Deliverable 3.1

Design and Prototyping of SliceNet Virtualised Mobile Edge Computing Infrastructure

Editor:	University of the West of Scotland
Deliverable nature:	Report (R)
Dissemination level: (Confidentiality)	Public (PU)
Contractual delivery date:	28/02/2018
Actual delivery date:	30/03/2018
Suggested readers:	Infrastructure providers; Communication service providers, Digital service providers; Network operators; Vertical industries
Version:	1.0
Total number of pages:	77
Keywords:	5G, MEC; Infrastructure; Data Plane programmability; ME platform; ME services; ME Apps; Network slicing; Virtualisation

Abstract

This document reports all the activities related to the design and prototyping of a virtualised Mobile/Multi-access Edge Computing (MEC) infrastructure segment as part of the SliceNet end-to-end slicing-friendly infrastructure. The design and prototyping ensures that the SliceNet MEC infrastructure is fully compliant with the ETSI MEC architecture and offers enablers for network slicing. SliceNet MEC architecture comprises a low-latency MEC platform, and a programmable Data Plane, multi-tenanted infrastructure, with Management Plane functionality support taken into account. Furthermore, representative MEC applications are described to show the practical use cases of the SliceNet MEC architecture.

Disclaimer

The research leading to these results has received funding from the European Union Horizon 2020 Programme under grant agreement number H2020-ICT-2016-2/761913.

Impressum

[Full project title] End-to-End Cognitive Network Slicing and Slice Management Framework inVirtualised Multi-Domain, Multi-Tenant 5G Networks

[Short project title] SliceNet

[Number and title of work-package] 5G Integrated Multi-Domain Slicing-Friendly Infrastructure

[Number and title of task] T3.1. Virtualised Mobile Edge Computing Infrastructure

[Document title] Design and Prototyping of SliceNet Virtualised Mobile Edge Computing Infrastructure

[Editor: Name, company] University of the West of Scotland

[Work-package leader: Name, company] Navid Nikaein, Eurecom

[Estimation of PM spent on the Deliverable]

Copyright notice

Executive Summary

Mobile/Multi-access Edge Computing (MEC) has increasingly become an integral part of a 5G mobile network due to the considerable benefits that can be gained from this architectural enhancement, taking advantage of edge cloud computing and software networking capabilities, among other technologies. The SliceNet MEC architecture described in this document aims to provide an execution system for accelerated slice deployment, and offers advantageous support for service quality assurance, which is especially beneficial for services and use cases that demand ultra-low latency and/or high throughput for instance.

Specifically, the following achievements are reported in this deliverable:

- An advanced MEC system that is fully compliant with the ETSI MEC architecture is defined, as part of the end-to-end SliceNet infrastructure;
- A fully functional MEC platform, named Low-Latency MEC (LL-MEC) platform, is presented with implementation details reported, including essential MEC services and Mobile Edge Application Framework and Software Development Kit (SDK) to allow further programmability;
- A programmable, multi-tenanted Data Plane is designed and prototyped with experimental empirical results illustrated, as a slicing-friendly infrastructure for MEC and other non-Radio Access Network (RAN) segments, enabling Quality of Service (QoS) aware slicing;
- The management and orchestration for the MEC system is discussed with specific open source managers and orchestrators considered and candidate solutions explored;
- Finally, a number of preliminary use cases that can explore the proposed MEC system for improved performance are presented, as representative yet not comprehensive examples to be further evolved and aligned with SliceNet primary use cases.

List of Authors

Company	Author	Contribution
UWS	Jose Maria Alcaraz Calero; Qi Wang; Zeeshan Pervez; Hector Marco; Rubén Ricart Sánchez; Pedro Malagon; Antonio Matencio	Data Plane Programmability and Virtualized Infrastructure; ETSI Mobile Edge Computing Standard and Compliance; Abstract; Executive Summary; Introduction; Conclusion
ECOM	Navid Nikaein; Tien Thinh Nguyen; Xenofon Vasilokos	MEC Platform; Management and Orchestration; ME Services and Use Cases
DellEMC	Thuy Truong; Zdravko Bozako	Management and Orchestration
TEI	Ciriaco Angelo	Review of the Deliverable
OTE	Agapiou Georgios	Review of the Deliverable
ORO	Marius Iordache; Elena Oproiu	ME Services and Use Cases; review of the Deliverable

Table of Contents

List of Figures

List of Tables

Abbreviations

3G	Third Generation (mobile/cellular networks)
3GPP	3G Partnership Project
4G	Fourth Generation (mobile/cellular networks)
5G	Fifth Generation (mobile/cellular networks)
5G PPP	5G Infrastructure Public Private Partnership
AE	Autoscaling Engine
AMF	Access and Mobility Function
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
ARP	Allocation Retention Parameters
ASIC	Application-Specific Integrated Circuit
BPF	Berkeley Packet Filters
CN	Core Network
CoAP	Constrained Application Protocol
COTS	Commercial Off-The-Shelf
CP	Control Plane
CPU	Central Processing Unit
CQI	Channel Quality Indicator
CU	Central Unit
DDoS	Distributed Denial of Service
DL	Downlink
DMA	Direct Memory Access
DNS	Domain Name System
DP	Data Plane
DPDK	Data Plane Development Kit
DU	Distribution Unit
EAL	Environment Abstraction Layer
eNB or eNodeB	evolved NodeB
EPC	Evolved Packet Core
EPS	Evolved Packet System
ETH	Ethernet
ETSI	European Telecommunications Standards Institute
FCAPS	Fault, Configuration, Accounting, Performance, Security
FCS	Frame Check Sequence
FDD	Frequency Division Duplex
FIFO	First In First Out
FMS	Fault Management System
FPGA	Field Programmable Gate Arrays
GBR	Guaranteed Bit Rate
GRE	Generic Routing Encapsulation
GPRS	General Packet Radio Service
GTP	GPRS Tunnelling Protocol
GW	Gateway

GWCN	Gateway CN (GWCN)
HDD	Hard Disk Drive
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
ID	Identifier
IoT	Internet of Things
IP	Internet Protocol
IPFIX	IP Flow Information Export
iSCSI	Internet Small Computer Systems Interface
JCC	JOX Clouds Controller
JOX	Juju-based Orchestrator
JSC	JOX Slices Controller
KPI	Key Performance Indicator
KVM	Kernel-based Virtual Machine
LAN	Local Area Network
LCM	Lifecycle Management
LCX	Linux Container
LL-MEC	Low Latency MEC
LPM	Longest Prefix Match
LTE	Long Term Evolution
M2M	Machine to Machine
MANO	Management and Orchestration
ME	Mobile Edge
MEC	Mobile/Multi-access Edge Computing
MEO	Mobile Edge Orchestrator
MIB	Management Information Base
MME	Mobility Management Entity
ML2	Modular Layer 2
MPLS	Multiprotocol Label Switching
MQTT	Message Queuing Telemetry Transport
NAPI	New API
NAS	Non-Access Stratum
NB-IoT	Narrow Band IoT
NETCONF	Network Configuration Protocol
NFS	Network File System
NFV	Network Function Virtualisation
NFVI	NFV Infrastructure
NGBR	Non-GBR
NIC	Network Interface Card
NS	Network Service
NSE	Network Slicing Engine
NSI	Network Slice Instance
NSSI	Network Slice Subnet Instance
ODL	OpenDayLight
ONF	Open Networking Foundation
OF	OpenFlow

OPEX	Operational Expenditure
OSS	Operations Support System
OVS	Open vSwitch
OVSDB	OVS Database
PCF	Policy Control Function
PCI	Peripheral Component Interconnect
PCIe	PCI Express
PCRF	Policy and Charging Rules Function
PGW	Packet Data Network Gateway
PLMN	Public Land Mobile Network
PMD	Poll Mode Driver
PoP	Point of Presence
PSA	Portable Switch Architecture
QoE	Quality of Experience
QoS	Quality of Service
RAB	Radio Access Bearer
RAN	Radio Access Network
REST	Representational State Transfer
RFC	Request for Comments
RIB	RAN Information Base
RNIS	Radio Network Information Service
RRC	Radio Resource Control
RRU	Remote Radio Unit
RTP	Real-time Transport Protocol
SCSI	Small Computer System Interface
SDK	Software Development Kit
SDN	Software-Defined Networking
SISO	Single-Input Single-Output
SGW	Serving Gateway
SLA	Service Level Agreement
SliceNet	End-to-End Cognitive Network Slicing and Slice Management Framework in Virtualised Multi-Domain, Multi-Tenant 5G Networks
SMF	Session Management Function
SON	Self-Organizing Networks
TCAM	Ternary Content Addressable Memories
TCP	Transmission Control Protocol
TEID	Tunnel Endpoint Identifier
TOS	Type of Service
TTI	Transmission Time Interval
UDP	User Datagram Protocol
UE	User Equipment
UL	Uplink
UP	User Plane
UPF	User Plane Function
USIM	Universal Subscriber Identity Module
VIM	Virtual Infrastructure Manager

VLAN	Virtual LAN
VNF	Virtual Network Function
VNFM	VNF Manager
VNFO	VNF Orchestrator
VM	Virtual Machine
VXLAN	Virtual Extensible LAN
ZC	Zero Copy

Definitions

- Mobile/Multi-access Edge Computing (MEC) segment: This primarily refers to the MEC infrastructure and MEC platform, together with essential MEC-level management and orchestration. It is noted that Mobile Edge Computing is equivalent to the ETSI (European Telecommunications Standards Institute) Multi-access Edge Computing when focused on the mobile access.
- Slicing-friendly infrastructure: SliceNet infrastructure that explores Data Plane programmability, software networking, cloud computing and other related technologies to allow QoS awareness and control at the infrastructure level to facilitate network slicing that aims to meet specific Service Level Agreement (SLA).

1 Introduction

Mobile/Multi-access Edge Computing (MEC) pushes cloud computing capabilities and resources to the edge of a network to allow end user and vertical applications to exploit this new IT/Telco service environment to achieve improved Quality of Service (QoS) and/or Quality of Experience (QoE). Therefore, MEC has emerged as a value-added paradigm to be increasingly integral and important in an end-to-end networking architecture. SliceNet fully adopts this vision and considers a MEC segment as an integral part of the SliceNet architecture [1].

1.1 Objectives

Virtualised MEC infrastructure in particular features the end-to-end SliceNet infrastructure, and more generally to future 5G systems, which regards the establishment of slice-friendly cross-domain physical and virtual infrastructure layers, to provide an execution foundation for the upper layers in the SliceNet architecture. Within this the context, SliceNet will leverage and extend the MEC segment to support the emerging 5G slicing paradigm, and the resulted vertical use cases. The following specific objectives are identified, based on the description of the work:

- Establish a MEC segment that is fully compliant with the ETSI MEC architecture under standardisation. This assures the standard compliance of the proposed SliceNet MEC segment and therefore all the interoperability with other ETSI MEC-compliant systems;
- Explore enablers for slicing friendliness and multi-tenancy for the MEC infrastructure. Slicing-friendly infrastructure will facilitate the network slicing operations in the system especially in terms of providing enablers for QoS-aware or even QoS-assured network slicing. Moreover, multi-tenant isolation in the infrastructure layer will allow the infrastructure provider to offer the same infrastructure to multiple operators, among other potential users (tenants);
- Create an execution system to accelerate slice deployment and support QoS. A Low Latency MEC (LL-MEC) platform will largely realise the ETSI MEC platform functionality required to run MEC applications on the SliceNet virtualisation infrastructure and provide MEC services. The MEC platform and the slicing-friendly and multi-tenanted infrastructure combined together will deliver the core of the SliceNet MEC execution system;
- Investigate the essential management and orchestration support for the MEC infrastructure and MEC platform.

1.2 Approach and Methodology

The main technical approach taken in this task is to align the design and prototype of SliceNet MEC segment with the ETSI MEC architecture. SliceNet MEC segment mainly includes slicing friendly and multi-tenant aware infrastructure and LL-MEC platform, together with essential management and orchestration at the MEC level.

- The slicing friendliness of the infrastructure is mainly achieved through programming the Data Plane to enable traffic flow classification and rule-based control, leading to

QoS support required by the slice to fulfil the SLA of a service. This will enable QoS-aware slice deployment.

- The multi-tenancy in the infrastructure layer is mainly achieved through the combination of virtual infrastructure management and software networking. The MEC-Core network Data Plane segmentation is achieved via availability zones provisioned by virtual infrastructure management.
- The Control Plane and Data Plane programmability is supported through OpenFlow and FlexRAN control protocols effectively realising the ETSI MEC Traffic Rule Control and Radio Network Information services, among other MEC services.
- Additional service quality assurance is enabled by the support of Mobile Edge services provided by the SliceNet MEC segment. These MEC services are especially beneficial for applications and use cases that demand ultra-low latency and/or high throughput for instance.

1.3 Document Structure

The remainder of the document is organised as follows. Section 2 reviews the ETSI MEC reference architecture and defines the scope and focus of this deliverable in terms of establishing the ETSI MEC-compliant SliceNet MEC segment. Section 3 presents details in the technical approach of designing and prototyping the SliceNet virtualised slicing-friendly and multi-tenanted MEC infrastructure especially the programmable Data Plane. Section 4 elaborates the SliceNet LL-MEC platform achieving all the main functionalities defined in ETSI MEC platform for MEC applications and services. Section 5 investigates the necessary management and orchestration for the MEC infrastructure and platform, in particular, focusing on reference implementation for each functional block in Management and Orchestration (MANO). Section 6 describes a number of MEC applications and use cases to highlight the potential usage of the SliceNet MEC segment and benefits that it can bring to different application scenarios.

2 ETSI Mobile Edge Computing Standard and Compliance

This section reviews the ETSI MEC reference architecture, and identifies the scope of MEC components and contribution in SliceNet for achieving an ETSI MEC compliant MEC segment. Moreover, the design challenges to achieve slicing-friendly infrastructure are highlighted.

2.1 The Overall Architecture

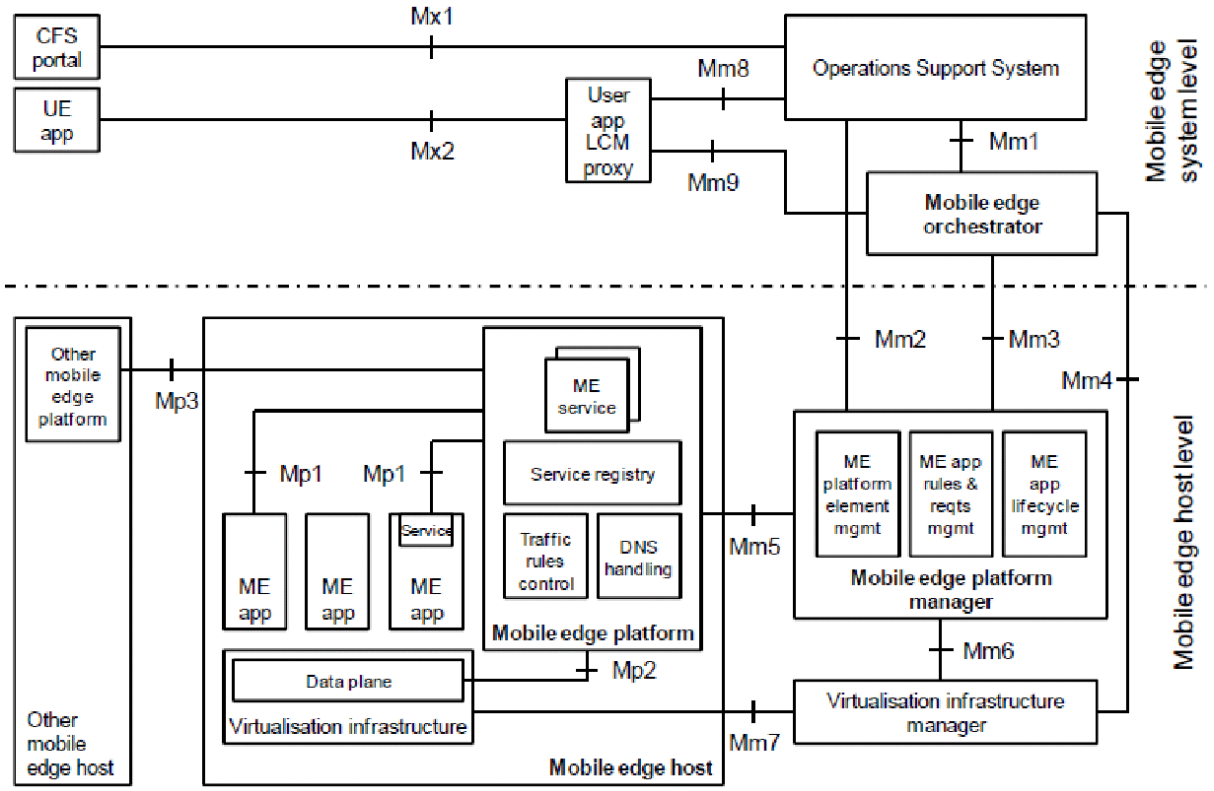


Figure 1. ETSI MEC reference architecture [2]

Figure 1 illustrates the ETSI MEC reference architecture [2]. The following summary highlights the key MEC functional components, which are grouped into two subsystems, together with corresponding reference points (Mp reference points for the Mobile Edge Platform, and Mm ones for the MEC management plane, respectively):

- Mobile Edge host subsystem: It contains Virtualization infrastructure (including the Data Plane), a Mobile Edge Platform (including ME Service, Service Registry, Traffic Rules Control and Domain Name System (DNS) handling) and ME applications (services). An ME service can be provided by the Mobile Edge platform or by an ME application (through service registration via Mp1 reference point).
- Mobile Edge management subsystem: The host-level management consists of a Mobile Edge platform manager (including ME platform element management, ME application rules & requirements management and ME app lifecycle management) and a Virtualisation infrastructure manager, which manages the Mobile Edge platform via Mm5 and the Virtualisation infrastructure via Mm7, respectively. The system-level management contains a Mobile Edge Orchestrator (MEO), which maintains an overview of the MEC system such as deployed ME hosts, available resources and ME services, topology etc., and an ME app catalogue. MEO interacts

with Mobile Edge Platform via Mm3 for the management of the ME application lifecycle, application rules and requirements, tracking available ME services, and Virtualisation infrastructure manager via Mm4 for managing virtualised resources of the Mobile Edge host, including tracking available resource capacity and managing ME application images.

In addition, an optional User app LCM proxy is proposed to allow the User Equipment (UE) to request on-boarding, instantiation, termination and possible relocation (if supported) of the UE's ME application and to be informed of the state of the ME application. An ME application package is on-boarded by the MEO, and the application can be associated with a set of application rules (especially traffic rules), and resources, services and/or QoS requirements (e.g., delay constraint) etc., and these requirements are validated by the MEO (and if necessary, adjusted to be compliant with the operator's policies). The MEO also selects the appropriate Mobile Edge host to trigger the instantiation of the ME application based on these requirements.

The Virtualisation Infrastructure provides the virtualisation resources to run the ME application as a Virtual Machine (VM). The Mobile Edge Platform provides an environment that enables the ME application to discover, advertise, provide and consume ME services and to run on the particular virtualisation infrastructure.

The Mobile Edge Platform receives traffic rules from the Mobile Edge Platform Manager via Mm5 (or applications/services), and poses these rules to the Data Plane via Mp2. The Data Plane then executes the rules and consequently routes the traffic as desired for the MEC use case, e.g., among applications, services, network entities and various networks.

2.2 Key Components and Interfaces Considered in SliceNet

Figure 2 depicts the mapping of the ETSI functional elements to the tasks in this project based on the scoping of the tasks, which also illustrates the compliance of the SliceNet MEC with the ETSI MEC reference architecture.

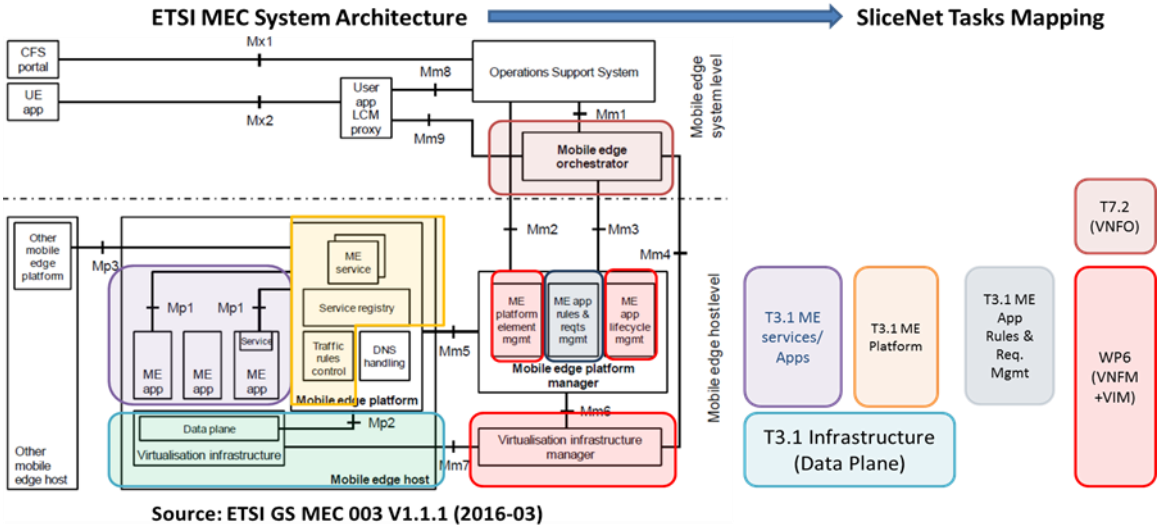


Figure 2. ETSI MEC compliance, scoping and tasks mapping

In light of the scope of this project, to achieve an ETSI MEC compliant MEC segment, most of the functional elements in the standard reference architecture are explored.

Firstly, a Mobile Edge Platform is designed and prototyped, and a number of ME services are also implemented to support use cases. Consequently, an ETSI MEC compliant Mobile Edge host subsystem is achieved. It is noted that the DNS Handling element in the Mobile Edge platform is optional and thus it is not covered. This MEC platform part is in yellow (MEC platform) and purple (MEC applications/services) in Figure 2 and presented in Sections 3 and 6, respectively.

Secondly, regarding the virtualisation infrastructure, the aim is to achieve a programmable, multi-tenanted Data Plane for a slice-friendly MEC infrastructure. This infrastructure and Data Plane part is highlighted in light blue in Figure 2 and is addressed in Section 4.

Thirdly, for the Mobile Edge management subsystem in red and brown in Figure 2, it is planned that this belongs to the Management Plane tasks to be further investigated in the corresponding following Work Package WP6 and task T7.2. In this deliverable, however, initial considerations regarding the Management Plane for the MEC segment are discussed in Section 5 to inform the design and implementation of the subsequent management tasks in other work packages.

2.3 Design Challenges for Slicing-Friendly Infrastructure

There are a number of challenges across different planes to achieve slicing-friendly infrastructure as envisioned in SliceNet:

- Data Plane: To have different logical data paths (lanes/queues) into the networking infrastructures in order to allow traffic to flow through such lanes/queues without horizontal collisions/interference between lanes/queues. The Data Plane paths should be programmed to designated QoS-aware ones and best effort ones to allow the realisation of paths with controllable QoS and those with best effort delivery service, respectively. They are isolated from each other so that the traffic in each category would not affect that of the other category for effective management and fair provisioning even for the best effort slices.
- Control Plane: To have a well-known semantics on the different priorities associated with each of the data path lanes/queues of the network infrastructure in order to allow the foundations of the specification of the definition of “network slicing”.
    - Hardware Isolation: To have a logical architecture generic enough to represent the main technologies to allow hardware resource sharing, e.g., Virtualization and Encapsulation.
- Application Plane: to enable Coordinated Control and User Plane programmability across Radio Access Network (RAN) and Core Network (CN) with real-time access to radio network information the flexibility to develop control apps on the top of the platform SDK and the support of low latency control apps and their priorities and deadlines.
- Management Plane: To have well-known semantics on the different rules used to identify how to select the lane/queue that is to be assigned to each of the network flows passing by the infrastructure and to select how much traffic is allowed over a given time period in order to allow the essential functionality of “network management”.
    - Multi-tenancy: To have an infrastructure to allow the isolated use of shared resources.

- Mobility: To have an infrastructure to allow user mobility along network resources.

