MEC Implementation in NFV Architecture Supporting 5G End-to-End Network Slicing

Yao-Chia Chan*, Li-Hsing Yen*, Tse-Han Wang*†, and Chien-Chao Tseng*
*Department of Computer Science, College of Computer Science, National Yang Ming Chiao Tung University, Hsinchu, Taiwan.
Email: josh0408tw@gmail.com, {lhyen, wangth, cctseng}@cs.nycu.edu.tw
†Network Management Laboratory, Chunghwa Telecom Laboratories, Taoyuan, Taiwan.

***Abstract*—Multi-access Edge Computing (MEC) utilizes computing platforms in the proximity of end users to provide cloud-based services with low latency and high reliability. The 3rd Generation Partnership Project (3GPP) introduces network slicing to provide diverse network services in 5G networks and has been working with European Telecommunications Standards Institute (ETSI) to standardize the integration of MEC with 5G to support network slicing. However, the realization of network slicing in MEC is still in early stage. All the existing studies on network slicing in MEC were based on 4G Evolved Packet Core (EPC), which do not conform to the network slicing standard developed by 3GPP. This work extends our prior work on management and orchestration (MANO) framework for 5G end-to-end (E2E) network slicing by incorporating the support for MEC. Our contribution is twofold. We report our MEC implementation in NFV architecture using all open-source software. We also make the MEC support network slicing and integrate it into an E2E networking slicing framework. The experimental results show that our MANO framework efficiently deploys k8s clusters, MEC platforms, and E2E network slices with short deployment time and low CPU and memory usage. Furthermore, the round-trip time to access services provided by MEC is much shorter than that of core networks.**

I. Introduction

The fifth-generation (5G) network aims to provide diverse network services to various applications. To this end, the 5G network requires elasticity to allow service customization and programmability, which differs from the one-size-fits-all architecture targeting mobiles only in the 4G networks. A key enabling technology to meet these requirements is *network slicing* [1]. Network slicing enables sharing a common 5G network infrastructure by more independent logical networks called *network slices* (NSs). Each NS is provided with customized network resources [2] regarding bandwidth, latency, and availability to support a specific type of network service.

Network function virtualization (NFV) is a network architecture concept that leverages virtualization technology to decouple software-based network functions (NFs) from the underlying hardware. The decoupling enables virtualized network functions (VNFs) to run on top of commercial off-the-shelf (COTS) equipment. NFV liberates network operators from proprietary equipment and rigid network architecture without sacrificing functionality. Furthermore, softwarized network functions facilitate agile deployment, structure evolution, and platform innovation. Consequently, network operators can increase flexibility and scalability while improving user experience and speeding time to market.

Multi-access edge computing (MEC) [3] provides low-latency cloud services by deploying cloud infrastructure to the proximity of mobile users. With less than 1 ms standard latency in 5G, MEC could meet strict delay constraints of most time-sensitive applications. European Telecommunications Standards Institute (ETSI) published a white paper on integrating 5G and MEC in 2018, describing the location of MEC in the 5G architecture, how to direct user traffic to MEC, etc. However, the realization of network slicing in MEC is still in its infancy. All the existing studies on network slicing in MEC [4] were based on 4G Evolved Packet Core (EPC), which are not aligned with the requirements proposed by 3rd Generation Partnership Project (3GPP).

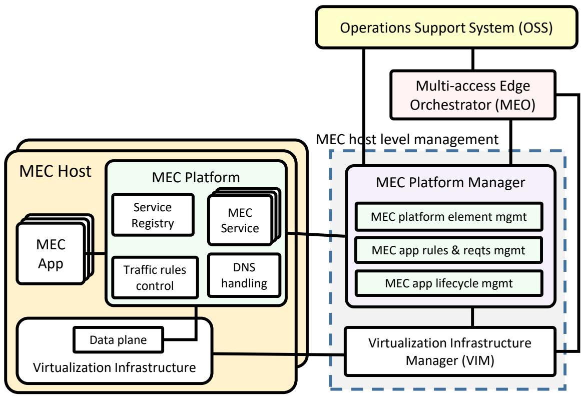
Our prior work in [5] proposed a cloud-native management and orchestration (MANO) framework for end-to-end (E2E) network slicing. The framework divides an E2E network slice into sub-slices of radio access network (RAN), transport network (TN), and core network (CN) and proposes a MANO for each sub-slice. All the sub-slice MANOs are coordinated by an E2E MANO. Our work in this paper complements the prior work by integrating a MANO for MEC into the framework. The proposed MEC MANO conforms to the recommendation of ETSI. Furthermore, our implementations of the whole framework and all associated sub-slices (including the MEC MANO) are based on open-source solutions, and can be deployed across Kubernetes (k8s) clusters.

