Scalable Network Slicing Architecture for 5G

Tulja Vamshi Kiran Buyakar, Amogh PC, Bheemarjuna Reddy Tamma, and Antony Franklin A
Department of Computer Science and Engineering, IIT Hyderabad, India
Email: [cs16mtech11020, cs15mtech01002, tbr, and antony.franklin]@iith.ac.in

ABSTRACT

The diversified use cases of next-generation mobile networks can be realized by the key concept of Network Slicing that enables mobile network operators to slice a single physical network into multiple virtual network instances optimized to specific services and business goals. Scaling of network slices is required to cope with the resources needed for peak traffic demand. In this paper, we demonstrate scaling of network slices based on the type of network slice such as enhanced Mobile Broadband (eMBB), massive Machine Type Communication (mMTC) in order to ensure Service Level Agreement (SLA) guarantees of the network slices with the help of our proposed Network Slicing Profiler (NSP) and Network Slice Scaling Function (NSSF) in an ETSI MANO based network slicing framework.

1. INTRODUCTION

Based on the requirements of diversified use cases of 5G, the International Telecommunication Union (ITU), classified the use cases into three broad families, namely enhanced Mobile Broadband (eMBB), massive Machine Type Communication (mMTC), and ultra-Reliable Low-Latency Communications (uRLLC). eMBB aims to focus on services that require high bandwidth and sustained high capacity network connections, such as High Definition videos, Augmented Reality, etc. The uRLLC services are required for applications like Factory Automation, Intelligent Transportation Systems, etc., which have latency constraints and need high reliability and availability. mMTC focuses on services that include high demands for connection density, such as smart agriculture, smart city, etc. Network Slicing is the key to 5G network architecture evolution to support diversified 5G use cases. Network Slicing allows the mobile network operator (MNO) to split a single shared physical network into multiple logical or virtual networks. As these logical networks (Network Slices) are isolated, the failure of one slice doesn't affect the other slices. Software Defined Networking (SDN) & Network Function Virtualization (NFV) capabilities combined with cloud technologies provide the necessary tools to enable Network Slicing. In [8], the authors developed an NFV-based LTE EPC [1] implementation by simulating the working of a typical EPC of LTE for handling signaling and data traffic across multiple virtual machines. In [9], the authors present a design of a flexible 5G architecture with the emphasis on techniques that provide efficient utilization of substrate resources for network slicing. In our previous work [6] we did the auto-scaling of dataplanes of network slice using the Mobility Management Entity (MME). In this work, we focus on the operational aspects of the network slicing of the 5G core network in the orchestrated environment. In our work, scaling of control plane components and Radio Access Network (RAN) are not discussed as it is outside the scope of this paper.

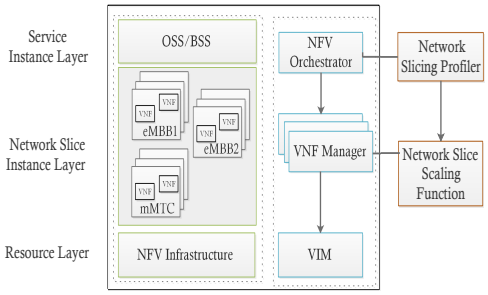

Figure 1: Proposed Scalable Network Slicing Architecture.

2. SCALABLE NETWORK SLICING ARCHITECTURE

In the proposed novel Scalable Network Slicing Architecture (SNSA), the ETSI MANO framework is extended with some additional components like Network Slicing Profiler (NSP), Network Slice Scaling Function (NSSF) as shown in Fig. 1. Adaptive management and orchestration of network slices is crucial in ensuring the performance requirements of the deployed services [7]. It should be efficient at utilizing underlying resources by making decisions based on the current state of slices as well as their predicted demands in the near future. As the network slices share the same underlying NFV Infrastructure (NFVI), there is a need to design adequate resource management mechanisms, that maintain isolation among slices and also meet the performance requirements of the slices. In order to address the above challenges, we propose a novel NSP in our architecture. NSP maintains profiles of various network slices with respect to set of avail-

