A Design Framework of Automatic Deployment for 5G Network Slicing

Wen-Ping Lai[*,#], Hong-Lun Lai[*] and Ming-Jay Lai[+]
[*]Department of Electrical Engineering, Yuan Ze University, Taoyuan, Taiwan
[+]Department of Communication Engineering, National Central University, Taoyuan, Taiwan
[#]Email: wpl@saturn.yzu.edu.tw

***Abstract*— Differential services driven user-end and operator-end challenges have been the main driving forces behind the 5G network, which is well perceived as an innovative platform for digital convergence of information, control and management. With both the *network slicing* (NS) and *service slicing* (SS) technologies, precious physical resources can thus be shared among multi-tenant mobile virtual network operators, such as *over-the-top* (OTT) service providers. This paper proposes a three-stage design for automatic slice deployment called LMA, namely (1) LCP: *local charm provision* for VNF services, (2) MSP: *model-based slice planning* for service chaining, and (3) ASD: *automatic slice deployment* for flexible and virtual resource allocation. The LMA is a *model-based slice-specific platform-neutral* design framework for deploying the NS and SS, not only automatically deployable on both the x86-based desk-top computers and data-center bare-metal servers, but also on public or private clouds, as well as the 5G mobile edge. Our design framework adopts the Juju-as-a-Service and Eurecom-Mosaic5G software technologies, where several customizable *virtual network function* (VNF) components can be flexibly chained together to form a desired NS or SS. This paper studies and presents two successful deployment showcases: a web-blog-database based SS and a virtual-evolved-packet-core based NS. Our preliminary results of performance benchmarking show a strong effect of the number of CPU cores on the average latency response of SS, in particular during congestions caused by concurrent user requests.**

Keywords— 5G, service slicing, network slicing, auto deployment, DevOps

I. INTRODUCTION

The novel landscapes of the fifth generation communication [1, 2] are not only remarked by the technology milestones in communication, but also by the ones in networking, which jointly open a brand new vision for creative and versatile applications over human-to-human, human-to-thing, or even thing-to-thing networks. From the perspective of customer premise equipment, these applications are expected be characteristic of very high data rates for wonderful user experiences in mobile broadband, extremely-low latency and ultra-high reliability for connected vehicles and tactile internets, and massive-but-seamless machine type communications everywhere for internet of things. On the other hand, from the perspective of telecom operators, the technologies to achieve these application scenarios should be of low cost in *capital expenditure* (CAPEX) and *operating expenditure* (OPEX), but still of high operation efficiency or even of *zero-maintenance agility* in response to versatile and differential user demands. These user-end and operator-end challenges have been the main driving forces of the 5G networking innovations, such as *software defined networking* (SDN), *network functions virtualization* (NFV) [3], *mobile* or *multi-access edge computing* (MEC) [4-6] etc.

Rooted from SDN, which decouples the *control* and *user* planes, NFV mainly addresses the decoupling of *software* and *hardware*, i.e., it completely releases those conventional *physical network functions* (PNFs) from the vendor-lock-in proprietary hardware, and amplifies the power of *virtual network functions* (VNFs) in terms of *modulization*, *softwarization* and *virtualization*. These make *wireless or radio democracy* based market places [7, 8] possible and feasible via open or COTS hardware design and open-source software implementation. Furthermore, NFV-based *network slicing* (NS) [9-11] and *service slicing* (SS) allow for customized and agile chaining of VNFs based on those differential and even real-time user demands. In other words, different service slices or network slices can coexist on the same system and network architecture, physically sharing computing and storage resources as well as network bandwidths, and yet logically independent of each other. However, the major open and challenging problem to network slicing is the resource orchestration among slices in harmonic slice scheduling for virtual computing power, slice allocation for virtual memory space, and slice arrangement for virtual network bandwidth, all under the balance constraints between maximizing the user-end satisfaction level and minimizing the operator-end resource consumption such that it can accommodate the largest number of multi-tenant application demands.

Research works on network slicing were mainly initiated by 3GPP and ETSI-NFV, where the former focuses on the impact of network slicing to network functions such as mobility, connection choices, charging rates etc., while the latter emphasizes the life-cycle management of virtual networking resources, i.e., *management and network orchestration* (MANO) [12]. In recent years, the ETSI-NFV-MANO issues have become a hot area attracting many studies and collaborative implementation efforts, such as OPNFV [13], ONF-MCORD [14], and Eurecom-Mosaic5G [15] etc. Although implemented on different hardware and software

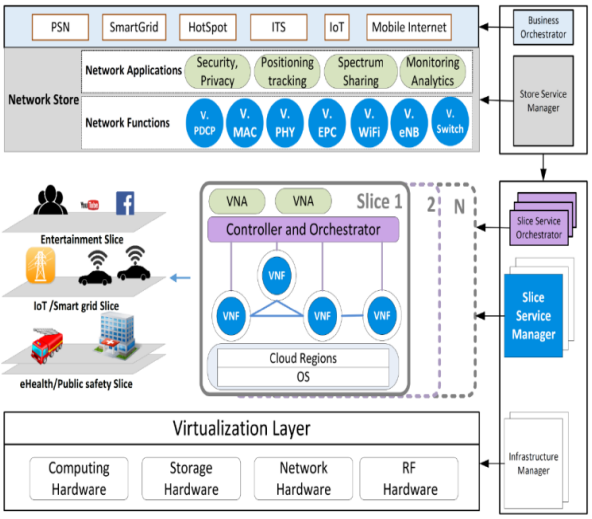

Fig. 1 The concept of Network Store proposed by Eurecom [18].