The remainder of this paper is structured as follows. Sec. II briefs the backgrounds and related work. Sec. III presents the proposed scheme. Sec. IV presents the experimental results and the last section concludes this paper.

II. Background and Related Work

A. Multi-access Edge Computing (MEC)

MEC is a technology that deploys cloud service in the edge of network. As the service access point is to the proximity of end users, MEC provides cloud services with low latency. A MEC framework consists of system-level, host-level, and network-level entities (Fig. 1). The system-level entities are all for management and interworking purposes, which include Operations Support System (OSS) and Multi-access Edge Orchestrator (MEO). The network level entities (not shown in Fig. 1) such as access network (AN) are external to the MEC system. The host-level entities include MEC hosts and associated host-level management entities. An *MEC host* is a physical machine which hosts and supports the executions of cloud-enabled MEC applications (MEC Apps). Executing a MEC App demands several functionalities from the MEC host. First, a *virtualization infrastructure* which



Operations Support System (OSS)
Multi-access Edge Orchestrator (MEO)
MEC host level management
MEC Host
MEC Platform
Service Registry
MEC Service
MEC App
Traffic rules control
DNS handling
Data plane
Virtualization Infrastructure
MEC Platform Manager
MEC platform element mgmt
MEC app rules & reqts mgmt
MEC app lifecycle mgmt
Virtualization Infrastructure Manager (VIM)

Fig. 1: MEC framework

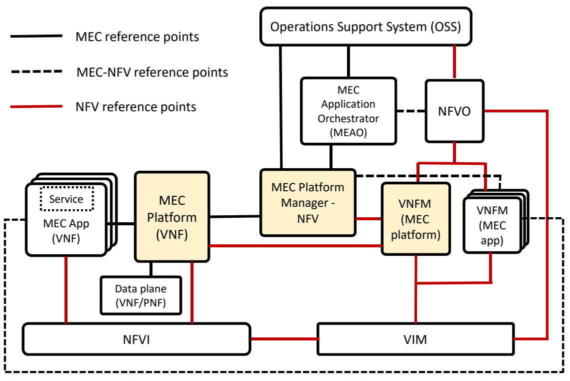


Fig. 2: MEC in NFV Architecture Variant

provides compute, storage, and network resources to MEC Apps running on top of it. Second, a set of functionalities called *MEC services* that are essential for MEC Apps to run on a particular virtualization infrastructure. An *MEC platform* hosts MEC services and offers an environment for MEC Apps in the same MEC host to discover, advertise, and consume these MEC services. Third, a *data plane* that enforces the traffic rules demanded by the MEC platform and routes the traffic among MEC Apps, MEC services, DNS server/proxy, access networks, and external networks. The MEC platform also receives traffic rule requests from the MEC platform manager, MEC Apps, or MEC services and configures and manages the data plane accordingly.

The MEC host-level management comprises MEC Platform Manager and Virtualization Infrastructure Manager (VIM). The former manages MEC-specific functionality of a particular MEC host and the MEC Apps running on it. The latter manages resources in the virtualized infrastructure.

B. MEC in NFV Architecture

ETSI proposed *MEC in NFV architecture variant* [6] (Fig. 2) as a way to softwarerize MEC. In this variant, both MEC platforms and MEC Apps are VNFs running on Network Function Virtualization Infrastructure (NFVI). The NFVI comprises both virtualized computing and networking platforms. In addition, a virtualized MEC platform also needs supports for its data plane service. The supports could come from virtualized or physical network function (PNF).

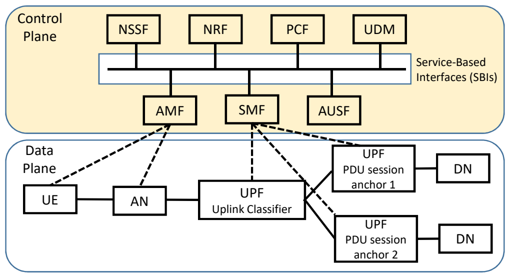


Fig. 3: 5G Core Network Architecture

The variant also differs from the original framework in the following ways.

