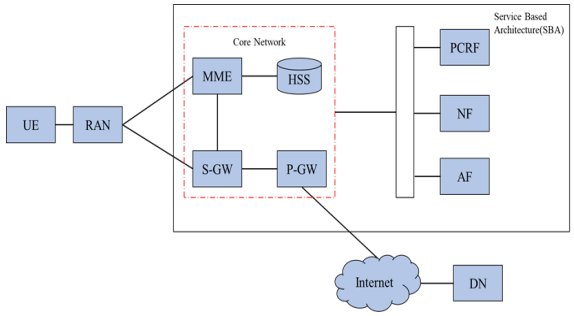

Fig. 4 Realizing NFVI through OAI C-RAN version and nextEPC in the proposed implementation mechanism

Because it is currently difficult to obtain complete open source software for the simulated 5G core, the implementation

and verification of this research is through 4G softwares, OAI eNB and nextEPC, where OAI is the C-RAN version, and nextEPC is used to replace the 5G core network. The proposed implementation of NFVI architecture is shown in Fig. 4, where RAN is implemented by OAI eNB, and core network including Mobility Management Entity (MME), Home Subscriber Server (HSS), Serving Gateway (S-GW) and Packet Data Network Gateway (P-GW) are implemented through nextEPC. In the proposed implementation mechanism, User Plane Function (UPF) of 5G core is implemented through S-GW and P-GW of nextEPC. SMF and AMF are implemented through some session and mobility functions in MME of those are mainly NSSFs (Network Slice Selection Functions. The authentication functionality of AUSF is replaced by MME and HSS of nextEPC. UDM is implemented by HSS of nextEPC.

In addition, we also adopt Virtual eXtensible Local Area Network (VXLAN) in the proposed implementation mechanism. VXLAN is a new type of network virtualization technology. Compared with traditional Virtual LAN (VLAN), VXLAN can provide better scalability and flexibility. In 5G, services need to be provided to more users, so it is one of the reasonable methods to improve large cloud computing deployments through VXLAN. Based on the OpenStack architecture, we modify the configuration files of the controller node, network node, and compute node to implement VXLAN.

B. Control plane

In order to implement the network slicing architecture, the control plane and the data plane must be separated in practice. The proposed implementation mechanism realizes the control plane interface through the open source SDN controller ODL and OpenFlow protocol.

We implement resource management of network slice instances (NSIs) through ODL, and transfer signals through Application Programming Interface (API). In order to realize that each network slice is isolated from each other, the system logically separates each slice through NFV and SDN, and each slice obtains the network functions it needs.

C. Signal processing and slice management

Fig. 5 illustrates the signal processing in the proposed implementation mechanism. First, the UE sends its resource requirements to Remote Radio Head (RRH), then RRH passes the requirements to SDN Conroller and Slice Orchestrator (SC&SO), and finally SC&SO allocates the required resources to the UE, and establishes a service-based slice to the UE.

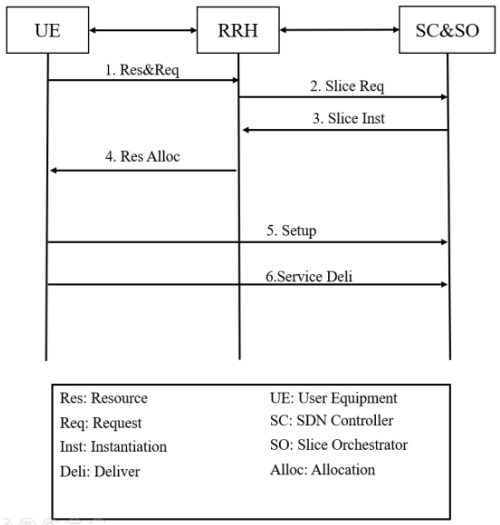

Fig. 5. Signal processing of the proposed system

In different service scenarios of 5G, there are different process sequences. In the two scenarios of eMBB service and mMTC service, first of all, the service requirements will be passed to SC&SO through API. Then SC&SO will set the required resources for slicing according to the network resources and space resources owned by the system. In contrast, if it is a URLLC service, the required resources will be created directly through the SC, and will not go through the network functions created by the SO, thus the URLLC service will be delivered more quickly.