- Network Segments: To have a logical architecture inclusive enough to represent the main network segments involved in multiple administrative domains involved in the end-to-end communications. As shown in Figure 3, it includes the following connectivity: (Domain 1) Vertical business enterprise network -> RAN -> MEC segment -> CN -> Inter-Domain Network -> (Domain 2) CN -> MEC -> RAN -> Vertical business enterprise network. Moreover, it is envisioned that the 5G/4G components/network functions will be distributed in the appropriate segments:
  - RAN: (5G) Remote Radio Unit (RRU), or Distributed Unit (DU);
    (4G) Remote Radio Head (RRH)
  - Edge: (5G) Central Unit (CU);
    (4G) BaseBand Unit (BBU)
  - CN: (5G) Access and Mobility Management Function (AMF), Session Management Function (SMF), User Plane Function (UPF), Authentication Server Function (AUSF), Unified Data Management (UDM), Policy Control Function (PCF) etc.;
    (4G) MME (Mobility Management Entity), Serving Gateway (SGW), Packet Data Network Gateway (PGW), Home Subscriber Server (HSS), Policy and Charging Rules Function (PCRF).

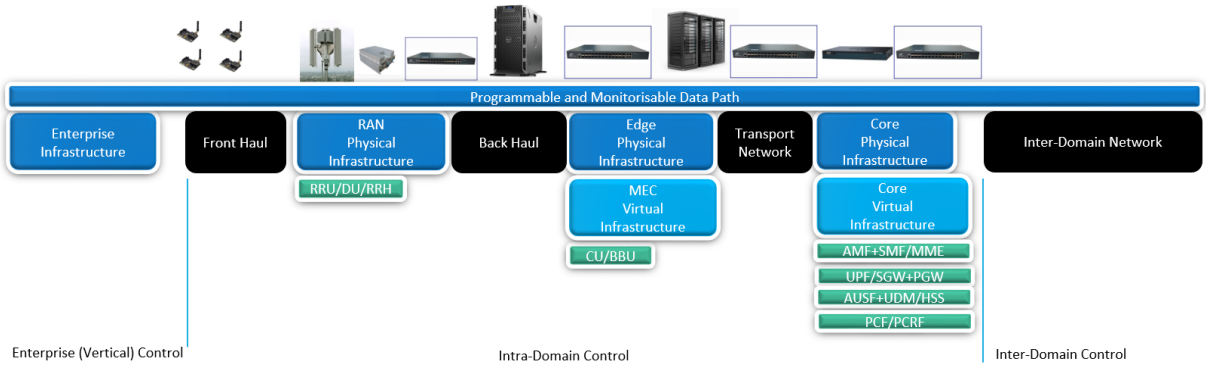


Figure 3. Data path (and 5G/4G network functions) across different network segments

3 Low Latency MEC (LL-MEC) Platform for Software-Defined Mobile Network

This section focuses on the design and prototyping of the SliceNet MEC platform, LL-MEC.

Considering one of the key requirements for MEC, programmability, Software-Defined Networking (SDN) is a promising solution and already exploited extensively in non-mobile networks. It provides a network architecture where the Control Plane has been migrated from physical network devices with a well-defined protocol, e.g., OpenFlow [3]. OpenFlow is an SDN standard by ONF (Open Networking Foundation), and defines the southbound interfacing of a compliant SDN controller with the forwarding plane (Data Plane); it is further discussed in Section 4. The underlying infrastructure can therefore be abstracted creating opportunities for innovation and customization of network applications. The noticeable success in non-mobile networks made by SDN gives the initiatives to apply it onto the CN of Long Term Evolution (LTE) [4]. With the separation of Control Plane and Data Plane, SDN virtualizes the mobile network components, such as MME, Control Plane of SGW (or S-GW) and PGW (or P-GW) as potential MEC applications. The programmability of the core network provided by SDN is exactly where MEC can leverage and extend its programmability in RAN and further delegate control decisions. Not surprisingly, there have been considerable research interests on SDN and MEC with most of them focusing on conceptual frameworks but no open source platform for researchers as a reference to evaluate the benefits of SDN-enabled MEC services. This gives the initiatives of LL-MEC to exploit the interplay between MEC and SDN in exploring and demonstrating coordinated network programmability through an ecosystem of network applications and SDK. Given the open specifications of MEC for vendor implementation, the SDN concept is applied in LL-MEC with OpenFlow [3] and FlexRAN [5] protocols.

3.1 High-level Overview

LL-MEC is a MEC platform thoroughly realizing SDN concept with much design ingenuity as well as many software components. In what follows, we provide an overview of LL-MEC architecture and the design challenges in realizing a low latency MEC platform that not only provides an ETSI-aligned MEC platform but also acts as a CN controller providing a clean separation between Control Plane (CP) and User Plane (UP) or Data Plane (DP) in CN [6]. Figure 4 shows that the MEC application manager lays the foundation for the upper-most layer and provides the programming interfaces (Mp1) for applications to be developed. Standing in the middle layer, the MEC platform includes two main core components, namely Radio Network Information Service (RNIS) and Edge Packet Service (EPS), which manage RAN and CN network services based on the C-plane and D-plane Application Programming Interfaces (APIs) from the abstraction layer respectively. At the bottom-most layer, the eNodeBs and OpenFlow-enabled switches comprises the Data Plane with the information abstracted by the FlexRAN and OpenFlow protocols and exposed through abstraction API (Mp2). The proposed MEC platform operates on a software-defined mobile network consisting of multiple LTE eNodeBs and OpenFlow-enabled switches, whether it is physical or software. Figure 4 depicts the application of LL-MEC to 4G. As seen in the figure, the control and Data Plane are separated, which without loss of generality also applies to 5G. In order to simplify the annotation in the figure, the Control Plane and Data Plane of SGW and PGW are respectively annotated as X-GW-C and X-GW-U, which represent the UPF and to some

extend SMF in 5G-CN architecture. As specified by ETSI, the Mp1 and Mp2 reference points are the interfaces between layers. In the following, we focus on the main design and implementation of LL-MEC, as a reference implementation of a subset of ETSI MEC specification, and highlight the key components to address the latency challenges.

One of the key features of LL-MEC is to provide a unified applications development and programming environment by means of SDK to allow coordinated control decisions to be applied across different network domains, namely RAN and CN. In order to complete LL-MEC, the abstraction protocols, i.e. FlexRAN towards RAN and OpenFlow towards CN, are exploited to facilitate the communication among network elements. These abstraction protocols and their corresponding APIs are developed within the MEC platform for allowing two-way interaction between them. In this way, LL-MEC is able to fulfil the requests coming from limitless edge applications and execute the precise tasks onto the underlying networks, which is exactly the merit of SDN. Moreover, the LL-MEC platform has been designed to support time-critical RAN operations and allow applications to be deployed with different level of priorities when interacting with the platform. The required low latency aspect has been considered throughout the design stages to fully utilise the power of LL-MEC at the network edge.

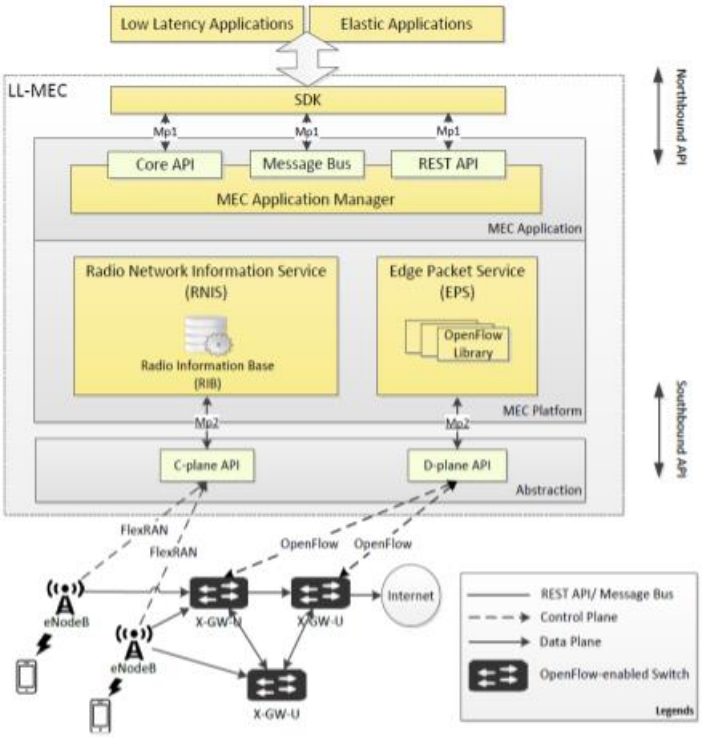


Figure 4. High-level schematic diagram of LL-MEC

Considering latency being the primary feature for a MEC platform to enable an ecosystem of rich edge applications with varieties of needs, LL-MEC distinguishes three types of latencies in its design as follows:

- **User latency:** this represents the end-to-end user transport latency;
- **Control latency:** this captures the latency for the MEC to perform an action, on behalf of an edge application, to the underlying networks, e.g. control and/or monitoring;

- **Application latency:** this represents the latency for an edge application to perform an action to the MEC.

To this end, the key LL-MEC design challenges to realize the low latency MEC platform are listed below:

- The separation of control and Data Plane throughout RAN and CN to have a programmable and coordinated network;
- Coordinated control and user plane programmability across RAN and CN with real-time access to radio network information;
- The scalability with the large number of users and services (i.e. application flows) with QoS support;
- The flexibility for applications and services to be registered as low latency to support the control decisions, their priorities and deadlines.

3.1.1 Workflow of Bearer Establishment with LL-MEC

Figure 5 shows the workflow of how an SDN-based mobile core network operates and interacts with LL-MEC to handle UE initial attach procedures for bearer's establishment. The main point of the sequence diagram starts from the message calls initiated all the way from X-GW-C through LL-MEC to X-GW-U. It is noted that the entity to initiate the API call is not limited to be SGW-C, and MME can start the procedure for default and dedicated bearer's establishment since they have the equal understanding about the bearer setup information.

- **Default Bearer:** As soon as the UE becomes attached to the network with MME and X-GW-C knowing the GTP information, X-GW-C will initiate the procedure to transmit the UE information with the "UE Setup Rule" message to LL-MEC. And then based on the rules, LL-MEC is able to add the UE to its internal information base and setup the OpenFlow rules with the "OF Rules Setup" message in the corresponding switches. By introducing the concept of SDN into mobile network through the integration of OpenFlow-enabled switches, the default bearer can be setup by configuring the OpenFlow rules when UE completes the attach procedure. In addition, notice in Figure that at this point, default bear is established along the path, UE, eNodeB, and X-GW-U switch, and that the UE can access the Internet as normal. This procedure is the same for each "Modify Bearer Request" and "Modify Bearer Response" message for each is the same for S5/S8 bearer
- **Dedicated Bearer:** When a UE attaches successfully to a mobile network, only the default bearer is created. Additional bearers that may be created after the default one are dedicated bearers, which also have different identities, e.g., bearer identifier (ID) and S1 SGW/eNB Uplink (UL)/Downlink (DL) Tunnel Endpoint Identifier (TEID), with default bearer. Another UE Setup Rules API call is required to setup the dedicated bearer right after the "Create Bearer Response" call for the default bearer, as shown in Figure. Like the case of default bearer, the dedicated bearer is established when X-GW-C receives the "UE Setup Rules Response" and exists only along the path between UE and X-GW-U switch.
- **QoS Setup for Guaranteed Bit Rate:** The established dedicated bearer can be either Guaranteed Bit Rate (GBR) or Non Guaranteed Bit Rate (non GBR). For example, if the dedicated bearer is established for voice service, it has to be GBR for guaranteed QoS. In order to have the required QoS for UE, X-GW-C initiates a new API call (QoS Setup Rules) through LL-MEC right after the dedicated bearer is setup. This call is

used not only to reserve the required bandwidth but also to manage the access control based on allocation retention parameters (ARP). This is also shown in the bottom of Figure 5 clearly. When X-GW-U receives the configuration from LL-MEC, it will setup the rules for UE by creating a new meter table or adding into a pre-defined meter group. At this point, UE can access the Internet with guaranteed QoS.

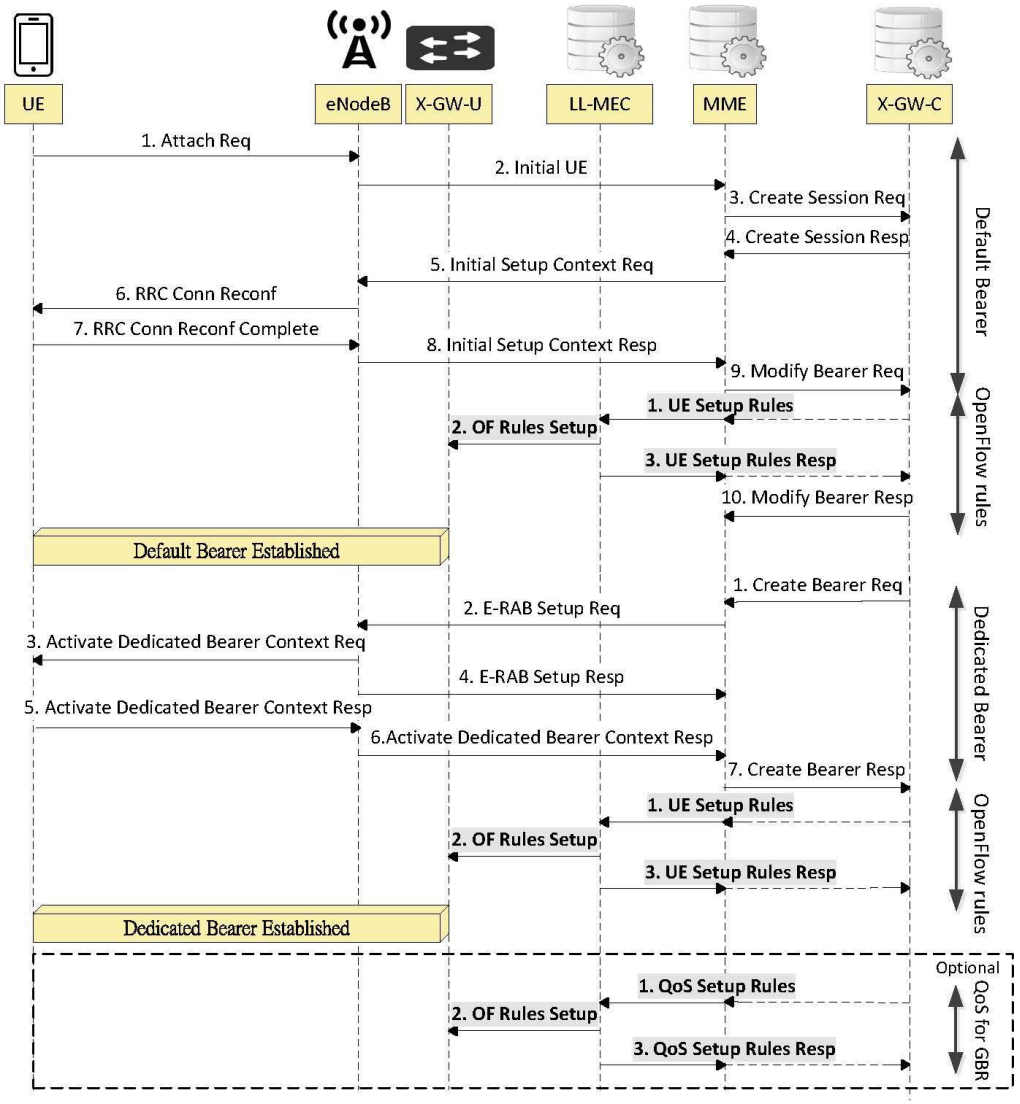


Figure 5. Workflow of bearer establishment with LL-MEC

3.2 Design and Implementation

The layered architecture of the LL-MEC and its main software components are shown in Figure 6. It can be seen that MEC application comprising the upper-most layer manages the Data Plane based on the information gathered through Mp1. The real-time RAN information is provided through a RNIS producer app to the MEC platform. The LL-MEC SDK abstracts the network information through a well-defined northbound interfaces and facilitates access to detailed network information based on which a decision can be made and enforced to the underlying network. The LL-MEC platform includes EPS, which implements ETSI Traffic Rule Control Service, to provide network services based on OpenFlow (OF) APIs located at the abstraction layer. In addition, the LL-MEC platform provides additional services to manage

events, UEs, context, stats, and switches. The eNodeBs and OpenFlow-enabled switches form the Data Plane as the bottom-most layer. The OpenFlow library is based on Libfluid.

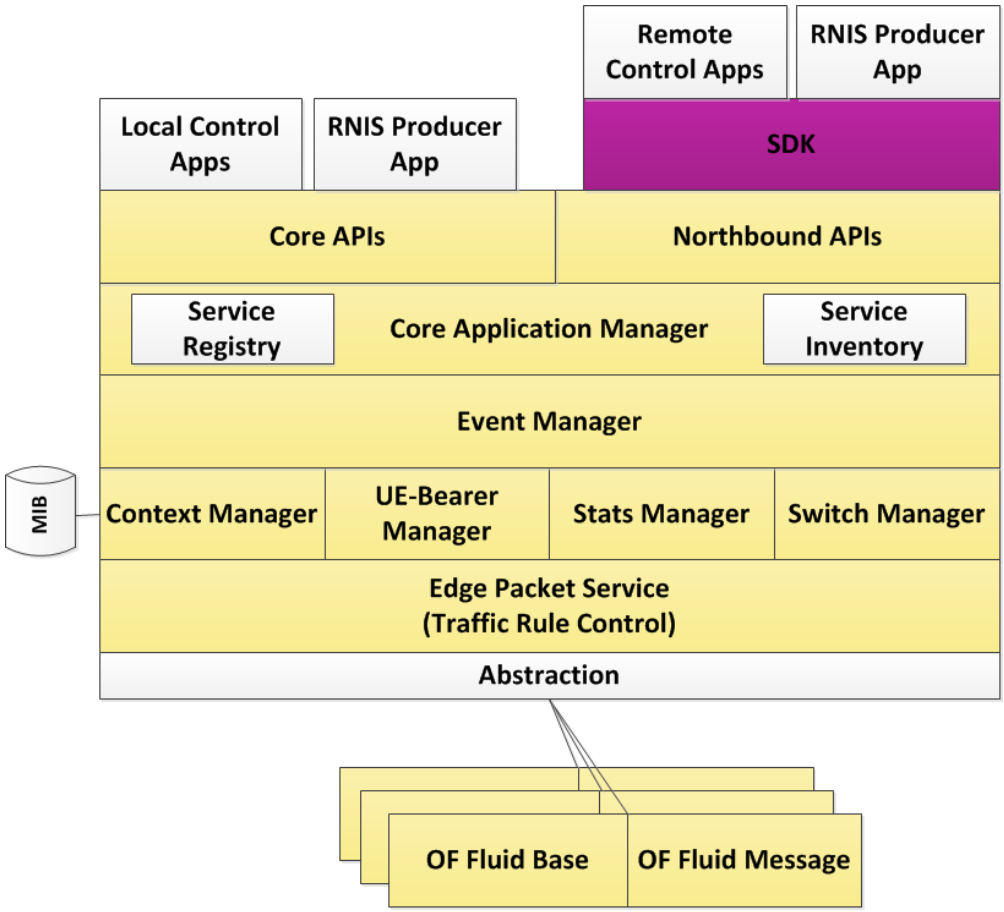


Figure 6. Software architecture of LL-MEC

3.2.1 Mobile Network Abstraction

The abstraction layer models and exposes the required operations for the underlying network through a unified interface. In LL-MEC, the Data-Plane APIs naturally comprise the abstraction layer for the edge of the network between RAN and CN by providing only the necessary information for the development of MEC applications and platform. The control protocols implemented in LL-MEC are divided into two domains, namely the RAN enabled by the RNIS producer App and the CN through OpenFlow. The RNIS producer app leverages the FlexRAN SDK to abstract view of the radio network status (e.g., topology, band, and signal strength) by extracting the parameters of interest from the RAN with the required level of granularity. Besides this, it also gives the possibilities to modify and control the state of the underlying network and passes the control decisions on the fly at a very fine time granularity (per subframe), e.g. reconfigure the resource block allocation policy for each connected UE and apply the policy on the fly in order to adapt service priorities. On the other hand, the OpenFlow protocol provides a fine-grain Data Plane programmability through the abstraction of the underlying data paths and allows the switch to handle GPRS (General Packet Radio Service) Tunnelling Protocol (GTP) packets in the core network and set the inner packet Type of Service (TOS) field to support QoS in the transport network. Last, all of the aforementioned features occur at the interaction between MEC platform and its underlying network enabled by CP and DP API.

3.2.2 Traffic Rules Control

LL-MEC EPS implements the ETSI MEC Traffic Rules Control Service, and it is one of the main components for managing Data Plane. EPS brings a native IP-service end-point to the MEC applications and acts as a local IP agent performing network functions, like IP forwarding and packet encapsulation/decapsulation. EPS also gives the abilities for MEC applications to adapt the routing/forwarding for their specific purpose. Traffic coming from UEs through OpenFlow-enabled switches goes along the routes based on the rules setup in the switches by EPS and can be changed or shaped dynamically to optimize the routing. The way LL-MEC abstracts the Data Plane is to utilise the OpenFlow library as the protocol and based on that to construct the essential endpoints for LL-MEC infrastructure. As one of the core entities in LL-MEC, EPS offers the interfaces towards its northbound and southbound, which are described respectively as Mp1 and Mp2.

Mp1 is the control interfaces for MEC applications to instruct the basic and advanced functionalities in the underlying network, such as default/dedicated bearers (re-)establishment, QoS for GBR traffic, and a custom control commands from MEC applications. Note that the S1/S5/S8 bearer establishments follow the same workflow as presented in Figure 5. When the “Modify Bearer Request” is requested by either one of the “LTE attach procedures” or “EPS Mobility Management Service Request”, X-GW-C will notify LL-MEC for bearer establishment through “UE Setup Rules Request” API call, allowing LL-MEC to trigger an OpenFlow rules to setup the switch accordingly. When this “UE Setup Rules Request” API is called, the message must include the user identities, as indicated in Table 1 (e.g. uplink/downlink tunnel ID and bearer ID). As soon as the X-GW-C receives the “UE Setup Rules Response” call, the S1/S5/S8 bearer establishment is confirmed to be complete. Similar procedure can be used to extend LL-MEC to support QoS for GBR traffics through OpenFlow meter and group tables allowing performing various operations such as rate limiting for a particular flow, user, or group. LL-MEC currently supports traffic redirection allowing a MEC application to request that all traffic for a certain UE and certain service (flow) to be redirected to a receiver inside the calling MEC application.

Mp2 is, from EPS point-of-view, used to instruct the Data Plane on how to route the traffic through OpenFlow rules. The types of rules that EPS creates and maintains in OpenFlow handler can be categorized into three groups:

(1) default rules, which are pushed to OpenFlow-enabled switches on connection established for handling Address Resolution Protocol (ARP) and Domain Name System (DNS) queries;
(2) UE specific rules, which are used to establish the default and dedicated bearers for UE;
(3) MEC application rules, which are pushed to OpenFlow-enabled switch on events registered by applications.

With the well-defined and full set of rules, the Data Plane can be fully separated from Control Plane and the user/nearer latency can be thus improved through dynamic programmability.

Table 1. Identifiers for user/bearer establishment/modifications

Identities	UE	eNB	MME	SGW	PGW
UE IP	YES		YES		YES
BEARER ID	YES	YES	YES	YES	YES
S1 SGW/eNB UL TEID		YES	YES	YES	
S1 SGW/eNB DL TEID		YES	YES	YES	
S5 SGW/PGW DL TEID			YES	YES	YES
S5 SGW/PGW UL TEID			YES	YES	YES
SGW IP			YES	YES	YES
PGW IP			YES	YES	YES

3.2.3 Radio Network Information Service

Under one of the specifications by ETSI MEC [2], RNIS is a service providing up-to-date radio network information, although the design and implementation details are under specification. In LL-MEC, RNIS exposes the real-time RAN information, such as radio bearer statistics, measurements related to UE, state changes of UE, and power measurements to MEC applications by interacting with C-plane API. The granularity of information can be adjusted based on parameters such as per cell, per UE, or per radio access bearer (RAB) and can be requested only once, periodically, or triggered when an event occurs. In addition, the control-plane API defines a set of functions that can be used by the Data Plane to notify the Control Plane about events such as the initiation of a new Transmission Time Interval (TTI) and the state change of a UE that has been powered off. In order to have a clean separation of control and Data Plane for RAN, FlexRAN protocol and RAN Information Base (RIB) are integrated into LL-MEC as a MEC producer app at the application level.

