Application-Driven End-to-End Slicing: When Wireless Network Virtualization Orchestrates with NFV-based Mobile Edge Computing

Kai Han, Shengru Li, Shaofei Tang, Huibai Huang, Sicheng Zhao, Guilu Fu
Zuqing Zhu, *Senior Member, IEEE*

***Abstract*—Recently, to adapt to the various quality-of-service (QoS) requirements of emerging applications, application-driven network slicing has attracted intensive interests. In this work, we apply the idea of software-defined wireless network virtualization (WiNV) to WiFi networks, and design and demonstrate a novel network system, namely, ADE^{2}WiNFV. The proposed system can orchestrate software-defined WiNV with network function virtualization (NFV) based mobile edge computing (MEC) to realize application-driven end-to-end (E2E) slicing over heterogeneous wireline/wireless networks. Our experimental demonstrations verify that ADE^{2}WiNFV can realize application-aware E2E slices on-demand, each of which contains not only guaranteed E2E bandwidth resources (*i.e.*, in the forms of virtual links, virtual switches and virtual access points (vAPs)) but also isolated IT resources (*i.e.*, in the form of virtual network functions (vNFs)) to carry specific applications with QoS guarantees.**

***Index Terms*—Software-defined networking (SDN), Network function virtualization (NFV), Mobile edge computing (MEC), Application-driven slicing, Heterogeneous networks.**

I. Introduction

OVER the past decades, the usage of wireless-enabled mobile devices has been growing rapidly. As a result, mobile data traffic has been increasing exponentially, and this trend will continue in the near future [1]. Meanwhile, it is known that among all the radio access technologies, the WiFi-based ones are the most widely used and have already become an indispensable part of people's lives. For instance, the Office of Communications of the United Kingdom (UK) recently reported that 81% of the mobile users in UK used WiFi frequently [2]. Despite its convenience and cost-effectiveness, WiFi usually works in the best-effort manner and thus can hardly provide any quality-of-service (QoS) guarantees on metrics such as access bandwidth, end-to-end latency, and service availability. However, more and more emerging applications would require such QoS guarantees [3–5], and an end-to-end solution is highly desired. This brings new challenges not only to WiFi but also to the wireline network that interconnects WiFi access points (APs).

The aforementioned challenges could be addressed by leveraging the concept of slice-as-a-service (SaaS), *i.e.*, application-driven end-to-end (E2E) network slicing. Specifically, the infrastructure provider (InP) creates various network slices (*i.e.*, virtual networks (VNTs)), assures certain types of QoS guarantees within the slices, and leases them to tenants for offering different services to end-users [6]. In Fig. 1, we consider future 5G network as an example, where three slices are created over the same substrate network (SNT), for delivering high-throughput, low-latency and real-time, and low-rate and non-critical services to smart-phones, autonomous cars, and massive Internet-of-things (IoTs), respectively [7].

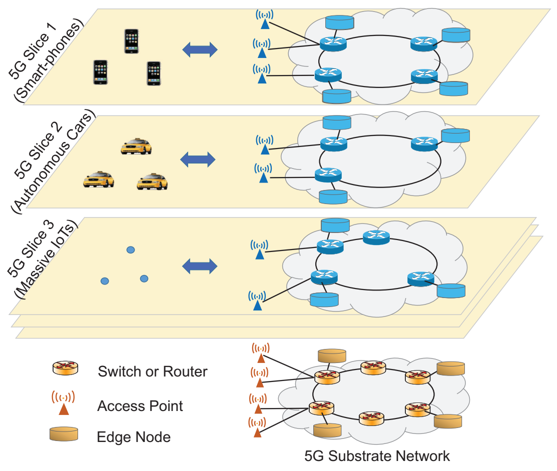

5G Slice 1 (Smart-phones)
5G Slice 2 (Autonomous Cars)
5G Slice 3 (Massive IoTs)
Switch or Router
Access Point
Edge Node
5G Substrate Network

Fig. 1. Application-driven E2E networking slicing in 5G network.

To realize SaaS, we first need an effective virtual network embedding (VNE) algorithm, which can determine how to map the virtual links and nodes in a VNT onto substrate links and nodes to satisfy certain optimization objective. Previously, the problem of VNE has already been studied intensively and numerous algorithms have been proposed to achieve various optimization objectives in different types of SNTs [8–10]. Then, from the perspective of system implementation, software-defined networking (SDN) and network virtualization are considered as the key enabling technologies for SaaS. SDN decouples network control and management (NC&M) from data forwarding and utilizes centralized NC&M to facilitate customized and QoS-aware routing and switching [11–14]. While network virtualization can realize VNTs over a shared SNT by isolating network resources to provide customized network environments. Therefore, by combining SDN with network virtualization, an InP can conveniently build a VNT according to the QoS requirements of a tenant for it to deploy

K. Han, S. Li, S. Tang, H. Huang, S. Zhao, G. Fu and Z. Zhu are with the School of Information Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, P. R. China (email: zqzhu@ieee.org).

Manuscript received April 1, 2018.

emerging applications, *i.e.*, realizing application-driven network slicing. Several existing network systems have been designed for such purpose, *e.g.*, OpenVirtex [15], CoVisor [16], SR-PVX [17] and DPVisor [18]. Nevertheless, the application-driven network slicing realized by these systems only covers the wireline segment of a network, while the access bandwidth in the WiFi segment can hardly be isolated properly, which means that the solution is not "E2E". This is because WiFi uses shared medium for data transmission without any service differentiation, and thus it would be difficult to apply SDN polices to it for QoS guarantees. Moreover, it is known that to ensure QoS for certain emerging applications, the InP not only needs the support from the network side but also has to leverage the network function virtualization (NFV) based on IT resource virtualization [19, 20].

In order to achieve an E2E solution, people have recently introduced the concept of wireless network virtualization (WiNV) [21], which can partition the radio resource in a wireless access network into logical slices according to application demands and isolate these partitions properly for QoS guarantees. Meanwhile, mobile edge computing (MEC) has attracted intensive interests to provide computing capabilities for mobile users at the edge of networks for simplifying core network operation and reducing E2E latency [20]. In this work, we apply the idea of WiNV to WiFi networks and extend our preliminary work in [22] to design and demonstrate a novel network system, namely, ADE^{2}WiNFV. The proposed system can orchestrate software-defined network virtualization with NFV-based MEC to realize application-driven E2E slicing over heterogeneous wireline/wireless networks. Our contributions can be summarized as follows:

- We design and implement a highly-programmable wireless SDN switch, which supports protocol-independent packet processing on the wireline side and can realize radio resource virtualization (*i.e.*, creating/removing virtual APs dynamically and allocating access bandwidth to them) on the WiFi side.
- We realize a MEC system to improve the performance of application-driven E2E slicing by leveraging the container-based NFV, *i.e.*, the virtual network functions (vNFs) are packaged as docker images that can be instantiated on commodity Linux servers on-demand with very short setup latency (*i.e.*, within a second).
- We integrate wireline/wireless SDN switches and the MEC system as ADE^{2}WiNFV, to provide a flexible virtualization layer for tenants to request for application-aware slices and implement applications with QoS guarantees.