D. Implementation of VNF through Tacker and Queue

Tacker is an official project of OpenStack that builds a Generic VNF Manager (VNFM) and an NFV Orchestrator (NFVO) to deploy and operate network services and VNFs on OpenStack platform. In the proposed implementation mechanism, we first create a Tacker database, then connect with OpenStack, and create a Virtualized Infrastructure Manager (VIM) and VNF through a configuration file. Then we write the configuration file of the Virtual Network Function Description (VNFD), and finally we can create the VNF required by the service through the VNFD configuration file. OpenFlow commands are used to create Queues to the corresponding ports, and Weighted Fair Queue (WFQ) is used to accurately ensure that services with higher priority are first guaranteed.

IV. EXPERIMENTAL RESULTS

A. Experimental Environment

This section describes the construction and testing experiments of the proposed implementation mechanism. The hardware for building this mechanism and the open source softwares used for development are listed in Table 1.

Table 1. Experimental environment

Experiment environment and equipments
• Operating System: Centos 7 • CPU: Intel® Core™ i7-6700 CPU @ 3.40GHz 3.41GHz • Main Memory: 16.0 GB • USRP:Ettus USRP B210、Ettus USRP B200 mini • 4610-3OT -O-AC-F with Pica8 NoS L2+OVS • Development Software: OpenAirInterface、OpenStack 、Opendaylight、Mininet、OpenvSwitch • Experimental Software: Mosquitto、VLC streaming、TFN2K

Here we briefly explain the contents of the five experiments in this paper. Experiment 1 is to test whether the slice bandwidth is guaranteed by Queue after establishing VXLAN on OpenStack. Experiment 2 is to test the variation of the latency of the service requested by the UE according to traffic load. Experiment 3 is mainly to verify the service guarantee of the slice. The purpose of experiment 4 is to test the number of connected MTC devices and packet loss rate of mMTC slice. Experiment 5 is to test when one slice suffers from User Datagram Protocol (UDP) Distributed Denial of Service (DDoS), will other slices be affected?

B. Experimental results

Experiment 1

The purpose of experiment 1 is to verify the error rate of bandwidth guaranteed by Queue. The test method is to set the upper and lower limits of the slice bandwidth guaranteed by the Queue to be the same, then generate traffic flow and measure the actual throughput through the iperf software. We measured once every 10 seconds and took the average of the other 10 times except the first measurement.

Fig. 6 shows the measurement before the Queue command has been issued. At the first second, because iperf is establishing a connection, the measured data is not accurate. Therefore, we do not take the measured result of the first second. After the measurement, the bandwidth is about 95-100 Mbits/sec before the OpenFlow Queue command is issued. Next, we issued Queue through ODL, and set the upper and lower slice throughput limits to 5 Mbits/sec and 50 Mbits/sec, as shown in Figure 7 and Figure 8, respectively.

```
Connecting to host 140.117.168.119, port 5201
[  4] local 140.117.168.124 port 59510 connected to 140.117.168.119 port 5201
[ ID] Interval           Transfer     Bandwidth
[  4]   0.00-1.00   sec  8.12 MBytes  68.0 Mbits/sec
[  4]   1.00-2.00   sec  11.2 MBytes  94.5 Mbits/sec
[  4]   2.00-3.00   sec  11.4 MBytes  95.3 Mbits/sec
[  4]   3.00-4.00   sec  11.2 MBytes  94.4 Mbits/sec
[  4]   4.00-5.00   sec  11.2 MBytes  94.4 Mbits/sec
[  4]   5.00-6.00   sec  11.4 MBytes  95.4 Mbits/sec
[  4]   6.00-7.00   sec  11.2 MBytes  94.4 Mbits/sec
[  4]   7.00-8.00   sec  11.2 MBytes  94.3 Mbits/sec
[  4]   8.00-9.00   sec  11.2 MBytes  94.4 Mbits/sec
[  4]   9.00-10.00  sec  11.4 MBytes  95.3 Mbits/sec
```