The FlexRAN protocol acts as an abstraction layer allowing the management of the higher-level control operations in a technology agnostic way, similarly to how OpenFlow abstracts the data path in the wired network. On the other hand, all the statistics and configurations about the RAN, i.e. UEs and eNodeB, are all maintained in the RIB as shown in Figure 6 and accessed by the applications. Furthermore, with the integration of RIB into MEC platform, LL-MEC RNIS module can have direct and high priority access into RIB on per millisecond basis to ease the control latency. For example, an edge application can query each user link quality to provide a quasi real-time indication on the throughput in the next time window.

3.2.4 Mobile Edge Application Framework and SDK

One of the main benefits coming with the separation of control and Data Plane is that the MEC applications have limitless possibilities to be developed for any specific purpose without knowing the detailed knowledge of the underlying network. In LL-MEC, the programming interfaces (Mp1) and the SDK built on top of it (depicted in Figure 6) enable the application development and programming environment. The SDK offers a uniform interface and abstracts the multiple choices of Mp1 including Representational State Transfer (REST) API, message bus, and local API for having different requirements of

applications developed, such as low latency and elastic. Examples include monitoring and constantly acquiring the information through message bus, managing the traffic rules based on application preferences through REST API within 100 ms, or optimising the content according to the radio quality through local API within 1 ms. In addition, MEC applications, through Mp1, can also access the basic functionalities provided by the MEC platform, such as service registration, service discovery, event mechanism as described in [7]. Another pivotal feature LL-MEC has is that the application can be deployed in different scheduling recipes such as round robin, first-in-first-out, or deadline scheduler for having different priorities when executing the task behind the scene. This significantly lowers the application latency and meets the required control deadlines from an edge application.

LL-MEC currently supports the following API endpoints in the UP through the OF Control Plane as shown in Table 2. LL-MEC currently supports the following API endpoints in the CP through FlexRAN Control Plane as shown in

Table 3.

API documentation and examples can be found at:

- LL-MEC: http://mosaic-5g.io/apidocs/ll-mec
- FLEXRAN: http://mosaic-5g.io/apidocs/flexran

LL-MEC and FlexRAN SDKs integrate all of the above APIs into a set of high-level user-friendly APIs that simplify the application development and enable an application to extend such APIs to monitor, control, and manage the underlying network.

Table 2. LL-MEC API endpoints in Data Plane

Type	Method	End point	Description
STATS	GET	/stats	Get all the traffic flow statistics in upstream and downstream.
STATS	GET	/stats/id/:id	Get a particular traffic flow statistics in upstream and downstream by ID
STATS	GET	/stats/imsi_bearer/:imsi_bearer	Get a particular traffic flow statistics in upstream and downstream by IMSI and EPS bearer ID
USER	POST	/bearer	Add a default/dedicated bearer context.
USER	GET	/bearer	Get all bearer context.
USER	GET	/bearer/id/:id	Get a specific bearer context by id
USER	GET	/bearer/imsi_bearer/:imsi_bearer	Get a bearer context by IMSI and EPS bearer ID
USER	POST	/bearer/redirect/imsi_bearer/:imsi_bearer	Redirect specific traffic flow for one bearer by IMSI and EPS bearer ID
USER	POST	/bearer/redirect/:id /bearer/redirect/id/:id	Redirect specific traffic flow for one bearer by its ID

USER	DELETE	/bearer	Remove all bearers context
USER	DELETE	/bearer/:id /bearer/id/:id	Remove a specific bearer context by its ID
USER	DELETE	/bearer/imsi_bearer/:imsi_bearer	Remove a specific bearer context by its IMSI and EPS bearer ID
USER	DELETE	bearer/redirect/:id bearer/redirect/id/:id	Remove the redirect flow for one bearer by ID
USER	DELETE	bearer/redirect/imsi_bearer/:imsi_bearer	Remove the redirect flow for one bearer by it IMS and EPS bearer ID
SLICE	GET	/slice	Get all the slices and its ID mappings
SLICE	GET	/slice/:id	Get a specific slice and its ID mapping by slice ID

Table 3. LL-MEC API endpoints in Data Plane

Type	Method	End point	Description
STATS	GET	/stats_manager/:stats_type	Get all the radio statistics in upstream and downstream by stats type in a human readable format. Stats type could be all, enb_config, mac_status
STATS	GET	/stats/[:type]	Get all the radio statistics in upstream and downstream for all eNBs and UES by stats type in json format. Stats type could be all (default), enb_config, mac_status
STATS	GET	/stats/enb/:id/[:type]	Get all the radio statistics in upstream and downstream for a particular eNB and the associated UES by stats type in json format. Stats type could be all (default), enb_config, mac_status. UE ID can be RNTI or IMSI.
STATS	GET	/stats/enb/:id/ue/:id	Get all the radio statistics in upstream and downstream for a particular eNB and UE by their ids in json format. UE ID can be RNTI or IMSI.
STATS	GET	/stats/ue/:id	Get all the radio statistics in upstream and downstream for a particular UE by its ids in json format. UE ID can be RNTI or IMSI.
STATS	POST	/record/[:type/[:duration]]	Record the radio statistics by type and for a predefined duration. Returns a record ID in the payload.

STATS	GET	/record/:id	Ge the record radio statistics by its id
USER	POST	/rrm_config -d@pathtopolicyfile	Post a RAN policy and commands to the underlying RAN

3.2.5 Helper Services

A set of helper services are also implemented in LL-MEC to complement the platform. These services are not the entities to provide core functionalities but necessary for having an effective MEC server.

- **Communication Service:** It handles the channel between MEC application and platform as well as the interactions among MEC applications; they are enabled by the REST API and the SDK for the remote control apps, and CORE APIs for the local control apps (see Figure 6). LL-MEC is using Pistache (http://pistache.io/) to handle REST call binding and processing.
- **Event Manager:** It facilitates the internal communication and monitoring among different MEC services.
- **App manager:** It allows LL-MEC application to be deployed with different policies, scheduling receipt (e.g. RR, DEADLINE, and FIFO) and operation including continuous, periodic, or event-driven app. An app can be local implementing an event callback and start functions to get runtime to perform its task, and remote interacting with the LL-MEC either directly through the REST API or indirectly through the SDK. Currently, the following applications are implemented:
    - **UE/Bearer Manager:** It handles add/remove/redirect of a new pair of user-bearer in LL-MEC with the associated OF rules that is transmitted to the underlying OF-enabled switches.
    - **Stats Manager:** It provides fine-grain flow-level statistics on per user per service in both upstream and downstream directions. These statistics are extended by the RNIS producer app.
    - **Context manager:** It stores the context related to UEs, bearer, slice, and switch accessible by all the other services. It is also responsible for generating and managing the internal identifier for the pair of UE and bearer based on the cookie in the OpenFlow rules.
    - **Switch manager:** It manages the connected switches to each LL-MEC instance, allowing a MEC app to apply a particular rule to a subset of underlying switches.
- **Core Apps Manager:** It provides a tight integration with OpenFlow library and event manager.
- **OpenFlow Library:** It enables the communication with OpenFlow-enabled switch based on Libfluid.

In addition, two services are currently under design and development:

- **Service inventory:** It identifies the available services supported by LL-MEC and the endpoints the service has.
- **Service Registry:** It implemented as a database, and includes the holistic information of the available MEC applications and gives the abilities for high-level applications to verify if the desired information is available.

3.2.6 LL-MEC Implementation

LL-MEC is a Low Latency Mobile/Multi-access Edge Computing platform licensed under Apache License V2.0 and delivered as part of mosaic-5g.io ecosystem. The bulk of the code is written from scratch using C++ and currently supports x64 Linux systems. The implementation aims to support core network programmability coordinated with RAN real-time operation and provide flexible application programming environment at the network edge. In addition to exposing the APIs specified in ETSI MEC, it was designed to easily expand the control function as well as the supported OpenFlow rules. The current OF library is based on version 1.3.

The open-source software Open vSwitch (OVS) [8] is employed as the software switch; it is further discussed in Section 4. Given the fact that OVS does not officially support GTP tunnel yet, GTP enabled OVS is packaged along with useful scripts in order to facilitate the deployment of LL-MEC for community. Currently, LL-MEC provides two versions of OVS with GTP support. The first version is based on the latest version of OVS with GTP support in Linux kernel that is available in 4.9, and the second version is based on the OVS version 2.7 with GTP management in the user space.

The salient features of LL-MEC that are currently supported can be summarised as follows:

- Northbound APIs and SDK
- Add/Redirect/Remove Default and dedicated bearers support
- Multiple switch support
- Flow status per user/bearer/switch
- Support of network slicing
- Support of packet TOS update
- LL-MEC S1 tester

4 Data Plane Programmability and Virtualized Infrastructure

Following the design and prototyping of the SliceNet MEC Platform in the previous section, this section further presents the SliceNet programmable Data Plane for MEC and other non-RAN segments. The focus is on investigating hardware-based and hybrid (hardware and software based) approaches, complementary to the software-based programmable Data Plane in the MEC platform.

4.1 Data Plane Programmability Enablers

4.1.1 Fundamental Architecture and Enablers

Towards achieving Data Plane programmability, it is proposed in SliceNet that a common reference architecture known as SimpleSumeSwitch [9] (and workflow) as shown in Figure 7 is adopted for prototyping forwarding devices (e.g., switches) to process packets at line rate and at affordable cost. This architecture comprises a single Parser, a single Match-Action pipeline, and a single Deparser. The different buses/channels (arrows) are coded in different colours in Figure 7:

- (Red) Packet parsing only happens internally (between the Parser and the Deparser).
- (Purple) Digest data are introduced by the Parser, and are one of the architecture's outputs. SliceNet proposes to explore packet hashing to be applied for data integrity based on digest data of a fixed size (80 bits). The hashed digest data are part of a rule for packet processing and are accessible by the Match-Action component of the NetFPGA to make a decision with respect to the packet being received.
- (Yellow) SUME metadata contain the metadata that indicate the port numbers of the packet and information related to the processing of the packet, and exposes to a programming language the programmable metrics. The SUME metadata struct is shown as follows:
    - struct sume_metadata_t {
        bit<16> dma_q_size; // measured in 32-byte words; DMA (Direct Memory Access)
        bit<16> nf3_q_size; // measured in 32-byte words
        bit<16> nf2_q_size; // measured in 32-byte words
        bit<16> nf1_q_size; // measured in 32-byte words
        bit<16> nf0_q_size; // measured in 32-byte words
        bit<8> send_dig_to_cpu; // send digest_data to Central Processing Unit (CPU)
        bit<8> drop;
        port_t dst_port; // one-hot encoded: {DMA, NF3, DMA, NF2, DMA, NF1, DMA, NF0}
        port_t src_port; // one-hot encoded: {DMA, NF3, DMA, NF2, DMA, NF1, DMA, NF0}
        bit<16> pkt_len; // (bytes) unsigned integer
      }

The interpretation of the SUME metadata struct proposed is as follows:

- pkt_len - the size of the packet (not including the Ethernet preamble of Frame Check Sequence or FCS) in bytes.
- src_port - the port on which the packet arrived.
- dst_port - the port or ports (if any) the packet should be sent out of.
- drop - if the least significant bit of this field is set to 1, the packet will be dropped.

  - send_dig_to_cpu - if the least significant bit of this field is set to 1, the digest data will be sent over DMA (Direct Memory Access) to the CPU.
  - X_q_size - the size of each output queue, measured in terms of 32-byte words (rounded up). This is the size of the output queues when the packet starts being processed.
- (Brown) User metadata contain any additional information from the Parser to the Match-Action pipeline and from the pipeline to the Deparser (internal only). In SliceNet, it is proposed that this bus will be utilised to classify the packets, for instance, to share the number of encapsulated layers available in the packet among the Parser, Match-Action and Deparser.
- (Green) Control bus. It is used to insert the rules into the hardware in order to determine the actions to be carried out over the packet.

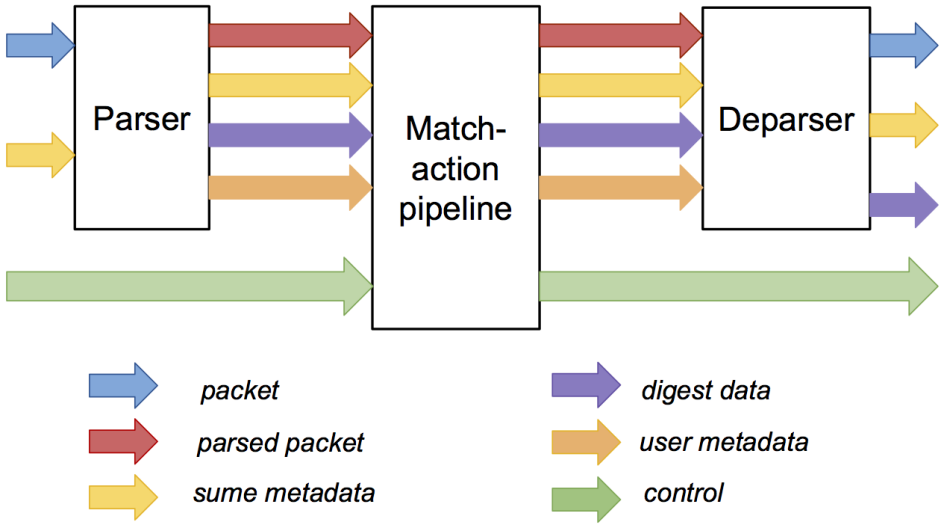


Figure 7. Reference architecture and workflow for a forwarding device to process packets [9]

- Packet classification by Parser using the User metadata channel

  Packet classification is a fundamental function required for QoS support. It maps a received packet against a mapping rule to categorise the packet to the best-matched class, and the categorization of incoming packets is based on selected fields of selected headers of the packets according to specific classification criteria. Packet classification can be based on packet hashing to speed up the process of determining if an incoming packet matches a certain classification rule.

- Packet hashing by Parser using the Digest data channel

  Packet hashing is a function that maps data of arbitrary size to data of fixed size. This facilitates classification-based QoS measurement and monitoring. On receiving a packet, the forwarding device hashes it and looks up to check if the packet belongs to a known flow (flow classification). If yes, statistical measurement regarding the flow will be continued (e.g., in terms of packet count and bytes for that flow). Otherwise, a new flow entry can be created.

Data Plane programmability entails the following enablers for network monitoring and control:

- P4 language for Data Plane classification, monitoring and control

P4 [10] is a common language for the description of Data Plane functionalities of different targets. However, not every target supports every language option, which reduces the portability. Moreover, in order to add flexibility to describe non-standard elements available in a concrete target, the language provides mechanisms to add external modules to the pipeline. There is an official P4 compiler, P4C, which is target agnostic and is used as a showcase of the P4 functionalities. P4 requires a target architecture, a structure with fixed elements and programmable modules, being the most used Portable Switch Architecture (PSA). PSA is a target architecture that describes common capabilities of network switch devices which process and forward packets across multiple interface ports with a pipeline with three input stages (parser, ingress and deparser) and three output stages (parser, egress and deparser). There are multiple compilers available for different target technologies, using the PSA, or a modified version, including software switches, Field Programmable Gate Arrays (FPGA), Network Processor Units (NPU) or Application-Specific Integrated Circuit (ASIC). In SliceNet, with respect to the reference architecture and workflow for a forwarding device to process packets as shown in Figure 7, P4 is the programming language for packet classification, monitoring and internal control of the hardware.

- OpenFlow for Data Plane monitoring and control

  OpenFlow [3] defines rule/policy-based traffic flow related processing, including flow matching, flow forwarding, flow QoS metering etc. Specific control actions may include packet forwarding to a particular port or ports, packet encapsulation and forwarding to the controller, packet forwarding to the normal processing pipeline or packet dropping etc. For monitoring purposes, statistics such as packet and byte count are available. OpenFlow is applicable to both physical and virtual (hypervisor-based) forwarding devices and it is used in SliceNet to expose the control capabilities of the forwarding devices to the SDN controller for the purpose of Data Plane control. SliceNet recommends that P4 be employed to programme the monitoring and Control Plane capabilities, which are exposed by OpenFlow to the SDN Controller.

- IPFIX/NetFlow and sFlow for Data Plane monitoring

  The IP Flow Information Export (IPFIX) protocol [11][12], as standardised in IETF RFC (Request for Comments) 7011 and a number of other associated RFCs, defines a standard mechanism to transmit uniform IP traffic flow information from an exporting process (in a network sensor or other reporting device) to a collecting process for the purpose of Data Plane monitoring. The IPFIX standard specifies the representation of different flows, IPFIX Data and Template Records sent over various transport protocols, additional data required for flow interpretation, packet format, and security concerns and so on. Monitoring is conducted on selected packets based on packet selection techniques such as sampling, filtering and hashing standardised by the Packet Sampling (PSAMP) protocol [13]. The Management Information Base (MIB) module [14] for monitoring and the data model [15] for configuring and monitoring IPFIX and PSAMP-compliant devices using the Network Configuration Protocol (NETCONF) [16] are specified as well. IPFIX is based on Cisco's NetFlow Version 9 [17].

sFlow [18] is an industry standard multi-vendor sampling technology embedded within switches and routers for network traffic monitoring. sFlow is scalable in collecting, storing and analysing traffic data, and enables monitoring links of speeds up to 10 Gbps and beyond without impacting the network performance or adding significant network load. A comparison of sFlow with Cisco's NetFlow is given in [19], which highlights a list of advanced features in sFlow across protocol layers and in terms of performance and cost considerations.

- OVS and OVSDB for Data Plane packet forwarding and control

  OVS [8] is capable of forwarding packets between VMs within the same physical host machine or between VMs and physical infrastructure. OVS is programmable and controllable using OpenFlow and the OVSDB (Open vSwitch Database) management protocol, an IETF standard [19]. OVSDB allows programmatic access to the OVS database, which holds the configuration for the OVS (daemon), to manage and configure this OVS.

  The main components in OVS and the space each component belongs to are illustrated in Figure 8 [21]. Firstly, ovs-vswitchd is the daemon that controls all the OVS switches in the system. The daemon implements switch features such as mirroring, bonding and Virtual LANs (VLANs). The OVSDB management protocol is employed by this daemon to obtain the initial configuration and new configurations from the ovsdb-server, where the configurations are persistent. Secondly, the kernel module, openvswitch_mod.ko, takes care of the data path consisting of physical and/or virtual ports in the kernel space. This kernel module applies switching or tunnelling actions, e.g., through Generic Routing Encapsulation (GRE) or Virtual Extensible LAN (VXLAN), to the arriving packets belonging to a known flow, i.e. a flow where the action to be performed is known. Otherwise, the packet is sent to the user space for the ovs-vswitchd daemon to process. The OpenFlow controller is responsible for the persistence of important flows.

The top-level configuration for the daemon is through the OVS tables in the OVSDB. The relationship among the tables are depicted in Figure 9 [22], where each node represents a different table. Table 4 [22] lists these tables and their purposes. It is noted that OVS supports NetFlow, IPFIX and sFlow for Data Plane monitoring.

It is noted that the current prototyping is based on the develop branch of OVS to provide a set of extensions. Thus, OVS is taken as it is, and extended by NetFPGA hardware-based approach.

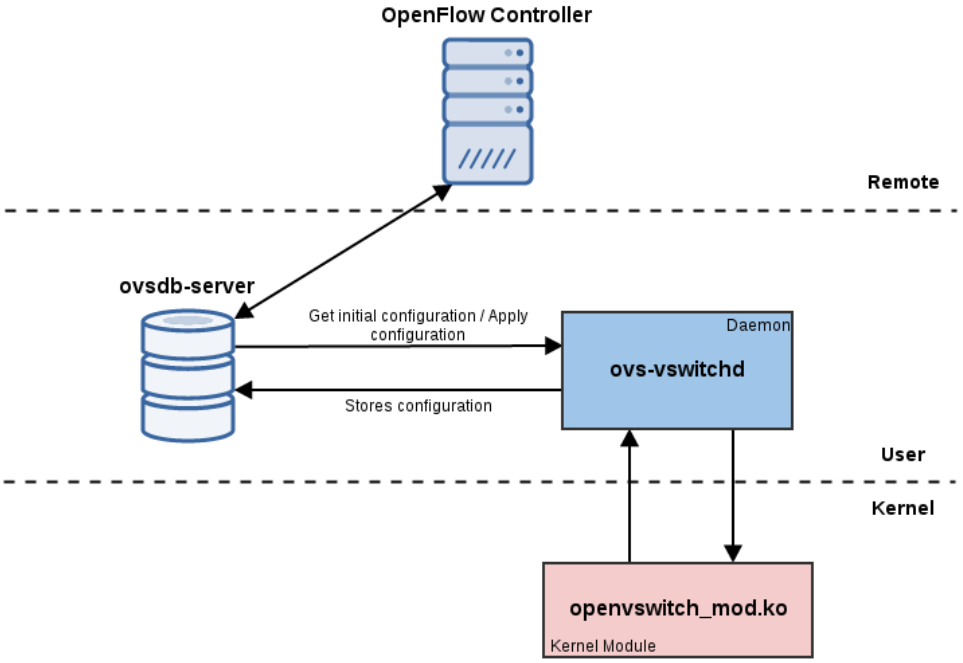


Figure 8. OVS components [22]

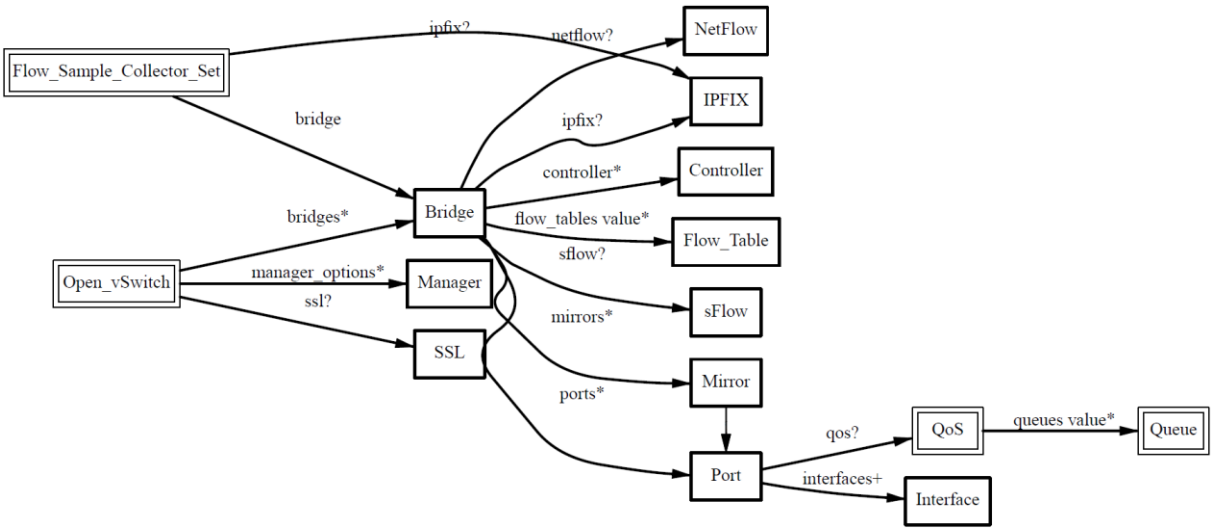


Figure 9. Tables relationship in OVSDB for OVS configuration [22]

Table 4. OVSDB tables [22]

OVSDB Table	Purpose
Open_vSwitch	Open vSwitch configuration
Bridge	Bridge configuration
Port	Port configuration
Interface	One physical network device in a Port
Flow_Table	OpenFlow table configuration
QoS	Quality of Service configuration

Queue	QoS output queue
Mirror	Port mirroring
Controller	OpenFlow controller configuration
Manager	OVSDB management connection
NetFlow	NetFlow configuration
Secure Sockets Layer (SSL)	SSL configuration
sFlow	sFlow configuration
IPFIX	IPFIX configuration
Flow_Sample_Collector_Set	Flow_Sample_Collector_Set configuration

- Kernel Network Control for Data Plane control

  In Linux, on receiving a packet, a Network Interface Card (NIC) sends it to a receive queue (RX). The packet is then copied to the main memory via the DMA (Direct Memory Access) mechanism. An sk_buff struct buffer is allocated for every received packet. The system is notified of the new packet and pass the data to the buffer. The process is based on an interrupt scheme, where several interrupts are incurred. The packet is then sent to the Linux networking subsystem. When an application (in the user space) needs to send or receive a packet, a system call is issued.