The rest of the paper is organized as follows. Section II discusses the related work. We describe the system architecture of ADE^{2}WiNFV and introduce its design in details in Section III. The experimental demonstrations are presented in Section IV. Finally, Section V summarizes the paper.

II. Related Work

A. Software-Defined Wireless Networks

As the best-known implementation of SDN, OpenFlow (OF) [23] specifies the southbound protocol for a logically-centralized controller to manage the forwarding tables in SDN switches remotely. However, OF does not provide effective support to WiFi, since even in its latest version (*i.e.*, OF v1.5), the match fields do not address IEEE 802.11 frames. To address this issue, the authors of [24] extended the match fields in OF to consider WiFi frames and developed open APIs to enable the NC&M of a home WiFi network with an SDN controller. However, the work only tried to improve the radio channel utilization and total throughput the WiFi network, but did not consider how to provide various QoS guarantees to different services with WiNV. In [25], Lee *et al.* designed a system called meSDN to realize WiNV and the management of WiFi uplink for QoS guarantees. However, the compatibility of meSDN is somewhat limited since it needs to apply modifications on mobile clients. By creating light virtual APs for mobile clients, the studies in [26, 27] tried to improve the performance of mobility management in WiFi networks and achieved reduced handover latency. Nevertheless, they did not address the problem of service provisioning with various QoS guarantees in WiFi networks. Moreover, these studies only worked on the network side but did not try to leverage NFV based on IT resource virtualization.

Another drawback of the OF-based approaches mentioned above is that their data planes are protocol-dependent, *i.e.*, the match fields are all defined based on existing network protocols. Hence, when the need of supporting new protocols appears, we have to extend OF to include more match fields, which would make OF more and more complicated. To achieve a future-proof and protocol-independent data plane, people have proposed both P4 [28] and protocol-oblivious forwarding (POF) [14]. POF refers to a packet field as a tuple <*offset, length*>, where *offset* represents the start location of the field in a packet and *length* indicates its length in bits [29]. Therefore, POF switches can utilize the tuple <*offset, length*> to locate any data in a packet, and then process it with the protocol-oblivious forwarding instruction set (POF-FIS) [14, 30] for packet parsing and forwarding. Previously, we have designed and implemented several network elements, subsystems and systems [31–33] to build the POF-enabled network environment. These investigations suggest that the protocol-independent nature of POF can fit into the requirement of WiNV perfectly, since there is no need to define new match fields for IEEE 802.11 frames. Therefore, in this work, we expand our POF-based mobility management system developed in [22], and add WiNV with radio resource isolation and NFV-based MEC into the big picture for realizing ADE^{2}WiNFV.

B. Network Function Virtualization

Traditionally, service providers rely on special-purpose middleboxes to realize network functions. This scheme, however, has the drawbacks of low cost-effectiveness, long time-to-market, difficult to maintain, *etc*. Hence, NFV was proposed to decouple network functions from dedicated hardware and realize network functions with software-defined elements by leveraging IT resource virtualization [34]. Specifically, NFV enables service providers to instantiate vNFs on-demand on general-purpose hardware for customized traffic processing

[35, 36]. An intuitive way to instantiate vNFs is to deploy normal virtual machines (VMs) in a cloud system [37]. Nevertheless, such VMs usually consume relatively large IT resources and might have difficulty to fit into the resource budget of a lightweight MEC system. More importantly, since both creating and migrating a VM cause a long latency (*i.e.*, in the order of tens of seconds or even minutes), the VM-based schemes could hardly support mobile clients. On the other hand, ClickOS [38] has been considered as a lightweight and fast-booting platform for vNF prototypes. Specifically, by using a high-performance I/O library as Netmap [39], people can create lightweight VMs with ClickOS quickly and move packets among them with a relatively high throughput.

Container-based platforms such as docker have recently attracted intensive interests and been considered as a promising solution for vNF prototyping [20]. This is because compared with VMs, containers consume much less IT resources and can be started within a second. Specifically, containers do not need to include their own operating systems (OS'), but rely on the name-space and resource isolation provided by their hosts' OS kernels to encapsulate the vNFs [40]. In other words, with containers, vNFs are essentially defined as configuration files, which can be easily copied, modified and transferred. In this work, we leverage container-based NFV to package each vNF as a docker image for fast deployment and mobility support.

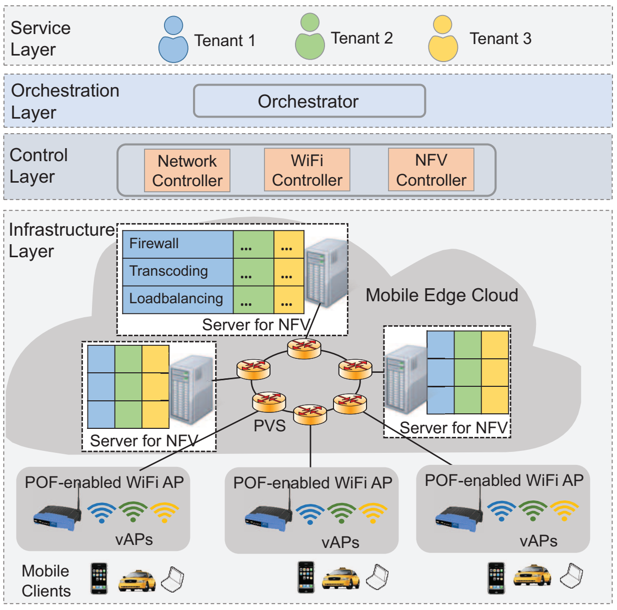


Fig. 2. Architecture of our proposed ADE^{2}WiNFV system.

C. Mobile Edge Computing

Recently, MEC has been considered to provide mobile and cloud computing capabilities in access networks to simplify core network operation and reduce E2E latency [20]. Since service providers need to customize computing capabilities for different mobile clients and services to adapt to their demands, IT resource virtualization becomes a key enabling technology for MEC to mitigate the negative effects due to the heterogeneity on hardware, feature and platform [41]. Hence, NFV-based MEC would be a promising solution to increase the flexibility of service deployment and improve resource utilization at the edge of networks. For instance, in [42], the authors leveraged NFV-based MEC to enhance the efficiency of content delivery. However, the work only addressed how to increase the efficiency and performance of network services with NFV-based MEC, but did not consider the resource allocation and isolation among different services, *i.e.*, the orchestration of WiNV and NFV-based MEC. In this work, we design and implement ADE^{2}WiNFV, which can orchestrate software-defined WiNV with NFV-based MEC to realize application-driven E2E slicing over heterogeneous wireline/wireless networks. This, to the best of our knowledge, has not been explored before in the literature.

III. System Design and Implementation

In this section, we first provide a high-level overview on the proposed ADE^{2}WiNFV system, and then explain the designs of the four layers in the system in details.

A. Architectural Overview of ADE^{2}WiNFV

With ADE^{2}WiNFV, we aim to integrate software-defined WiNV and NFV-based MEC to achieve application-driven E2E slicing over heterogeneous wireline/wireless networks. Here, an application-driven E2E slice refers to a virtual network (VNT) that contains not only guaranteed E2E bandwidth resources (*i.e.*, in the forms of virtual links, virtual switches and virtual APs (vAPs)) but also isolated IT resources (*i.e.*, in the form of vNFs) to carry specific applications with QoS guarantees (*e.g.*, high throughput for a Big Data related application, and low latency for a delay-sensitive application). Fig. 2 shows the architecture of ADE^{2}WiNFV, where there are four layers in it as follows.

