JOX: An Event-Driven Orchestrator for 5G Network Slicing

Kostas Katsalis, Navid Nikaein, and Anta Huang
Communication System Department, Eurecom
Email: firstName.lastName@eurecom.fr

***Abstract*—Network slicing is expected to be the main pillar around which virtualization technologies together with SDN control and NFV, will provide on-demand network and cloud infrastructures and facilitate rapid service deployment. In this paper we present JOX, an event-driven orchestrator for the virtualized network, operating on top of the Juju management system, that inherently supports network slicing. JOX is a python-based generic network slicing orchestrator, with a plugins architecture that is able to support different segments of a modern mobile edge network. We present a concrete prototype implementation of JOX for LTE, with experimental results considering footprint analysis, performance metrics, and implementation experience for slicing and orchestrating of an operational LTE network.**

***Index Terms*—Network slicing, orchestration, cloud computing, 4G, 5G, SDN, NFV, RAN.**

I. Introduction

In many ways virtualization technologies are solving fundamental weaknesses in traditional IT environments. However, they have little to offer when they are not well suited to support diverse services, when they are not agile with open interfaces for management and control, while also when they are not able to be optimized for specific use cases that are concurrently exploiting shared infrastructures.

In order to support such functionalities, the concept of network slicing has come about towards integrated 5G communications. A network slice is used to support the communication service of a particular connection type with a specific way of handling the control plane and user plane for this service [1], [2]. The consensus of both academia and the industry is that network slicing will rely on cloud-based approaches and the main gears will be build around cloud technologies, Software Defined Networking (SDN), Network Function Virtualization (NFV), together with advanced orchestration tools [3], [4], [5]. Network slicing will offer a way to enhance operational efficiency, while reducing time-to-market for new services and opening new business opportunities for Over-The-Top (OTT) players and vertical industries.

In this paper we present JOX, an open-source orchestrator for the virtualized network that natively supports network slicing. Each slice supports one or multiple service offering tailored to specific business segment. Using JOX, each network slice can be independently optimized with specific configurations on its resources, network functions and service chains. Inside the JOX core, a set of services is used to operate and control each network slice, while at the same time support the necessary interplay between resource and service orchestration, Virtual Network Function Management (VNFM) and Virtual Infrastructure Management (VIMs) as these are defined in the ETSI MANO architecture [6]. VNFM is the tool used to control the life-cycle management of Virtual Network Function (VNFs) and Virtual Network Applications (VNAs), whereas the VIM is responsible for managing the virtualized infrastructure. From the implementation perspective, JOX is tightly integrated with the Juju VNFM framework provided by Canonical [7]. The Juju system is also one of the main VNFM for ETSI OSM [8].

Although there are many orchestration solutions available like Raft-IO in OSM [8], ONAP [3], and OpenStack Tacker [9], they do not offer life-cycle management support on per network slice basis. Furthermore, the operation at the edge of network is fundamentally different from that of the data-center, with a very sophisticated control plane and strict requirements for the Radio Access Network (RAN) processing. For the LTE network, issues like resource management and isolation between slices are discussed in 3GPP TR 28.801, but still no mature orchestration solution exists in the context of network slicing. In that sense, the goal of JOX is to create new vantage points on 5G orchestration targeting the edge network, that will allow new policies to handle strict latency or bandwidth requirements on per slice basis.

The core JOX characteristics and innovations are summarized as follows and are analyzed in the following sections:

- slice-specific lifecycle management and a powerful northbound API;
- core services facilitate the optimization of the orchestration procedures;
- a message-bus based plugin framework is exploited for communicating with VIMs. JOX also supports RAN specific plugins, like for example FlexRAN [10], in order to control the physical or virtualized LTE eNodeBs;
- slice descriptors are coupled with the service configuration.
- network slice logic can be easily introduced as an application for slice optimization.

In the following we describe the JOX architecture and we analyze its core components, together with the functionalities offered. JOX partially covers the main network slice lifecycle phases as identified by 3GPP TR 28.801, regarding network slice pre-provisioning,instantiation,runtime operation and decommissioning. To the best of our knowledge JOX offers the first open-source orchestration implementation that is able to support network-slicing targeting virtualized mobile networks. We evaluate JOX performance, the efficiency of the design and the ability to support network slicing. We also demonstrate JOX in practice by running a use case for building end-to-end LTE VNF service chains. For the

978-1-5386-3416-5/18/$31.00

necessary LTE network components and VNFs in all of our demonstrations we exploit the open-source OpenAirInterface (OAI) framework [11] and the FlexRAN protocol [10]. OAI offers an open-source software implementation of the whole LTE protocol stack for both the RAN and the core network.

The rest of the paper is organized as follows. In Section II we present the related work. In Section III we present the design goals and primitives of operation. In Section IV, we present the JOX architecture, together with the main functionalities and JOX core components. In section V we evaluate JOX through footprint analysis and proof of concept demonstrations. We conclude the paper and highlight some future directions in Section VI.

TABLE I: Summary of MANO terminology

Acronym	Description
VNF	Is the virtualized network element like Router VNF, Switch VNF, Firewall etc.
VNF Catalog	A repository of all usable VNF Descriptors (VNFD). VNFD describes a VNF in terms of its deployment and operational behavior requirements
NFVI Resources	A repository of NFVI resources utilized for the purpose of establishing NFV services.
VNFO	software responsible to automate the creation, monitoring and deployment of resources and services.
VIM	Virtualized Infrastructure Manager (VIM), manages NFVI resources in one domain.
VNF Manager (VNFM)	Manages life cycle of VNFs. It creates, maintains and terminates VNF instances, installed on VMs which the VIM creates and manages.