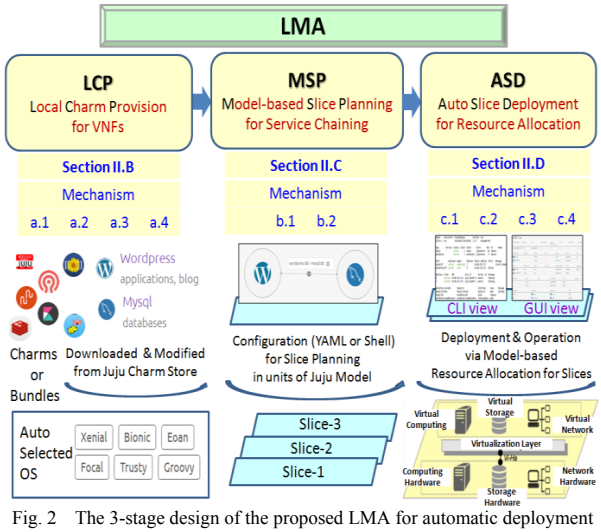

Fig. 2 The 3-stage design of the proposed LMA for automatic deployment of network slicing: from provision and planning to deployment and operation.

platforms, it is expected that the ETSI NFV-MANO-compliant and VNFs-based NS/SS systems be interoperable [16], which is feasible via *loose coupling*, e.g. RESTful APIs.

The Eurecom *open air interface* (OAI) [17] is an open-source software technology for 4G/5G. Eurecom also proposed an innovative concept called Network Store [18], as shown in Fig. 1. The basic idea of Network Store is to create an APP-like market place of telecom VNFs so as to speed up the innovation of telecom via NFV-MANO. The idea has spread out, and an international collaboration from academia and industry has been formed, known as the Eurecom-Mosaic5G Ecosystem, which is continuing the efforts toward an orchestrated 5G network.

This paper proposes a platform-neutral design framework for implementing both NS and SS, not only automatically deployable on both the x86-based desk-top computers and data-center bare-metal servers, but also on both the public and private clouds, and even the 5G mobile edge. Our design framework adopts the Canonical *Juju-as-a-Service* (JaaS) technology and the Eurecom-Mosaic5G open software, where customizable service or network function components (from Canonical's Juju Charm Store [19] or Eurecom's Network Store of Charmed VNFs) can be flexibly modified and chained together to form a desired NS or SS.

This paper proposes a three-stage design for automatic slice deployment called LMA, with stage-specific mechanisms for *slice provision, slice modeling, slice deployment and further operations of slice resources*. In addition, two showcases are presented based on such a multi-stage design process: (1) a simple user-space SS, exemplified by a web-blog-database system, and (2) a complex and kernel-space-involved NS, demonstrated by a *virtual evolved packet core* (vEPC) system of LTE.

The remainder of this paper is organized as follows. Section II details the multi-stage mechanisms of the proposed LMA design. Section III presents the two showcases, as aforementioned. Section IV concludes this paper.

II. Proposed Multi-stage Design (LMA)

This paper proposes a 3-stage design called LMA for automatic slice deployment, as shown in Fig. 2, consisting of the LCP-MSP-ASD stages sequentially, i.e., (1) LCP: *local charm provision* for VNF services, (2) MSP: *model-based slice planning* for service chaining, and (3) ASD: *automatic slice deployment* for flexible and virtual resource allocation. The following firstly describes the selection considerations of *Linux Container* (LXC) for the virtualization layer over the physical infrastructure, followed by three sub-sections giving the details of stage-specific design mechanisms and underlying principles for LMA in three stages.

In terms of virtualization, conventionally the provision of a VNF can be based on a *virtual machine* (VM). VMs can be divided into two categories: *user-space* VMs (such as VMware, VirtualBox, Xen etc.) and *kernel-space* VM (aka Linux KVM), where the latter is less user-friendly but faster in the run time. In terms of virtualization, each VM has its own well-defined *boundary* so that its system resources such as *computing* and *storage* will need to be *predefined before booting*, which also means that run-time resource adjustment is not possible. In addition, due to the need of a guest OS, the resource consumption levels of a VM is usually heavy and inefficient. In addition, the booting of the guest OS is usually intolerably slow to the timely user-demands. A new trend to avoid the above problems is to adopt the Container-based technology, such as Docker or LXC Containers [20-22]. Instead of *OS confinement* in the case of VMs, Containers can be viewed as *process confinement* due to the fact that they get rid of the need of a guest OS in order to be light-weight in system operations and resources consumption. However, different containers on the same physical host still share the same Linux kernel.