Infrastructure Layer: This layer is the substrate infrastructure that consists of a few edge network devices (*e.g.*, WiFi APs and mobile clients) and a mobile edge cloud. The APs are homemade and POF-enabled, and can provide data plane programmability like the wireline POF switches [29, 33]. Moreover, we program each AP to enable WiNV, *i.e.*, being able to create and host vAPs with isolated virtual interfaces and access bandwidth. In other words, each vAP works as an independent AP dedicated to an application-driven E2E slice, and the traffic to/from it is processed with POF-FIS to ensure customized forwarding for the mobile clients in the slice. The mobile edge cloud covers the wireline part of the substrate infrastructure, which includes high-throughput homemade POF switches (*i.e.*, PVS [33]) for packet forwarding and high-performance Linux servers for container-based vNF deployment. Here, the vNFs for firewall, transcoding, loadbalancing, *etc.*, can be instantiated on and removed from the servers dynamically according to application demands.

Control Layer: This layer is in charge of the NC&M of the devices in the infrastructure layer, with three controllers, *i.e.*, the network controller, the WiFi controller, and the NFV controller. The network controller manages the wireline part of the network in the substrate infrastructure, which includes the PVS' and the Ethernet side of the POF-enabled APs, to provide customized packet processing and forwarding for each E2E slice. The WiFi side of the POF-enabled APs is managed by the WiFi controller, which is responsible for configuring system parameters related to wireless channel, SSID, BSSID, client authentication, *etc*. Meanwhile, to accomplish WiNV, the WiFi controller also controls the creation and removal of vAPs and associated access bandwidth allocation and isolation. The NFV controller handles the vNF deployment for each E2E slice, with the IT resources in the mobile edge cloud.

Orchestration Layer: This layer interacts with the control layer to gather the real-time information regarding the substrate infrastructure. Meanwhile, this layer also provides a set of application programming interfaces (APIs) to the tenants in the upper service layer, for instantiating/removing vAPs and vNFs, customizing traffic forwarding schemes, and monitoring the operation status of their E2E slices.

Service Layer: This layer is the place where tenants access ADE^{2}WiNFV to request for application-driven E2E slicing. Note that, this layer is only responsible for the high-level administrative tasks from tenants, while the actual E2E slicing is accomplished by the orchestration and control layers.

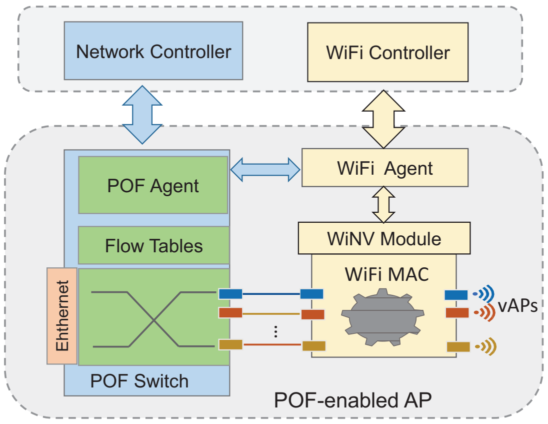


Fig. 3. Design of our POF-enabled WiFi AP.

B. Software-Defined Wireless Network Virtualization

The media access control (MAC) layer of WiFi can be logically divided into two sub-layers, *i.e.*, an upper layer and a lower layer. The upper layer handles the management frames, including the beacon, probe, authentication, and association requests and the corresponding responses, while the lower layer processes the control frames, including the acknowledgement (ACK), request-to-send (RTS) and clear-to-send (CTS) frames. Note that, as the processing of the control frames is delay-sensitive, the lower layer is usually taken care of in hardware. On the other hand, the management frames can be handled by software tools since they do not have time constraints. This actually provide us a good opportunity to design the software-defined WiNV system for ADE^{2}WiNFV. The detailed design of the POF-enabled AP is illustrated in Fig. 3. We integrate three modules in it, *i.e.*, the WiNV module, WiFi agent, and POF switch. The POF-enabled AP runs on a Linux system, and thus it can be implemented either on a commodity server with a WiFi card or a portable wireless router running OpenWrt.

1) WiNV Module: In the POF-enabled WiFi AP, the WiNV module handles the creation and removal of vAPs with isolated virtual interfaces and access bandwidth. We realize it by extending Hostapd [43], which is a user-space daemon running on a Linux system to create APs and associated authentication servers. Specifically, Hostapd can intercepts the management frames from the MAC layer of WiFi and processes them in a customized way. In order to support on-demand application-driven E2E slicing in ADE^{2}WiNFV, we expand Hostapd to support dynamic creation and removal of multiple vAPs on a physical radio interface. Also, we program an interface on Hostapd to facilitate the communication with the WiFi agent, *i.e.*, reporting association and disassociation of mobile clients to and receiving instructions from the WiFi agent.

2) WiFi Agent: We program the WiFi agent as a lightweight Python daemon to bridge the communication between the POF-enabled AP and the WiFi controller in the control layer. When a mobile client connects to a vAP in a slice, the WiNV module reports the association event to the WiFi agent, which in turn notifies the remote WiFi controller about the event. Similarly, when the mobile client leaves, the disassociation event is reported as well. In this way, the WiFi controller knows about the location and status of each mobile client in real time. To operate on the application-driven E2E slices, the WiFi agent receives instructions (*e.g.*, create/remove vAP(s), change channel, and remove client) from the WiFi controller, and executes them by invoking the APIs provided by the WiNV module. Meanwhile, the execution results are also sent back to the WiFi controller. When necessary, the WiFi agent can connect a newly-added vAP to the POF switch and activate the data transfer through it.

3) POF Switch: The POF switch is realized based on our homemade software switch PVS [33], and it bridges data traffic between the WiFi and wireline Ethernet interfaces of the POF-enabled AP, since the WiNV module only handles the management frames of WiFi MAC layer. With POF-FIS, the network controller in the control layer can install flow tables in the POF switch to realize the protocol translation between the WiFi and wireline Ethernet interfaces and achieve per-flow and per-vAP based traffic steering. For example, the POF switch can limit the access bandwidth of any vAP with the meter instruction in POF-FIS, and thus realize bandwidth allocation and isolation for the vAPs on the POF-enabled AP.

As the management frames of WiFi MAC layer are processed locally in the WiFi agent, our design of the POF-enabled WiFi AP does not fully decouple the control and data planes as that in previous work [26]. This is because sending management frames to a remote controller for processing would bring in extra latency, and thus might limit the performance of mobility management.

C. NFV-based Mobile Edge Cloud

ADE^{2}WiNFV utilizes an NFV-based mobile edge cloud to bring the computing and storage capabilities close to mobile clients for improving the QoS of the applications running in E2E slices. Fig. 4 shows the design of an MEC node in the mobile edge cloud, which consists of docker containers, an MEC agent, and a POF switch.