Fig. 6 Bandwidth measurement before issuing the Queue command

```
ccepted connection from 140.117.168.124, port 59509
[  5] local 140.117.168.119 port 5201 connected to 140.117.168.124 port 59510
[ ID] Interval           Transfer     Bandwidth
[  5]   0.00-1.00   sec  106 MBytes   88.5 Mbits/sec
[  5]   1.00-2.00   sec  5.50 MBytes  4.61 Mbits/sec
[  5]   2.00-3.00   sec  5.62 MBytes  4.72 Mbits/sec
[  5]   3.00-4.00   sec  5.62 MBytes  4.72 Mbits/sec
[  5]   4.00-5.00   sec  5.62 MBytes  4.72 Mbits/sec
[  5]   5.00-6.00   sec  5.62 MBytes  4.72 Mbits/sec
[  5]   6.00-7.00   sec  5.62 MBytes  4.72 Mbits/sec
[  5]   7.00-8.00   sec  5.62 MBytes  4.72 Mbits/sec
[  5]   8.00-9.00   sec  5.62 MBytes  4.72 Mbits/sec
[  5]   9.00-10.00  sec  5.62 MBytes  4.72 Mbits/sec
[  5]  10.00-10.04  sec   506 KBytes  4.72 Mbits/sec
```

Fig.7 Bandwidth measurement uder the lower throughput limit of 5Mbits/sec

```
ccepted connection from 140.117.168.124, port 59509
[  5] local 140.117.168.119 port 5201 connected to 140.117.168.124 port 59510
[ ID] Interval           Transfer     Bandwidth
[  5]   0.00-1.00   sec  157 MBytes   132 Mbits/sec
[  5]   1.00-2.00   sec  57.2 MBytes  47.8 Mbits/sec
[  5]   2.00-3.00   sec  57.2 MBytes  47.8 Mbits/sec
[  5]   3.00-4.00   sec  57.4 MBytes  47.8 Mbits/sec
[  5]   4.00-5.00   sec  57.2 MBytes  47.8 Mbits/sec
[  5]   5.00-6.00   sec  57.2 MBytes  49.6 Mbits/sec
[  5]   6.00-7.00   sec  57.4 MBytes  48.7 Mbits/sec
[  5]   7.00-8.00   sec  57.2 MBytes  48.7 Mbits/sec
[  5]   8.00-9.00   sec  57.2 MBytes  49.6 Mbits/sec
[  5]   9.00-10.00  sec  57.2 MBytes  49.6 Mbits/sec
[  5]  10.00-10.04  sec   506 KBytes  48.7 Mbits/sec
```

Fig. 8 Bandwidth measurement uder the upper throughput limit of 50Mbits/sec

Fig. 9 shows that the variation ratio between the actual bandwidth and the Queue setting bandwidth is about 4%-6%, and the variation ratio is not large. This experiment shows that it is effective to manage the network resources required by individual slices through the OpenFlow Queue command.

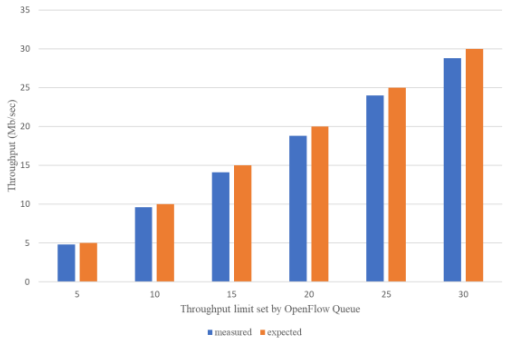

Fig. 9 Experimental result of bandwidth guarantee

Experiment 2

The purpose of experiment 2 is to explore the end to end delay of the service slice. The architecture of experiment 2 is shown in Fig. 10. This experiment is divided into two kinds of tests. The first test is to check the variation of end to end latency from UE1 to the Google server (eMBB slice) and that from UE3 to the Google server (URLLC slice) under normal transmission conditions. The second test is to observe the variation of end to end latency by increasing traffic load. To be more specifically, UE2 and UE4 used iperf to generate heavy traffic, we then used IP-Tools to observe the end to end latency from UE1 to the Google server and that from UE3 to the Google server. Note that in Fig. 10, the setting of URLLC slice (in red) is to install the core network in the physical machine instead of in the Virtual Machine (VM) of OpenStack and the transmission does not go through Open vSwitch (OVS), so it can reduce the latency slightly.