II. Related Work

A. Background Information on Network Slicing

The concept of network slices has been refined by NGMN [1], adopted and adapted by the main Telecom manufacturers like Huawei, Ericsson and Nokia. According to the NGMN definition, a 5G network slice supports the communication service of a particular connection type with specific requirements and configurations for handling the control and data plane. A more generic definition is described in [12] where a network slice can be defined as a composition of adequately configured network functions, network applications and underlying cloud infrastructures that are bundled together to meet the requirement of a specific use case.

The concept is strongly coupled with SDN/NFV technologies [13], [14]. In [2], SDN and NFV technologies are exploited to enable an dynamic sharing of network resources among operators. In the LTE domain 3GPP SA2,SA5 and 3GPP radio access network (RAN) groups are building technical specifications to integrate Network Slicing in the upcoming specifications. Network Slicing requirements are described in [15] and 3GPP TR22.864 and TR23.799. 3GPP considers a solution to enforce Network Slicing using the eDECOR concept (3GPP, TR 23.711, release 14), and more recently a technical report on study and provisioning of network slicing for 5G networks and services (3GPP, TR 28.801 and TS 28.531, release 15). One of the objectives of these activities is to combine the concept of 3GPP RAN sharing and eDECOR to create an end-to-end Network Slice. In [16] a technical approach is presented for slicing the LTE network.

B. ETSI MANO and 3GPP Orchestration Terminology

The ETSI NFV-MANO architectural framework is a collection of functional blocks, data repositories and interfaces through which information is exchanged for managing and orchestrating network functions virtualization infrastructure (NFVI) and VNFs. For ease of reading in Table I, we provide a summary of definitions for the terminology used by ETSI.

VNF Orchestrator (VNFO): The main components of the architecture that we focus in this work is the VNF Orchestrator (VNFO). This is the software responsible to automate the creation, monitoring and deployment of resources and services in the underlying environment (software and hardware). According to ETSI, the following distinction must be considered:

- **Resource Orchestrator (RO):** coordinates, authorizes, releases and engages NFVI resources among different Point of Presence (PoPs) or within one PoP, and
- **Service Orchestrator (SO):** SO overcomes the challenge of creation of end-to-end services among different VNFs (that may be managed by different VNFMs). Service Orchestration creates end-to-end service between different VNFs.

According to 3GPP a Network Slice Instance (NSI) is a set of functions and the resources of these functions which are arranged and configured, forming a complete logical network (3GPP TR23.799, TR28.801). A NSI may operate over multiple network slice subnetworks. For each sub-network a specific set of requirements is driving the creation of a Network Slice Sub-network Instance (NSSI). The NSSI is controlled through interaction between the Network Slice Managemenent Function (NSMF) and the corresponding Network Slice Subnet Management Functions (NSSMF). The NSSIs are integrated in the NSMF in order to provide and instantiate the end-to-end NSI. JOX is targeting RAN-NSSMF functionalities, however it can also be used as a NSMF solution.

C. Orchestration Software Landscape

Towards 5G, standardization and open-source are becoming complementary to allow fast innovation, which is why most of the current solutions are open-source.

- Open Source MANO (OSM) [8]: published under Apache v2 license, and includes the SO, the RO and a configuration manager. In one of the realization of the architecture, the Juju framework is used to provide the VNFM functionalities, while Riftware is used to support orchestration.
- Tacker [9]: is an official OpenStack project building a VNFM and a NFVO to deploy and operate Network Services and VNFs on the OpenStack infrastructure platform. It targets data center environments.
- ONAP [3]: brings together Open ECOMP from AT&T and Open-O projects as a comprehensive platform for real-time, policy-driven orchestration and automation of physical and virtual network functions. It is supported by Linux Foundation. Open-O had adopted standard service modeling languages YANG and TOSCA to ensure commonality.
- OPNFV: is another realization of ETSI MANO framework, supported by the Linux Foundation. OPNFV integrates OpenStack as the supporting cloud management system and also considers for a number of SDN controllers.

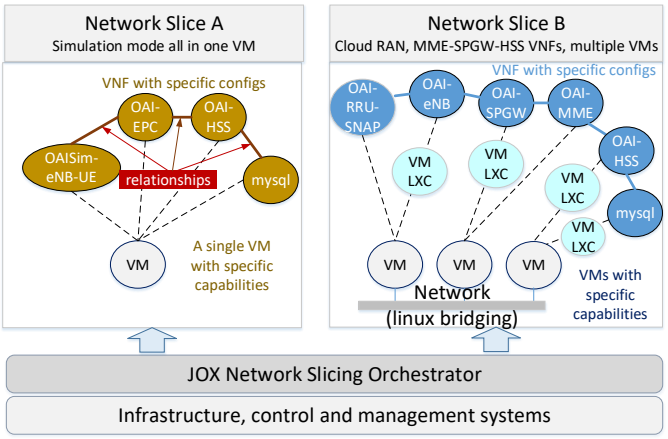


Fig. 1: JOX support for two network slices. In *slice A* the OAI eNodeB and EPC are deployed in a single VM. In *slice B* a cloud-RAN chain using disaggregated RAN is deployed over different VMs.

The SONATA and 5GEx projects[1][2] introduce two new types of orchestrator following the ETSI MANO design [17], [18]. The architecture of JOX is similar to the SONATA design, exploiting a message bus to support the plugin communication subsystem. A similar proposal comes from OpenBaton NFVO [19], designed and implemented following the ETSI MANO specification. Similarly,OpenBaton uses message queueing for the communication with the VNFM. Note, however both the OpenBaton and the SONATA solutions do not inherently support life-cycle management of network slices. Other frameworks such as CloudNFV, Puppet, Chef, Cloud Foundry provide a fast applications development and deployment cycle. We also mention the Unify solution that describes a multi-layer service orchestration in a multi-domain network [20] and [21]. Third party solutions also exist like Hurtle that delivers software as services and can easily compose and chain network functions to deliver end-to-end services [22].