In fact, both Docker and LXC actually stemmed from two important virtualization concepts and techniques of Linux, namely the kernel-space supported *namespace* and *control-*

group, where the former is responsible for the *isolation* among containers (basically, there are 6 different types of namespaces, and *network namespace* is the typical one), and the latter for the *control* and *management* of containers, including the CPU, memory, I/O subgroups etc.

In this design, either LXC or KVM can serve as the *infrastructure virtualization layer* since the adopted *VNF manager* (VNFM) Juju supports both. In other words, Juju can manage their corresponding virtual infrastructure manager, i.e., the LXD or KVM server. For simplicity, the paper focuses on the studies of LXC/LXD to magnify the functioning of the Container-based technology. However, for those slices consisting of a component service involving any kernel-space function module or device driver, KVM would be the preferred choice since LXC will need some extra effort for importing those kernel modules or device drivers from the host directory trees, such as */lib/modules* or */dev*. This issue will be further discussed in Showcase 2, which deals with the deployment of a vEPC slice, where the LXC based VNF functioning of its SPGW cannot work without considering the above.

A. Local Charm Provision (LCP) *for VNF Services*

The first stage of the proposed design is the provision of the component VNFs of a target slice. Each component VNF is usually a unique service, working with other component VNFs or services to customize the target slice functions in terms of service chaining of multiple component VNFs.

In order to allow for *life-cycle management* of VNFs, each VNF should be provisioned with an accompany package dealing with the various *states* and *state transitions* in its life cycle, including *installation*, *build*, *configuration*, *running*, and even *future upgrades*. This design adopts the Canonical Charm toolset to provide the above needs for generating a Charmed VNF, denoted as a Charm in short. Many popular Charm templates have been prepared in public repositories such as the Canonical Juju Store, since *no need for re-making the wheels*. Furthermore, these Charm templates can be modified as wished if one understands its technical details.

The proposed design leverages our understanding of the Charm toolset and proposes a concept called *local charm provision* (LCP). The basic mechanism of LCP is described below:

a.1 Pre-downloading the target VNF Charm to save the run-time downloading latency of a *remote* Charm from the Charm Store via networking, in particular to avoid any potential network congestion or unexpected server malfunction of the Charm Store.

a.2 Once a Charm is located locally, its life-cycle management package can thus be modified as desired.

a.3 When a modified Charm is ready, say charm-1, it can thus be deployed locally from a *local provider, i.e.,* from a Charm-based directory tree, as shown in Fig. 3 for a typical Charm's directory structure, where the basic functions of each file and sub-directory are explained below:

- ✓ *config.yaml*: This file is a hierarchical key-value configuration file for describing the default values of various system parameters in the YAML file format.
- ✓ *hooks*: This is a subdirectory for the life-cycle management package of the target Charm, including how to install relation-based events (broken, changed, departed, joined) with other Charms, how to start, stop, react to config-changed events, and how to upgrade the target Charm.
- ✓ *icon.svg*: This is the icon of the target charm in the SVG format.
- ✓ *metadata.yaml*: This file describes all the *provider-requirer* relations and/or the *peer* relations of the target Charm with others.
- ✓ *README.ex*: This file describes the intended usage of the target Charm and how it relates to others.
- ✓ *revision*: This is the revised number of the target Charm.

a.4 The target Charm can also be generated from scratch in different programming language styles such as Linux Shells or Python etc.

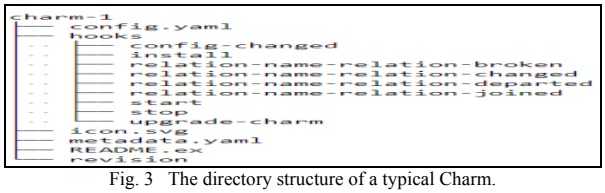

Fig. 3 The directory structure of a typical Charm.

Fig. 4 The interface for the link relation between a web-blog server (Wordpress, the W icon) and its associated database (MySQL, the *Dolphin* icon), where the left end is *wordpress:db* and the right end is *mysql:db*.

```
root@wpl-VB:/home/wpl/my-lxd-juju# juju clouds
Cloud          Regions  Default          Type         Description
aws                 15  us-east-1        ec2          Amazon Web Services
aws-china            1  cn-north-1       ec2          Amazon China
aws-gov              1  us-gov-west-1    ec2          Amazon (USA Government)
azure               26  centralus        azure        Microsoft Azure
azure-china          2  chinaeast        azure        Microsoft Azure China
cloudsigma           5  hnl              cloudsigma   CloudSigma Cloud
google              13  us-east1         gce          Google Cloud Platform
joyent               6  eu-ams-1         joyent       Joyent Cloud
oracle               5  uscom-central-1  oracle       Oracle Cloud
rackspace            6  dfw              rackspace    Rackspace Cloud
localhost            1  localhost        lxd          LXD Container Hypervisor
```