Mobicom '18 New Delhi, India

able physical and virtual network resources. These profiles help in the efficient allocation of resources to network slices. A new profile can be created based on the SLA requirements of the network slice that needs to be deployed. NFV Orchestrator (NFVO) interacts with NSP to get resource allocation profile for each slice. The requested resources are then allocated to the network slice based on its profile. NSP keeps track of the resource usage of various network slices and depending on the current load, NSP contacts NFVO to dynamically update the resources to the network slice. The concept of dynamically increasing resources to a given slice on runtime is called Vertical Scaling. The operators need to define the maximum amount of resources that can be allocated to individual slices in the NFVI so that the NSP can limit the resources that are being allocated to each slice in the NFVI. The isolation is maintained by the NSP in terms of resource allocation by dedicating resources up to the maximum limit for a network slice. If the network slice requires resources more than a maximum limit specified in NSP, then NSP triggers the NSSF. NSSF involves the creation of new Virtual Network Functions (VNFs) of the slices by monitoring of various metrics such as CPU load, traffic load, bandwidth, etc. of the VNF. To ensure the SLA requirements, the scaling metrics should be based on the profile of the network slice. Triggering for scale up/down may happen upon various conditions depending on the type of the network slice. For example, if a slice is of type eMBB, to ensure minimum bandwidth to the UEs, it has to scale based on the bandwidth consumption of the slice. NSSF runs the slice specific scaling algorithm. When a slice has to be scaled, it interacts with VNFM to perform scaling of slice's VNFs. As the resource allocation by NSSF requires the creation of new VNFs, it incurs an overhead of booting and setup. So appropriate thresholds have to be set to keep the service continuity. NSSF maintains the resource pool of various hosts in the underlying physical network and allocates them to network slices on demand.

3. IMPLEMENTATION FRAMEWORK

In this section, the SNSA is realized with the help of open source tools that provide a wide range of open development models to large operators and enterprises. Various platforms that are used to realize the SNSA are OpenStack [3], Open Baton [2], Zabbix [5] and RabbitMQ [4]. Fig. 2 shows the implementation framework regarding ETSI NFV architecture. OpenStack is an open source cloud virtualization platform which offers the ability to design the NFV system to build large and scalable services. Open Baton is an open-source implementation based on ETSI NFV MANO architecture which provides modular and extensible architecture realized by RabbitMQ. The significant components of Open Baton are NFV Orchestrator (NFVO), Generic VNF Manager (VNFM), Auto Scaling Engine (ASE). Communication among these components happens via RabbitMQ. Open Baton provides the OpenStack plugin mechanism to communicate with cloud environments of OpenStack. Open Baton integrates with a Zabbix monitoring system via the Zabbix monitoring plugin. A Network Service Descriptor (NSD) file is created in JSON format which contains the network slicing setup. Open Baton NFVO uses the NSD to launch the setup. This is done either via dashboard in a web browser or via command line interface. Once the NSD is launched, the VNFs are created, and links are set up among the VNFs on top of OpenStack. Open Baton NFVO interacts with the Generic VNFM to create new VNFs. The Element Management System (EMS) is responsible for the performance and fault management of VNF.

Table 1: Simulation Parameters

Parameter	Value
Number of UEs	0 to 300
Simulation Time	360 Seconds
Network Slices	[eMBB1, eMBB2, mMTC]
SliceIDs	[s1, s2, s3]
Packet Size [s1, s2, s3]	[800, 800, 100] Bytes
Min. Bandwidth per flow for [s1, s2, s3]	[5, 10, -] Mbps
$NUEs$ for [s1, s2, s3]	[10,10,-]
BW_{init} for [s1, s2, s3]	[60 Mbps, 120 Mbps, 80 Mbps]
BW_{max} for [s1, s2, s3]	[1920 Mbps, 960 Mbps, 80 Mbps]
UE Data Transfer Duration for [s1, s2, s3]	[60-180s, 30-80s, 5s]
Mean Arrival Rate (λ) for eMBB1 [0:250s]	12
Mean Arrival Rate (λ) for eMBB2 [0:150s]	12
Mean Arrival Rate (λ) for mMTC [0:50s]	4
Mean Arrival Rate (λ) for mMTC [100:150s]	15
Mean Arrival Rate (λ) for mMTC [150:200s]	8