Fig. 11 shows the end to end latency variation under normal transmission conditions, and Fig. 12 shows the end to end latency variation under heavy traffic load. Since this experiment uses nextEPC and OAI eNB to implement, there is a transmission delay error in the experiment, so the latency of both eMBB and URLLC performs zigzag style curve under normal transmission condition in Figure 11. From Fig. 12, it can be found that both URLLC and eMBB slices have a significant increase in latency when the traffic load is greater than 0.6.

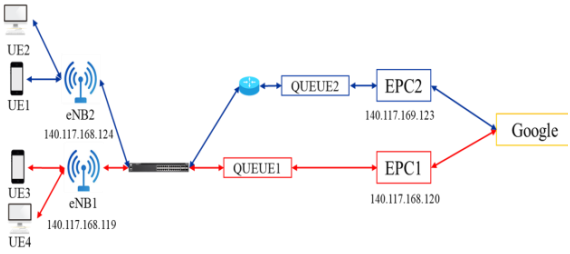

Fig. 10 Architecture of experiment 2

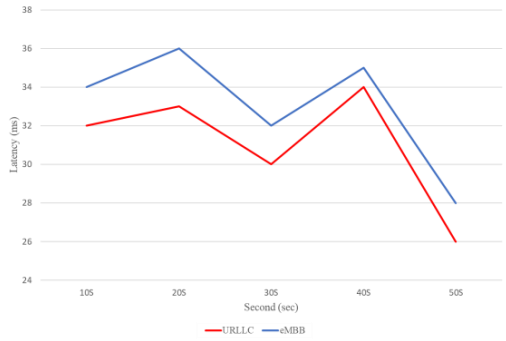

Fig. 11 Variation in end to end latency under normal transmission conditions

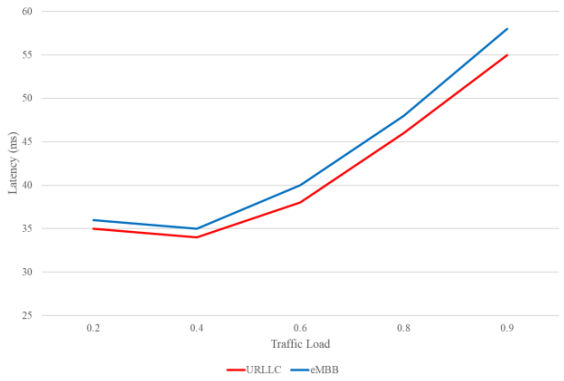

Fig. 12 Variation in end to end latency under heavy traffic load

Experiment 3

The purpose of experiment 3 is to test whether the service of slices can be guaranteed through Queue. The experiment architecture is shown in Fig. 13.

First, we used Queue to set the upper and lower throughput limits of of eMBB and mMTC slices, and then injected heavy traffic flow to the mMTC slice to simulate network congestion, in order to verify whether the slice service can be guaranteed. The traffic of eMBB and mMTC slices in this experiment are all simulated by iperf. The upper and lower limits of slices throughput are set as follows. The upper throughput limit of eMBB slice is 70 Mb/sec, the lower limit is 30Mb/sec. The upper throughput limit of mMTC is 50Mb/sec, and the lower limit is 30 Mb/sec.