Fig. 5 The *cloud types* that Juju supports, including the major public clouds (e.g. Amazon AWS, M.S. AZURE, Google GCE), the private cloud (e.g. Rackspace OpenStack), and the LXD on the *localhost* adopted by this paper.

```
root@oai:~# juju models
Controller: yzu

Model       Cloud/Region         Status     Machines  Access  Last connection
controller  localhost/localhost  available         1  admin   just now
default     localhost/localhost  available         0  admin   2020-08-13
slice-1     localhost/localhost  available         2  admin   20 hours ago
slice-2*    localhost/localhost  available         4  admin   8 hours ago
```

Fig. 6 The two initial models after the bootstrapping of the Juju-based VNFM: (1) the *controller* model, where the controller called *yzu* (Yuan Ze University) lives, and (2) the *default* model, which is empty initially. Note that more models such as *slice-1* and *slice-2* can be added for slice deployments later on, one for each slice.

```
series: xenial
services:
  "wordpress":
    charm: "cs:xenial/wordpress-0"
    num_units: 1
    annotations:
      "gui-x": "300.0"
      "gui-y": "500.1"
    to:
      - "1"
  "mysql":
    charm: "cs:xenial/mysql-58"
    num_units: 1
    annotations:
      "gui-x": "100.0"
      "gui-y": "500.1"
    to:
      - "2"
relations:
  - - "wordpress:db"
    - "mysql:db"
machines:
  "1":
    series: xenial
    constraints: "arch=amd64 cpu-cores=1 mem=150M"
  "2":
    series: xenial
    constraints: "arch=amd64 cpu-cores=1 mem=200M"
```

Fig. 7 A typical content of a Bundle template file, say called *bundle.yaml*, consisting two VNF services: *wordpress* and *mysql*, corresponding to the slice-specific *model planning* for Fig. 4.

B. Model-based Slice Planning (MSP) *for Service Chains*

The second stage of the proposed design is to deal with the *service chaining* and *slice planning* of the Charmed VNFs. At this stage, a slice is formed by a chain of VNFs, considering their mutual relations to be established based on each pair of *provider-requirer* relationship, which can be realized based on some predefined interface protocols. A typical example can be the link relation between a web-server and its associated database, as shown in Fig. 4, or the HTTP protocol between web clients and the web server. A more complicated example can be the control-plane and user-plane signaling systems among the component functions of an EPC slice, as discussed in Showcase 2.

In order to achieve slice-specific deployment and management, this design proposes a concept called *Model-based slice planning* (MSP). Such a concept is inspired from our understanding of the Canonical Juju and Charm toolsets. As a companion of the Charm toolset designed for the creation of VNFs, the Juju toolset is designed to achieve the deployment of a Charmed VNF or a bundle of Charmed VNFs, which can be pre-planned in terms of a Bundle template. Again, the Canonical Juju Store also prepares some popular Bundle templates, which can be referenced for implementation. As shown in Fig. 5, these Bundle templates can be deployed onto all the cloud types supported by Juju, including *public* or *private* clouds if the cloud credentials are provided, or onto a desk-top *localhost*, which is the adopted case by the paper.

This design takes the *local provider* as the example of the deployment platform. Once the Juju toolset is initialized, it will come up with two basic Models: the *controller* Model and the *default* Model, where the former is exclusively designed for the Juju Controller application to live on top of an LXC machine, and the latter is empty and waiting for further deployment of a Service Bundle. More Models can be added and named as wished to deploy more Bundles. However, *one model* should be prepared specifically for the deployment of *one Bundle* to serve as *one Slice*. In other words, from our understanding on the operational behaviors of Juju Models, this design proposes the following concept: a Juju Model can serve as a unique *canvas* for the deployment of NS or SS, with its detailed mechanism listed below:

b.1 A bundle of Charmed VNFs, or a Bundle in short, can be planned in advance on a Bundle Descriptor, say *bundle.yaml* as shown in Fig. 7., to serve as a slice template for the provision of a slice. Such a Bundle Descriptor is responsible for describing the identities of component VNFs from the Juju Store, the interfaces among them, and the (x, y) positions on the Model canvas. Optionally, the system resources can also be assigned in this descriptor, such as the allowed constraints on the limits of CPU cores and memory space. The Bundle Descriptor template can either be built up by oneself or just be downloaded from the Juju Store, and then be modified to be the desired one. To gain the speed during the deployment, the aforementioned concept of LCP at the first stage should also come in to play in this descriptor to reduce the deployment time of each Charmed VNF so as to achieve a great reduction in the overall slice deployment time.

b.2 Once different slices are deployed on their specific Juju Models, these slices are expected to be well separated during their own run time since Juju creates and visits these Models in a well-separated way spatially and temporally. Namely, the *life cycle* of each slice, including its *creation* and *destroying*, as well as its *resource constraints* or *scalability adjustment* can thus be manipulated in such Model-based planning to pursue fast and automatic slice deployment.