III. Design Goals and Primitives

JOX is a multi-service orchestrator, developed as a part of the Mosaic5G opensource ecosystem[3]. Mosaic5G aims to provides agile mobile network service delivery platforms for 4G-5G R&D. JOX satisfies the following design goals:

- `Support Network Slicing`: JOX enables network slice lifecycle management and allows to orchestrate each network slice independently. As depicted in Fig.1 using JOX one can deploy a number of network slices with different end-to-end logical networks. These are optimized for a particular service through custom logic. Each slice operates over common VNFM and VIM infrastructures (see Fig. 2).
- `Support Orchestration for the Mobile Network`: JOX exploits RAN and CN specific plugins to efficiently orchestrate the edge network resources and services. For example, orchestrate a new slice across multiple eNodeBs, partition the radio resources and deploy a dedicated CN for this newly generated slice.
- `Optimize the operational environment`: JOX runs slice-specific logic (e.g. where to schedule VNFs and how to optimize a slice) as an application running on top through a northbound API (see Fig. 2). Global optimization (considering all slices) is also possible as a Python application within the JOX orchestrator.
- `Isolation`: JOX guarantees that different network slices are isolated from each other in the management plane.

[1]www.sonata-nfv.eu
[2]5gex.eu
[3]mosaic5g.io

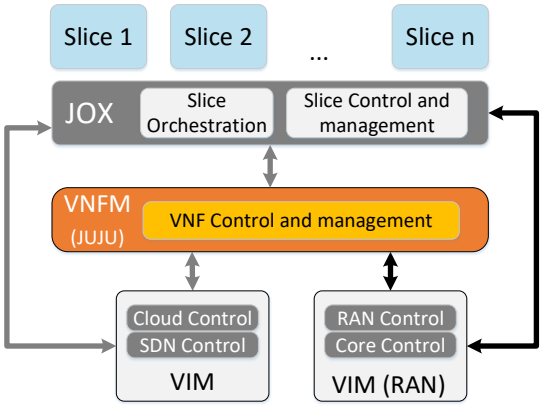


Fig. 2: JOX interactions with VNFM and VIMs

Fig. 3 presents the main components of JOX including `JOX Network Slices (JSlices)` and `JOX Clouds (JClouds)` and their relationship, which are detailed below.

1) `JOX Network Slices (JSlices):` In JOX a slice is represented by a `JSlice` object that is defined as a set of models (called `JModels`) together with a policy specification. This policy may be global for all the slice models, while every model is deployed over a specific cloud infrastructure that is controlled by a single VIM (e.g. RAN-VIM). Every `JModel` is a bundle of:

- *Resources:* include all physical (e.g. servers, spectrum) and virtual resources (e.g., VMs).
- *Services:* include physical or virtual network functions (PNFs and VNFs) such as eNodeB and vMME and virtualized network applications (VNAs) (e.g. monitoring).
- *Service chains:* describe the relationship between PNFs/VNFs/VNAs (e.g. between eNodeB and vMME).
- *Policy:* a `JModel`-specific policy (e.g. do not use server A).

Each service or resource is defined using a JSON descriptor that defines requirements, capabilities and configurations.

2) `JOX Clouds (JClouds):` Every *JCloud* object hosts all the underlying cloud resources and interacts with the physical infrastructure and the cloud control mechanisms through two channels: (1) the VNFM for a set of basic funtionalities, and (2) directly with the VIM for fine-grain monitoring and control. Although VNFM is able to interact with the VIM, it is the direct communication between the orchestrator and the VIM that can offer the maximum level of control of the underlying physical and virtual infrastructure.

JOX is a single VNFM - multi VIM orchestrator. The VNFM is Canonicals's Juju, that interacts with *charms* that act as structured NFV element managers driven by Juju. A charm encapsulates a VNF as a service and contains all the

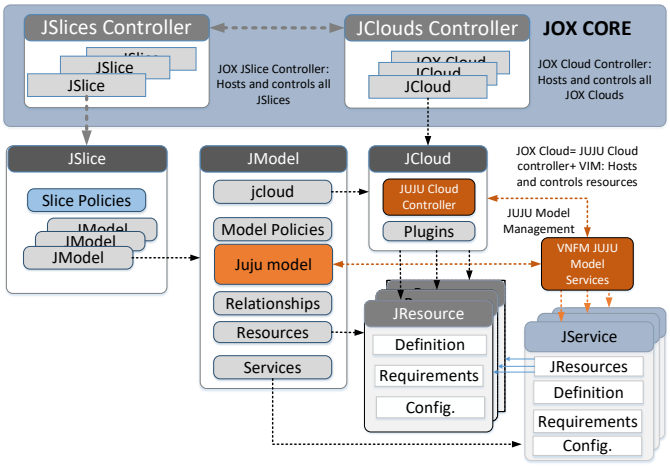


Fig. 3: JOX main components and properties

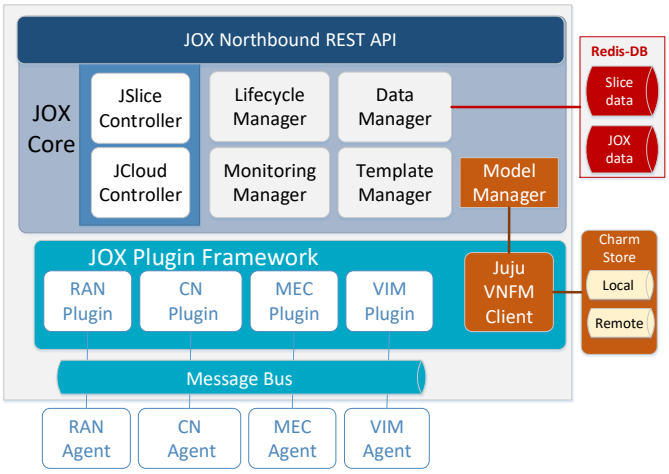


Fig. 4: JOX architecture

necessary hooks (i.e. scripts and primitives) to manage the life-cycle of the VNF and its relationships within service chains. It contains all the logic required to deploy, configure, integrate, scale, and expose the service to the outside world, that are available to JOX through a rich Juju API. We highlight that a rich set of OAI-based 4G and a subset of 5G VNFs (for both RAN and CN) are already available as Juju charms in the Juju store (an online VNF catalog [7]).