1) Docker Containers: They run on a Linux system and share the OS' kernel, and thus they can be started instantly and only consume a small amount of IT resources. Hence, we use docker containers as our vNF deployment platform, and package vNFs as docker images. This implementation provides us the flexibility of either realizing vNFs with commonly-used software tools (*e.g.*, *iptables* for firewall, and *haproxy* for load-balancing) or programming our own vNFs with the assistance of the Intel data plane development kit (DPDK) [44]. For the former case, the vNFs connect to the POF switch through virtual Ethernet ports. For the latter case, the vNFs communicate with the POF switch through the vhost-user driver of DPDK for high-performance I/O. As an MEC node can host multiple vNFs belonging to different E2E slices, we use a two-Byte ***tenant_id*** to identify a slice and another two-Byte ***instance_id*** to index a vNF within the slice. The ***tenant_id*** and ***instance_id*** can be combined as a four-Byte ***vnf_id***, which is the unique ID of a vNF. In order to steer traffic through the vNFs, we reuse the ***vnf_id*** of each vNF as the IP address of its input interface.

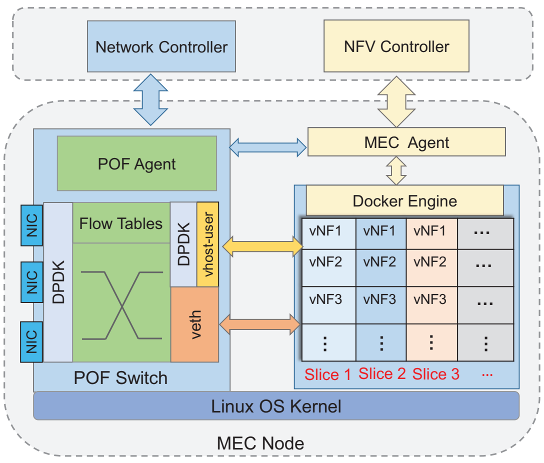


Fig. 4. Design of an MEC node in mobile edge cloud.

2) MEC Agent: It is a python daemon, which is in charge of starting, maintaining and stoping vNFs. When a tenant sitting in the service layer requests for a new vNF, the orchestration layer works with the NFV controller to find the most suitable MEC node for vNF deployment, and then they instruct the MEC agent in the MEC node to instantiate the vNF. Next, the MEC agent selects the right image for the vNF and uses it to start a container, and the processing pipeline of the vNF will be activated when the container finishes booting. After that, the MEC agent creates the I/O interfaces for the container and connects them to the POF switch. Meanwhile, the MEC agent assigns ***tenant_id*** and ***instance_id*** to the vNF, combines them to obtain the IP address (*i.e.*, the ***vnf_id***) of the vNF's input interface, and records the vNF's information in its database. When a vNF needs to be removed, the MEC agent looks up its database to find the ***container_id*** of the vNF, deletes the corresponding container, and then removes the vNF's I/O interfaces that are connected to the POF switch. Note that, the MEC agent also needs to report to the NFV controller periodically about the working status and resource usages of the vNFs on its MEC node.

3) POF Switch: The POF switch in Fig. 4 is responsible for routing traffic externally *(i.e.*, to/from other MEC nodes) and steering traffic internally among the local vNFs. To process traffic to/from other MEC nodes, the POF switch uses DPDK on the physical linecards (NICs) to achieve high-throughput packet forwarding, while for internal traffic routing, it uses the virtual Ethernet ports or the vhost-user driver of DPDK.

D. Controllers and Orchestrator

As shown in Fig. 2, there are three controllers, *i.e.*, the network controller, WiFi controller, NFV controller, and an orchestrator in the control and orchestration layers of ADE^{2}WiNFV. We implement all of them based on the ONOS platform [45]. By leveraging our previous work in [17, 18], we extend ONOS to support POF and use it as the network controller, while the WiFi controller, NFV controller and orchestrator are programmed as applications in the extended ONOS. The orchestrator can coordinate the three controllers, and provide a set of restful APIs to tenants in the service layer for application-driven E2E slicing. Table I lists the northbound APIs (*i.e.*, from tenants in the service layer to the orchestrator), with which tenants can create/remove E2E slices and start/stop vAPs and vNFs in the slices dynamically. The WiFi and NFV controllers use TCP connections to communicate with the WiFi agents and NFV agents, respectively, with the southbound APIs in Table II.

TABLE I
NORTHBOUND APIS FOR SERVICE LAYER TO CALL ORCHESTRATOR

Call Orchestrator	Descriptions
addTenant()	Create an E2E slice
getTenants()	Get a global view of all the tenants' slices
removeTenant()	Destroy an E2E slice
addVapToTenant()	Add a vAP to a slice
removeVap()	Remove a vAP from a slice
getVaps()	Get status of all the vAPs in a slice
addVnfToTenant()	Add a vNF to a slice
removeVnf()	Remove a vNF from a slice
getVnfs()	Get all the vNFs in a slice
getWifiAgents()	Get all the WiFi agents in the infrastructure layer
getMecAgents()	Get all the MEC agents in the mobile edge cloud
getClients()	Get all the mobile clients in a slice

IV. EXPERIMENTAL DEMONSTRATIONS

In this section, we discuss the experimental demonstrations to evaluate the performance of our proposed ADE^{2}WiNFV, with the setup in Fig. 5. Here, we include five Linux servers in the setup, each of which is equipped with a 2.10GHz Intel Xeon CPU and 32 GB DDR3 memory. Among these servers, three are used as MEC nodes, each of which equips with six Ethernet ports (1GbE or 10GbE), one runs ONOS to carry the service, orchestration and control layers of ADE^{2}WiNFV, and the last one equips a WiFi card to work as a POF-enabled AP

TABLE II
SOUTHBOUND APIS FOR CONTROL LAYER TO CALL INFRASTRUCTURE LAYER

Call WiFi Agent	Descriptions
createVap()	Create a vAP in a physical AP
delVap()	Remove a vAP in a physical AP
getVaps()	Get all the vAPs in a physical AP
getStatus()	Get radio information of a physical AP
changeChannel()	Change a physical AP's channel
getClient()	Get information of a mobile client
removeClient()	Disassociate a mobile client from a vAP
getTxRate()	Get sending rate of a vAP
getRxRate()	Get receiving rate of a vAP
Call MEC Agent	Descriptions
createVnf()	Create a vNF in an MEC node
delVnf()	Remove a vNF in an MEC node
getVnfs()	Get all the vNFs in an MEC node
getStatistics()	Get IT usage in an MEC node

(*i.e.*, AP2 in Fig. 5). In order to demonstrate the compatibility of our design, we implement another POF-enabled AP (*i.e.*, AP1) on a portable wireless router running OpenWrt.

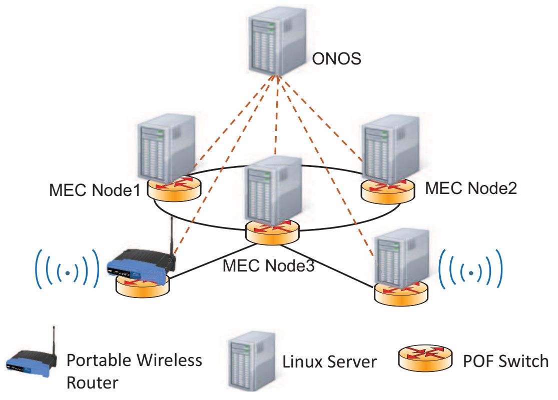


Fig. 5. Experiment setup.