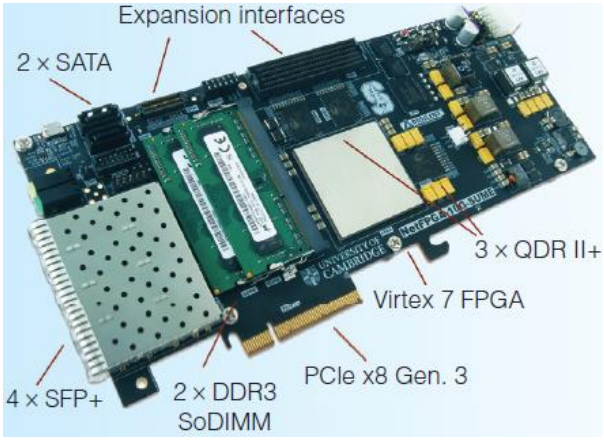
  There are drawbacks in terms of performance in this kernel space packet processing. Firstly, the buffer allocation based on the sk_buff struct consumes considerable bus cycles for copying the packet from CPU to the main memory. Moreover, the sk_buff struct is a complicated struct that was designed to be inclusive for different network protocols and thus is not optimised for performance. Thirdly, the mode switching between kernel and user spaces for packet sending and receiving also introduces latencies. Finally, the numerous interrupts slow down the system too.

  To address some of these problems, since version 2.6, Linux kernels employ the New Application Programming Interface or API (NAPI), which combines interrupts with requests. To further mitigate the performance issues in kernel network control, further optimisations (e.g., kernel space speed-up enhancements), new approaches (e.g., kernel bypass) and/or hardware acceleration are entailed, as explained in the subsequent subsections.

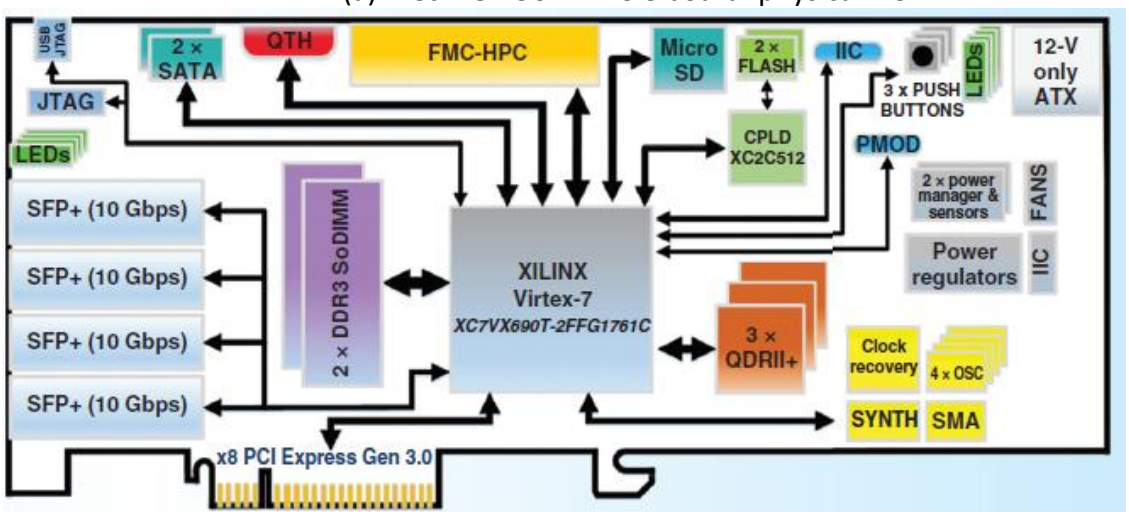
4.1.2 Programmability of the Hardware Data Path

- NetFPGA platform

  NetFPGA [23] is a line-rate, open networking platform that enables hardware-based programmable data path. The state-of-the-art NetFPGA SUME is an FPGA-based PCI (Peripheral Component Interconnect) Express (PCIe) board with I/O capabilities for 10 (and up to 100) Gbps operation, and the workflow is based on SimpleSumeSwitch depicted in Figure 7. The platform can be employed as NIC, multiport switch, or firewall, among other Data Plane networking or testing devices. The open source and low cost nature of the platform allows prototyping 10 Gbps solutions in R&D projects like SliceNet. More details on SliceNet prototyping are presented later.



(a) NetFPGA SUME PCIe board: physical view



(b) NetFPGA SUME PCIe board: block diagram

Figure 10. NetFPGA SUME platform [23]

Figure 10 [23] shows the NetFPGA SUME PCIe board (a) and the block diagram of the board (b) respectively. The core element is a Xilinx Virtex 7 690T FPGA. There are five peripheral subsystems on this full-sized PCIe adaptor:

- A PCIe x 8 Generation 3.0 interface between the board and the host device's motherboard for packet transfer between them.
- High-speed serial interfaces subsystem that comprises 30 serial links running at up to 13.1 Gbps speed, connected to 4 x 10 Gbps SFP+ Ethernet interfaces, 2 x expansion connectors, and a PCIe edge connector to the FPGA.
- Memory subsystem consisting of both SRAM (3 x 36-bit QDRII+ @500 MHz) and DRAM (2 x 64-bit DDR3 @933 MHz).
- Storage subsystems allowing both a MicroSD card and external disks through two Serial Advanced Technology Attachment (SATA) interfaces.
- FPGA configuration and debugging subsystem has 2 x NOR Flash devices, storing the FPGA's programming file, initial bootup image etc., and contains the debug and control capabilities through additional interfaces, LEDs, buttons etc.

- Netcope platform

  Netcope [24] is an alternative FPGA-based networking platform, which supports up to 100 Gbps Ethernet interfaces. Figure 11 [24] shows the physical view of the various Netcope boards. NFB-100G2Q has been investigated in SliceNet as an alternative to deal with 100Gbps. It provides architecture similar to the SimpleSUMESwitch but with other capabilities such as multiple DMA channels per interface and DPDK Driver support. It is noted that P4 is the recommended language for programming both NetFPGA and Netcope platforms (e.g., [25]).

Figure 11. Netcope platform (left: NFB-200G2QL; middle: NFB-100G2Q; right: NFB-100G2C) [24]

4.1.3 Programmability of the Software Data Plane

- DPDK (Data Plane Development Kit)

  DPDK [26] is an open source project that provides mechanisms for fast data processing for Data Plane applications, mostly running in the Linux user space and being agnostic to processors (Intel and others). As illustrated in Figure 12 [27], DPDK employs the kernel bypass approach, which allows the applications in the user space to directly communicate with the hardware (physical) or virtual devices (NICs) without involving the Linux kernel and thus circumvent the performance limitations of the Linux kernel caused by interrupts, the complexity of the sk_buff struct etc. After the interface receiving incoming packets is unbound from the Linux kernel driver, the communications between the application and the device is organised by the DPDK Poll Mode Driver (PMD).

  The DPDK framework creates a set of libraries for specific hardware and operating system environments through the creation of an Environment Abstraction Layer (EAL), which in turn is created through make and configuration files. Consequently, the user can link with the EAL library to create customised applications for the Data Plane. There are also other libraries to support Data Plane packet processing, e.g., the Hash for packet classification, and the Longest Prefix Match (LPM and LPM6) for packet forwarding based IP addresses.

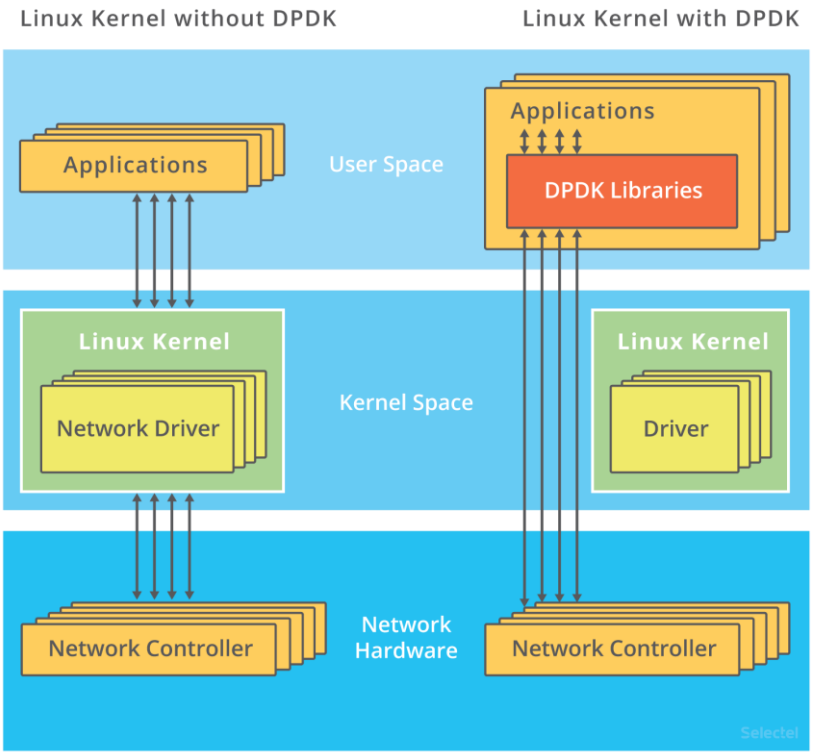


Figure 12. Linux kernel with DPDK vs. without DPDK [27]

- XDP (eXpress Data Path)

  In contrast to the kernel bypass approach taken e.g., in DPDK, XDP [28] through the IO Visor Project [29] addresses the performance limitations of kernel space in packet processing by enabling bare-metal packet processing at the lowest point in the software stack in the kernel space for improved speed whilst achieving Data Plane programmability. Figure 13 shows the packet processing overview in the XDP approach. Essentially, XDP creates an integrated fast path in the kernel stack. The in-kernel XDP Packet Processor intercepts the incoming packet before it is sent to the normal kernel process. It processes RX packet-pages directly out of driver via a functional interface and avoids early allocation of sk_buff's or software queues as seen in conventional kernel based packet processing. Basic XDP packet processor actions include packet forwarding, dropping, normal receiving and steering (to another CPU for processing), and generic receive offloading etc. The BPF (Berkeley Packet Filters) program is leveraged to perform processing actions such as packet parsing, table lookups, stateful filters creation/management, packet manipulation e.g. encapsulation/decapsulation, and returns action. Powered by the XPD programmability, a number of use cases can be built by exploring XDP, e.g., filtering for mitigating DDoS (Distributed Denial of Service) attacks, packet forwarding, load balancing, and flow sampling, lookup, inspection and monitoring based on hash, and flow analytics. The XDP approach is agnostic to CPU/hardware.

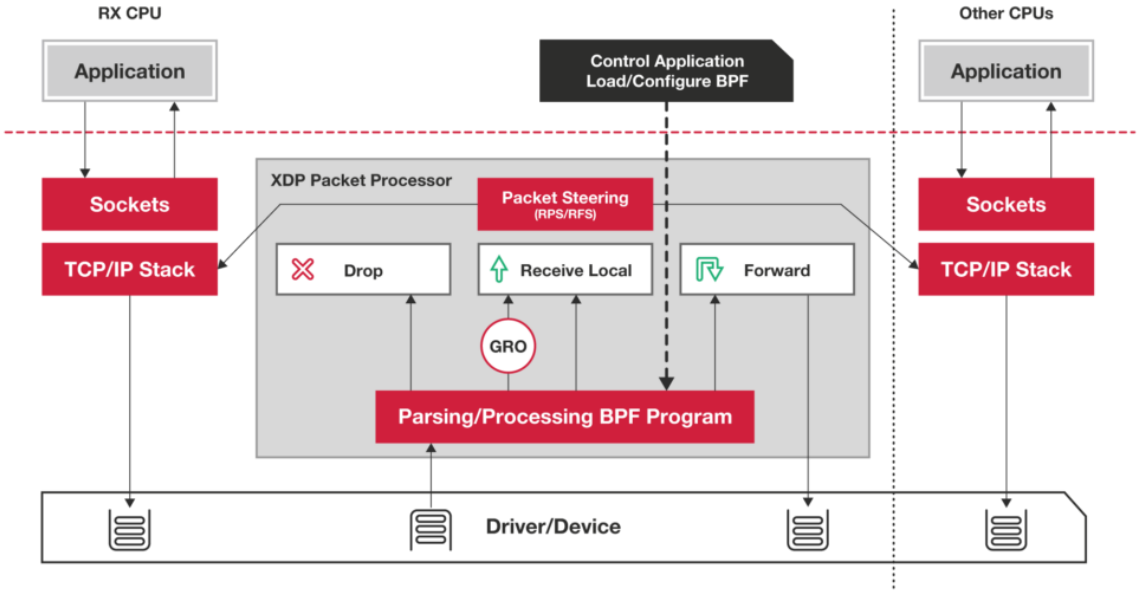


Figure 13. XDP packet processing [28]

- PF_Ring

  PF_Ring [30] is another technology that speeds up packet capture via kernel bypass. As shown in Figure 14, PF_RING polls packets from NICs through the Linux NAPI, which copies packets to the PF_RING circular buffer (ring), bypassing the kernel stack, and then the user-space application reads packets from the ring. PF_RING can distribute incoming packets to multiple rings, and thus multiple applications can read their packets simultaneously.

  PF_Ring is not in integrated in the mainstream Linux, and special kernel modules need to be launched. In particular, the PF_RING ZC (Zero Copy) [31] module offers a flexible packet processing framework that can achieve 1/10 Gbps line-rate packet processing for both RX and TX at various packet sizes. ZC implements zero copy operations for inter-process and inter-VM (Kernel-based Virtual Machine (KVM)) communications as a cloud-ready solution. Moreover, ZC can operate in either kernel bypass or normal kernel mode. In addition, PF_Ring also provides support for a range of FPGA vendors such as Netcope etc. through the FPGA-based card modules, among other additional modules.

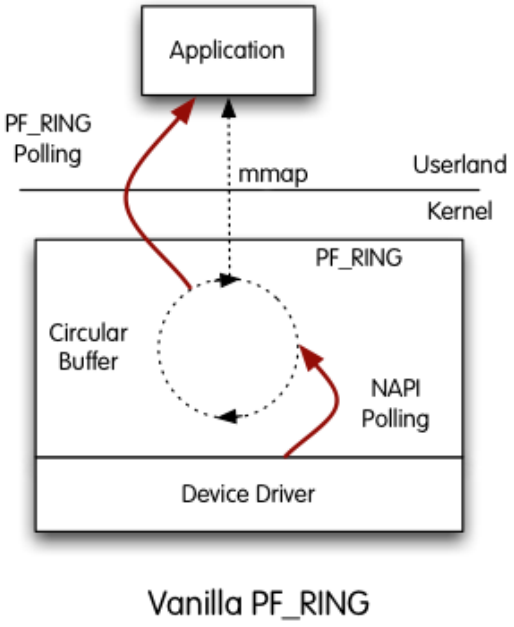


Figure 14. PF_Ring operation [30]

4.2 Data Path Architecture in SliceNet

Two different approaches have been considered in the Data Path architecture of SliceNet. The first approach "Hardware Approach" is based on an intensive hardware-offline approach where all the functionality required to the Data Plane is implemented in hardware. Figure 15 and Figure 16 show an overview of this approach, where two different hosts are illustrated as an example. One compute node is allocated in the edge network and another compute node in the core network. They are interconnected through a fibre-optical physical switch. The antenna and RRH or RRU/DU are connected to the MEC compute by mean of another physical switch. The solid red circles indicate possible control points (programmable points) in the data path.

The proposal is to use a NIC that allows in hardware to bypass the Linux kernel of the Host Machine and connect the hardware directly into the VM deployed in the edge (MEC VNF). It is achieved by means of the Single Root I/O Virtualization (SR-IOV) technology [32]. Then, in order to achieve slicing-friendly capabilities into the 5G infrastructure, the NIC should provide the following. Firstly, at least VLAN tagging support should be employed in order to allow essential multi-tenant isolation; or ideally, VXLAN/GRE encapsulation is applied to provide a true capability to allow tenants to define networks in software and perform such off-loading into hardware. Secondly, the NIC should provide the exposition of the different lanes/queues directly to the VMs. This can be achieved by the VMDq technology [33]. This approach allows every VM to have a dedicated set of queues/lanes available to be used in an exclusive use and the scheduling of these lanes is an enabler of the slicing-friendly infrastructure.

In this approach, the Infrastructure Provider only has the control capabilities exposed by the hardware. It means that in order to allow a slicing-friendly infrastructure in this "Hardware Approach", the programmability of the scheduler is required in order to enforce the slice QoS control. Both the NetFPGA and Netcope architectures presented before allow these

capabilities by using a TCAM (Ternary Content Addressable Memory) structure for inserting the Match-Action rules. It is noted that packets in this approach still need to be processed by the Linux kernel of the Guest VM and thus this is where kernel-bypass approaches can be employed in the Guest VM as a complement to achieve efficiency at high rates. In case of considering a deployment where the programmability of the Data Plane is centralised in the SDN controller, then an OpenFlow agent is required to perform the programmability of the hardware rules of the scheduler implemented in the NIC.

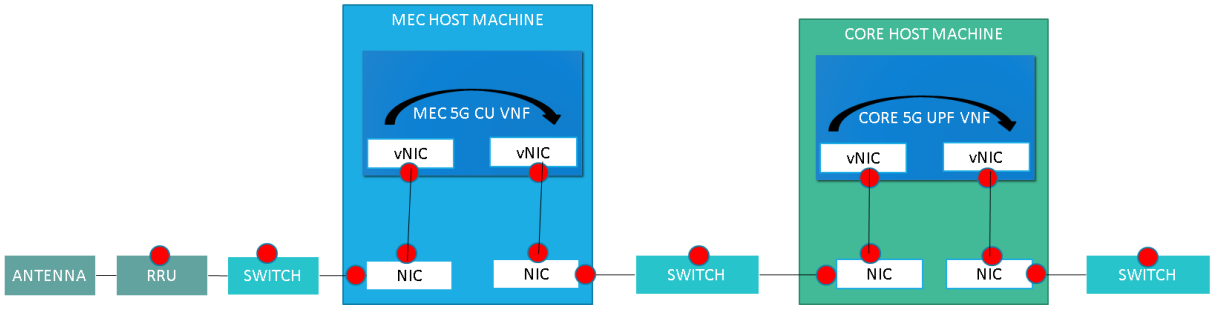


Figure 15. Programmable Data Plane (hardware approach)

The second approach "Hybrid Approach" is the combination of both software and hardware forwarding devices in order to separate the required roles to achieve a slicing-friendly infrastructure between the software and hardware components. The proposed architecture is shown in Figure 16 and Figure 15. It can be seen how the traffic coming from the hardware NIC is now received in the OVS hosted in the Host machine. It would require the Linux kernel to deal with the packets and thus would suffer from scalability issues. In order to sort out this problem, it is proposed to ensure that DPDK (or alternatively PF_Ring) be integrated with OVS. In this architecture, OVS is the responsible element to forward packets to the VMs and thus introduces a control point where policies can be applied. Due to the experimental nature of XDP, it is suggested that XDP will only be monitored in SliceNet and will not be further explored for prototyping.

In terms of role separation between software and hardware, it is proposed to minimise the possible use of the software-based approach, by off-loading as much as possible workload into the hardware capabilities. Compared with the Hardware Approach, this Hybrid Approach would yield significantly lower performance whereas it provides more flexibility in the Control Plane since it allows having double control points layers for the Infrastructure Provider and Network Operator. The Infrastructure Provider's control points layer is composed by two control points: one flexible yet slow software control point and a limited yet fast hardware control point. The Network Operator's control points layer, however, only has one control points layer for the tenant in the kernel space of the VM.

SliceNet is a research and development project and thus it has been decided to explore this approach even knowing in advance that it is not as fast as a pure hardware-based approach in order to investigate how better make use of all these control points to maximise slicing flexibility, along the execution of the different stages of the project.

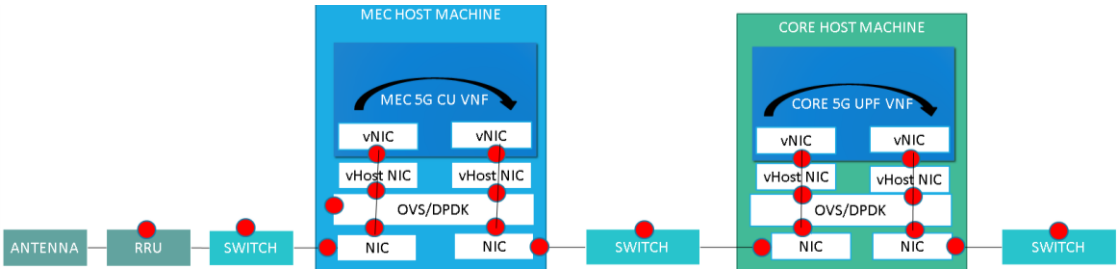


Figure 16. Programmable Data Plane (hybrid approach)

4.3 Infrastructure Multi-Tenancy Support

4.3.1 Overview of Multi-Tenancy Based on Integrating VIM and SDN

The SliceNet infrastructure supports multi-tenancy at the virtualisation level. Different VMs belonging to different tenants are isolated from each other, even co-located in the same physical machine. This tenant isolation can be achieved by employing a proper Virtual Infrastructure Manager (VIM) such as OpenStack [34]. OpenStack allows the management of the control provided by the OVS in order to manage tenant isolation of networking resources by using tagging and encapsulation. Furthermore, OpenStack allows the management of the control provided by the Hypervisor in order to manage tenant isolation of memory, disk and CPU resources. OpenStack is further discussed in Section 5.

Moreover, VMs belonging to the same tenant, even distributed in different physical machines, are able to reach each other via virtual switches such as OVS. This VM connectivity can be established by integrating the VIM with a compatible Software-Defined Networking (SDN) controller such as OpenDayLight (ODL) [35], as illustrated in Figure 17 [36]. After a new VM is created and attached to the network, the SDN controller creates a tunnel e.g., through VXLAN, between the OVS switches of the compute nodes hosting the VMs. Once the tunnel is created, the tenant's VMs on the different compute nodes are able to communicate with each other. In addition, when 5G or LTE traffic flows between the VMs, additional tunnelling e.g., the GTP tunnelling in LTE, applies.

It is noted that there are various ways to integrate ODL with OpenStack to achieve the multi-tenancy support [37]:

- ODL-OpenStack integration using ODL Group Based Policy's Neutron VPP Mapper
- ODL-OpenStack integrating using ODL Group Based Policy
- ODL-OpenStack integration using ODL NetVirt
- ODL-OpenStack integration using ODL Virtual Tenant Network

In principle, in all the methods, OpenStack employs ODL as its network management provider through the Modular Layer 2 (ML2) northbound plugin, and ODL controls the Data Plane network flows for the OpenStack compute nodes through the OVSDB southbound plugin. The next section takes the ODL Virtual Tenant Network approach as an example to elaborate the ODL-OpenStack integration for multi-tenancy.

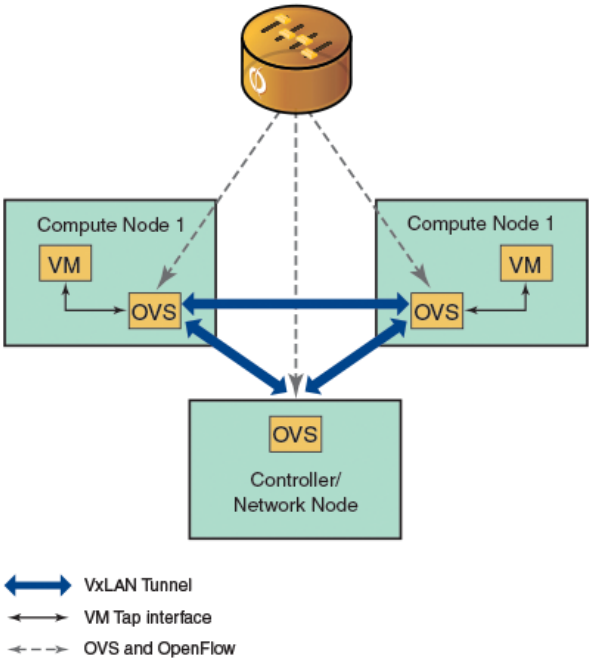


Figure 17. Integrated SDN controller and OVS model for multi-tenancy [36]

4.3.2 Multi-Tenancy Support Based on OpenDayLight Virtual Tenant Network

The Virtual Tenant Network (VTN) in ODL provides multi-tenant virtual networks on an ODL SDN controller. VTN consists of two main components: the VTN Coordinator application on the top as part of the network application, orchestration and services layer, and the VTN Manager plugin as part of the Network Services below the ODL APIs in the overall ODL architecture.

The VTN Coordinator provides a REST interface to the user to employ VTN, realises the VTN provisioning in ODL instances, and orchestrates multiple ODL to support VTNs that span across different ODL. The VTN Manager provides a REST interface to configure and manage the lifecycle of VTN components, and provides the integration with Neutron interface to provide network services for OpenStack by utilising the OVSDB plugin.