4. EVALUATION

The experiments are performed on a Intel Xeon CPU E5-2690 server, with 64GB RAM, running Ubuntu 16.04.2 LTS OS. The objective is to show how our testbed guarantees bandwidth isolation and auto-scaling using NSP and NSSF for the network slices. We used NFV-LTE-EPC [1] for our network slicing setup. We define a network slice as a combination of Serving Gateway (S-GW) and Packet Data Network Gateway (P-GW). Three network slices of types 'eMBB' and 'mMTC' are considered in our setup. It is to be noted that, in our setup uRLLC type network slice is not demonstrated. The eMBB type slice is scaled based on the maximum bandwidth (BW_{max}) of the slice as it has to guarantee the SLA requirements to the UEs in terms of bandwidth. The BW_{max} values and SLA requirements of various slices are mentioned in Table 1. Since mMTC type slices need not provide any minimum bandwidth guarantees to UEs, it need not scale based on the bandwidth of the slice. As mMTC type slices are dependent on VNF processing, they are scaled based on the CPU load of the VNFs in the slice. The current bandwidth consumed by a slice and CPU load of the VNFs is fetched by Zabbix monitoring system. The ScaleUp and ScaleDown operations are performed by the Open Baton's ASE. We considered three network slices, two of type eMBB and one of type mMTC. There is also a default slice which does the EPS bearer setup for other types of network slices. The eMBB mMTC type of slices will only do data forwarding.

We simulate concurrent UE threads of eMBB1, eMBB2, and mMTC using RAN-Simulator of [1] with the traffic load as shown in Fig. 3. Poisson distribution is used for modeling the UE arrival rate, with the mean arrival rates mentioned in Table 1. To meet the minimum bandwidth guarantees to the UEs of eMBB slices, NSP starts provisioning with initial bandwidth (BW_{init}) for a given number of UEs ($NUEs$). NSP doubles the current bandwidth provisioned to a slice for an increase in every $NUEs$. The current number of UEs in a slice are fetched by Zabbix. Fig. 4 shows the bandwidth provisioning for various slice over time. The NSSF scales up the eMBB1 slice as shown in Fig. 6 during time t=36 sec, as the BW_{max} for eMBB1 slice is reached. Simi-

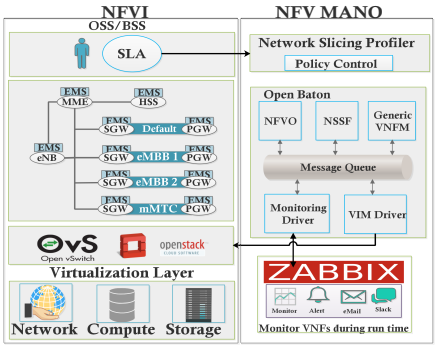

Figure 2: Network Slicing Implementation Framework.

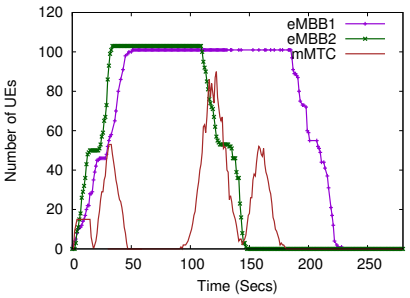

Figure 3: UE Load Distribution over Simulation Time.

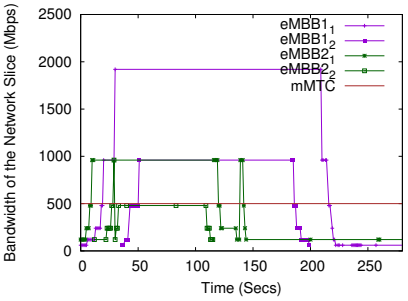

Figure 4: Bandwidth Provisioned for Slice over Time.

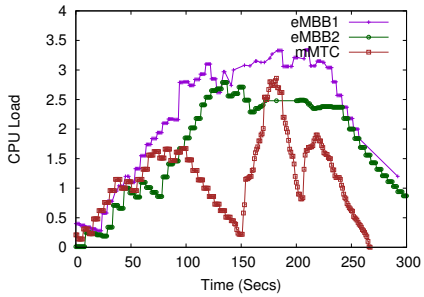

Figure 5: Average CPU Load over all the Instances with Time.