Juju supports a number of VIMs and the clouds including OpenStack, LXC, MaaS, Joyent, Amazon, Cloudsigma, Vmware, HPCloud, GCE and Azure. With JOX, the underlying cloud resources are extended with physical or virtual RAN and CN elements and a set of resources that also include for example radio spectrum and resource blocks. Note that while JOX exploits all the services that are exposed by Juju regarding the resource management of the infrastructure, the API between Juju and VIMs is restricted to a basic set of functionalities (e.g., deploy VMs with specific requirements). In order to retrieve/analyze the real-time performance and also trigger custom events, JOX also exploits direct communication with the VIMs through a plugins framework.

IV. Architecture, Services, and Implementation

The architecture of JOX is depicted in Fig. 4. JOX exposes a REST northbound API to enable monitoring, control, and programming of each slice. A set of core services is used in support of slice-specific life-cycle management, data handling, monitoring and template management. In the following we provide design and implementation details for each components.

A. JOX Core Services

JOX provides two main services in support of slicing:

- `JOX Slices Controller (JSC)`: responsible to host and control all the instantiated `JSlices`. This is the place where global optimizations can be performed.
- `JOX Clouds Controller (JCC)`: responsible to host and control all the instantiated *JClouds*. JCC offers services to the JSC.

Other core services are related to the life-cycle management of JOX itself and a logging subsystem.

B. Northbound API

JOX exposes a northbound REST API. Using the exposed methods one can create a *JSlice*, connect to a *JCloud* and adjust all the models, resources, services and service relationships. In more detail, JOX is aligned with the recommendations of the basic operations that are defined by 3GPP in TR 28.801. According to this the Network Slice Instance (NSI) life-cycle phases are preparation, instantiation, configuration, activation, runtime control and decommissioning. Through the API methods related to these phases are exposed.

C. Network Slice Definition, Control and Management

1) JSlice Preparation Phase: The first step in the preparation phase is related to the NSI design. In this step the NSI owner defines a set of resources, requirements and services, service relationships and the corresponding configurations. Currently, JOX provides a simple *JSlice* definition. At this stage of developments and the current research status, it is impossible to support a very large set of functionalities, since standard templates definitions is currently an open research issue.

In Fig.5 an empty network slice (without any resources or services) id depicted that can be created as a POST message to JOX. In the subsequent release of JOX, template descriptions will be aligned with the work delivered in OASIS TOSCA and the modeling work in IETF [23].

For every model we utilize different namespaces for resources, services and the bindings to services. This way a resource (e.g. a VM) is described using a *JSlice-JModel* specific name. For example in the top right of Fig. 5 the definition of a LXC-VM is described. The container identifier for the specific model is “k2” while the container is hosted in machine “cF45” that can be a physical machine or a container or a KVM-VM. Note that in order to efficiently manage the resources through the plugin framework, for each resource a number of identifiers is used. For example a container carries its LXC id “juju-97ba7d-0”, its Juju identifier “0” and its JModel id “k2”. In the service provisioning time, each JSlice/JModel is isolated, so only the JModel identifiers are required, and there is no need to know what is the name the

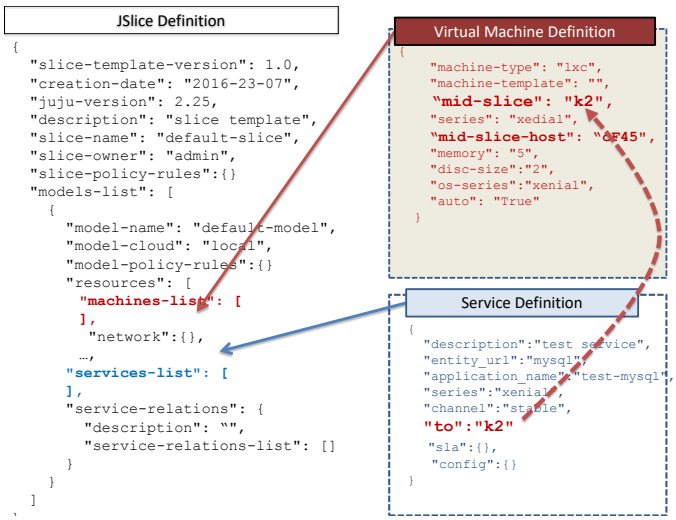


Fig. 5: JOX JSlice definition in JSON representation.

hypervisor assigns to the VM. These are discovered/assigned internally by JOX. At the bottom of the figure the definition of a ”mysql” service is described where the prefix “to” simply places the deployment of the service to container “k2”.

Besides the information related to “where to deploy” the service, the configuration of the service is passed together with its definition; otherwise default configuration is loaded. This is very important and can greatly assist the preparation, the instantiation and activation phases. For example in custom made solutions, fine-tuning a mysql server and connecting it to the HSS service requires from minutes to hours. In JOX this flexibility is enabled by the Juju framework, which triggers the corresponding Juju charm hooks (e.g. configuration changes and relation changed).

In the pre-provisioning phase a negotiation routine with the VIM can also be executed. Such negotiation procedures were described in [16], [5]. While a logical definition of services and networks are initially requested by the slice, the VIM supports the requested capabilities, SLA and QoS levels based on its current state and the book-keeping information maintained by the top-level orchestrator. The negotiation routine is currently not supported by JOX.