A. Performance of Container-based vNFs

We first conduct experiments to measure the performance of our container-based NFV platform. Here, we build a ClickOS-based NFV platform as the benchmark, since according to [38, 39], ClickOS also provides a lightweight and fast-booting platform for vNF prototypes and can realize high-throughput traffic processing. The experiments compare the two NFV platforms in terms of traffic processing performance.

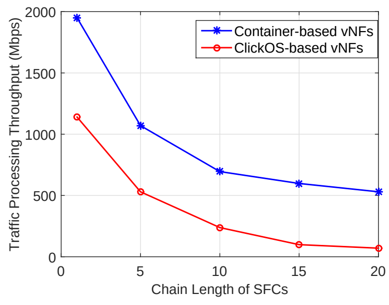
To push the platforms to their extremes, we use them to realize a simple vNF that directly forwards packets from input to output without any further processing. Then, in a single MEC node, we can concatenate such vNFs multiple times to realize a service function chain (SFC). For fair comparisons, the container-based and ClickOS-based NFV platforms run on the same Linux system with the same software/hardware configuration. We use iperf3 [46] as the traffic generator to pump traffic through the SFCs with a fixed packet size of 1500 Bytes, and the experiments measure the traffic processing throughput, traffic processing latency and memory usage of the SFCs when their chain lengths change. Fig. 6(a) shows the results on traffic processing throughput. It can be seen that our container-based NFV platform achieves a peak throughput of 1.95 Gbps when there is only one vNF in the SFC. As expected, the throughput decreases with the chain length of the SFC, but the throughput of our container-based NFV platform still maintains at 531 Mbps even when there are as many as 20 vNFs in the SFC. In contrast, the ClickOS-based one can only achieve a peak throughput of 1.14 Gbps, and its throughput quickly decreases to 70 Mbps with 20 vNFs in the SFC.

The results on average traffic processing latency are plotted in Fig. 6(b), which are obtained by using the *ping* program. The results suggest that the traffic processing latency of our container-based NFV platform is also much shorter than that of the ClickOS-based one. The comparison on memory usage is illustrated in Fig. 6(c). Since the ClickOS-based NFV platform is still based on VMs (*i.e.*, each vNF is based on a VM that consumes around 8 MBytes of memory), its memory usage increases sharply with the chain length of the SFC. On the other hand, since our docker containers share memory with the host's Linux OS, its memory usage only increases slightly with the chain length and is much less. The results in Fig. 6 confirm that our container-based NFV platform performs much better than the benchmark. This is because the container-based vNFs directly process packets in Linux kernel without extra memory copies. While in ClickOS, the vNFs are based on VMs and thus packets have to first go through the virtualization layer and then be copied from kernel space to user space, which applies a strict performance bottleneck.

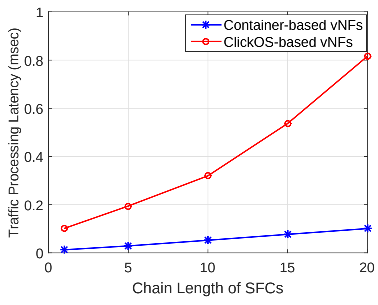
Note that, the results in Fig. 6 are obtained without applying any high-performance I/O tools for accelerating. Actually, the traffic processing throughput of both the container-based and ClickOS-based NFV platforms can be further improved with such tools. Therefore, we incorporate netmap in the ClickOS-based platform and add DPDK support in our container-based platform, to accelerate their packet processing. Then, we remeasure their packet processing throughput with one vNF in the SFC and plot the results in Fig. 7. The experiments consider different packet sizes, *i.e.*, from 64 to 1500 Bytes. We observe that our container-based platform achieves a throughput of 7.66 million packets per second (Mpps) with the smallest packet size (*i.e.*, 64 Bytes), and its data throughput reaches the line-rate of a 10GbE port when the packet size is 256 Bytes. Nevertheless, the packet processing throughput of the ClickOS-based benchmark is only 5.12 Mpps with 64-Byte packets, and it cannot reach the 10 Gbps line-rate until the packet size increases to 1024 Bytes.

B. Setup and Removal Latencies

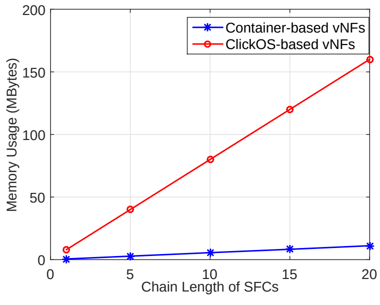
As we have explained before, the setup and removal latencies of the vAPs and vNFs would affect ADE^{2}WiNFV's performance on dynamic slicing and mobility management. Hence, we conduct experiments to measure the latencies with the setup in Fig. 5. Here, we have two physical APs, *i.e.*,



(a) Traffic processing throughput



(b) Average traffic processing latency



(c) Memory usage

Fig. 6. Performance comparisons between container-based and ClickOS-based NFV platform.

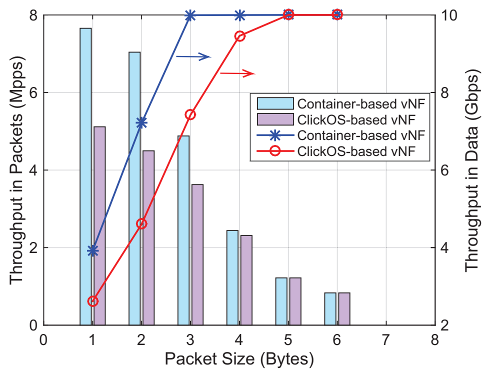


Fig. 7. Packets processing throughput with acceleration tools.

AP1 and AP2, and due to the hardware limitation on them, AP2 can support 8 vAPs while AP1 can only support 4 vAPs at most. We measure the latencies on AP2 since it has a larger capacity, and use the restful APIs on orchestrator to send instructions of creating and removing vAPs to the WiFi agent on AP2. The corresponding setup/removal latencies are defined as the time interval from when the WiFi agent receives the instructions to when the vAPs have been started/removed. Fig. 8 shows the results on the setup/removal latencies to operate on different numbers of vAPs, which indicate that on average, our ADE^{2}WiNFV only consumes around 30 and 100 msec to create and remove a vAP, respectively.

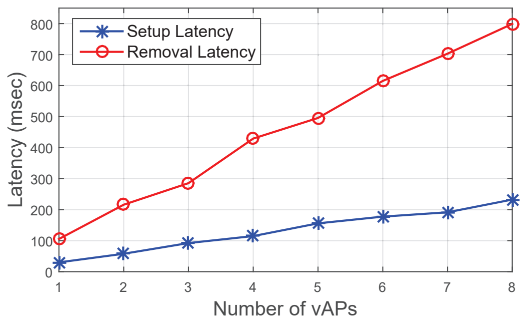


Fig. 8. Setup and removal latencies of vAPs in ADE^{2}WiNFV.