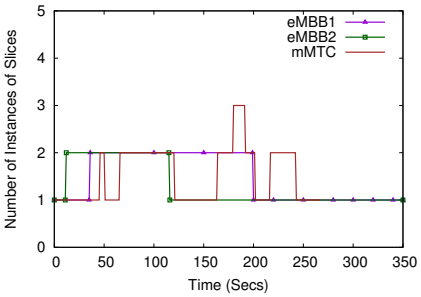

Figure 6: Number of Slice Instances (S-GW+P-GW) over Time.

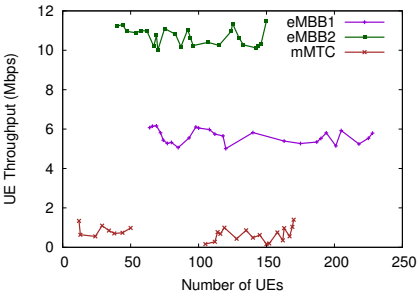

Figure 7: Average Per UE Throughput Observed over Time.

larly, NSSF scales up eMBB2 slice during time t=12 sec. As load decreases, NSSF scales down the eMBB1 slice at time t=200 sec and eMBB2 slice at t=116 sec. As the bandwidth that a slice can consume is limited, we are making sure that the bandwidth provisioned to one slice isn't affected by the other slices. In this way, the bandwidth isolation is ensured among slices. Fig. 5 shows the total CPU load of three slices over all the instances. We observe that as the number of UEs increases, the CPU load is also increased beyond the capability the VNFs can handle. So, for mMTC type slices the auto scaling metric is chosen as the CPU load. We set the auto scaling threshold of the CPU load as 1.0 which means fully loaded. By continuous monitoring the CPU load of mMTC slice, when the CPU threshold is reached, NSSF waits for cooldown period of 10 secs (for checking if the CPU threshold is consistently above 1.00) and scales up the slice as shown in Fig. 6. Fig. 7 shows an average per UE throughput of 5 Mbps for eMBB1 and 10 Mbps for eMBB2 slices that meets the minimum requirements of UEs in eMBB1 and eMBB2, respectively.

5. CONCLUSIONS AND FUTURE WORK

Next-generation mobile networks need network slicing to meet the requirements of various use cases. In this work, we proposed a novel NSP and NSSF modules on the open source technologies to realize the network slicing environment. Bandwidth isolation among slices and scaling of the slices are evaluated by considering three slices, two of type 'eMBB' and one of type 'mMTC'. It is also demonstrated that SLAs of the eMBB slices (as shown in Fig. 7) is ensured when all three slices are running. In future, we like to extend NSP and NSSF with the latency requirements of network slices considered. NSP and NSSF can also be tested with other scaling techniques based on time series analysis, control theory, reinforcement learning and queuing theory.

6. REFERENCES

[1] NFV-LTE-EPC. https://github.com/networkedsystemsIITB.
[2] OpenBaton. https://openbaton.github.io/.
[3] OpenStack. https://www.openstack.org/.
[4] RabbitMQ. https://www.rabbitmq.com.
[5] Zabbix. https://www.zabbix.com/.
[6] T. V. K. Buyakar, A. K. Rangisetti, A. A. Franklin, and B. R. Tamma. Auto scaling of data plane VNFs in 5G networks. In *Network and Service Management (CNSM), 2017 13th International Conference on*, pages 1–4. IEEE, 2017.
[7] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina. Network slicing in 5G: Survey and challenges. *IEEE Communications Magazine*, 55(5):94–100, 2017.
[8] A. Jain, N. Sadagopan, S. K. Lohani, and M. Vutukuru. A comparison of SDN and NFV for re-designing the LTE packet core. In *Proc. of IEEE NFV-SDN*, pages 74–80. IEEE, 2016.
[9] F. Z. Yousaf, M. Gramaglia, V. Friderikos, B. Gajic, D. von Hugo, B. Sayadi, V. Sciancalepore, and M. R. Crippa. Network slicing with flexible mobility and qos/qoe support for 5G networks. In *Communications Workshops (ICC Workshops), 2017 IEEE International Conference on*, pages 1195–1201. IEEE, 2017.