C. Automatic Slice Deployment (ASD) *for Flexible and Virtual Resource Allocation*

The third stage of the proposed design is to achieve an agile deployment for a slice demand and maintain its operational health and scalability to the external stress. Such a *scalable slice deployment* scheme is denoted as ASD, and its mechanism for *flexible* and *virtual* resource allocation is detailed below:

c.1 Slice-specific resource allocation for limiting the use of CPU cores and memory space can be achieved by the model-based constraints in the Juju toolset via the *set-model-constraints* and *get-model-constraints* options, where the former option is for resource allocation, and the latter option is for status checking of resources. However, such a model-based slice-specific allocation scheme is somewhat tricky because the model-based resource constraints should be assigned on the target model *before* deploying the target slice, otherwise no effect will actually happen even if the check by *get-model-constraints* says so, which is quite misleading.

c.2 The reasoning of the above misleading is that the constraints at the level of Model will also passed onto the level of Machine, e.g. the level of Container. But the key is that constraints need to be set *before* the deployment of the target slice.

c.3 The Slice-specific resource allocation is certainly no problematic to *static allocation* of resources. But how

about *dynamic allocation*? It seems to be a big trouble. For the moment, one should pray for a further improvement of the Canonical Juju toolset, or do it in another way, namely change the resource constraints from the level of Machine, which is also achievable by the Juju or LXC toolsets.

c.4 The real issue following *dynamic allocation* is how to get the *decision-making* rules for a better or even optimal use of the system resources. The solution may be *obvious* but *uneasy*. It is *obvious* from the viewpoint of a single slice when the remaining system resources are still enough: *traffic adaptation based dynamic reallocation* is the key. On the other hand, *resource orchestration* among slices is an *uneasy* task, which may involve a choice dilemma between *fairness* and *priority*, and even among multi-tenants. These are beyond the scope of this paper, and more future studies are definitely needed.

III. Showcases and Analyses

A. Experimental Setup

This section presents two showcases for the proposed 3-stage design called LMA (formed by the LCP, MSP, ASD stages sequentially) for slice deployment on the two sample models of Fig. 6, namely *slice-1* and *slice-2*, with both the component Charmed VNFs from the Juju Charm Store, where *n* in the VNF-*n* notation system stands for the *revised number* of the target Charm for the purpose of version control on Charmed VNFs.

- Showcase 1: a simple and user-space service slice (formed by two Charmed VNFs: *wordpress-0* and *mysql-58*) was deployed on the *slice-1* model.
- Showcase 2: a more complex and kernel-space-involved network slice (formed by four Charmed VNFs in the SNAP [23] version of the Eurecom OAI-CN: mysql-56, oai-hss-17, oai-mme-19, oai-spgw-19) was deployed on the *slice-2* model.

Both the slices were built and operated on a physical x86 machine installed with the LXC/LXD and Juju environments. For simplicity and focusing on the Container technology, the KVM technology was avoided. However, the proposed design should still apply to KVMs in principle.

B. Showcase 1: Application Slicing of Web-Blog-Database

Showcase 1 demonstrates a simple and user-space service slice (*slice-1*) consisting of two Charmed VNFs: one is the *wordpress* Charm serving as a Web-Blog, and the other is the *mysql* Charm serving as the database behind. The objective of Showcase 1 is to observe the modeling, deployment and operational behaviors of *slice-1*'s VNFs.

Fig. 8 presents a typical GUI view of the two Charmed VNFs of Showcase 1, which was successfully deployed on the *slice-1* model, based on the model planning of *bundle.yaml*, as aforementioned and discussed for Fig. 7. From the top-left corner, it can be seen easily that 2 *applications* (*wordpress* and *mysql*) consume 2 *machines* in total, namely 1 *machine* for each *application*, as pre-planned by *bundle.yaml*. The second thing noteworthy is the green circular button with a sign of +, which actually means that it allows for *adding* some other external VNFs from the Juju Store to extend the extra functionalities of *slice-1*, or more VNF(s) of the same type to enhance *slice-1*'s scalability. Both of the above are beyond the pre-planning by *bundle.yaml*, making the slice modeling *flexible* and *scalable*. However, *manual scalability* is actually not practically useful to timely changes of the external traffic, and thus *automatic scalability* is much more preferred but will take more research efforts. The third thing (not so visible before pulling down the rectangular button labeled with: 2 *applications*) is a pull-down online form containing application-specific tunable running parameters, which allows for *continuous delivery*, namely *without service disruption* during system tuning.

A click on the *status* button of Fig. 8 gives another GUI view of the statuses of both the deployment and the operation stages, as shown in Fig. 9. This GUI view presents almost-identical contents when compared to those presented in the *command-line-interface* (CLI) view from Fig. 10. Although both the views present mostly the same contents from bottom to top, namely in terms of the Relation level, the Machine level, the Unit level, and the Application level. The key difference between them lies in the presented views: GUI offers a more friendly interface than CLI, in particular to those telecom operator employees not engineering-based. In

Fig. 8 A typical GUI view of the *wordpress* and *mysql* Charmed VNFs after a successful deployment on the *slice-1* model, provided by the Juju controller running on a local LXC container and serving as a Web server at 10.80.253.152 via the 17070 port, whose password can be provided by delivering 'juju gui' on the Linux command line interface.