Figure 18 [38] illustrates the system diagram of integrating VTN with OpenStack and OVSDB. In this integrated system, ML2 allows the OpenStack control node to use ODL as its L2 network management provider. The ODL VTN Manager's Neutron deals with the interface creation notification (Event) from the OVSDB plugin and creates the mapping between the OpenFlow (OF) ports and the virtual interfaces in VTN. Moreover, the Neutron allows the VTN Manager to associate the VMs in the OpenStack compute nodes with the virtual networks, and install flow entries to OpenFlow switches between the VMs. In the Data Plane, the OVS switches in the OpenStack compute nodes communicate with each other through an optional OpenFlow network (e.g., a physical or logical OpenFlow switch) to achieve the inter-connectivity among the VMs belonging to the same tenant.

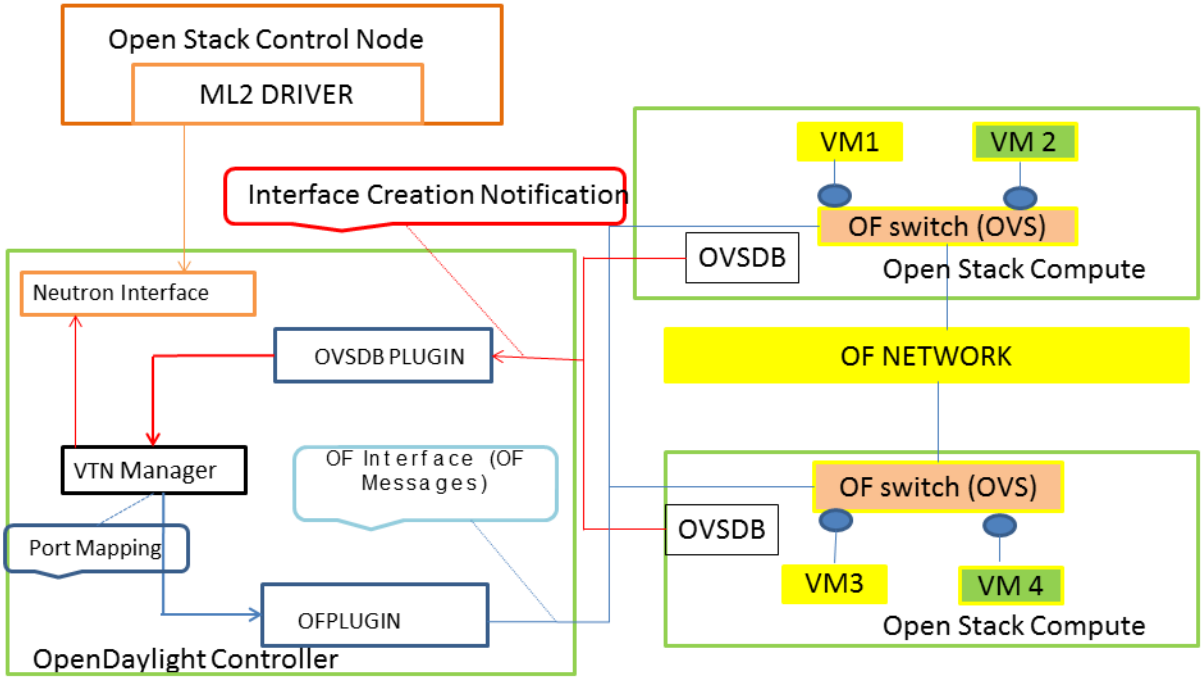


Figure 18. ODL (Lithium) VTN integration with OpenStack and OVSDB for multi-tenancy [38]

The VTN may be leveraged by MEC in two ways: (1) associate different MEC instances to a particular VTN (1:1 mapping), and (2) associate a single MEC to multiple VTN (1:N mapping) and apply the programmability on the logical networks instead of the physical network. In the latter case, the MEC platform is unaware of the separation of physical network plane from the logical one.

4.4 Mobile Edge-Core Network Data Plane

In 5G networks, the Mobile Edges and the Core Network (CN) are geographically distributed. SliceNet achieves this infrastructure view by exploring the different availability zones provided by OpenStack. Essentially, an individual edge can be positioned in a specific availability zone, and the core network is in a different zone from any individual edge. Consequently, multiple zones corresponding to edges and the core network can be created. For instance, for an MEC #1 edge, all MEC #1 related network nodes including MEC #1 VNFs such as a pool of LTE BBUs (or 5G CUs) are grouped into one availability zone. For the core network, all core network nodes including the core VNFs such as MME, SGW, PGW and PCRF (or the 5G counterparts such as AMF, SMF, UPF and PCF) are grouped into another availability zone. Figure 19 illustrates this MEC-CN Data Plane segregation based on availability zones.

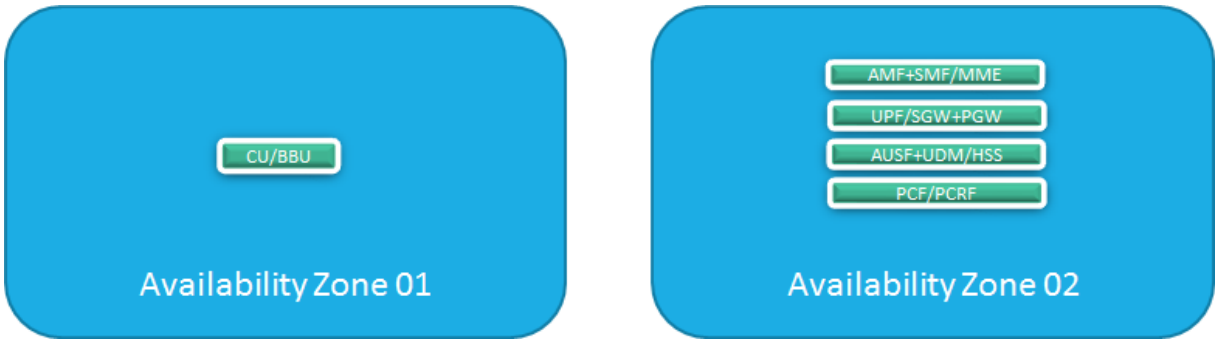


Figure 19. MEC-CN Data Plane segregation based on availability zones

Figure 20 [39] shows two availability zones consisting of their own compute nodes, which can run the mentioned VNFs for the edge and the core network, respectively. Furthermore, the difference between the concepts of Availability Zone and Host Aggregate in OpenStack is also illustrated for the sake of clarification and justification. In brief, nodes geographically distributed should be separated with availability zones, whilst nodes with the same specification should be clustered with host aggregates. Clearly, availability zones are more appropriate to be adopted in segregating the different edge and core network segments. In addition, Regions are of a higher geographical hierarchy and are usually employed to separate cloud computing systems of larger scale. For example, it is reported that there are only 18 geographic Regions (and 49 Availability Zones) worldwide in the Amazon Web Services Global Infrastructure [40]. In addition, availability zones have a unified control of the network connectivity between all the computes involved whereas regions have different network connectivity going out of the administrative domains, which add additional complexity to the control in the creation of slices. Therefore, it is not recommended that the edge and core networks are segmented based on Regions.

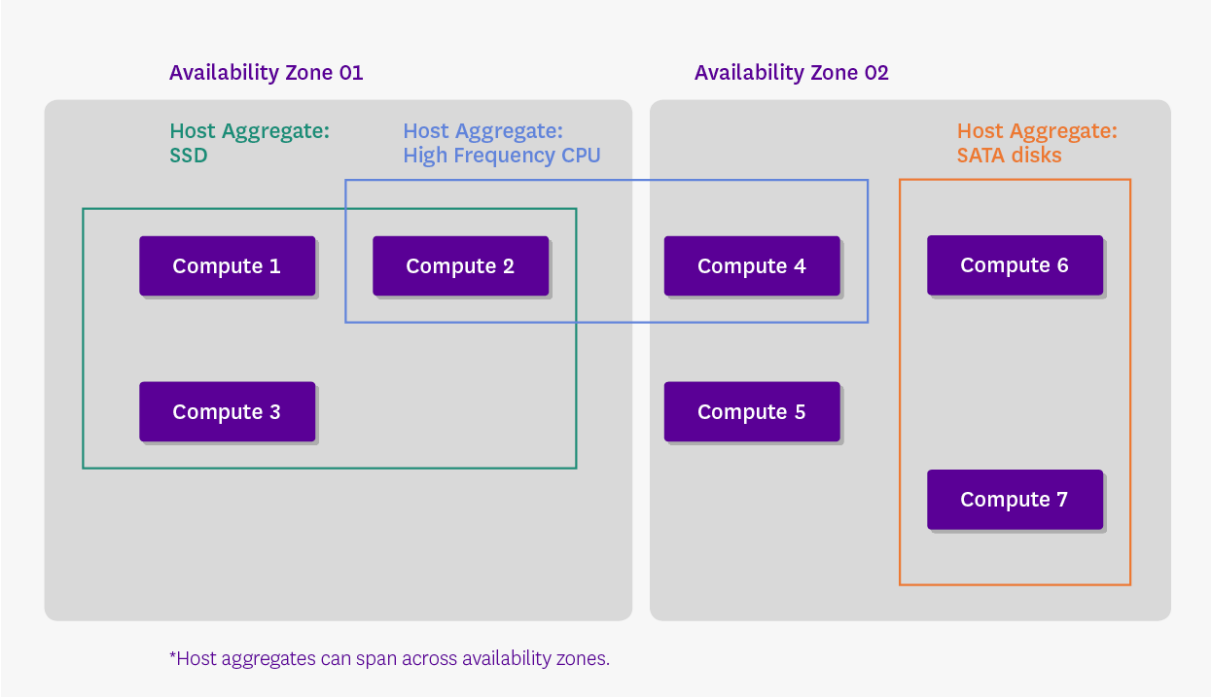


Figure 20. Availability zones and host aggregates in OpenStack [39]

4.5 SliceNet Programmable Data Plane Prototyping

In order to validate and test the above design, prototyping has been conducted to achieve slicing-friendly infrastructure especially programmable Data Plane for multi-tenanted 5G MEC infrastructure, and the technologies are largely applicable to the Data Plane of other non-RAN segments (MEC to core, and core network). The main prototyping is based on NetFPGA (10 Gbps) and P4, following the SimpleSumeSwitch architecture [7] as shown in Figure 7, to create various actions (queue setting, ToS setting) as hardware-based slicing enablers, as described below.

4.5.1 SliceNet Programmable Data Plane Prototyping Tools and Platform

The prototyping utilised the P4 NetFPGA reference implementation recently released by the NetFPGA Team employing the Xilinx SDNet P4 compiler. The SDNet Compiler v2017.1.1 [41] was set up on Ubuntu (64-bit) with Vivado Design Suite installed and licensed, together with additional supporting tools including gcc 6.2.0, Questa v10.4c, GraphViz DOT graph visualization software library, and Wireshark. Moreover, Xilinx SDK 2016.4 [42] was employed for the development. The design and development procedure followed the official SDNet framework design flow presented in Figure 21 [43].

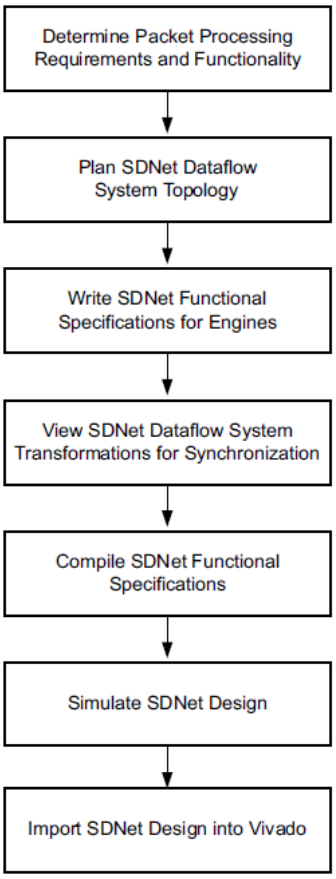


Figure 21. SDNet framework design flow [43]

4.5.2 SliceNet Programmable Data Plane Prototype

Figure 22 illustrates the system diagram of SliceNet Data Plane traffic classification and control prototype. The prototype operates as follows.

1. The traffic classification rules are inserted in the NetFPGA's Ternary Content Addressable Memories (TCAM) table through a REST API.
2. An inbound packet arrives at the NetFPGA.
3. The packet is sent to the Parser for classification.
4. Once the packet have been classified based on its headers, it is sent to the Match/Action component.
5. In the Match/Action component, the TCAM table is checked.
6. If there is any rule that matches the packet, the "action data" will be received by the Mach/Action (step 6).
7. Match/Action applies to the packet the action received in the "action data" of the rule (step 6) and it is sent to the Deparser component.
8. The Deparser builds the packet that is going to be sent to the PCI or to the outbound interface depending on the specific action.
9. The digest data will be sent to the CPU for further processing.

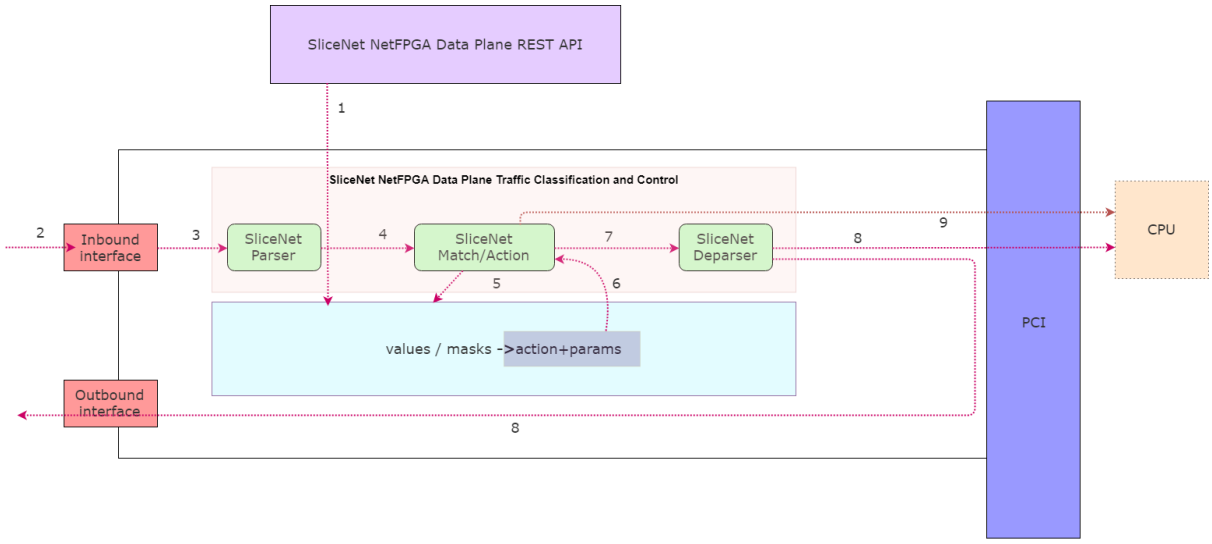


Figure 22. SliceNet Data Plane traffic classification and control prototype

The operations listed in Table 5 allows the flow control and management of the traffic processed by the NetFPGA in terms of lifecycle management (add, delete, clean) of the traffic classification rules installed in the table used by the Match-Action component and recording statistics of the flows offline. An operation defined in Table 5 is invoked on demand by a REST API with the parameters indicated in Table 6, among others.

Table 5. SliceNet traffic flow control and management operations

End Point	Description
/add	Adds a new rule in a table
/delete	Deletes a rule of a table
/clean	Deletes all the rules of a table
/totalMatchedBytes	Returns the total of bytes matched
/totalMatchedBytesPerRule	Returns a list of tuples with the number of bytes matched by rule
/ruleMatchedBytes	Returns the number of bytes matched by a specific rule

/totalMatchedPackets	Returns total number of packets matched
/totalMatchedPacketsPerRule	Returns a list of tuples the number of packets matched by rule
/ruleMatchedPackets	Returns the number of packets matched by a specific rule
/numRules	Returns the total of rules in the table in a determinate moment
/resetCounters	Resets all the measures, included the variable with the total of bytes matched
/resetCounter	Returns the bytes matched by a determined rule and reset the counter of that rule
/availableAddress	Returns if exist some rule in a determined address of the table

Table 6 shows the actions supported by the NetFPGA with respect to a flow:

- A flow can be dropped.
- A flow can be mirrored to another interface.
- The TOS value of the outermost IP header of a packet belonging to a flow can be dynamically set to configure its priority.
- In addition to the above action, a flow can be sent to a specific queue at the FPGA to be further processed by the CPU.

The instructions regarding these actions for matched flows are communicated through the REST API to establish the corresponding traffic classification rules beforehand or on demand.

Table 6. SliceNet traffic flow actions supported by NetFPGA-based prototype

Action Description	Action value (5 bits)	Parameter 1 (48 bits)	Parameter 2 (2 bits)	Parameter 3 (3+1 bits)
DROP	1	-	mirror interface	-
NOPE	2	-	mirror interface	QueueID + Enable
SET TOS	7	TOS (outer)	mirror interface	QueueID + Enable

Headers supported by the P4 implementation are listed in Table 7, where Ethernet (ETH), User Datagram Protocol (UDP), Transmission Control Protocol (TCP) etc. are employed. The following different NIC Modes are supported:

- NIC Mode I supports the traffic classification of pure IP flows, which may employ different L4 protocols.
- NIC Mode II is able to classify flows with one-level of encapsulation applied, via either GTP or VXLAN, corresponding to 5G/LTE traffic that is GTP tunnelled or IP multi-tenanted traffic respectively.
- NIC Mode III enables the classification of flows with two-level of encapsulations applied, via both GTP and VXLAN, which means 5G multi-tenanted traffic.

Table 7. SliceNet traffic classification (headers supported) by the P4 implementation

#	NIC Mode	Flow Type	L2	L3	L4							
1	I		ETH	IPv4	ICMP							
2	I	1	ETH	IPv4	UDP/TCP							
3	II		ETH	IPv4	UDP/TCP	GTP	IPv4	UDP/TCP				
4	II	2	ETH	IPv4	UDP/TCP	VXLAN	ETH	IPv4	UDP/TCP			
5	III	3	ETH	IPv4	UDP/TCP	VXLAN	ETH	IPv4	UDP/TCP	GTP	IPv4	UDP/TCP

4.5.3 Empirical Results

Experimental tests have been conducted to empirically validate the design and prototyping. In the tests, the following scenarios have been considered, with reference to Table 7:

- Scenario 1: NetFPGA NIC Mode I, II and III; Flow Type 1, 2 and 3 (UDP); 1 traffic classification rule; Ethernet frame size is 1500 bytes.
- Scenario 2: NetFPGA NIC Mode I, II and III; Flow Type 1, 2 and 3 (UDP); 512 traffic classification rules, Ethernet frame size is 1500 bytes.
- Scenario 3: NetFPGA NIC Mode I, II and III; Flow Type 1, 2 and 3 (UDP); 1 traffic classification rule; Ethernet frame size is 144 bytes.

Scenario 1 and Scenario 3 are configured to establish a benchmarking performance for the single-rule case (expected best-case performance in terms of rule scalability test). The difference between them is that Scenario 1 employs flow packet size of the Maximum Transmission Unit (MTU), whilst Scenario 3 employs small sized packets for the baseline packet size case (expected best case performance in terms of packet size). In contrast, Scenario 2 employs both high number of rules (512) and packet size of MTU, and thus it represents a challenging case (expected worst-case performance).

Figure 23 shows the experimental testing results. The left, middle and right sub-figures depict the measured results in Scenarios 1, 2 and 3 respectively. Based on those results, the following observations can be made:

- In every scenario, the delays increase as the NIC Mode varies from 1 to 3, corresponding to the increasing complexity levels of traffic classification logics deployed for pure IP, multi-tenanted IP and 5G multi-tenanted flows respectively.
- When Scenario 1 and Scenario 2 are compared with each other, it can be seen that the differences in the delays between the two scenarios are negligible (insignificant) despite the significant difference in the number of classification rules applied. Therefore, this approach is scalable in terms of supporting an increasing number of traffic classification rules.
- When Scenario 1 and Scenario 3 are compared with each other, it can be concluded that the size of the Data Plane traffic packets has an impact on the processing delays although the differences made by this do not change the overall profile of the delays and do not contribute to the overall delay significantly. Therefore, this approach is also scalable in terms of the size of packets.
- In all scenarios, the Match-Action and Deparser steps combined together introduce the majority of the delays. The delays caused by the Parser are much lower.

- In all scenarios, the delays for pure IP flows are under 3000 ns (3 ms), which is the benchmarking performance. For multi-tenanted IP flows, the delays are about 4 ms; and for 5G multi-tenanted ones, the delays are about 6 ms. The extra delays in the latter two cases are the performance penalties paid to achieve multi-tenancy for IP flows and multi-tenancy for 5G (GTP tunnelled) flows, respectively.
- To further reduce the delays, FPGA cards that are capable of realising higher speed such as 100 Gbps would be required. Nevertheless, in this proof-of-concept prototype, the experimental tests have shown promising results in achieving fast programmable Data Plane for 5G multi-tenanted traffic.

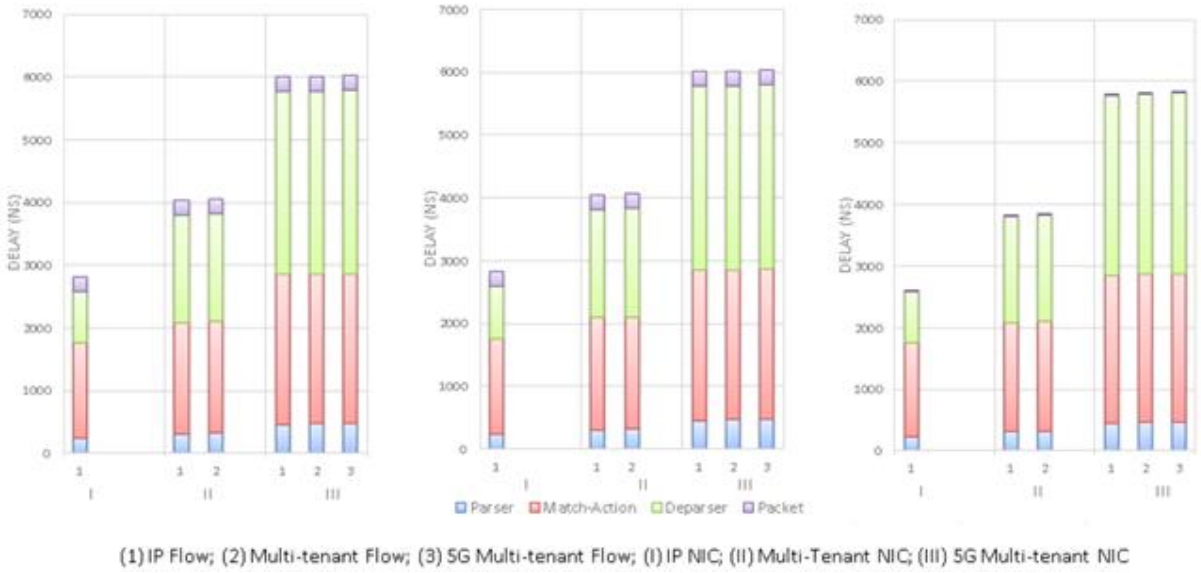


Figure 23. SliceNet Data Plane traffic classification and control prototype

5 Management Plane Considerations for Mobile Edge Segment

This section focuses on the Management Plane for MEC with description of management components including the Virtual Infrastructure Manager (VIM), Mobile Edge App/Platform Lifecycle Manager (VNFM), and the MEC Orchestrator (NFVO) where the functionalities and interfaces of those components are aligned as in ETSI MEC [2]. The section also focuses on the Management Plane considerations specifically for MEC.

5.1 Virtual Infrastructure Management (VIM)

A VIM, according to the MANO specification [2], is a virtual infrastructure manager that provides the Infrastructure-as-a-Service (IaaS) by assembling different NFVI, each with different technologies/vendors, and abstracting them into compute, storage and network nodes/resources. More specifically, VIM operations include:

- managing software images, such as add/delete/update/query/copy, and allocating those images to run on the NFVI, as requested by other functional blocks (the VIM obtains information from NFVO for managing application images, virtual resources, and it also interacts with VNFM to manage the NFVI resources associated with the Mobile Edge application lifecycle);
- orchestrating the allocation (of the virtual resources assigned to Mobile Edge applications), management and release of NFVI (compute, storage and networking) resources. The VIM will maintain an inventory of the allocation of virtual resources to physical resources for this operation;
- collecting and reporting performance and fault information about the NVFI;
- other operations that involve the NFVI management such as the security policies for access control, optimizing the use of resources, application relocation from/to external cloud environments, etc.