2) Runtime: We consider that a modification of the runtime state of a network slice is performed in two modes: `auto` and `manual`. In the former the network slice controller detects a performance degradation that leads to a SLA violation. In such case, specific actions need to be performed through interactions with the plugin framework. For example, increase in memory and CPU power to support the current workload, or increase the radio resources of a particular slice to increase its data rate. In the `manual mode`, the network slice owner is able to adjust the parameters and configurations of the network slice and the corresponding sub-elements through the northbound APIs. In both cases, a set of monitoring services is exploited at the level of network slice (e.g `slice is healthy`), the VNF, and the cloud infrastructure. These are used to either trigger the necessary action sets or facilitate the decision maker procedures.

The ability to monitor and adjust the network slice behavior and characteristics in runtime is an extremely powerful feature of JOX since it allows network slice logic to be easily introduced. In the current version of JOX, this is achieved through direct interactions with KVM, LXD hypervisors and the plugins framework for the RAN. Automatic events creation will be supported in future release.

3) Data Management: For the data back-end, the design is based on the non-SQL REDIS server serving as in-memory data structure store. In JOX we consider two types of data: the *configuration data* and the *runtime data*. For each *JSlice* the former are the data that are posted or pushed by the *JSlice* owner to JOX to update the configuration of the slice and its sub-elements. Example of such data updates are *add model*, *delete model*, *add machine to model* or *update a service configuration*. The runtime data are related to the actual current state of the *JSlice* models on metrics like I/Os or the memory usage of a VM.

One great challenge is related to the recreation of the slice in case of failure. The reason is that JOX operates on top of VNFM and VIMs that are unaware of the existence of network slices. The state management of a *JSlice* is under the full responsibility of the JOX orchestrator. If JOX fails for some reasons, the VMs and the configured services (through Juju VNFM) will still be operational; the *JSlice* state however will be lost. Since the underlying VIM and VNFM systems are still operational, the infrastructure and the services deployed needs to be associated again with their corresponding slices. The approach we currently consider is the serialization of the entire *JSlice* object as a REDIS entry. This is only related to configuration data (e.g. VM ”k2” belongs to *JModel* ”model-k21” of JSlice ”default-slice”), which is an assembly of the configuration requested by the slice owner. During operation the runtime performance metrics (like CPU utilization, or memory usage) are periodically stored in Redis as well and thus in case of failure they can be restored.

D. Plugin Framework and Interplay with Juju

JOX interacts with the Juju VNFM using the Juju-python 2.25 API. A specific Juju plugin is responsible to update the status of network slices services in runtime depending on the events/messages received by JOX driven by the *JSlice* owner. In order to interact with VIMs for the cloud infrastructure and the RAN, JOX relies on a message-bus-based plugin framework. The message bus implementation is based in the RabbitMQ solution (v3.5.7 AMQP), while we use the Pika library to exploit the RabbitMQ services.

In Fig. 6 a high level description of the agent-plugin approach is described. For each resource, a local agent (called *JAgent*) is instantiated responsible for providing two functionalities: (1) monitoring, and (2) event handling. The former is related to the statistics, measurements and status reports that JOX and the *JSlices* need to know. The other is related to the events that are driven by the orchestration logic, or generated by the slice-specific applications, or even manually from the *JSlice* owner. Inside JOX, a specific plugin (referred as *JPlugin*) is loaded to interact with the JOX agent over the message bus interface as well as with the JOX core services. Regarding monitoring services, a Pub/Sub mechanism is used for the JAgent/JPlugin messaging. The agent publishes the monitoring information and the consumer plugin inside JOX

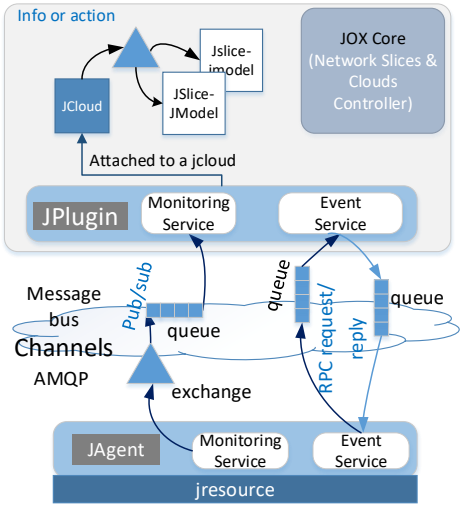


Fig. 6: General JAgent-JPlugin architecture.

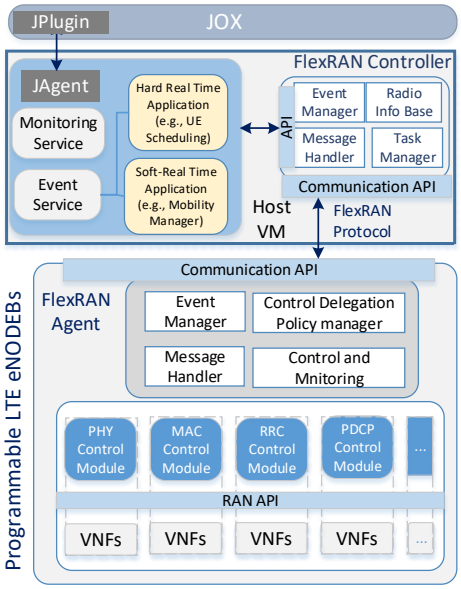


Fig. 7: JAgent integration with FlexRAN controller.

receives information from a queue. For the event handling, JOX relies on the RPC communication with a request/reply pattern.