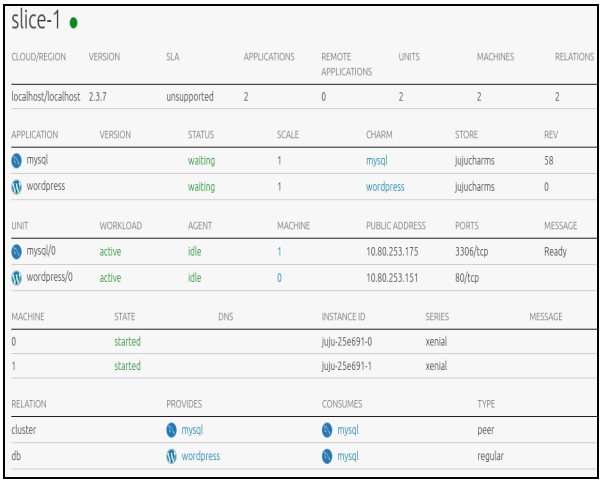

Fig. 9 Another GUI view of *slice-1*, offering the statuses of deployment and operation for each component Charmed VNF, with different levels in Relation, Machine, Unit and Application respectively.

```
Model    Controller  Cloud/Region         Version  SLA
slice-1  yzu         localhost/localhost  2.3.7    unsupported

App        Version  Status  Scale  Charm      Store       Rev  OS      Notes
mysql               active      1  mysql      jujucharms   58  ubuntu
wordpress           active      1  wordpress  jujucharms    0  ubuntu

Unit          Workload  Agent      Machine  Public address  Ports   Message
mysql/0*      active    executing  1        10.80.253.175           (start) Ready
wordpress/0*  active    idle       0        10.80.253.151   80/tcp

Machine  State    DNS           Inst id        Series  AZ  Message
0        started  10.80.253.151  juju-25e691-0  xenial      Running
1        started  10.80.253.175  juju-25e691-1  xenial      Running

Relation provider       Requirer                Interface     Type     Message
mysql:cluster           mysql:cluster           mysql-ha      peer     joining
mysql:db                wordpress:db            mysql         regular
wordpress:loadbalancer  wordpress:loadbalancer  reversenginx  peer
```

Fig. 10 A CLI view of *slice-1*, with nearly identical information levels and messages of running statuses to those presented by the GUI view.

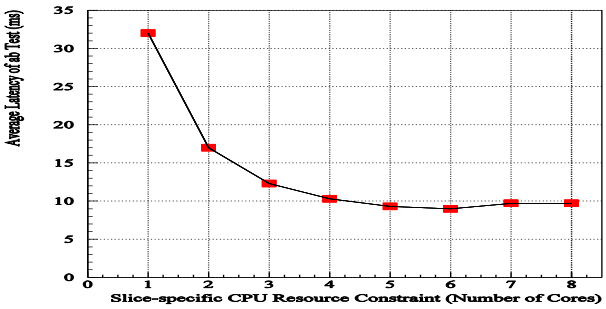

Fig. 11 The resource-constraint effect of CPU cores on the average latency of *ab* tests.

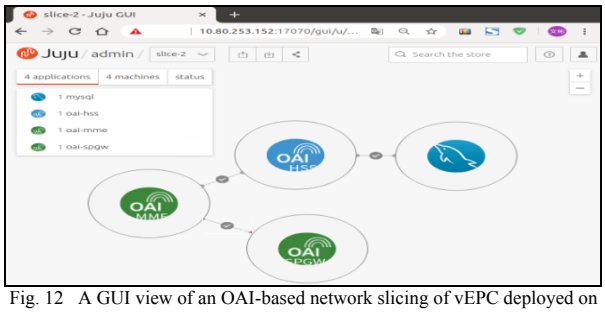

Fig. 12 A GUI view of an OAI-based network slicing of vEPC deployed on the *slice-2* Model, with 4 Charmed VNFs: mysql, oai-hss, oai-mme, and oai-spgw.

```
Model    Controller  Cloud/Region         Version  SLA
slice-2  yzu         localhost/localhost  2.3.7    unsupported

App       Version  Status  Scale  Charm     Store       Rev  OS      Notes
mysql     5.7.31   active      1  mysql     jujucharms   56  ubuntu
oai-hss            active      1  oai-hss   jujucharms   17  ubuntu
oai-mme            active      1  oai-mme   jujucharms   19  ubuntu
oai-spgw           active      1  oai-spgw  jujucharms   19  ubuntu

Unit          Workload  Agent  Machine  Public address  Ports                          Message
mysql/0*      active    idle   0        10.80.253.29    3306/tcp                       Ready
oai-hss/0*    active    idle   1        10.80.253.205                                  Running
oai-mme/0*    active    idle   2        10.80.253.218   36412/tcp,2123/udp,2152/udp    Running
oai-spgw/0*   active    idle   3        10.80.253.172   2123/udp,2152/udp              Running

Machine  State    DNS           Inst id        Series  AZ  Message
0        started  10.80.253.29   juju-ade9f2-0  xenial      Running
1        started  10.80.253.205  juju-ade9f2-1  xenial      Running
2        started  10.80.253.218  juju-ade9f2-2  xenial      Running
3        started  10.80.253.172  juju-ade9f2-3  xenial      Running

Relation provider  Requirer       Interface  Type     Message
mysql:cluster      mysql:cluster  mysql-ha   peer
mysql:db           oai-hss:db     mysql      regular
oai-hss:hss        oai-mme:hss    S6a-hss    regular
oai-spgw:spgw      oai-mme:spgw   S11        regular
```