Existing solutions for VIM include OpenStack [34], VMware vSphere, CloudStack, Google Kubernetes VIM, etc., all come in the form of complete software stacks.

There are some considerations regarding the VIM performance, fault, and security, for example, how fast it can handle the provisioning of applications; how well it can allocate the physical resources necessary to deliver network services, keep track of the allocation of software images for applications/services onto the virtual resources, and then from virtual resources onto the physical resources and use that information to optimize/coordinate the use of resources; the issues of resource sharing and isolation; scaling up/down and scaling in/out; the decision whether to spin up VMs or containers for a VNF. The VIM should also be flexible to integrate/manage/orchestrate numerous hardware resources from different vendors/technologies/etc. Further, with multiple domains, it should be able to support multiple VIMs, and thus, communication and coordination among multiple VIMs should be taken into consideration to best utilise the resources across domains.

Towards the functionalities and considerations for the VIM above, it is proposed that OpenStack VIM is adopted for NFVI management in SliceNet. The logical architecture of OpenStack is in Figure 24 [34], [44], mainly consists of functional blocks for Compute, Storage and Networking.

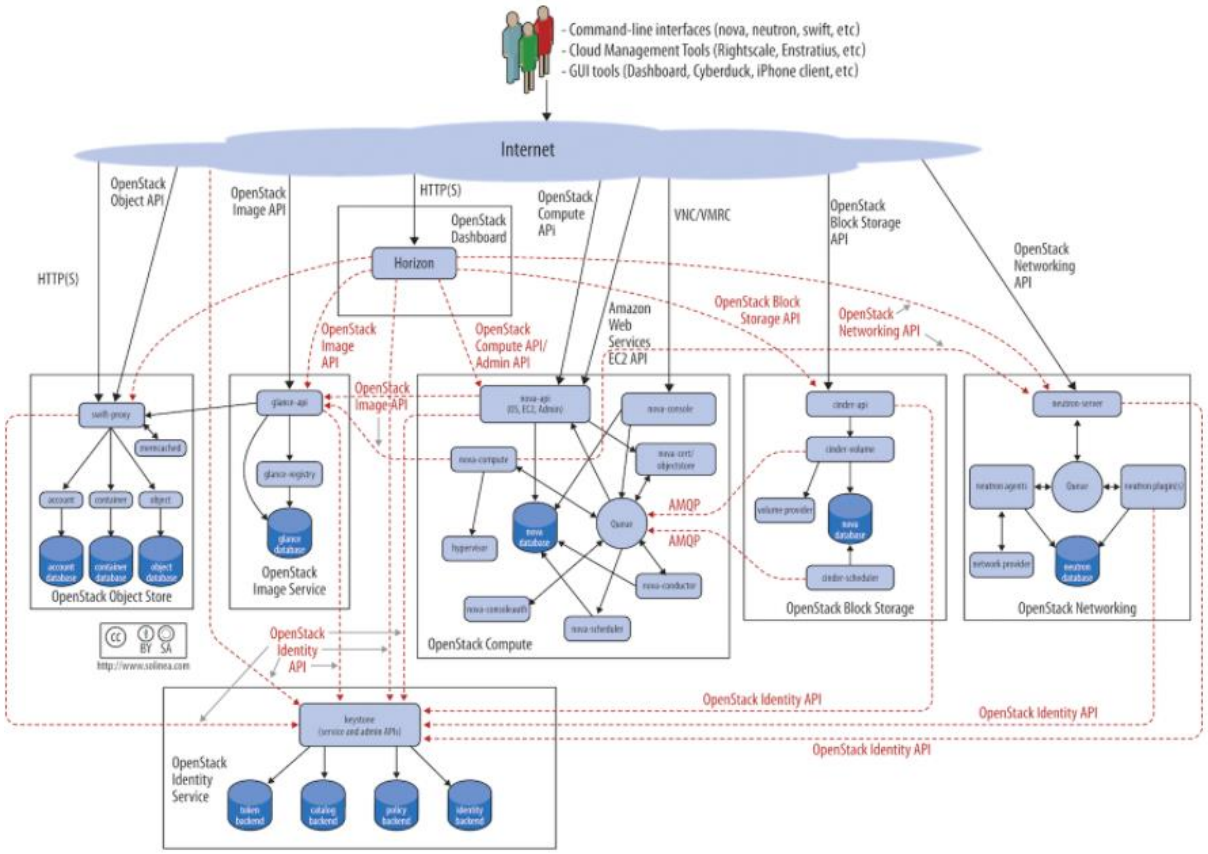


Figure 24. OpenStack logical architecture [34], [44]

With different considerations listed in the architecture design guide [44], the logical compute, storage and networking functionalities are fully designed and implemented in OpenStack Nova, Cinder, and Neutron respectively.

The OpenStack Compute Nova handles the provisioning computer instances (virtual servers), i.e. creating virtual machines, bare metal servers and some support for system containers. It requires additional OpenStack services including Keystone for identity and authentication services, Glance for compute image repository (as all compute instances launch from glance images), and Neutron for provisioning the virtual/physical networks connecting the compute instances. Nova supports a wide variety of compute technologies (hypervisor layer) such as KVM, Xen, LXC, Hyper-V, VMware, XenServer, OpenStack Ironic and PowerVM, which provides the flexibility in choosing a hypervisor(s). The OpenStack storage functionality is provided by three main components: Swift for object storage, Cinder for block storage and Glance for a repository for VM images, which can use storage from Cinder using standard protocols such as Internet Small Computer Systems Interface (iSCSI), Fibre Channel, NFS or object storage from Swift via the Swift API or HTTP protocols with simple PUT/GET commands. Finally, Neutron provides networking functionality between interface devices (e.g. vNICs) managed by other OpenStack services and supports advanced network services like firewall, load balancing, intrusion detection, VPN, etc.

In addition, OpenStack also supports running containers on bare metal or VMs with full storage and networking support. One can easily run containers on top of Nova as it has everything needed to run compute instances. However, in complex environments, it is required to have a container orchestration solution to ease the task of managing many containers in data centre environments. For this, OpenStack offers the Magnum system [45]

that supports multiple container orchestration tools including Docker Swarm, Kubernetes, Mesos, etc. The architecture of Magnum system is presented in Figure 25, which mainly shows the integration of OpenStack Heat with backend container technology (Kubeneters/Swarm/Mesos).

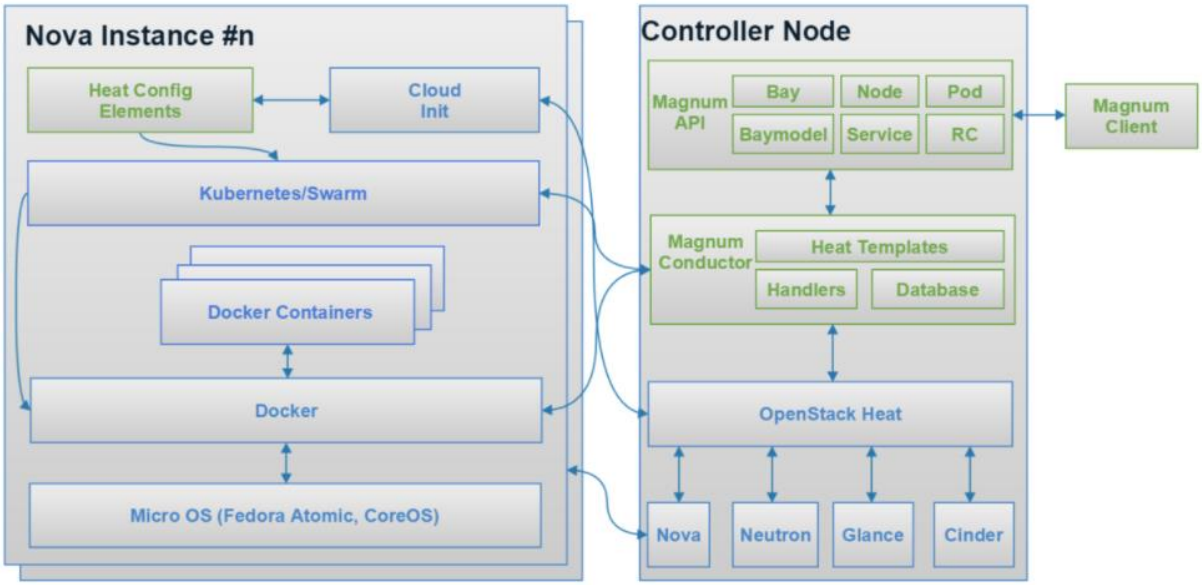


Figure 25. OpenStack Magnum architecture [45]

5.2 Mobile Edge App Lifecycle Management (VNFM)

According to ETSI GS MEC 001 [46] and 003 [2], the Mobile Edge Platform lifecycle manager is responsible for managing the life cycle of applications including informing the Mobile Edge orchestrator of relevant application related events. VNFM operations for managing application lifecycle include:

- Instantiating/Terminating an Application instance;
- Supporting the request to change the state (starting/stopping) of an application instance; supporting querying information about an application instance, status of an ongoing application lifecycle management operation, status of an application instance, etc.;
- Operations on an Application Package Management such as querying application package information (release date, vendor info, manifest, descriptor, files contained in the package, etc.); providing notification as a result of changes on application package states or the on-boarding of the application packages; fetching an application package or selected files contained in a package.

There are many considerations on managing the Mobile Edge application lifecycle. Firstly, to deploy an application, the consideration is on the number of instances per user and per host, etc. Then, how to bring the application on board and where to, in order to meet all the requirements regarding the virtual resources, latency, location (to be closer to the user), dependencies (other Mobile Edge services need to be running before this application gets on-boarded), etc. During the runtime of the application, mobility might occur in which VNFM should be able to handle application instance relocation (closer to the user to meet latency requirement), change of states, and so on. In addition, as many application instances and/or

different types of application might be running simultaneously, fairness among them or the users that requested to run those applications should be considered.

5.3 Mobile Edge App Rules & Requirements Management (VNFM)

The VNFM is also responsible for managing the application rules and requirements, which includes service authorizations, traffic rules, DNS configuration, mobility support, resolving conflicts, requirements on resources, services and/or QoS (e.g., delay constraint), and requirement validation, etc. For this, some considerations should be taken into account. Firstly, VNFM should be able to quickly create a ruleset for new and/or existing applications. Then, it is important to maintain a clean set of rules and to avoid unnecessary complexity. The VNFM should take a good care of the issues of unused/shadowed/expired rules, which create unnecessary costs and overhead in management. A clean set of rules also means no conflicting rules, no unwieldy rulesets that could break the applications or create risks to the system, for example, conflicting security rules can create backdoor entry points. Also, when there are many rules applied to the same object or one rule to different objects, etc. these rules configuration/application should be ordered optimally. At some stages, there should be a clean-up process to validate the existing rules and remove them if necessary (rules become invalid or expired). Finally, the VNFM should support all of those functionalities in different system scales.

5.4 Mobile Edge Platform Element Management (VNFM)

In Mobile Edge Platform Element Management, the VNFM is responsible for the functional management of VNFs running at MEC, i.e. FCAPS (Fault, Configuration, Accounting, Performance and Security Management). Technically, when a new VNF is created, the VNFM notifies this Element Management System to provide an element management for this new VNF, and the Element Manager associated with that VNF will take care of the management of functional components in that VNF, e.g. the functionalities that the VNF supposes to deliver/support.

As there are many VNFs running simultaneously in the system, and also the dynamicity of the system (bringing up/tearing down VNFs), some considerations should be taken care of. For instance, when a VNF is instantiated, how the VNFM can quickly provide an element management for this new VNF, and how well this element management can handle the functional components of the VNF; the issues of managing FCAPS for a large number of VNFs/PNFs running on the same system, each might have different FCAPS requirement. Besides common functionalities in FCAPS management, e.g. monitoring, managing and reporting FCAPS for each and all VNFs/PNFs, the VNFM should take into account some other details on each feature. For example, in fault management, beside considerations on the strategy/toolsets/mechanism to monitor the NFV, consideration on fault collection and on alarm mediation, the VNFM should also consider some further processing, e.g. analysis on the root cause and fault correlation, in order to fix the issue optimally. However, the virtual environment with many-to-many relationship between VNFs, VMs and physical resources, and a fault could be VNF/VM/NFVI related, it is high complex to do such analysis, not to mention some services that are created by service function chaining across both VNFs/PNFs and in different domains. For configuration, the VNFM should be intelligent enough to automate the (re-)configuration operations, especially for those that react to runtime events, but also guarantees the configuration integrity. For accounting management, it

should extend to track network utilisation to ensure that individual parties can be appropriately billed for their use, especially the case of different parties coexist on the same device through virtualisation, etc. Then, the VNFM can extend to manage FCAPS with a global view and in an optimal way, e.g. configuration and performance optimisation can be automated responding to faults and accounting (billing). Finally, the management approach i.e. centralized/distributed/policy-based/self-managed (or self-healing) should also be taken into consideration.

In SliceNet, it is proposed to have Juju [47] as a VNFM in MEC. Juju is an open source application modelling tool to quickly and efficiently deploy, configure, scale, integrate, and perform operational tasks on a wide choice of public and private clouds along with bare metal servers and containers. The central mechanism behind Juju is Charms. Charms contain all necessary instruction for deploying and configuring a service. A collection of Charms that link services together is called a Bundle, which allows to deploy whole chunks of app infrastructure in one go. According to [48], a Charm corresponds to a service definition and a collection of Charms and Bundles corresponds to the NS catalogue in ETSI model, and the process of uploading and deploying Charms into Juju corresponds to the NS on-boarding and instantiation process respectively. A global Charm catalogue containing all available Charms and Bundles can be found in Juju store [49].

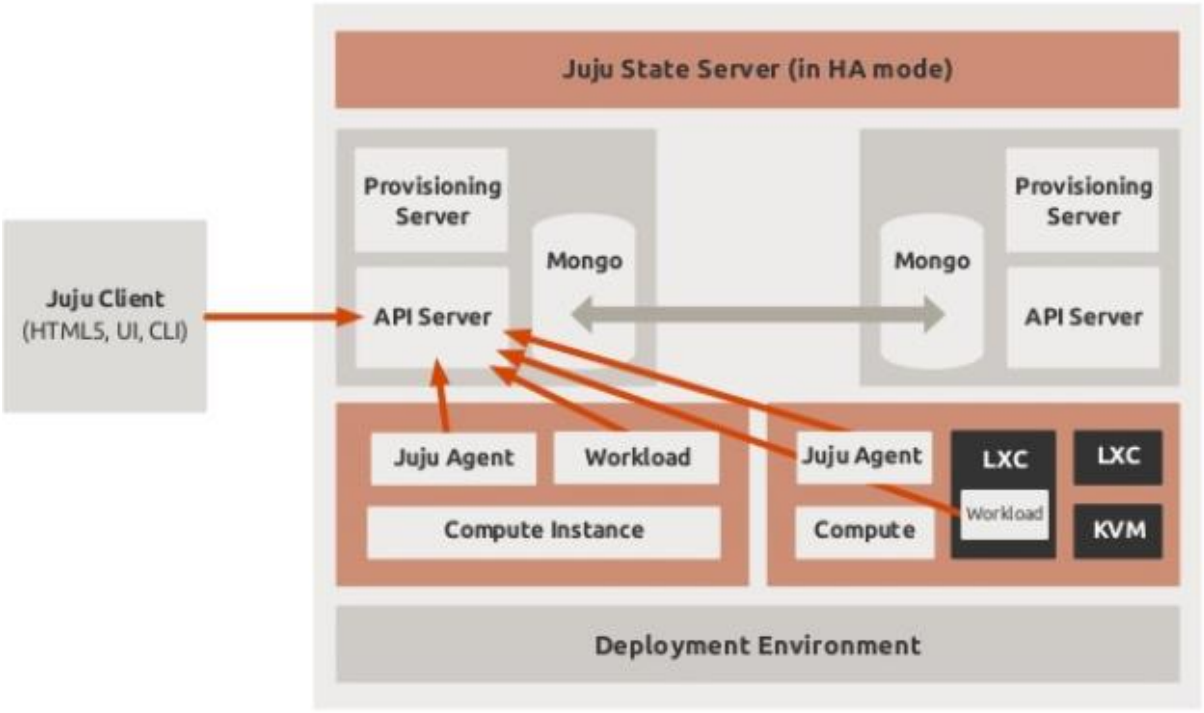


Figure 26. Juju architecture [47]

Juju manages the service lifecycle with hooks (or scripts) implemented inside Charms. Currently, there are five unit hooks including install, config-changed, start, upgrade-charm, and stop. There hooks are called during the lifecycle of a service, specified in the Charm's configuration file. Besides, for each interface (e.g., loadbalancer) that a Charm supports, there are four relation hooks, named after the interfaces: ifaceName-relation-joined, ifaceName-relation-changed, ifaceName-relation-departed and ifaceName-relation-broken to handle cases where the interface is connected to it, or disconnected, or the configuration or settings of that interface are changed.

For management, Juju creates a special node, called Juju Controller during bootstrap/installation stage. This controller houses the database, manages all the machines in the running models and responds to all events that are triggered throughout the system. It also manages scale out, configuration and placement of all models/applications, user account and identification, access and sharing.

In conclusion, we think that with Charms and Bundles mechanism and Juju Controller (also other supported functionality), Juju is a suitable tool to adopt as a VNFM in MEC MANO in SliceNet.

5.5 Mobile Edge Orchestrator (NFVO)

As in ETSI GS MEC 003 [2], the Mobile Edge Orchestrator (NFVO) is the core component in the Mobile Edge system level management. With an overview of complete Mobile Edge System, this NFVO is responsible for the following:

- maintaining a global view of MEC system, including the view on all deployed Mobile Edge hosts, available services and resources on each edge host, instantiated applications running on each host and also the network topologies;
- on-boarding of application packages, e.g. Mobile Edge application installations, application integrity check and authentication, application rules and requirements validation, also preparing the VIM(s) to handle the applications, while keeping track of the on-boarded packages;
- triggering application instantiation and termination;
- making decision on selecting a host(s) for application instantiation based on latency requirement, available resources and services;
- triggering application relocation if supported.

The main purpose of having MEC is to bring services closer to the users, in order to provide low latency services. However, mobility often occurs as users keep moving (ambulances, cars, phones, etc.); the communication between geographically edge hosts should be taken into account. Also, each edge host has its own NFVI managed by the local VIM, there should be multiple VIMs for multiple edge hosts, and thus multi-VIM management and orchestration, load balancing among edge hosts (also come with the issues how to select the best host for which applications, host relocation for the running application, etc.) should be taken into consideration. In addition, in 5G, the number of users grows significantly, there will be more Mobile Edge hosts added in a large scale, and for this, security and scalability should be supported, also the issue of monitoring and collecting KPIs from millions of edge objects should be counted. Overall, depending on the environments/conditions, the most suitable strategy to design and implement the NFVO, e.g. centralized/decentralized NFVO, distributed NFVO platform (with local NFVO for each edge and a centralized NFVO master), etc., is selected. However, in any decision, the NFVO should take into consideration the issues of mobility, scalability, load balancing, and orchestration on many different types of VIMs simultaneously; it should also be flexible to changes; and support for availability zones is a requirement for an orchestrator due to the edge and core networks segregation based on availability zones.

Based on a preliminary investigation, SliceNet proposes three solutions to explore: Open Baton [50], Juju-based orchestrator (JOX) [51], and OSM implemented in MEC as an NFVO.

5.5.1 Open Baton

Open Baton [50] is an open source platform providing an aligned implementation of the ETSI MANO specification. The architecture of Open Baton is presented in Figure 27 which shows the interoperability (integrating heterogeneous VIMs and VNFMs via a plug and play model with the exposed Restful API and SDKs) and extensibility (flexible for supporting any kind of use case) of the framework. In addition, it supports a publish/subscribe Event Engine for dispatching of the lifecycle events execution, a Fault Management System (FMS) for automatic runtime management of faults which may occur at any level, an Autoscaling Engine (AE) for automatic runtime scaling operation of VNFs, a Network Slicing Engine (NSE) to ensure a specific QoS for a Network Slice Instance (NSI) or Network Slice Subnet Instance (NSSI), and a Monitoring Plugin to allow whatever monitoring system preferred. The communication between the components is via RabbitMQ, which implements the Advanced Message Queuing Protocol (AMQP). Importantly, the NFVO is completely designed and implemented and fully compliant with the ETSI MANO, and thus it is suitable to adopt this as the NFVO functional block for MEC MANO in SliceNet.

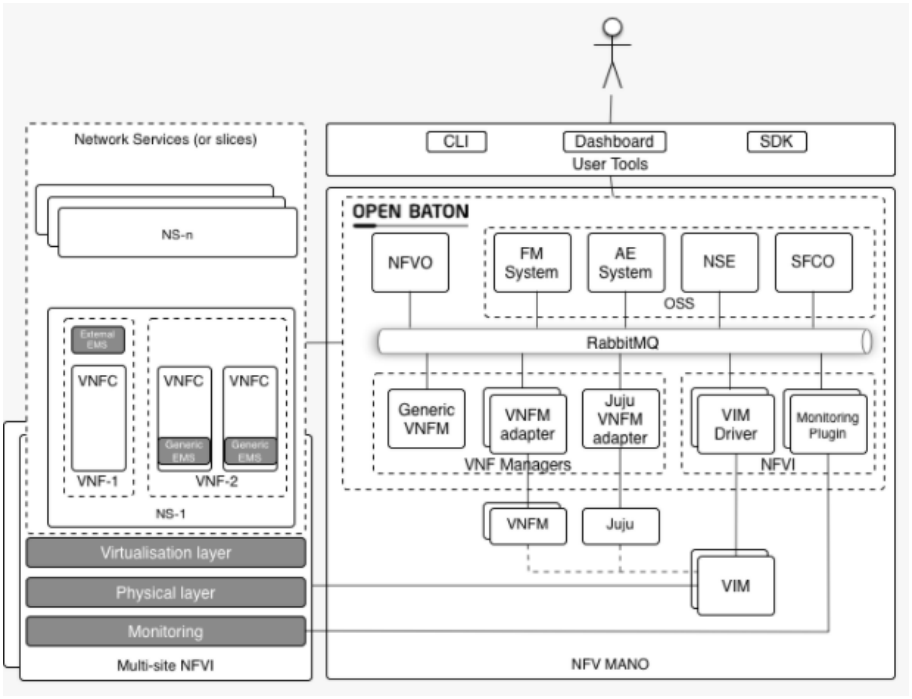


Figure 27. Open Baton architecture [50]

Open Baton is an extensible and customizable NFV MANO-compliant framework supporting different types of VIM (which can be easily added via a VIM instance Point of Presence (PoP)), VNFM, monitoring system, etc. Therefore, it is proposed in SliceNet that this framework will be adopted in SliceNet MANO where it uses existing Open Baton NFVO and integrates with OpenStack VIM; and either uses the provided generic VNFM and/or integrates with Juju VNFM to complete the comprehensive functional blocks in the ETSI MANO. Some other supported components (SSL, FM, AE, etc.) can be enabled, if needed. For multi-VIMs, the other VIMs can be integrated into this framework by providing their own VIM drivers via a plug and play model provided by Open Baton as shown in Figure 27. Alternatively, they can be integrated and managed by OpenStack (Docker Swarm, Kubernetes, Mesos) as shown in Figure 28, which illustrates the integration of the open source implementation selected for the three MANO functional blocks.