Then, the experiments also measure the setup and removal latencies of vNFs, and we still use the ClickOS-based platform as the benchmark. The definitions of the setup/removal latencies for vNFs are similar as those for vAPs. The results are shown in Fig. 9, and we can see that the ClickOS-based platform actually achieves shorter setup/removal latencies than our container-based one, despite the worse performance on traffic processing. The average setup and removal latencies per vNF of our container-based platform are 0.95 and 0.73 second, respectively, which are still less than a second and can fit into the requirement of dynamic operation well.

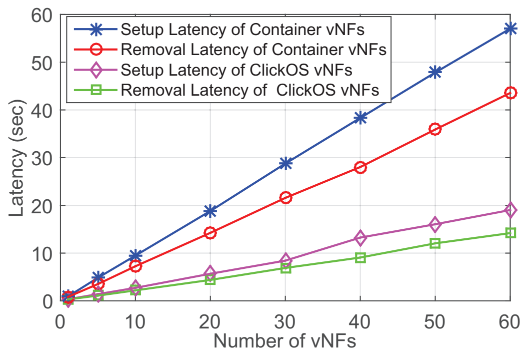
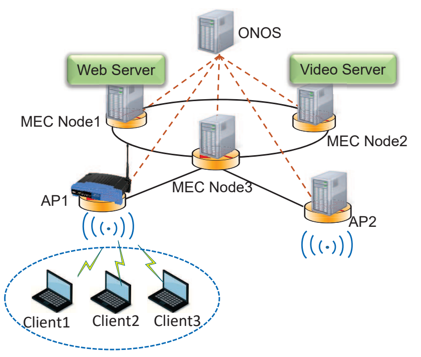


Fig. 9. Setup and removal latencies of vNFs.

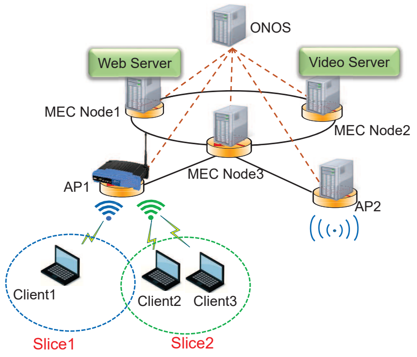
C. Application-Driven E2E Slicing Demonstrations

Since the mobile clients connect to a WiFi network share the wireless media for data transmission, it would be difficult to ensure QoS guarantees to them, especially for bandwidth-hungry applications such as video streaming. Although IEEE 802.11e has included the WiFi multimedia scheme to improve the QoS of multimedia related applications in WiFi networks, it needs to apply modifications on mobile clients and would have compatibility issues. In this subsection, we will conduct experiments to verify that ADE^{2}WiNFV can realize application-aware slices and implement applications with QoS guarantees through the orchestration of software-defined WiNV and NFV-based MEC, without requiring any modifications on mobile clients.

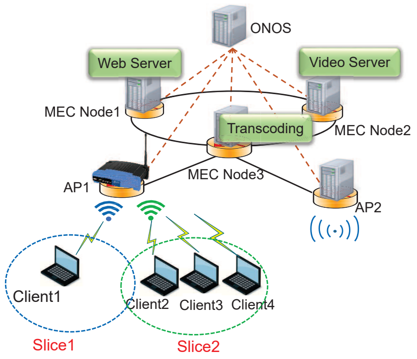
The experiments consider three scenarios, as shown in Fig. 10. In Scenario 1, we do not create any application-aware slices and just let the three mobile clients to stay in the same WiFi physical network (*i.e.*, hosted by AP1) and compete for access bandwidth with each other. As shown in Fig. 10(a), ADE^{2}WiNFV deploys a vNF for web server and a vNF for



(a) Scenario 1: without slicing



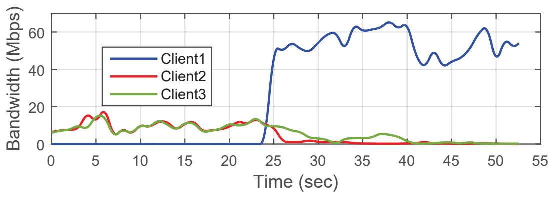
(b) Scenario 2: with application-aware slices



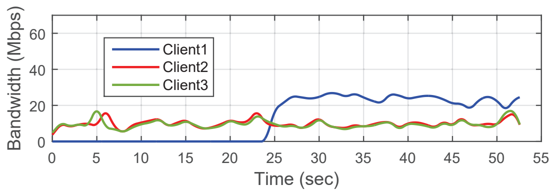
(c) Scenario 3: application-aware slices with dynamic vNF deployment

Fig. 10. Experimental scenarios to demonstrate application-driven E2E slicing.

video streaming server on MEC Node1 and MEC Node2, respectively. In Scenario 2, we create two application-aware slices with two vAPs on AP1. Here, as shown in Fig. 10(b), Slice1 is for the web application, which is delay-tolerant but only allows a small amount of packet losses, while Slice2 is for video streaming, which is a delay-sensitive application and needs guaranteed access bandwidth. Scenario 3 in Fig. 10(c) considers the dynamic joining of a mobile client in Slice2, and to adapt to the increased bandwidth requirement due to the new client, ADE^{2}WiNFV deploys a vNF for transcoding on MEC Node3 instantly for video traffic adaption.



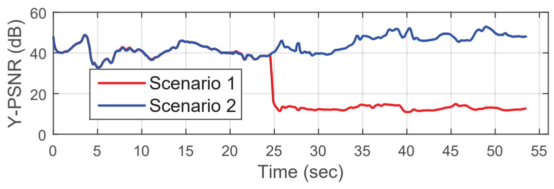
(a) Scenario 1



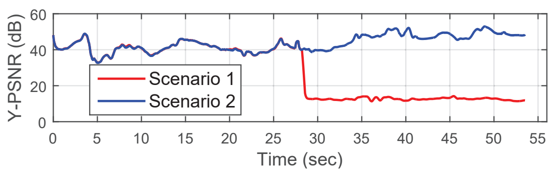
(b) Scenario 2

Fig. 11. Receiving bandwidths on clients in Scenarios 1 and 2.

In the experiments, we measure the total down-link and up-link bandwidths of AP1, and confirm that they are 60 and 30 Mbps, respectively. Then, Client2 and Client3 subscribe to the video server on MEC Node2 for 1080P video streaming services, and each of them consumes 10 Mbps access bandwidth on AP1. After the video streaming services running for 25 seconds, Client1 starts to download a large data file from the web server on MEC Node1 with 10 simultaneous threads. In Scenario 1, ADE^{2}WiNFV does not create any slices and lets the mobile clients compete for the access bandwidth on AP1 freely, while in Scenario 2, it creates two slices, assigns Client1 to Slice1 for web services, and puts Client2 and Client3 in Slice2 for video streaming services. Slice2 gets a guaranteed access bandwidth of 30 Mbps. Fig. 11 shows the receiving bandwidths of the services running on the three mobile clients in the two scenarios. As we can see in Fig. 11(a), when there is no application-driven E2E slicing, the receiving bandwidths of Client2 and Client3 can easily drop to almost zero when Client1 starts its download process at $t = 25$ second and seizes all the access bandwidth on AP1. In contrast, the results in Fig. 11(b) indicate that in Scenario 2, the application-driven E2E slicing provided by ADE^{2}WiNFV can guarantee the receiving bandwidths of Client2 and Client3 in Slice2 even when the download of Client1 starts. Specifically, the access bandwidth of Client1 is limited below 30 Mbps and thus it would not affect the video streaming services of Client2 and Client3.