Fig. 13 A CLI view of the *slice-2* Model, offering the statuses of deployment and operation for each component Charmed VNF, with different levels in Relation, Machine, Unit and Application respectively.

addition, a much shorter training time is also achievable, making operational cost-down and experiences-inheritance possible. On the other hand, CLI is still quite valuable to engineers since both the *LXC* and *Juju* toolsets come with very rich, unique and complementary functional options, allowing for innovative joint development and operation.

Lastly but not least, Fig. 11 summarizes the resource-constraint effect of CPU cores on the average latency of the *wordpress* Charm in slice-1 based on the well-known *apache benchmarking* (*ab*) tests. Every data point stands for the average of 3 data sets, with each running for 10,000 HTTP requests under the con-currency level of 100 requests to generate some level of congestion. The *slice-specific constraint* on *the number ofCPU cores* (N_{cores}) runs from 1 to 8, which is feasible and checkable via both the *set-model-constraints* and *get-model-constraints* options of the Juju toolset. It is clearly seen that such *slice-specific constraints* do have the largest impact when N_{cores} is relaxed from 1 to 2, and the slope gets much smaller as N_{cores} continues to increase. Finally, it is no more helpful when N_{cores} $\geqq$4 since the con-currency level is no more dominant, and the max-latency event becomes *randomly and uncertainly large*, giving a *slight but un-expected fluctuation* in the average value of latency.

C. *Showcase 2: Network Slicing of vEPC*

Showcase 2 demonstrates a more complex and kernel-space-involved *network slice* (*slice-2*) based on the Eurecom OAI-CN software to perform as a virtual evolved packet core (vEPC) of LTE, consisting of four Charmed VNFs: (1) *mysql* (with the *oai-db*) as the back-end server of SIM database, (2) *oai-hss* as the Home Subscriber Server, formed by the *apache2* web server as the front-end, (3) *oai-mme* as the Mobility Management Entity, in charge of the *S1-Control* (S1C) plane for the eNodeB and UE, aided by the S6a protocol interface with *oai-hss*, and interfacing with the user-plane of *oai-spgw* via S11, (4) *oai-spgw* as the joint S-Gateway and P-Gateway (via omitting the S5/S8 interface), where the former is in charge of the *S1-User* (S1U) plane for the eNodeB and UE and the latter is in charge of the Internet connection via the SGi protocol.

Similar to Fig. 8, Fig. 12 also presents a GUI view of *slice-2,* where all the above Charmed VNFs and their necessary interfaces were successfully modeled and deployed. All the benefits of deploying and operating *slice-1* as mentioned for Fig. 8 also apply to the case of *slice-2*. Again, similar to Fig. 10, Fig. 13 provides a CLI view of the deployment and operational statuses of *slice-2*.

However, the domain knowledge to successfully deploy and operate slice-2 is more complex in the sense that not only more interface protocols are needed among the Charmed VNFs, but also the *GPRS tunneling protocol* of *user plane* (GTP-U) needed by the *oai-spgw* Charm cannot be provisioned in advance because the function of GTP-U is played by a Linux kernel module called *gtp.ko*, which should be inserted during the run time at the Machine level (in terms of LXC). In other words, failing in doing so will lead to an error message at the stage of configuration. Another

configuration error type comes from the incompatible device names of *network interface cards* (NICs) due to the change of interface naming convention of NICs, particularly for Ubuntu Linux versions 16.04. In this design, both the */lib/modules* and */dev* directory trees in the *host* namespace should be imported and shared by the *guest* namespace of the *oai-spgw* container, in terms of the LXC volume-based sharing and mounting.

Hence, the domain knowledge still plays some important role for a successful design of slice deployment of operation like *slice-2*. However, this is again an engineering problem. Once it is conquered, *slice-2*-like deployments and operations should be as easy and convenient as slice-1's.