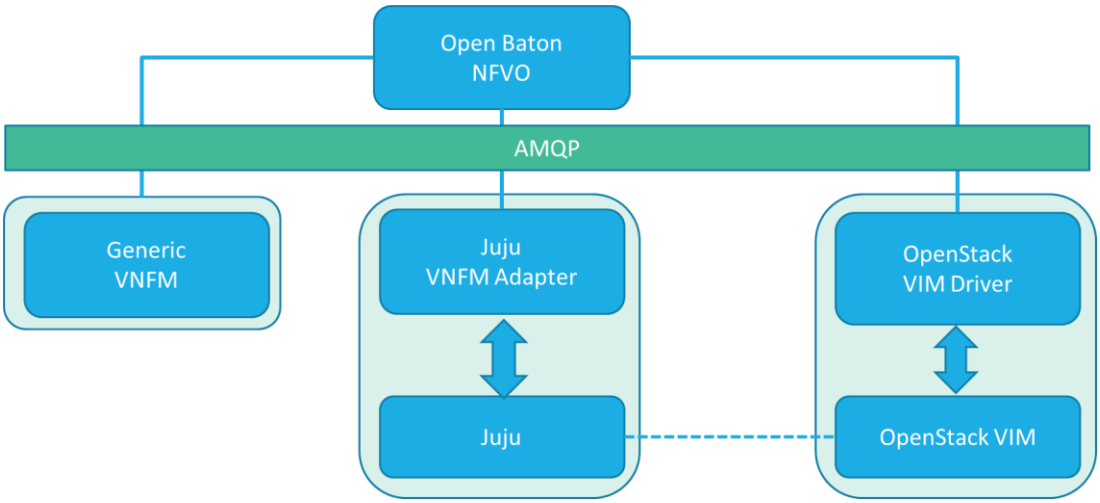


Figure 28. Proposed MANO implementation for MEC in SliceNet

5.5.2 JOX- a Juju-based Slice Orchestrator

JOX [51] is an open-source, event-driven orchestrator for the virtualized network that natively supports network slicing that can be used not only for MEC platform and its application, but also for RAN and CN segments. Inside the JOX core, a set of services is used to operate and control each network slice, while at the same time it supports the necessary interplay between resource and service orchestration, VNFM and VIMs. From the implementation perspective, JOX is tightly integrated with the Juju VNFM framework provided by Canonical.

Figure 29 [51] shows the architecture of JOX including:

- the JOX core with a set of core services to support of slice-specific life-cycle management, data handling, monitoring and template management;
- JOX Plugin Framework where each plugin element interacts with the corresponding agent via a message bus;
- the Northbound REST API to enable monitoring, control and programming of each slice.

In more details, 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. Every JModel is a bundle of resources, services, service chains and policy. JOX Slices Controller (JSC) is responsible to host and control all the instantiated JSlices. This is the place where global optimizations can be performed. JOX Clouds Controller (JCC) is responsible to host and control all the instantiated JClouds. JCC offers services to the JSC. Every JCloud object hosts all the underlying cloud resources and interacts with the physical infrastructure and the cloud control mechanisms through two channels: (i) the VNFM for a set of basic functionalities, and (ii) directly with the VIM for fine-grain monitoring and control.

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, the RAN and the MEC, 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.

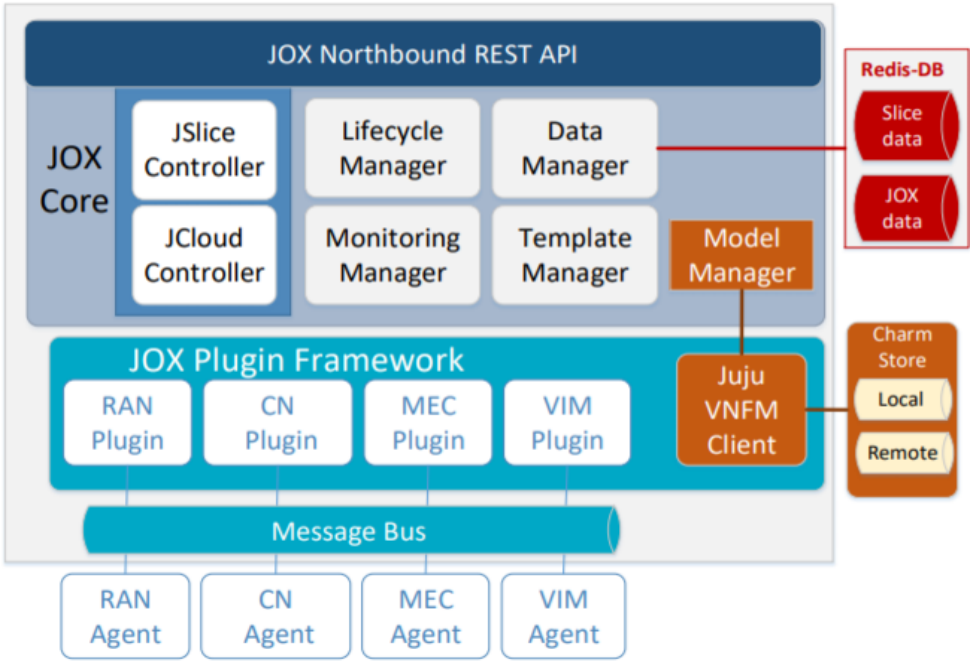


Figure 29. JOX architecture [51]

Different from Open Baton, JOX inherently supports lifecycle management of network slices and orchestration for the mobile network. Specifically, it supports basic operations defined by 3GPP in TR28.801 to manage the lifecycle (preparation, instantiation, configuration, activation, runtime and decommissioning phase) of a NSI, where all phase related API methods are exposed via the Northbound API. Besides, JOX also supports orchestration for the Mobile Network where it exploits RAN and CN specific plugins to efficiently orchestrate the edge network resources and services, e.g. orchestrating a new slice across multiple eNBs, partitioning the radio resources and deploying a dedicated CN for this newly generated slice. Furthermore, JOX also supports optimising the operational environment, for example, running a slice-specific logic or global optimisation on all slices applications on top of the Northbound API.

Besides the implementation proposal in Figure 28, SliceNet has also investigated the option of JOX framework with JOX NFVO, Juju VNFM and multi-VIMs as shown in Figure 30. It is noted that JOX can also be used for RAN and CN segments. As JOX is a single JOX NFVO, single Juju VNFM and multi-VIMs, it can also integrate with other VIMs via the plugins framework.

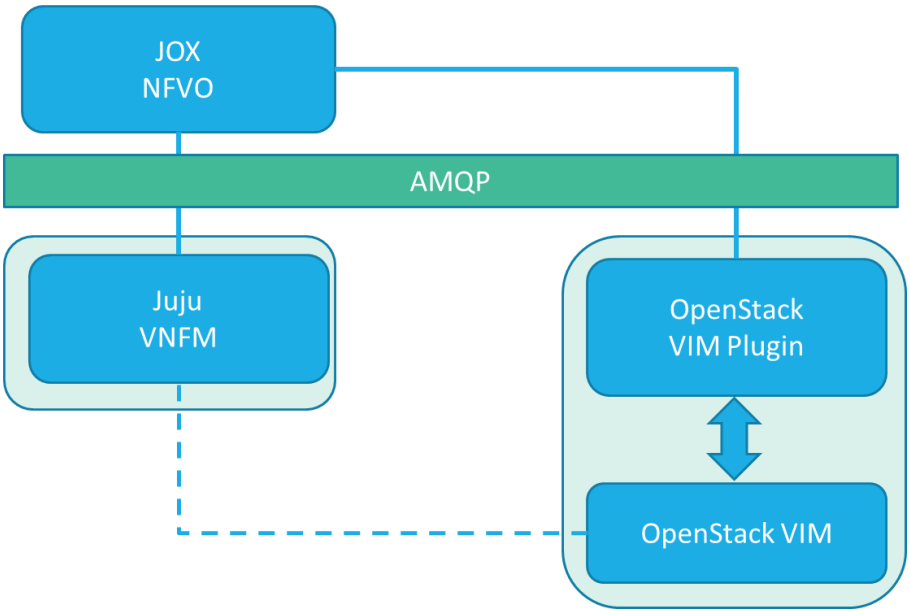


Figure 30. Proposed MANO implementation for MEC in SliceNet with JOX NFVO

5.5.3 OSM - ETSI's Open Source Mano

Open Source MANO (OSM) [52], [53] is an ETSI-hosted project to develop an Open Source NFV MANO software stack aligned with ETSI NFV, suitable for all VNFs, operationally significant and VIM-independent. The architecture of OSM is shown in Figure 31 [53], where it can be approximately mapped to ETSI NFV MANO logical view as in Figure 32 [53].

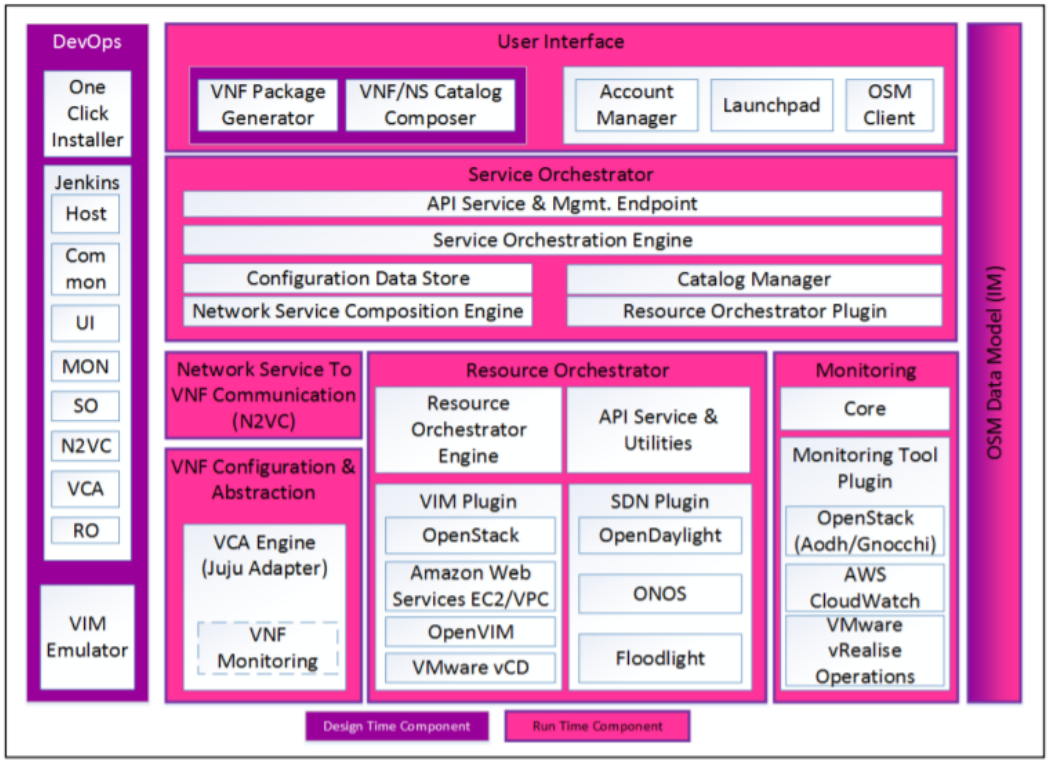


Figure 31. OSM Release THREE architecture [53]

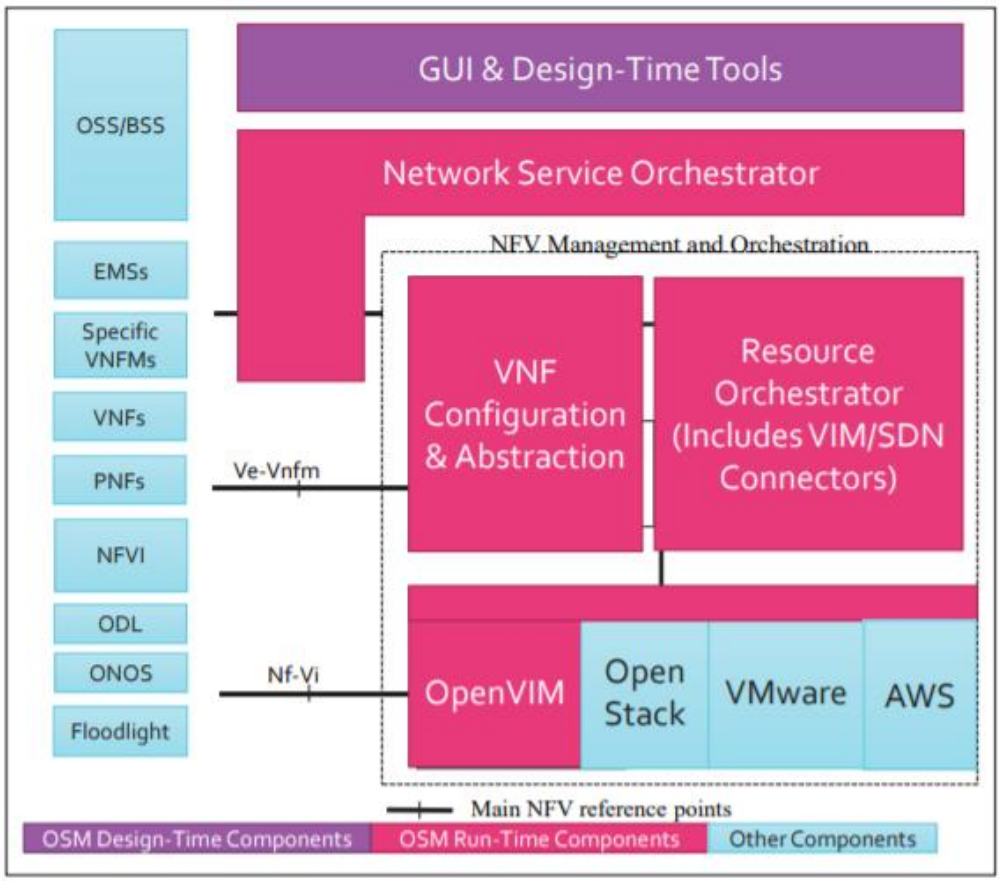


Figure 32. OSM mapping to ETSI NFV MANO [53]

The OSM has defined an expansive scope for the project covering both design-time and run-time aspects related to service delivery for telecommunications service provider environments. The run-time scope includes:

- An automated Service Orchestration environment that enables and simplifies the operational considerations of the various lifecycle phases involved in running a complex service based on NFV;
- A superset of ETSI NFV MANO where the salient additional area of scope includes Service Orchestration but also explicitly includes provision for SDN control;
- Delivery of a plugin model for integrating multiple SDN controllers;
- Delivery of a plugin model for integrating multiple VIMs, including public cloud based VIMs;
- Delivery of a plugin model for integrating multiple monitoring tools into the environment;
- One reference VIM that has been optimised for Enhanced Platform Awareness (EPA) to enable high performance VNF deployments;
- An integrated “Generic” VNFM with support for integrating “Specific” VNFMs.
- Support to integrate Physical Network Functions into an automated Network Service deployment;
- Being suitable for both Greenfield and Brownfield deployment scenarios;
- GUI, CLI, Python based client library and REST interfaces to enable access to all features;

The design-time scope includes:

- Support for a model-driven environment with Data Models aligned with ETSI NFV MANO;
- The capability for Create/Read/Update/Delete (CRUD) operations on the Network Service Definition;
- Simplifying VNF Package Generation;
- Supplying a Graphical User Interface (GUI) to accelerate the network service design time phase, VNF onboarding and deployment.

As shown in Figure 31, with the plugins model for VIMs and SDNs, the Resource Orchestration Engine is connected to specific interface provided by the VIMs and SDN controllers for managing and coordinating resource allocations across multiple geo-distributed VIMs and multiple SDN controllers. In addition, the VNF Configuration and Abstraction (VCA) layer enables configurations, actions and notifications to/from the VNFs and/or Element Managers. When backed by Juju, it provides the facility to create generic or specific indirect-mode VNFMs, via Charms that can support the interface the VNF/EM chooses to export. Overall, the OSM Release THREE substantially enhances interoperability with other components (VNFs, VIMs, SDN controllers, monitoring tools) and provides a plug-in framework to make platform maintenance and extensions significantly easier to provide and support, and thus it is proposed in SliceNet that this framework will be adopted in SliceNet MANO.

6 Practical Case Studies – Mobile Edge Apps

Studying the benefits of MEC as well as its practical cases was not possible due to the lack of implementation of ETSI MEC platform. We analyse three sample case studies that can be enabled with MEC in general and LL-MEC in particular. As a next step, the proposed SliceNet will be investigated leveraging the flexibility offered by the LL-MEC platform.

The considered demonstration scenario is illustrated in Figure 33 and consists of 2 commercial LTE-enabled smartphones (Huawei Nexus 6p), National Instrument/Ettus USRP B210 as RF front-end, and 4 Linux-based PC running OAI eNodeB, OAI core network, Open vSwitch v2.7, and Mosaic5G LL-MEC. The experiment is deployed in FDD SISO mode with 5MHz channel bandwidth. The target frequencies will be band 7 (Europe) radio environment.

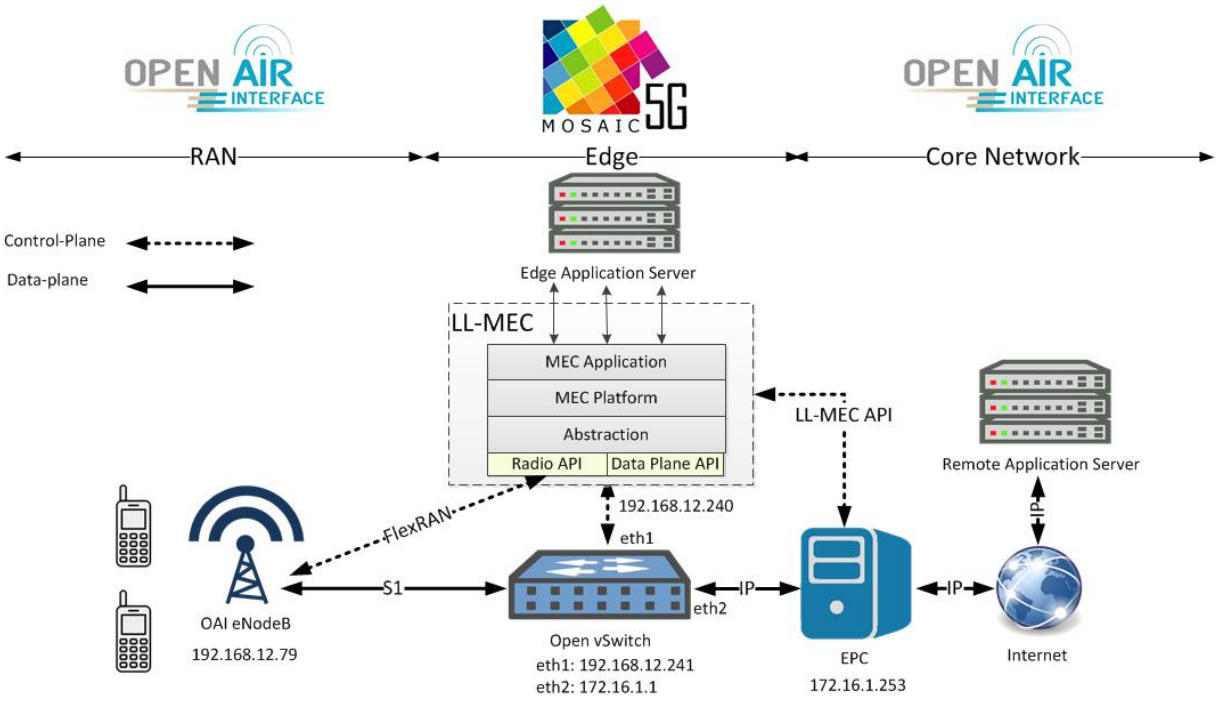


Figure 33. SliceNet MEC demonstration platform

6.1 End-to-End Mobile Network Slicing

Future 5G networks are envisioned to support a wide range of vertical segments with a diverse set of performance and service requirements. Network slicing can be seen as an enabler to share the physical network across multiple logically isolated networks. We consider LL-MEC as a platform to deploy network slicing leveraging 3GPP Gateway CN (GWCN) towards the enhanced dedicated core network (eDECOR 3GPP TR 23.711) to achieve isolation and performance guarantee in the Data Plane. In eDECOR, a UE indicates a slice ID that allows the eNodeB to select the appropriate CN elements for its traffic. The slice ID is indicated by the UE based on the encoded information in the UE, i.e. in the Universal Subscriber Identity Module (USIM), or can be simply mapped to the Public Land Mobile Network (PLMN). Moreover, the UE communicates the slice ID during the RRC connection procedure as well as in the Non-Access Stratum (NAS) procedure, which allows both eNB and MME to contain the UE within the requested slice(s) and treat it accordingly.

We assume no traffic differentiation within a slice and thus the same policy is applied to all the UEs within the slice whereas the policy between different slices are generally mutually independent. When realizing an E2E network slicing, the significant challenges come mostly from RAN due to the dependency on dedicated hardware (i.e. radio frontend), the time-varying radio resources, and mobility management among the others. Figure 34 shows how network programming (i.e. RAN and CN) is enabled in LL-MEC to create two network slices, where one gets higher over-the-air performance and served locally by MEC and the other gets the best-effort performance and directed to the backend server. This experiment is related to the SliceNet Smart Grid use-case, where the reaction time to an event is significantly reduced with the help of MEC.

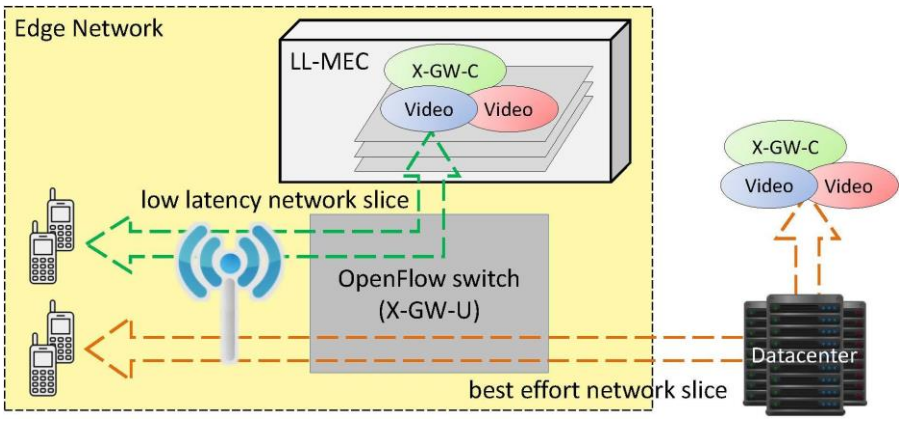


Figure 34. LL-MEC programmability in creating slices

More specifically, we design a slice policy enforcement algorithm to apply different resource allocation strategies to RAN and implement it as a low latency MEC application interfacing with the LL-MEC platform through the SDK. The Data-Plane programmability is enabled by the EPS and the real-time control decision can be delegated back to RAN.

Two slices are created and assigned with one Commercial Off-The-Shelf (COTS) UE each, and the percentage of radio resources and switching bandwidth for each slice are adjusted according to the applied slicing policy. To demonstrate the benefits of end-to-end slicing, we consider both uncoordinated and coordinated programmability for RAN and CN and change the enforced policy on-the-fly to measure the resulted downlink throughput. As illustrated in Figure 35, for uncoordinated case, the policy is first enforced at t=10s with 1Mbps for slice 1 and 15 Mbps for slice 2. Then at t=20s, a second policy is enforced only to RAN to lower the rate down to 8Mbps for both slices (equivalent to 50% of radio resources per slice). Finally, a third policy is enforced only to CN at t=33s to increase the switching bandwidth to 6 Mbps. In case of coordinated programmability, only one policy is enforced at t=18s to both RAN and CN to create a best-effort slice with 1Mbps and low latency slice with 15 Mbps.

The results confirm the benefits of MEC and SDN to allow coordinated programmability and enable the network slicing. In the case of uncoordinated slicing, some bandwidth is occupied but not used efficiently due to the asynchronous resource allocation between RAN and CN. However, for coordinated slicing, it can be clearly seen the performance gap between these two slices and the resources are appropriately allocated to each slice according to their specific requirements. While the results demonstrate the power of programmability in changing the behaviour and performance of the network, it has to be mentioned that unauthorized or inconsistent control decisions made from MEC applications can potentially lead to inefficient network utilization and performance or even a network failure. This

suggests that access control policy enforcement and conflict resolution among different MEC applications are definitely required in the production environment.