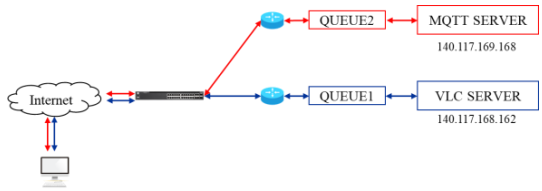

Fig. 13 Architecture of experiment 3

Fig. 14 shows the experimental result of traffic guarantee of slices. In this experiment, in order to facilitate the experiment process, we started the mMTC slice service and kept it stable through the Queue command, and then added the eMBB slice to verify whether the upper and lower throughput limits were guaranteed. It can be seen that when the traffic load of mMTC slice is heavy, the throughput of the mMTC slice gradually drops to about the guaranteed lower limit, and then does not continue to decrease. Thus it can be confirmed that the slice throughput can be guaranteed by Queue. Note that because the setting bandwidth and the actual measured bandwidth have a variation rate of about 4%-6% (as shown in experiment 1), the actual minimum throughput of mMTC slice in Fig. 14 is slightly lower than the lower limit of 30Mb/sec set by Queue.

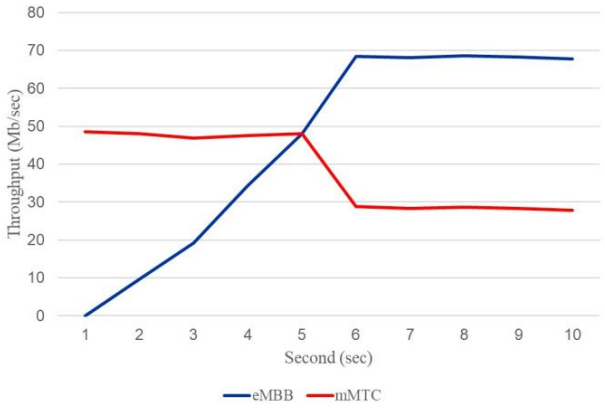

Fig. 14 Experimental result of throughput guarantee of slices

Experiment 4

Experiment 4 is to test the number of connected MTC devices and packet lost rate of mMTC slice. The mMTC simulation software used in this experiment is Moquette, which is an open source software Message Queueing Telemetry Transport (MQTT) broker used to simulate transmission of MQTT Protocol. This experiment is mainly divided into two parts. The first part is to test the limit of the number of connected devices (MQTT subscribers) of mMTC slice in our

architecture. The second part is to test the packet lost rate of mMTC slice when many MTC devices process unlink transmission at the same time. The architecture of experiment 4 is shown in Fig. 15. The MQTT broker delivers the publisher's message to each MQTT subscriber according to the topics subscribed by the subscriber. During this experiment, all messages sent are in plain text format. In addition, in order to verify the error rate, we set MQTT QoS to 0, which means that each message is only sent once regardless of success or failure.

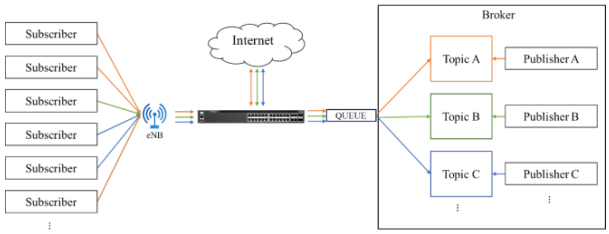

Fig. 15 Architecture of experiment 4

We tested MQTT publish/subscribe pattern. On average, each subscriber sent a message once in 10 seconds, and sent more than 100 messages per second. In addition, we set that when the device was disconnected, it started to reconnect after 60 seconds. Table 2 and Table 3 show the results of the first part and the second part, respectively.

From Table 2, it can be seen that when 5000~6000 MTC devices are connected, some devices start to be disconnected and then reconnect. In Table 3, when there are more than 4000 devices, some devices start to disconnect and have packet loss. When more devices reconnect, the packet loss rate also increases significantly.

Table 2. Experimental result of the limit of the number of connected devices of mMTC slice in the proposed architecture

Number of mMTC Devices	Connection Quality
3000	stable
4000	stable
5000	stable
6000	unstable

Table 2. Experimental result of packet lost rate of mMTC slice when many MTC devices process unlink transmission at the same time

Number of mMTC Devices	Connection Quality	Packet Loss Rate
2000	stable	0%
3000	stable	0%
4000	unstable	3.56%
5000	unstable	18.75%

Experiment 5

The scenario of experiment 5 is that there are three slices in the system, two of which are in normal service, and the third slice suffers from UDP DDoS. The experiment checks whether the throughput of these two normally operating slices were also affected by DDoS. The architecture of experiment 5 is shown in Fig. 16.