IV. Conclusion and Outlook

In this paper, a three-stage design for automatic slice deployment called LMA has been proposed, namely (1) LCP: *local charm provision* for VNF services, (2) MSP: *model-based slice planning* for service chaining, and (3) ASD: *automatic slice deployment* for flexible and virtual resource allocation. Based on LMA, two showcases have also been successfully presented: (1) a simple user-space SS, exemplified by a web-blog-database system, and (2) a complex and kernel-space-involved NS, demonstrated by a vEPC system of LTE. These two showcases together deliver the following important messages:

- From the perspective of *development* for successful slice deployment, the *domain knowledge* of each slice type is important and may be as tricky as those encountered in slice-2. But it is always a challenge to engineers, not to telecom operator employees.
- From the perspective of *operation* for slice deployment, *automation* is the key and will lead to easy training of telecom operator employees and conveniently meet those on-demand differential requirements from multi-tenants, such as those OTT service providers.
- The LMA is a platform-neutral design in the sense that a successful slice deployment of Charmed VNFs locally on a physical machine can be easily transplanted onto other platforms, including public/private clouds and even the 5G mobile edge cloudlets, since the Juju toolset is universal to these platforms.

As an outlook, end-to-end network slicing would be even more challenging because it involves slicing the RAN [24] and the Mobile Edge, where the former needs to handle the Cloud-RAN (C-RAN) related issues while the latter needs to combine both parts from the C-RAN and the micro-services driven *service based architecture* of 5G core networking [25].

Acknowledgement

This work was supported by Taiwan's Ministry of Science and Technology under grants 107-2221-E-155-013 and 108-2221-E-155-022-MY2.

References

[1] *System Architecture for the 5G system Stage 2*, 3GPP TS 23.501 v15.00, Dec. 2017.
[2] *View on 5G Architecture*, 5GPPP White Paper v2.0, Dec. 2017.
[3] *Network Functions Virtualization* (NFV) *Ecosystem Report on SDN Usage in NFV Architectural Framework*, ETSI NFV-EVE White Paper 005, Dec. 2015.
[4] M. Patel et al., Mobile-Edge Computing - introductory technical white paper. ETSI White Paper, Sep. 2014.
[5] Heli Zhang, Jun Guo, Lichao Yang, et al., "Computation offloading considering fronthaul and backhaul in small-cell networks integrated with MEC," *IEEE Conf. on Computer Communications Workshops*, Atlanta, GA, USA, May 2017, pp. 115–120.
[6] W.P. Lai and K.C. Chiu, "NUMAP: NUMA-aware multi-core pinning and pairing for network slicing at the 5G mobile edge," *Proc. APSIPA Annual Summit and Conference 2019 (APSIPA ASC'19)*, Lanzhou, China, Nov. 2019, pp. 22–27.
[7] Crowd Supply Market Place, https://www.crowdsupply.com/lime-micro/limesdr
[8] Lime Microsystems, https://limemicro.com/
[9] *Description of Network Slicing Concept*, NGMN White Paper, Jan. 2016.
[10] X. Foukas, G. Patounas, A. Elmokashfi and M. K. Marina, "Network slicing in 5G," *IEEE Comm. Magazine*, vol. 55, no. 5, pp. 94–100, May 2017.
[11] *Network Functions Virtualization (NFV) Release 3 on Management and Orchestration; Network Service Templates Specification*, ETSI GS NFV-IFA 014 v3.4.1, June, 2020.
[12] *Network Functions Virtualization (NFV) Release 3 on Management and Orchestration; Functional Requirements Specification*, ETSI GS NFV-IFA 010 v3.4.1, June, 2020.
[13] OPNFV, the Linux Foundation Projects, https://www.opnfv.org/
[14] ONF, the M-CORD project, https://www.opennetworking.org/
[15] Eurecom, the Mosaic-5G project, http://mosaic-5g.io/
[16] *Network Functions Virtualization (NFV) Release 2 on Testing;* , *Guidelines on Interoperability Testing for MANO,* ETSI GR NFV-TST 007 v2.6.1, Jan, 2020.
[17] N. Nikaein, M. K. Marina, S. Manickam et al., "OpenAirInterface: a flexible platform for 5G research," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 33-38, 2014.
[18] N. Nikaein et al., "Network store: exploring slicing in future 5G networks," *Proc. 10th Int. Workshop on Mobility in the Evolving Internet Architecture* (*MobiArch '15*), Paris, France, Sep. 2015, pp. 8–13.
[19] Canonical, Juju Charm Store, https://jaas.ai/store
[20] W.P. Lai, Y.H. Wang and K.C. Chiu, "Containerized design and realization of network functions virtualization for a light-weight evolved packet core using OpenAirInterface," *Proc. APSIPA Annual Summit and Conference 2018 (APSIPA ASC'18)*, Honolulu, Haiwaii, USA, Dec. 2018, pp. 472–477.
[21] I. Miell and A. H. Sayers, *Docker in Practice*. New York: Manning, 2016.
[22] D. Bernstein, "Containers and cloud: from LXC to Docker to Kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.
[23] Canonical, Snapcraft, https://snapcraft.io/docs/getting-started
[24] Chia-Yu Chang and Navid Nikaein, "RAN runtime slicing system for flexible and dynamic service execution environment," *IEEE Access*, vol. 6, pp. 34018–34042, July 2018.
[25] T. Li, L. Zhao, R. Duan and H. Tian "SBA-based mobile edge computing," *Proc. IEEE Globecom Workshops 2019 (Globecom'19)*, Waikoloa, Hawaii, USA, Dec. 2019.