- The MEO is replaced by MEC Application Orchestrator (MEAO) and an NFV Orchestrator (NFVO). The MEAO will send a request to the NFVO after it completes requests related to MEC platform or MEC App.
- The MEC Platform Manager becomes MEC Platform Manager-NFV, where the life-cycle management of MEC platforms, now VNF instances, has been delegated to a VNF manager (VNFM).
- One or more instances of VNFM are created to manage the life cycle of MEC Apps as VNFs.

C. 5G Core Network

Fig. 3 shows the 5G core network architecture proposed by 3GPP [2], which decomposes control-plane applications and services into service-oriented components realized as fine-grained cloud-native network functions. These components communicate with one another through service-based interfaces (SBIs). In control plane, network slices are managed by NSSF, AMF, and SMF. In data plane, user equipment (UE) can send packets to and receive packets from data network (DN) through AN and user plane function (UPF).

3GPP defines one particular type of UPF called uplink classifier (ULCL). ULCL classifies uplink user traffic by the destined DN and accordingly dispatches traffic to different PDU session anchors (PSAs). Note that all downlink traffic also flows over ULCL. Therefore, ULCL can be used for traffic steering in 5G with the existence of MEC.

III. Our Design and Implementation

Our contribution is twofold. We report our MEC implementation in NFV architecture using all open-source software. We also make the MEC support network slicing and integrate it into our E2E networking slicing framework.

A. Softwarized MEC

Fig. 4 shows the proposed MEC scheme in NFV architecture. For implementing NFVI, we used Docker and Open vSwitch (OVS) [7] for compute and network virtualizations, respectively. We used k8s [8] to control and manage Docker containers. For the management of OVS, we used Open Network Operating System (ONOS) controller.

We used Smart Edge Open together with ONOS controller to implement MEC Platform. Smart Edge Open [9] developed by Intel provides an edge application agent (EAA) for customer applications to receive notifications from producer applications. EAA enables MEC App to access MEC services.

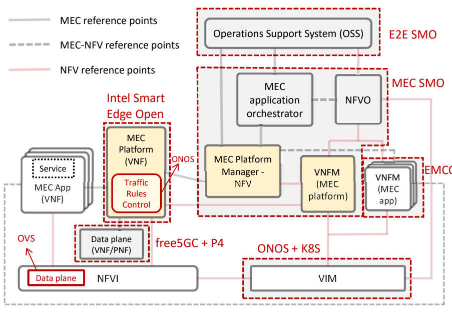


Fig. 4: Proposed design and implementation of MEC in NFV architecture

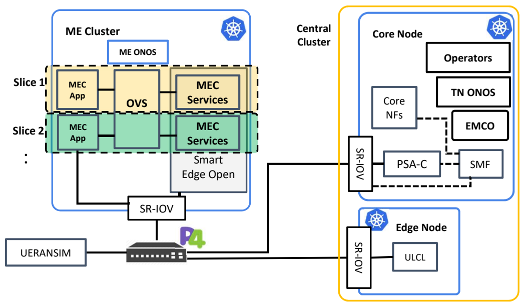


Fig. 5: Experiment Environment

ONOS controller, on the other hand, handles traffic rules. We also used free5GC together with P4 switches to implement user data plane for MEC platform. *free5GC* [10] is an open-source project for 5G core networks based on 3GPP Release 15. We took free5GC V3.0.6 as 5G core network component and modified it to support network slicing and ULCL.

We used Intel Edge Multi-Cluster Orchestrator (EMCO) [11] as the VNFM for MEC App. EMCO is able to deploy cloud-native applications to a set of k8s clusters. It provides a Cluster Registration Controller for the inclusion of k8s clusters under its management. Users can instruct EMCO to do application deployment via a command line interface.

B. MEC With Network Slicing Capability

We design a network slice subnet for MEC called MEC App slice (MEA slice). Each MEA slice consists of a MEC App, a collection of MEC services that supports the MEC App, and a configurable data plane with QoS rules for the MEC App to access the MEC services. For privacy concerns, MEA slices are isolated from one another such that a MEC App cannot access MEC services of other MEA slices.