First of all, we set the upper and lower limits of the throughput of three slices. The upper throughput limit of mMTC slice is 30Mb/s, and the lower limit is 10Mb/s. The upper throughput limit of eMBB slice is 60Mb/s, and the lower limit is 30Mb/s. The upper throughput limit of NS slice is 5Mb/s, and the lower limit is 1Mb/s. We then set the priority of the three slices to be NS>eMBB>mMTC. Note that the NS slice is created to test and compare other slices and its internal VNF does not contain EPC core.

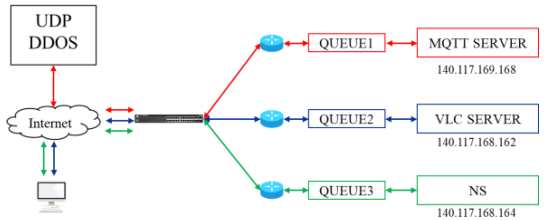

Fig. 16 Architecture of experiment 5

In this experiment, three slices mMTC, eMBB and NS were activated first and the traffic of these three slices is generated using iperf. Then we used open source software tfn2K to inject UDP DDoS traffic flow into the mMTC slice from the third second. Fig. 17 shows the result of experiment 5. From Fig. 17, it can be seen that when the UDP DDoS attack is launched, the mMTC slice cannot be connected intermittently after the fourth second, but it has no effect on the other two slices. eMBB and NS slices can still provide normal services. Due to the scale problem, it is not easy to see the throughput variation from Fig. 17, so we present the throughput variation of mMTC slice separately in Fig. 18. As can be seen from Fig. 18, UDP DDoS makes the connection of the mMTC slice extremely unstable after the fourth second. This intermittent connection causes the throughput of the mMTC slice to oscillate between approximately 1 Mbps and 0 Mbps. At this time mMTC slice is difficult to serve normally.

Noted that the guarantee of Queue in this experiment can only be achieved based on the fact that the switch can properly handle traffic packets. If the switch shared by the three slices cannot work because of a larger-scale DDoS, then the guarantee of all slices will not be able to achieve.

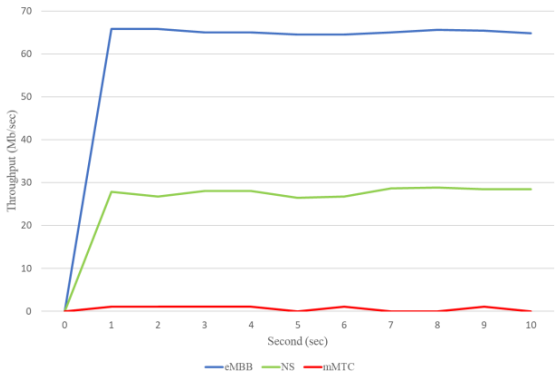

Fig. 17 Experimental result of throughput guarantee under UDP DDoS

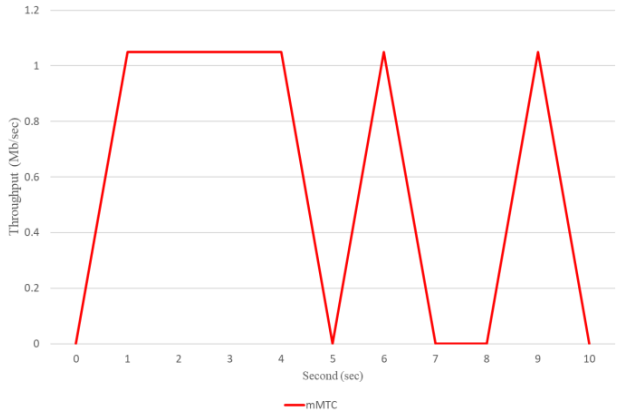

Fig. 18 Throughput variation of mMTC slice under UDP DDoS

V. Conclusions

This research integrates various open source softwares, SDN controller and cloud platform to achieve network slicing. The proposed implementation mechanism guarantees the priority of use of each service, and uses VXLAN to cut the network segment, so as to protect the independence of each service. In terms of service deployment, Tacker can also be used to quickly set up the required VNFs to efficiently create different services. Therefore, it is very convenient to set up different slices for difference service needs in the proposed implementation mechanism. In the future research, we hope to ensure the service resources on the user side and achieve better network utilization and also add appropriate scheduling scheme to adaptively improve resource utilization.