- **Example 1 :** In a physical host, a LXD *JAgent* is installed for controlling LXC containers. Using the *pylxd* library the *JAgent* retrieves information from the lxd hypervisor or using custom libraries it can control the cgroups for reallocation physical resources to each container. The *JAgent* interacts with a LXD JPlugin inside JOX.
- **Example 2 :** For orchestrating LTE eNodeB resources a *JAgent* is integrated with a FlexRAN controller, as described in [10], using OAI-based LTE systems. In Fig. 7, a high level representation of the solution is depicted. A FlexRAN agent resides within the eNodeB communicating and interacting with a FlexRAN real-time controller entity. In contrast to traditional wired-SDN approaches, realtime control and time-critical operations are also considered for the programmability of the data plane, while retaining the service continuity. The FlexRAN agent exposes a set of RAN APIs, which can be used to monitor and control one or many local network functions in the eNodeB. In this case the *JAgent* is integrated with the FlexRAN controller (by exploiting services from the controller's northbound API), while the orchestration procedures in JOX and the JSlice drive the system operation.

E. Implementation

Table II provides a summary of the main libraries and software used in JOX.

TABLE II: Summary of Implementation details

Component	**Library or framework used**
JOX Core	All JOX framework built in Python 3.5
RestAPI	Flask v0.12.2
Database	Redis server v 3.0.6, Redis python client v.2.10.5
Message bus	RabbitMQ Server v3.5.7, JPlugins-JAgent: Pika v0.11.0 *note that JAgents can be implemented in any language that supports the Rabbitmq bus
Juju VNFM	Juju v2.2.4-xenial-amd64,juju python client v0.5.2

In Table III, we summarize an indicative list of the main features that are available, under development or planned. As JOX is a living open-source orchestrator, there will be regular updates and feature releases.

TABLE III: Summary of features supported

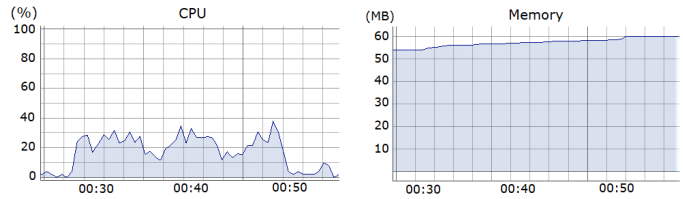
Feature	Status
Provisioning, deployment-deletes of JSices	supported
NF, VNF,VNA service chaining	supported
JSlice runtime/config status save to Redis	supported
JSlice runtime/config recovery	under development
Auto action/events mechanisms	planned
Manual Action mechanisms	supported
Clustering support	not in plans
Linux host, KVM, LXD, FlexRAN JPlugins	supported
LL-MEC JPlugin	under development
DOCKER, Openstack, FlexCore JPlugins	planned
Negotiation routine with VIM	planned
Negotiation routine with VNFM	planned
SDN controller interaction	planned
Support multiple VNFMs	not in plans

The core of JOX framework is technology-agnostic, and is able to support 4G/5G technologies through the plugin architecture, as it is shown in Fig. 4 for RAN and CN elements. Cloud-RAN concepts and 5G technologies can be easily supported, using the necessary plugins that will interact with the JOX Core.

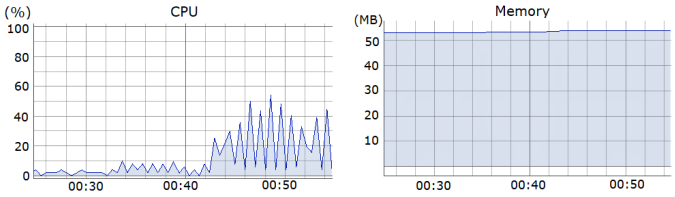
V. Evaluation

The goal of the evaluation process is to investigate the JOX footprint, deployment times and network slicing operational efficiency. We first present an analysis on the JOX footprint, regarding issues like memory usage and CPU power consumption. We then describe network slicing related performance testing and investigate service chaining deployment times, service scaling, and REST API response times.

In principle, a number of variables can be tuned which could affect the effectiveness of the orchestration procedures. For example, such parameters are the number of slices, the number of models per slice, number and type of services or the number and type of host machines, and statistical parameters, like the



(a) CPU overhead



(b) JAgent with FlexRAN

Fig. 8: PyVMMonitor profiling reports for JOX footprint

service request arrival rate distribution or the service lifetime distribution.

A. JOX Footprint Analysis

In order to perform the necessary footprint evaluation, we installed JOX in a 64bit Ubuntu Linux VM with 5GB memory and 2 cores operating in 64 bit host equipped with an Intel i5-5200U and 12GB memory. For each *JModel*, all the instantiated VMs were LXC container-based and the connectivity is provided using the Linux bridging driven by Juju.

In Fig.8 presents the VM footprint of JOX using the PyVmMonitor profiler for eclipse (10 samples/per second). In all cases, a *JAgent* is activated to report statistics regarding CPU, memory, storage, and network of the host machine. Inside the *JAgent*, we exploit the *psutil* linux lib, with a report interval of 10 secs. In Fig. 8(a) the CPU and the memory footprint is presented when deploying a single slice with 1 JModel including the `mysql` service managed by its charm[4] and hosted in 1 LXD container. It can be seen that during the startup, the CPU utilization is close to 4%, while the memory consumption is 47MB on average. During transient state the average CPU utilization is close to 20%, but is decreasing after deployment while the memory consumption is increasing to 57MB. In principle a *vanila JSlice* deployment will affect the memory usage (because of heap consumption), but not the CPU utilization. The CPU utilization is increased when the API is used to retrieve information or activate services. In Fig.8(b), a single slice is deployed (1 model, with 1 machine and 1 service), where 3 remote clients are generating HTTP GET requests to retrieve the serialized *JSlice* objects (internally JOX retrieves this information from REDIS). It can be observed that such requests greatly affects the CPU performance, while the effects in memory consumption are negligible.