We took an integrated design that combines MEC host level management with MANO for MEA slices. The design consists of two modules (Fig. 4). One named E2E Slice Management and Orchestrator (E2E SMO) is the OSS in NFV architecture as well as Network Slice Management Function (NSMF) in the framework proposed by 3GPP [12]. As an NS MANO, E2E SMO handles network slice requests and slice-related information and activates and deactivates E2E network slices. The other module named MEC Slice Management and Orchestrator (MEC SMO) collectively implements MEAO, NFVO, MEPM, and VNFM for MEC platform. In particular, MEC SMO performs the following tasks:

- Deployment of MEC platform, including the creation of k8s clusters
- Deployment of MEA slices, including the creation of MEC App and MEC services
- Storage of information specific to MEA slice
- Flow and request management for MEC App and MEC services

E2E SMO and MEC SMO collaboratively handle requests to deploy MEA slice, initiate the deployment, notify EMCO (as a VNFM for MEC Apps) to deploy MEC Apps across clusters, life-cycle management of MEC platform, etc.

In the framework proposed by 3GPP [12], Network Slice Subnet Management Function (NSSMF) manages the life cycle of NSSIs. We realized NSSMF by designing four specific SMOs, namely, RAN, CN, TN, and MEC SMOs, for the life cycle management of NSSIs in the RAN, CN, TN, and MEC domains, respectively. Each SMO orchestrates the corresponding NSSI when receiving management requests from the E2E SMO. In particular, the CN and MEC SMOs play the roles of NFVO/VNFM for their corresponding domains. For the implementation of MEC SMO, we used a k8s software extension called Operator pattern [13]. We used Operators to customize and automate the deployment and management of slice resources on k8s and undertake automation tasks.

IV. Experimental Results

We conducted experiments to measure the time performance of the proposed framework. We used SR-IOV [14] to allow containers to directly access the network interface card of the host, which is a way to accelerate the transmission rate. We also changed UPF in the CN slice to a ULCL with PDU Session Anchor-Core (PSA-C) mode. We physically deployed four hosts. One was used as UE plus RAN simulator (UERANSIM). Another was used as MEC cluster. The other two hosts were used as central cluster consisting of a core node and an edge node. We used k8s to manage containers for all hosts except the first one.

The experiment environment is shown in Fig. 5. The experiment environment is shown in Fig. 5. The detailed deployment procedure is as follows. We first deployed MANO framework (including four SMOs, TN ONOS, and EMCO) in the core node. We then created MEC platform in the MEC cluster, which deployed k8s cluster, Smart Edge Platform, MEC ONOS, and OVS. After that, the system was ready to deploy end-to-end network slices. The deployment of an end-to-end network slice began with the deployment of the CN slice, including the onboarding of core network components, PSA-C (in the core node), and ULCL (in the edge node to facilitate redirecting UE traffic to MEC). The deployment of MEA slice followed the deployment of the CN slice. The MANO framework then deployed MEC App and MEC services spanning across multiple clusters. It also configured OVS in the MEC cluster through MEC ONOS. After that, the MANO framework requested TN ONOS to configure P4 switches in the transport network for bandwidth management.

For the deployment of MEC applications, we used Edgecore SAU5081-2X which ran Ubuntu 16.04 on top of 36 CPU cores and 32 GB memory. We used Inventec D10056 P4 switch for data plane. The deployment of MANO framework

TABLE I: Deployment Time of MANO framework and MEC Slice

Deployment Object	Deployment Time (sec.)
MANO Framework	73.4
MEC Cluster Setup	89.6
MEA Slice Creation	10.3

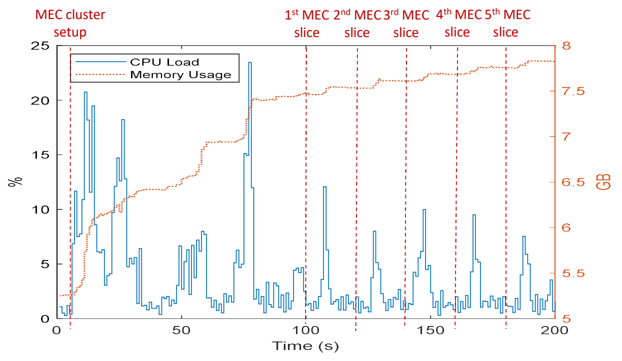


Fig. 6: CPU load and memory usage of MEC cluster setup and MEC slice deployment

includes the creation of four SMOs plus EMCO and the preparations of TN ONOS and P4 switch. The setup of MEC cluster consists of the initialization of k8s cluster and the preparations of MEC platform, including the deployments of Smart Edge Open, MEC ONOS, and OVS. The creation of a MEC sub-slice includes a MEC App and a MEC service.

A. *Deployment Time*

Table I shows the deployment time that is averaged over 10 trials. MEC cluster setup time primarily due to Smart Edge Open platform creation. Kafka deployment in k8s Pod is time-consuming. Nevertheless, an efficient MEC cluster setup in 2 minutes demonstrates the effectiveness of our work.