(a) Y-PSNR of video playback on Client2

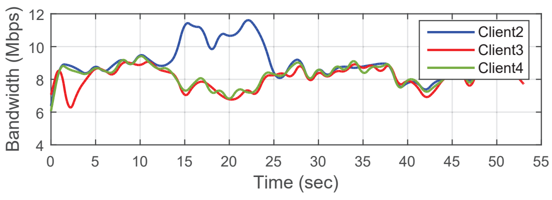


(b) Y-PSNR of video playback on Client3

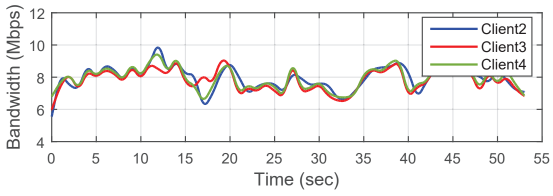
Fig. 12. Y-PSNR of video playback on Client2 and Client3.

To further verify the QoS of the video streaming services, we measure the luminance component's peak signal-to-noise ratio (Y-PSNR) of the video playback on Client2 and Client3 in Scenarios 1 and 2, and plot the results in Fig. 12. We observe that in Scenario 1, the QoS of the video playback on Client2 and Client3 decreases sharply when Client1 starts to download at $t = 25$ seconds. On the other hand, the application-driven E2E slicing in Scenario 2 maintains the Y-PSNR of the video playback at a relatively high value all the time.

In Scenario 3 in Fig. 10(c), we consider the situation in which Client4 joins in Slice2 dynamically and tries to share the access bandwidth with Client2 and Clien3. This, however, would lead to insufficient bandwidth in Slice2. To address this issue, ADE^{2}WiNFV deploys a vNF for transcoding on MEC Node3 for Slice2 on-demand, which decreases the bandwidth of each video stream to 8 Mbps. Fig. 13(a) shows the receiving bandwidths on the three clients in Slice2, when there is no vNF for transcoding. It can be seen that without the vNF for transcoding, the bandwidth variation on the clients can affect each other since the total bandwidth usage in Slice2 approaches its upper limit after Client4 joining in. For instance, from $t = 15$ to $t = 22$, there is a peak on the bandwidth of Client2, which suppresses the bandwidths of Client3 and Client4. With the vNF for transcoding, the receiving bandwidths in Fig. 13(b) do not have the issue anymore. This actually can be further verified with the Y-PSNR of the video playback on the clients in Fig. 14. Specifically, the results indicate that with the vNF for transcoding, the Y-PSNR of the video playback always stays at a relatively high value for all the clients, while the vNF for transcoding is absent, the Y-PSNR on each client can have sudden and large drops due to the bandwidth competitions among the clients. Therefore, Scenario 3 confirms that our ADE^{2}WiNFV can orchestrate software-defined WiNV and NFV-based MEC to realize traffic adaption and improve the QoS of mobile clients, especially when the WiFi access bandwidth becomes the bottleneck for service provisioning.



(a) without vNF for transcoding



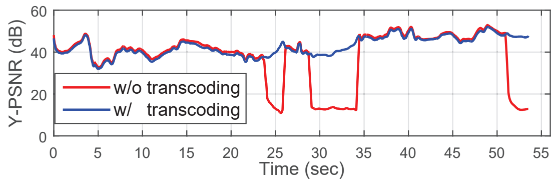
(b) with vNF for transcoding

Fig. 13. Receiving bandwidths on clients in Scenario 3.

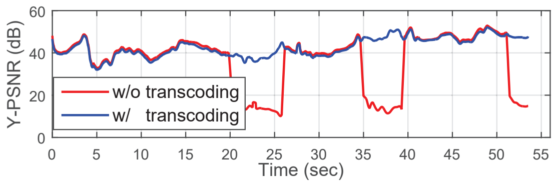
Finally, we hope to point out that same as the preliminary system in [22], our ADE^{2}WiNFV also supports NFV-assisted mobility management and can achieve relatively short handover latency. However, since this part has already been discussed intensively in [22], we omit it from this paper.

V. Conclusion

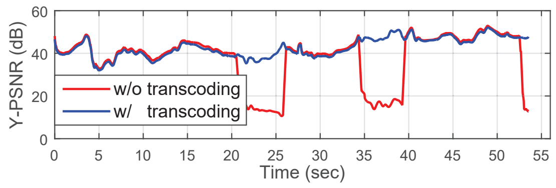
In this work, we designed and demonstrated ADE^{2}WiNFV, *i.e.*, a novel network system that can orchestrate software-defined WiNV with NFV-based MEC to realize application-driven E2E slicing over heterogeneous wireline/wireless networks. Our experimental results confirmed that ADE^{2}WiNFV can realize application-aware E2E slices on-demand, each of which contains not only guaranteed E2E bandwidth resources (*i.e.*, in the forms of virtual links, virtual switches and vAPs) but also isolated IT resources (*i.e.*, in the form of vNFs) to carry applications with QoS guarantees.



(a) Client2



(b) Client3



(c) Client4

Fig. 14. Y-PSNR of video playback on clients in Scenario 3.

Acknowledgments

This work was supported in part by the NGBWMCN Key Project under Grant No. 2017ZX03001019-004 and the Key Project of the CAS (QYZDY-SSW-JSC003).

References

[1] Cisco visual networking index: Global mobile data traffic forecast update, 2016-2021. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html
[2] Infrastructure report 2014: Ofcom's second full analysis of the UK's communications infrastructure. [Online]. Available: http://stakeholders.ofcom.org.uk/binaries/research/infrastructure/2014/infrastructure-14.pdf
[3] P. Lu, Q. Sun, K. Wu, and Z. Zhu, "Distributed online hybrid cloud management for profit-driven multimedia cloud computing," *IEEE Trans. Multimedia*, vol. 17, pp. 1297–1308, Aug. 2015.
[4] N. Xue *et al.*, "Demonstration of OpenFlow-controlled network orchestration for adaptive SVC video manycast," *IEEE Trans. Multimedia*, vol. 17, pp. 1617–1629, Sept. 2015.
[5] J. Yao, P. Lu, L. Gong, and Z. Zhu, "On fast and coordinated data backup in geo-distributed optical inter-datacenter networks," *J. Lightw. Technol.*, vol. 33, pp. 3005–3015, Jul. 2015.
[6] Z. Zhu *et al.*, "Build to tenants' requirements: On-demand application-driven vSD-EON slicing," *J. Opt. Commun. Netw.*, vol. 10, pp. A206–A215, Feb. 2018.
[7] 5G White Paper, 2015. [Online]. Available: https://www.ngmn.org/fileadmin/ngmn/content/downloads/Technical/2015/NGMN_5G_White_Paper_V1_0.pdf
[8] L. Gong and Z. Zhu, "Virtual optical network embedding (VONE) over elastic optical networks," *J. Lightw. Technol.*, vol. 32, pp. 450–460, Feb. 2014.
[9] H. Jiang, Y. Wang, L. Gong, and Z. Zhu, "Availability-aware survivable virtual network embedding (A-SVNE) in optical datacenter networks," *J. Opt. Commun. Netw.*, vol. 7, pp. 1160–1171, Dec. 2015.
[10] L. Gong, H. Jiang, Y. Wang, and Z. Zhu, "Novel location-constrained virtual network embedding (LC-VNE) algorithms towards integrated node and link mapping," *IEEE/ACM Trans. Netw.*, vol. 24, pp. 3648–3661, Dec. 2016.