Another very important issue to investigate is related to JOX scalability. Fig. 9 shows the effects of increasing the number of JSlices/JModels on the memory usage. In our experiments, every model is a VNF service chain, deployed in LXC. As

[4]https://jujucharms.com/mysql/

TABLE IV: JOX performance Metrics

Description	Average value
JOX loading time	70 ms
Access the configuration file of JOX	61.3 ms
Retrieve info for JUJU models	144 ms
Retrieve JSlice configuration	57 ms
Retrieve serialized JSlice json object	58.3 ms
JSlice loading times	
Empty, with no models	1077 ms
1 model, 1 service, 1 machine	33 sec
2 models with 2 chained services, 2 machines each	7.37 min

expected, the memory footprint increases with the number of *JModels*, however its amount is irrelevant of the service deployed and remains constant among different type of slices. Such a constant increase is because JOX only keeps the service status and functionalities, and delegates service management and maintenance to the underlying VNFM. The memory usage of course of the overall system (JOX, VNFM and VIM) is increased with the number of deployed services.

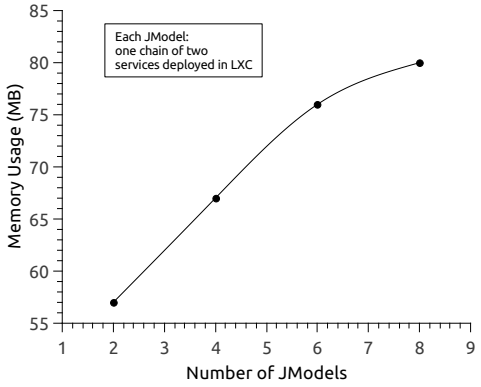


Fig. 9: Effect of increasing the number of slices/models on the memory usage. Every model is a chain of two services (either mysql-OAI HSS, or mysql-apache2) deployed in LXC.

Other relevant performance metrics are presented in Table IV, where the response times are the ones reported by the HTTP client when using the REST APIs. These are machine-depended metrics, however it can be seen that the JOX loading time is very low. The reason is that in the back-end, JOX exploits the non-SQL REDIS server, where all the GET, SET, DELETE operations are extremely fast.[5] In a local environment, under normal load conditions, any information/configuration stored or retrieved in sub-second timing.

Regarding *JSlice* deployment/updates/deletion times, an important observation can be made. As shown in Table IV, the JOX's response time for the deployment of a *JSlice* with two models, carrying two service chains can be close to 7 min. This is the timing seen by JOX based on the interaction and reports received by the JUJU VNFM. Note however that the time to actually bring up the service greatly depends on the chained VNFs and their associated charms, the related hook and the actual scripts that needs to be executed. For example, the HSS service deployment time can be in the order of few

[5]See also the official REDIS website for benchmarking tools.

minutes, whereas in case of the eNodeB it increases to tens of minutes. Note also that as in any real deployment hardware dependencies greatly affect the performance of the integrated system.

B. Use Case: Building LTE VNF chains with JOX

JOX exploits a rich set of functions to enable network slicing through Juju VNFM and charms. Each charm encapsulates every underlying LTE network module as a VNF, leveraging the OAI platform [11].[6] As also shown in Fig.1, for every slice the OAI solution can be chained using Juju relationship hooks and appropriate VNFs. For example in a Cloud-RAN setting, different functional splits can be exploited between the RRU and the BBU VNFs.[7]

In the considered use case, JOX orchestrates the deployment of standard LTE chain (eNB, MME, SPGW,MySQL, HSS) for a new *JSlice*.[8] Throughout this use case, OAI service chains were deployed as real-time LTE platform. With the support of Juju VNFM, JOX has the ability to fully automate the deployment of a LTE network service chain in different execution environments ranging from physical machine to a container or a VM.

The physical server infrastructure was based on commodity Linux-based machines, equipped with 6-cores i7-3930K CPU at 3.2GHz and 16GB of RAM. Also two virtualization environments (Linux LXC containers and KVM) were considered as the targets to deploy the LTE services. We highlight that during the deployment for all the VNFs all the kernel dependencies were automatically satisfied (because of the scripting inside the charms installation hooks). In Juju, a common life-cycle for a service has the following order (1) initialization: where the target environment is instantiated, such as LXC, KVM, or physical machine; (2) installation: where the service is installed on the environment; (3) configuration: where the service is reconfigured; (4) start and stop: where the service is started or stopped depending on the whether the service relationships are met, and (5) relationship: where the service chain is built and dependencies are met. In our setup during installation, the Juju charms were available from Juju remote repositories.

From Figure 10, it can be observed that the initialization phase in the case of LXC is faster and uniform for all the services, where as in case of KVM, it is higher and has a high variability. We observe that the installation delay dominates in most of the service life-cycle, because the services chains are deployed and built from source. The installation time can be reduced drastically when deploying the service from a local package or an image.

We highlight that when chains are deployed, service dependencies may exist. For instance between MySQL and HSS, the relationship cannot be built until the HSS is installed and configured. This is the same between MME and SPGW/HSS, and between eNB and MME. This imposes time delays that cannot be avoided, because of the way the relation hooks operate in Juju. On the other hand, JOX orchestrates the service deployment and automatically handles dependencies and conflicts through JUJU, without requiring any other action to be taken.

[6]OAI charms can be found at jujucharms.com/q/oai

[7]Example can be found at jujucharms.com/u/navid-nikaein/oai-5g-cran/

[8]Example of this service chain can be found at jujucharms.com/u/navid-nikaein/oai-nfv-4g/