B. *Memory Storage & CPU Load*

Fig. 6 shows the per-second memory usage. We started the MEC cluster setup at the fifth second, since then memory usage increased as the MANO framework created the k8s cluster, OVS, and MEC ONOS, and deployed Smart Edge Open. We started the deployment of the first MEA slice at the 100th second and added a new one after every 20 seconds since then. The memory usage by MEA slices was not significant compared with the MEC cluster setup, which indicates that MEA slices are not memory demanding. The total memory usage did not exceed 3GB.

Fig. 6 also shows the per-second CPU load. The CPU load increased after the fifth second due to the setup of MEC cluster (particularly for k8s to deploy containers). It dropped after the setup completed. Then, after the 100th second and every 20 seconds after, the CPU increased for a short time for the deployment of MEA slices. After the deployment of the last slice, the CPU load returned to 1% to 3%. In short, the maximum CPU load was below 25% and the minimum CPU load (without handling any UE traffic) was below 5%. The result confirms that the impact on CPU load is light.

C. *Network Latency*

We also studied the difference of network latencies between two cases: when user traffic was handled by a MEC App and when user traffic was handled by a server in the Internet. The round-trip time from a UE to a MEC App and Internet content were measured 0.79 and 3.46 ms, respectively. The delay was 4.38 times larger if UE traffic was not served locally in the MEC. This result demonstrates the benefit of MEC in reducing network latency of user traffic.

V. Conclusions

We summarize the design and implementation of MEC in NFV architecture as follows. First, we used Docker and OVS for compute and network virtualizations, respectively, in NFVI. Second, we used k8s and ONOS controller for the VIMs of Docker and OVS, respectively. Third, we used Smart Edge Open together with ONOS controller for MEC platform. Fourth, we used free5GC together with P4 switches for user plane in MEC platform. Finally, we used Intel Edge Multi-Cluster Orchestrator (EMCO) as the VNFM for MEC App.

For slice management and orchestrator (SMO) in MEC, we design a MEC SMO which works together with E2E SMO for the deployment of a network service involving MEC sub-slice. We used a k8s software extension for the implementation of MEC SMO. We conducted experiments to measure the time performance of the proposed framework. The results show that a MEC cluster can be deployed within two minutes, the total memory usage did not exceed 3 GB, and the impact of the implementation on CPU load is light.

Acknowledgement

This work was supported in part by the National Science and Technology Council, Taiwan, under Grants 110-2221-E-A49-044-MY3, 110-2221-E-A49-064-MY3, NSTC 112-2218-E-011-004 and NSTC 112-2218-E-011-006.

References

[1] "5G; management and orchestration; concepts, use cases and requirements," 3GPP, TS 28.530, V16.2.0, Release 16, Aug. 2020.

[2] "System architecture for the 5G system (5GS)," 3GPP, TS 23.501, V16.4.0, Release 16, Mar. 2020.

[3] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1657–1681, 2017.

[4] A. Ksentini and P. A. Frangoudis, "Toward slicing-enabled multi-access edge computing in 5G," *IEEE Network*, vol. 34, pp. 99–105, Apr. 2020.

[5] Y.-S. Chiu, L.-H. Yen, T.-H. Wang, and C.-C. Tseng, "A cloud native management and orchestration framework for 5G end-to-end network slicing," in *Proc. 16th IEEE Int'l Conf. on Service-Oriented System Engineering*, CA, USA, Aug. 2022.

[6] "Multi-access Edge Computing (MEC); framework and reference architecture," ETSI, GS MEC 003, V2.2.1, Dec. 2020.

[7] "Open vSwitch," https://www.openvswitch.org/.

[8] "kubernetes: Production-grade container orchestration," https://kubernetes.io, accessed: 2021-06-14.

[9] "Smart Edge Open," https://github.com/smart-edge-open/specs, accessed: 2022-05-21.

[10] "free5gc," https://www.free5gc.org, accessed: 2021-06-14.

[11] "EMCO," https://github.com/smart-edge-open/EMCO, accessed: 2022-05-21.

[12] "Telecommunication management; study on management and orchestration of network slicing for next generation network," 3GPP, TR 28.801, V15.1.0, Release 15, Jan. 2018.

[13] "Operator framework," https://github.com/operator-framework, accessed: 2021-06-14.

[14] "SR-IOV CNI plugin," https://github.com/intel/sriov-cni, accessed: 2022-10-21.