Acknowledgement

This research was supported in part by the Ministry of Science and Technology of Taiwan under contract No. MOST 107-2221-E-036-MY3.

References

[1] 3GPP, "Study on scenarios and requirements for next generation access technologies," *3GPP TR 38.913,* version 14.2.0, release 14, May 2017.

[2] X. Li, R. Casellas, G. Landi, A. de la Oliva, X. Costa-Perez, A. García-Saavedra, D. Thomas, C. Luca and R. Vilalta, "5G-crosshaul network slicing enabling multi-tenancy in mobile transport networks," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 128-137, 2017.

[3] M. Yuki, T. Atsushi, K. Taichi, S. Norio and S. Katsuhiro, "An architecture and implementation of automatic network slicing for microservices," *2018 IEEE/IFIP Network Operations and Management Symposium (NOMS 2018)*, pp. 1-4, Apr 2018.

[4] C. Salvatore, F. Ilhem, A. Nadjib and L. Rami, "DEMO: SDN-based network slicing in C-RAN," *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC 2018)*, pp. 1-2, Jan 2018.

[5] J. Kwak, J. Moon, H. W. Lee and L. B. Le, "Dynamic network slicing and resource allocation for heterogeneous wireless services," *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC 2017)*, pp. 1-5, Oct 2017.

[6] S. Vassilaras, L. Gkatzikis, N. Liakopoulos, I. N. Stiakogiannakis, M. Qi, L. Shi, L. Liu, M. Debbah, and G. S. Paschos, "The algorithmic aspects of network slicing," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 112-119, 2017.

[7] J. Ordonez-Lucena, O. Adamuz-Hinojosa, P. Ameigeiras, P. Munoz, Juan J. Ramos-Munoz, J. F. Chavarria and D. Lopez, "The creation phase in network slicing: from a service order to an operative network slice," *2018 European Conference on Networks and Communications (EuCNC 2018)*, pp. 1-6, Jul 2018.

[8] C. Song, M. Zhang, Y. Zhan, D. Wang and L. Guan, "Hierarchical edge cloud enabling network slicing for 5G optical fronthaul," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 11, no. 4, pp. B60 - B70, 2019.

[9] W. Guan, X. Wen, L. Wang, Z. Lu and Y. Shen, "Service-oriented deployment policy of end-to-end network slicing based on complex network theory," *IEEE Access*, vol. 6, pp. 19691-19701, 2018.

[10] A. Farrel, "Recent developments in Service Function Chaining (SFC) and network slicing in backhaul and metro networks in support of 5G," *2018 20th International Conference on Transparent Optical Networks (ICTON 2018)*, pp. 1-4, Jul 2018.

[11] A. A. Barakabitze, A. Ahmad, R. Mijumbi and A. Hines, "5G network slicing using SDN and NFV: A survey of taxonomy, architectures and future challenges," *Computer Communications*, vol. 167, pp. 1- 40, 202.

[12] M. Afaq, J. Iqbal, T. Ahmed, I. U. Islam, M. Khan and M. S. Khan, "Towards 5G network slicing for vehicular ad-hoc networks: An end-to-end approach," *Computer Communications*, vol. 149, pp. 252-258, 2020.

[13] H. T. Chien, Y.R. Lin, C. L. Lai and C. T. Wang, "End-to-end slicing as a service with computing and communication resource allocation for multi-tenant 5G systems," *IEEE Wireless Communications*, vol. 26, no. 15, pp. 104-112, 2019.