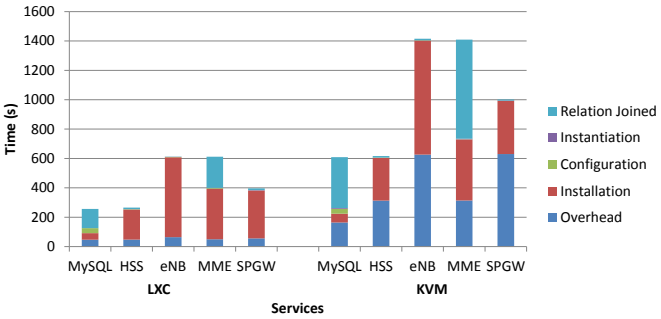


Fig. 10: Deployment time of VNF chains

VI. Conclusions and Future Work

Network slicing is becoming extremely important towards 5G, for a number of stakeholders like vertical industries that are now trying to enhance their service offering as much as possible. In this work, we presented JOX, an open-source orchestrator for the virtualized mobile network that inherently supports network slicing. The JOX architecture was presented together with the core functionalities and mechanisms used. We also presented JOX footprint analysis and a LTE chaining use case where JOX can be used as a fast prototyping tool for network slices provisioning and operation. Our future plans include the enrichment of the JOX features offering, in the directions of slice life-cycle management, plugins extensions, auto-placement and auto-scaling. We are also in the process of prototyping three additional use cases, namely Cloud-RAN with flexible functional split and RAN Sharing with multiplexed radio resources within the same eNodeBs. We are also planning to investigate interaction with SDN controllers for orchestrating the necessary transport network resources.

Acknowledgement

Research and development leading to these results has received funding from the European Union (EU) Framework Programme under Horizon 2020 grant agreement no. 762057 for the 5G-PICTURE project and no. 761913 for the SliceNet project.

REFERENCES

[1] N. Alliance, "Description of Network Slicing concept," *NGMN 5G P*, vol. 1, 2016.

[2] P. Rost, A. Banchs, I. Berberana, M. Breitbach, M. Doll, H. Droste, C. Mannweiler, M. A. Puente, K. Samdanis, and B. Sayadi, "Mobile network architecture evolution toward 5G," *IEEE Communications Magazine*, vol. 54, no. 5, pp. 84–91, May 2016.

[3] "Open Network Automation Platform (ONAP)." [Online]. Available: http://onap.org/

[4] A. Manzalini, D. R. Lopezz, H. Lonsethagen, L. Suciu, R. Bifulcozz, M.-P. Odinixi, G. Celozzixiii, B. Martinixv, F. Rissoy, J. Garayx *et al.*, "A unifying operating platform for 5g end-to-end and multi-layer orchestration," in *Network Softwarization (NetSoft), 2017 IEEE Conference on*. IEEE, 2017, pp. 1–5.

[5] K. Samdanis, X. Costa-Perez, and V. Sciancalepore, "From Network Sharing to Multi-tenancy: The 5G Network Slice Broker," *IEEE Communications Magazine*, 2016.

[6] G. ETSI, "001 (MANO), online https://www. etsi. org/deliver/etsi_gs," *NFV-MAN/001_099/001/01.01. 01_60/gs_NFV-MAN001v010101p. pdf*.

[7] "Juju VNFM Framework," http://jujucharms.com/.

[8] "Open source MANO (OSM), url= "http://osm.etsi.org/"."

[9] "OpenStack Tacker." [Online]. Available: http://wiki.openstack.org/wiki/Tacker/

[10] X. Foukas, N. Nikaein, M. M. Kassem, M. K. Marina, and K. P. Kontovasilis, "FlexRAN: A Flexible and Programmable Platform for Software-Defined Radio Access Networks." in *CoNEXT*, 2016, pp. 427–441.

[11] N. Nikaein, M. K. Marina, S. Manickam, A. Dawson, R. Knopp, and C. Bonnet, "OpenAirInterface: A flexible platform for 5G research," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 33–38, 2014.

[12] N. Nikaein, E. Schiller, R. Favraud, K. Katsalis, D. Stavropoulos, I. Alyafawi, Z. Zhao, T. Braun, and T. Korakis, "Network Store: Exploring Slicing in Future 5G Networks," in *MobiArch*. ACM, 2015, pp. 8–13.

[13] B. Astuto *et al.*, ""A Survey of SoftwareDefined Networking Past, Present, and Future of Programmable Networks"," *IEEE Communications surveys & tutorials*, vol. 16, no. 3, 2014.

[14] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network Function Virtualization: State-of-the-art and Research Challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. c, pp. 1–1, 2015.

[15] I. Da Silva, S. E. El Ayoubi, O. M. Boldi, Ö. Bulakci, P. Spapis, M. Schellmann, J. F. Monserrat, T. Rosowski, G. Zimmermann, D. Telekom *et al.*, "5G RAN Architecture and Functional Design."

[16] K. Katsalis, N. Nikaein, E. Schiller, A. Ksentini, and T. Braun, "Network slices toward 5g communications: Slicing the lte network," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 146–154, 2017.

[17] M. Peuster and H. Karl, "Understand your chains: Towards performance profile-based network service management," in *Software-Defined Networks (EWSDN), 2016 Fifth European Workshop on*. IEEE, 2016, pp. 7–12.

[18] R. Mijumbi, J. Serrat, J. l. Gorricho, S. Latre, M. Charalambides, and D. Lopez, "Management and orchestration challenges in network functions virtualization," *IEEE Communications Magazine*, vol. 54, no. 1, pp. 98–105, January 2016.

[19] "OpenBATON, url= "http://openbaton.github.io"."

[20] A. Csoma, B. Sonkoly, L. Csikor, F. Németh, A. Gulyás, D. Jocha, J. Elek, W. Tavernier, and S. Sahhaf, "Multi-layered service orchestration in a multi-domain network environment," in *Software Defined Networks (EWSDN), 2014 Third European Workshop on*. IEEE, 2014, pp. 141–142.

[21] B. Sonkoly, J. Czentye, R. Szabo, D. Jocha, J. Elek, S. Sahhaf, W. Tavernier, and F. Risso, "Multi-domain service orchestration over networks and clouds: a unified approach," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. ACM, 2015, pp. 377–378.

[22] "Hurtle Project," http://hurtle.it/.

[23] "Technology Independent Information Model for Network Slicing." [Online]. Available: IETF: draft-qiang-coms-netslicing-information-model-00