[11] D. Kreutz *et al.*, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, pp. 14–76, Jan. 2015.

[12] Z. Zhu *et al.*, "Demonstration of cooperative resource allocation in an OpenFlow-controlled multidomain and multinational SD-EON testbed," *J. Lightw. Technol.*, vol. 33, pp. 1508–1514, Apr. 2015.

[13] X. Chen, S. Zhu, L. Jiang, and Z. Zhu, "On spectrum efficient failure-independent path protection p-cycle design in elastic optical networks," *J. Lightw. Technol.*, vol. 33, pp. 3719–3729, Sept. 2015.

[14] S. Li *et al.*, "Protocol oblivious forwarding (POF): Software-defined networking with enhanced programmability," *IEEE Netw.*, vol. 31, pp. 58–66, Mar. 2017.

[15] A. Al-Shabibi *et al.*, "OpenVirteX: A network hypervisor," in *Proc. of ONS 2014*, pp. 25–30, Aug. 2014.

[16] X. Jin, J. Gossels, J. Rexford, and D. Walker, "CoVisor: A compositional hypervisor for software-defined networks," in *Proc. of NSDI 2015*, pp. 87–101, May 2015.

[17] S. Li *et al.*, "SR-PVX: A source routing based network virtualization hypervisor to enable POF-FIS programmability in vSDNs," *IEEE Access*, vol. 5, pp. 7659–7666, 2017.

[18] H. Huang *et al.*, "Realizing highly-available, scalable and protocol-independent vSDN slicing with a distributed network hypervisor system," *IEEE Access*, vol. 6, pp. 13 513–13 522, 2018.

[19] J. Liu *et al.*, "On dynamic service function chain deployment and readjustment," *IEEE Trans. Netw. Serv. Manag.*, vol. 14, pp. 543–553, Sept. 2017.

[20] R. Cziva and D. Pezaros, "Container network functions: Bringing NFV to the network edge," *IEEE Commun. Mag.*, vol. 55, pp. 24–31, Jun. 2017.

[21] M. Richart, J. Baliosian, J. Serrat, and J. Gorricho, "Resource slicing in virtual wireless networks: A survey," *IEEE Trans. Netw. Serv. Manag.*, vol. 13, pp. 462–476, Sept. 2016.

[22] K. Han *et al.*, "Leveraging protocol-oblivious forwarding (POF) to realize NFV-assisted mobility management," in *Proc. of GLOBECOM 2017*, pp. 1–6, Dec. 2017.

[23] OpenFlow Switch Specifications. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf

[24] A. Patro and S. Banerjee, "COAP: a software-defined approach for home WLAN management through an open API," in *Proc. of MobiArch 2014*, pp. 31–36, Sept. 2014.

[25] J. Lee *et al.*, "meSDN: Mobile extension of SDN," in *Proc. of MCS 2014*, pp. 7–14, Jun. 2014.

[26] L. Suresh *et al.*, "Towards programmable enterprise WLANS with Odin," in *Proc. of HotSDN 2012*, pp. 115–120, Aug. 2012.

[27] J. Schulz-Zander *et al.*, "OpenSDWN: programmatic control over home and enterprise WiFi," in *Proc. of SOSR 2015*, pp. 1–12, Jun. 2015.

[28] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *Comput. Commun. Rev.*, vol. 44, pp. 87–95, Jul. 2014.

[29] S. Li *et al.*, "Improving SDN scalability with protocol-oblivious source routing: A system-level study," *IEEE Trans. Netw. Serv. Manag.*, vol. 15, pp. 275–288, Mar. 2018.

[30] D. Hu *et al.*, "Flexible flow converging: A systematic case study on forwarding plane programmability of protocol-oblivious forwarding (POF)," *IEEE Access*, vol. 4, pp. 4707–4719, 2016.

[31] S. Li, D. Hu, W. Fang, and Z. Zhu, "Source routing with protocol-oblivious forwarding (POF) to enable efficient e-health data transfers," in *Proc. of ICC 2016*, pp. 1–6, Jun. 2016.

[32] D. Hu *et al.*, "Design and demonstration of SDN-based flexible flow converging with protocol-oblivious forwarding (POF)," in *Proc. of GLOBECOM 2015*, pp. 1–6, Dec. 2015.

[33] Q. Sun, Y. Xue, S. Li, and Z. Zhu, "Design and demonstration of high-throughput protocol oblivious packet forwarding to support software-defined vehicular networks," *IEEE Access*, vol. 5, pp. 24 004–24 011, 2017.

[34] M. Chiosi *et al.*, "Network functions virtualisation," 2012. [Online]. Available: https://portal.etsi.org/nfv/nfv_white_paper.pdf

[35] W. Fang *et al.*, "Joint spectrum and IT resource allocation for efficient vNF service chaining in inter-datacenter elastic optical networks," *IEEE Commun. Lett.*, vol. 20, pp. 1539–1542, Aug. 2016.

[36] M. Zeng, W. Fang, and Z. Zhu, "Orchestrating tree-type VNF forwarding graphs in inter-DC elastic optical networks," *J. Lightw. Technol.*, vol. 34, pp. 3330–3341, Jul. 2016.

[37] J. Soares *et al.*, "Cloud4NFV: A platform for virtual network functions," in *Proc. of CloudNet 2014*, pp. 288–293, Oct. 2014.

[38] J. Martins *et al.*, "ClickOS and the art of network function virtualization," in *Proc. of NSDI 2014*, pp. 459–473, Apr. 2014.

[39] L. Rizzo, "Netmap: A novel framework for fast packet I/O," in *Proc. of USENIX ATC 2012*, pp. 9–9, Jun. 2012.

[40] J. Fontenla-Gonzalez *et al.*, "Lightweight container-based OpenEPC deployment and its evaluation," in *Proc. of NetSoft 2016*, pp. 435–440, Jun. 2016.

[41] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya, "Heterogeneity in mobile cloud computing: Taxonomy and open challenges," *IEEE Commun. Surveys Tuts.*, vol. 16, pp. 369–392, First Quarter 2014.

[42] J. Fajardo, I. Taboada, and F. Liberal, "Improving content delivery efficiency through multi-layer mobile edge adaptation," *IEEE Netw.*, vol. 29, pp. 40–46, Nov. 2015.

[43] Hostapd. [Online]. Available: http://w1.fi/hostapd/

[44] DPDK: Data Plane Development Kit. [Online]. Available: https://dpdk.org/

[45] ONOS. [Online]. Available: https://onosproject.org/

[46] iperf. [Online]. Available: https://iperf.fr/