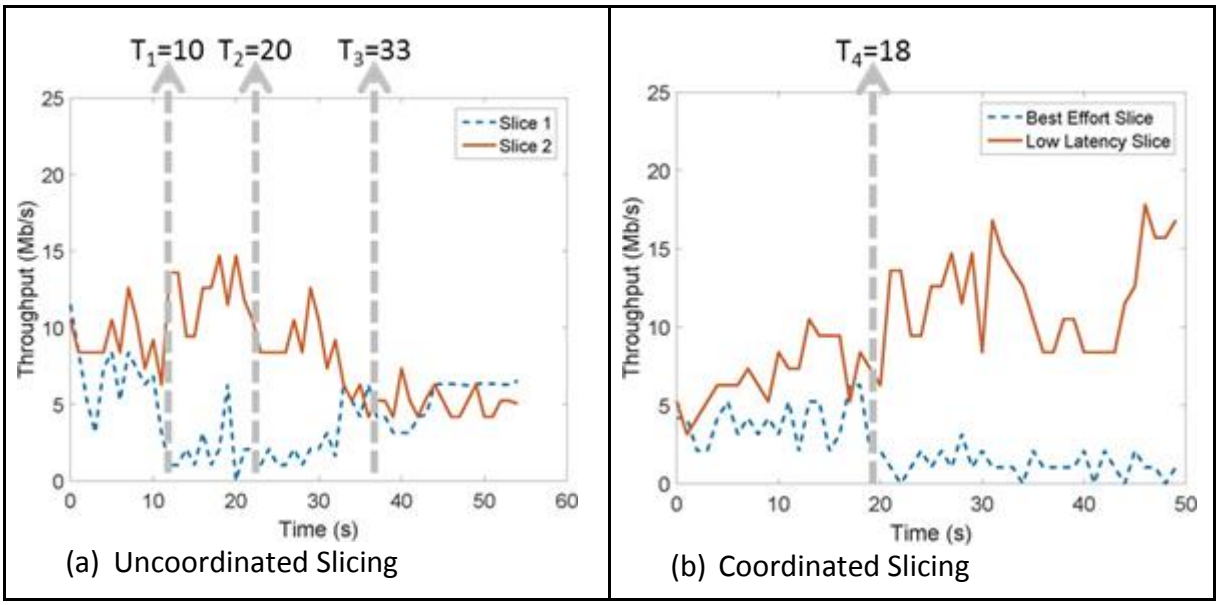


Figure 35. Mobile network slicing use case

6.2 RAN-Aware Video Optimization

As a showcase of mobile network slicing, RAN-aware content optimization is chose as the second use case. To demonstrate the benefits of LL-MEC, we consider video optimization as one of the MEC applications and study the benefit of RAN information reported by the eNodeB on improving user QoE. For example, the application can monitor the cell load status and radio link quality obtained from RNIS in order to enforce a new resource allocation policy or change the content quality. In the latter case, the video transcoding is further adapted based on the RAN status so as to improve the network efficiency (e.g. by avoiding TCP congestion control) and user QoE (e.g. avoid buffer freeze).

This use case is built on the top of the low latency network slice described above. We implement a simple video streaming application over HTTP on top of LL-MEC and choose Channel Quality Indicator (CQI) as a flag to reflect radio link quality of each UE. When UE accesses the video service, LL-MEC has the ability to (a) program the routing path and redirect the traffic to one of the MEC applications if the requested service matches (e.g. destination IP address), and (b) adapt dynamically the streaming rate according to the estimated UE throughput. Multiple approaches to provide throughput guidance can be applied on the top of LL-MEC RNIS producer app such as a statistical method, e.g. exponential moving average or even a discrete link quality to throughput mapping. This experiment is related to the SliceNet eHealth use-case, where the video streaming rate is adapted to the radio link condition with the help of MEC.

In Table 8, we show the maximum TCP bitrate of a video stream through a discrete mapping between user CQI and sustainable TCP throughput identified during experiments. This value is then used as a predicted user throughput allowing the video server to adjust the transcoding accordingly. The observed buffer freeze and perceived QoE (results are not shown here) at the user confirm such a RAN-aware content optimization enabled by LL-MEC.

Table 8. Measured maximum sustainable TCP bitrate with discrete congestion level based on CQI

CQI	Downlink (Mb/s)	Uplink (Mb/s)
11- 15	15.224	8.08
9- 11	11.469	6.04
7- 9	9.88	4.47
4- 7	5.591	2.49
0-4	1.08	0.69

The results reveal the benefit of the coordinated slicing and joint programmability managed by authorized MEC applications to achieve an effective mobile network. It is noted that the timescale of detecting CQI changes is much less than the one in TCP congestion mechanism. Instead of recovering congestion reactively, adapting the service demand proactively is also feasible through RNIS.

6.3 IoT Gateway

The MEC technology may be extended to the IoT services, as a platform used to aggregate and process the different IoT packets, with respect of scalability of resources. A specific implementation of IoT related to MEC is the Smart City vertical (Smart Lighting use case), which is in fact the massive Machine Type Communications UCs, aligned with the 3GPP. The MEC framework provides the networks communication aspects, 3GPP network, as an Mx interface between the UE app (users in general, sensors) and the MEC system. In case of different application and communication systems, it may be used also non-3GPP communication networks, as the LoRa or Wi-Fi.

The scenario is based on the massive IoT sensors deployment (tens of thousands of lighting poles with sensors), connected through seamless type technology to the provider network (MEC), by using dedicated IoT Gateways that plays the role of the network connectors. The scenario is relevant if the application used for the UCs is also considered critical and there are not local Data Centres. In the E2E vision, the IoT sensors may generate tens of millions of messages in a month, that are translated into thousands of messages per second, requiring a place of data processing before sending them to the edge or core application. There is also a requirement for security, as the application hosted in cloud (virtualized environment) are exposed to malicious attacks. These aspects and are treated at the locally, at the IoT Gateways level, that will handle the gap between the devices (sensors) and cloud apps.

The IoT Gateway, as a connector, must support any application and any device and it may be a physical device or a software application and pass raw data into a secure way to the central application (IoT platform, dashboard, command and control). In this aspect, the IoT Gateway module integrates a two-way communication between any 3rd party edge/core platform apps and the devices (sensors), assuring the interoperability and scalability and simplified communication, distributes the messages of different services and expose multiple IoT protocols.

The overall IoT network architecture for our use case, Smart Lighting, whose particularities and functional blocks were explained above, is presented on Figure 36.

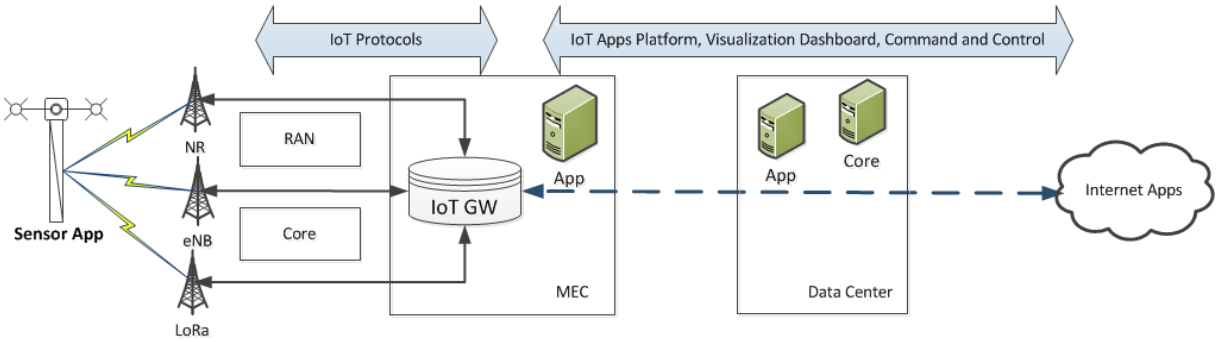


Figure 36. Overall IoT network architecture

As a third use case, we consider LL-MEC as a platform to deploy such an IoT Gateway at the edge leveraging the newly-introduced network slicing concept. Figure 37 shows a simplified workflow diagram on how the IoT traffics are directed to a dedicated user-plane function denoted as dedicated X-GW-U (De-X-GW-U) LL-MEC, based on the slice ID. Following the reception of the slice ID through attach request, the SMF, also referred as MME/SGW-C in 4G, maps the UE slice ID (stored in HSS) to the De-X-GW-U, and initiates a set of OpenFlow rules for this newly instantiated switch. Then, the tunnel information setup for De-X-GW-U is included in Initial Context Setup Request and sent to eNodeB. At this point, the dedicated Data Plane of the UE is established between the eNodeB and switch. This experiment is related to the Smart City use case, where a large number of sensory devices are served with the help of MEC.

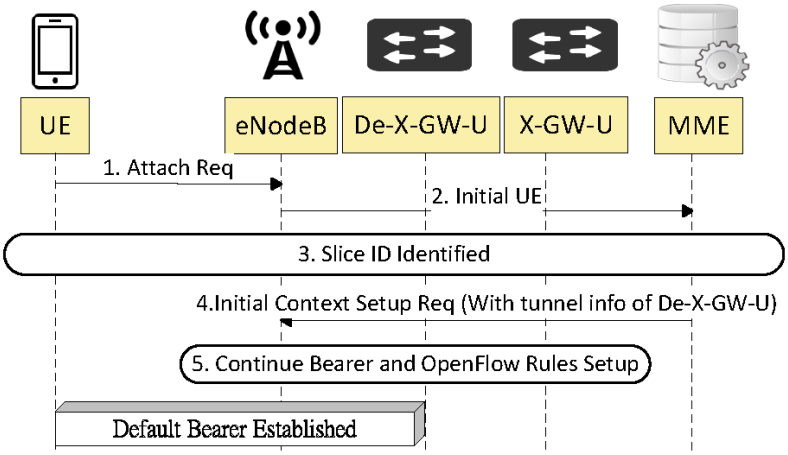


Figure 37. Workflow to establish a dedicated user-plane function

In this use case, 2000 UEs are considered, which are grouped into two slices of 1000 UEs. The massive LL-MEC S1-U emulator is used for sending sensory data to dedicated switches depending on the UE slice ID. The result of latency measurement is shown in Figure 38 with and without slicing. It can be observed that with the dedicated Data Plane, not only traffic isolation and scalability can be achieved, but also performance can be greatly improved by lowering the latency and its variability. With the help of LL-MEC, current architecture is ready to deploy IoT gateway for different island of sensory devices. As a result, IoT devices can be directed to a dedicated gateway for S1-U capabilities based on slice ID and achieve traffic isolation and security in terms of data privacy.

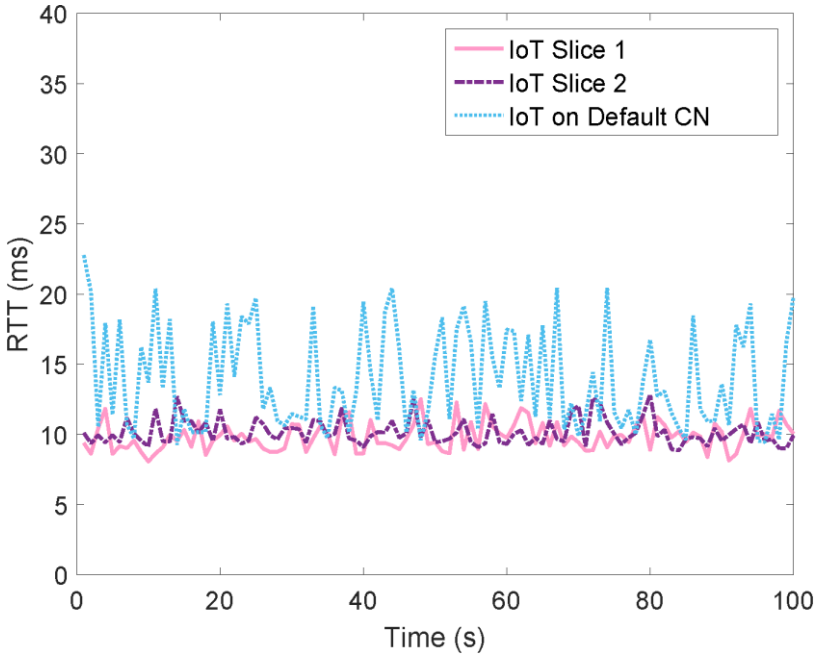


Figure 38. Latency measurements of isolated IoT slices

With MEC platform, an IoT Gateway control module can be deployed at the edge of the network for an efficient aggregation and management of messages sent by devices (sensors) towards the cloud and processing applications. It is designed to connect any type of sensors using different radio access network (4G, LTE-M, NR, LoRaWAN, etc.) or different dedicated UPF (see above). This device is built to assure bidirectional communication. Taking this into account, any admin or 3rd party applications can send commands through cloud to sensors/devices in the field.

Such an IOT gateway shall support divers IoT protocols. A general solution should have at least four main available connectivity protocols:

- **MQTT (Message Queuing Telemetry Transport):** MQTT is a lightweight connectivity protocol for IoT applications. It is based on the TCP/IP stack which uses the publish/subscribe method for transportation of data. MQTT consists of two broad categories of participating devices - they are called brokers and clients. MQTT works on a paradigm called the publish/subscribe method. A client can publish data regarding a certain parameter to the broker under a topic. Another client interested in this topic can subscribe to this topic and receive regular updates on messages under the topic.
- **CoAP (Constrained Application Protocol):** CoAP is a web transfer protocol based on the REST model. It is mainly used for lightweight M2M communication owing to its small header size. CoAP is built upon the UDP stack, which is the primary difference when compared with HTTP or MQTT. This makes it faster and more resource optimized rather than resource intensive. However, this also makes it less reliable than HTTP or MQTT, and QoS factors remain static in case of CoAP.
- **REST:** RESTful HTTP - Hyper Text Transfer Protocol, the most popular protocol for communication over the Internet. It runs on a client-server model, with the server responding to any client demands and it is necessary for this protocol to be built upon the TCP/IP stack.

- **LoRaWAN:** LoRaWAN is a media access control protocol for wide area networks. It is designed to allow low-powered devices to communicate with Internet-connected applications over long-range wireless connections. Using those protocols mentioned above, all sensors should connect to IoT Gateway over different type of access networks. E.g. 2G/3G/4G, LTE-M, LoRaWAN, NB-IoT or Bluetooth/Wi-Fi.

Figure 39 below depicts the IoT Gateway connectors architecture.

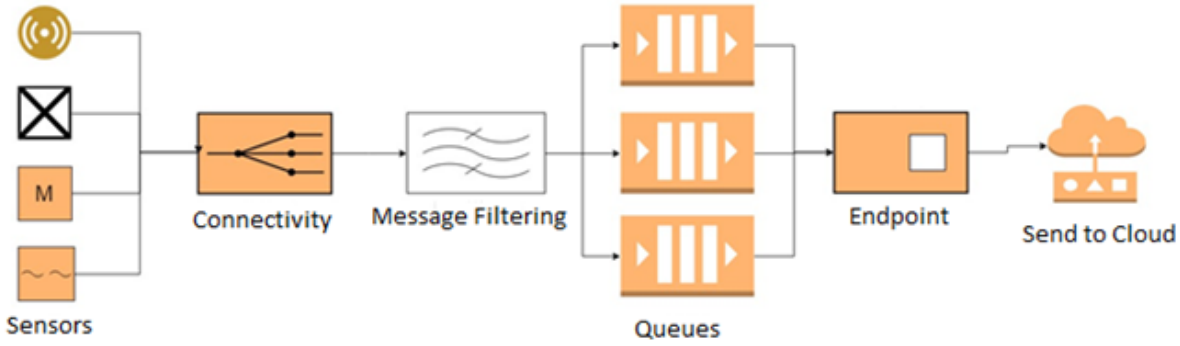


Figure 39. IoT gateway logical architecture

As mentioned above, an IoT Gateway should be a component that can allow bidirectional communications.

1. First flow, should gather data from all kind of sensors. Connectivity module is the interface that can be configured for specific protocol (according to each use case and access type). Message filtering is a function that sorts messages based on tags, id`s and forward the message to queue. Queues are configured based on clients/applications who listen for specifics messages. Endpoint block should expose an interface where some data can be extracted by anyone with access to it.
2. Second flow, should relay also on endpoint block where can call some methods or functions such that sensors can receive commands or some specific configurations through APIs.

IoT Gateway exposes a REST API with the following functionalities:

- Device management (command operations, parameters, inventory, etc.);
- API Key operations;
- Queues management;
- Alerts management;
- Web portal.

Software package for this type of application can be installed on a virtual machine with following amount of resources:

- CPU: 8 cores;
- RAM: 8 GB;
- Storage: 750 GB (depends on the type of architecture – if, for example, choose to use a database as cache system for faster search and indexing).

The IoT gateway is a key component of every IoT solution. Before decision regarding hardware for the gateway platform, it is important to analyse message flow and the data formats of the payloads and try to filter out or aggregate as much data as it can. In addition, while the choice of proper hardware for IoT solution is very important, picking up the right

gateway software and infrastructure is a factor that will highly affect the total maintenance and cost of entire system.

Specific 5G IoT use case requirements, based on the principle of deploying applications lower in the network, in order to meet various customers' requirements, starts from today implementations, as we face today's high latency apps implementations. 5G applications in this context requires medium or low latency or data processing, as close as possible to the client, using a MEC architecture.

7 Conclusions

This deliverable has presented the main activities related to the design and prototype of an open virtualised Mobile/Multi-access Edge Computing (MEC) infrastructure segment as part of the SliceNet end-to-end slicing-friendly infrastructure. An exhaustive analysis of the different programmable data path mechanisms has been carried out to provide a concrete design and prototype including selected suitable enablers towards allowing implementation of QoS-aware Network Slices in the MEC-Core network segment. It will enable the Slice Control and Slice Management capabilities envisioned in SliceNet to be created on top. An ETSI MEC platform has been prototyped to enable CP and UP programmability and the capabilities to facilitate the deployment of diverse applications over the edge of the network while enabling a slice-friendly infrastructure. Moreover, a number of considerations about the network Management Plane have been provided for the approach taken in the SliceNet consortium to address the management of MEC architecture. Prototyping details and empirical results have been provided to validate the essential technical approaches proposed in the MEC architecture. More empirical results will be presented in scientific publications.

References

[1] SliceNet, Deliverable 2.2 - Overall Architecture and Interfaces Definition, Jan. 2018.

[2] ETSI GS MEC 003, “Mobile Edge Computing (MEC); Framework and Reference Architecture”, Mar. 2016, available online at http://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/01.01.01_60/gs_MEC003v010101p.pdf

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: Enabling innovation in campus networks,” SIGCOMM Comput. Commun. Rev., 2008.

[4] X. Jin, L. E. Li, L. Vanbever, and J. Rexford, “Softcell: Scalable and flexible cellular core network architecture,” in Proc. the 9th ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT’13), 2013.

[5] X. Foukas, N. Nikaein, M. M. Kassem, M. K. Marina, and K. Kontovasilis, “Flexran: A flexible and programmable platform for software-defined radio access networks,” in Proc. the 12th International on Conference on Emerging Networking EXperiments and Technologies (CoNEXT’16), 2016.

[6] C.-Y. Chang, K. Alexandris, N. Nikaein, K. Katsalis, and T. Spyropoulos, “MEC architectural implications for LTE/LTE-A networks,” in Proc. ACM Workshop Mobility Evol. Internet Archit. (MobiArch), New York, NY, USA, Oct. 2016, pp. 13–18.

[7] A. Huang, N. Nikaein, T. Stenbock, A. Ksentini, and C. Bonnet, “Low latency MEC framework for SDN-based LTE/LTE-a networks,” in Proc. IEEE International Conference on Communications Conference, ser. ICC ’17, 2017.

[8] Open vSwitch, Feb. 2018. [Online]. Available: http://openvswitch.org/

[9] SimpleSumeSwitch Architecture, Feb. 2018. [Online]. Available: https://github.com/NetFPGA/P4-NetFPGA-public/wiki/Workflow-Overview

[10] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, “P4: Programming protocol-independent packet processors,” SIGCOMM Comput. Commun. Rev., vol. 44, no. 3, pp. 87–95, Jul. 2014

[11] B. Claise, B. Trammell, and P. Aitken, “Specification of the IP flow information export (IPFIX) protocol for the exchange of flow information,” IETF, Fremont, CA, USA, RFC 7011, Sep. 2013. [Online]. Available: http://www.ietf.org/rfc/rfc7011.txt

[12] B. Trammell and E. Boschi, “An introduction to IP flow information export (IPFIX),” IEEE Commun. Mag., vol. 49, no. 4, pp. 89–95, Apr. 2011. [Online]. Available: http://dx.doi.org/10.1109/MCOM.2011.5741152

[13] Claise, B., Ed., Johnson, A., and J. Quittek, "Packet Sampling (PSAMP) Protocol Specifications", RFC 5476, Mar. 2009.

[14] Dietz, T., Kobayashi, A., Claise, B., and G. Muenz, “Definitions of Managed Objects for IP Flow Information Export", RFC 6615, Jun. 2012.

[15] Muenz, G., Claise, B., and P. Aitken, "Configuration Data Model for the IP Flow Information Export (IPFIX) and Packet Sampling (PSAMP) Protocols", RFC 6728, Oct. 2012.

[16] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, Jun. 2011.

[17] Claise, B., Ed., "Cisco Systems NetFlow Services Export Version 9", RFC 3954, Oct. 2004.

[18] sFlow specifications, Feb. 2018. [Online]. Available: http://www.sflow.org/developers/specifications.php

[19] Comparison of sFlow with other technologies, in Traffic Monitoring using sFlow, Feb. 2018. [Online]. Available: http://www.sflow.org/sFlowOverview.pdf

[20] B. Pfaff and B. Davie, “The Open vSwitch Database Management Protocol”, RFC 7047, Dec. 2013.

[21] A. Bregman, Open vSwitch: Introduction – Part 2, Oct. 2016, [Online]. Available: http://abregman.com/2016/10/19/open-vswitch-introduction-part-2/

[22] Open vSwitch Manual, Feb. 2018. [Online]. Available: http://www.openvswitch.org//ovs-vswitchd.conf.db.5.pdf

[23] N. Zilberman, Y. Audzevich, G. Covington, and A. Moore, “NetFPGA SUME: Toward 100 Gbps as Research Commodity”, IEEE Micro, vol.34, no.5, pp.32,41, Sep.-Oct. 2014

[24] Netcope Technologies, “Netcope FPGA Boards,” Feb. 2018. [Online]. Available: http://www.netcope.com/en/products/fpga-boards

[25] The P4 Language Consortium, “P4→NetFPGA: A low-cost solution for testing P4 programs in hardware.” [Online]. Available: https://p4.org/p4/p4-netfpga-a-low-cost-solution-for-testing-p4-programs-in-hardware.html

[26] DPDK, Feb. 2018. [Online]. Available: https://dpdk.org/

[27] Introduction to DPDK: Architecture and Principles, Feb. 2018. [Online]. Available: https://blog.selectel.com/introduction-dpdk-architecture-principles/

[28] XDP, Feb. 2018. [Online]. Available: https://www.iovisor.org/technology/xdp

[29] IO Visor Project, Feb. 2018. [Online]. Available: https://www.iovisor.org/

[30] RF_Ring, Feb. 2018. [Online]. Available: https://www.ntop.org/products/packet-capture/pf_ring/

[31] PF_Ring Zero Copy, Feb. 2018. [Online]. Available: https://www.ntop.org/products/packet-capture/pf_ring/pf_ring-zc-zero-copy/

[32] Overview of Single Root I/O Virtualization (SR-IOV), Apr. 2017. [Online]. Available: https://docs.microsoft.com/en-us/windows-hardware/drivers/network/overview-of-single-root-i-o-virtualization--sr-iov-

[33] Intel, “Virtual Machine Device Queues”, Technical White Paper, Feb. 2018. [Online]. Available: https://www.intel.co.uk/content/www/uk/en/virtualization/vmdq-technology-paper.html

[34] OpenStack, Feb. 2018. [Online]. Available: https://www.openstack.org/

[35] OpenDayLight, Feb. 2018. [Online]. Available: https://www.opendaylight.org/what-we-do/current-release/lithium

[36] “Brocade SDN Controller OpenStack Integration Guide”, Feb. 2018, available online at http://workflowcomposer.net/content/html/en/deployment-guide/brocade-sdn-openstack-integration-dp/GUID-3230BA97-29D2-4E05-B7A3-5BB9390A1B3A.html

[37] OpenDayLight, “OpenDaylight with OpenStack Guide”, Feb. 2018. [Online]. Available: http://docs.opendaylight.org/en/stable-nitrogen/opendaylight-with-openstack/index.html

[38] OpenDayLight, “Release/Lithium/VTN/Developer Guide/OpenStack Support”, Feb. 2018. [Online] Available: https://wiki.opendaylight.org/view/Release/Lithium/VTN/Developer_Guide/OpenStack_Support

[39] Comparison of Availability Zone and Host Aggregate in OpenStack, Feb. 2018, available online at https://www.datadoghq.com/blog/openstack-host-aggregates-flavors-availability-zones/

[40] AWS 101: Learning About Regions and Availability Zones, Feb. 2018, available online at https://cloudarchitectmusings.com/2017/02/07/aws-101-learning-about-regions-and-availability-zones/

[41] SDNet Compiler Installation, Release Notes, and Getting Started Guide, Feb. 2018. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug1018-sdnet-installation.pdf

[42] Xilinx SDK 2016.4, Feb. 2018. [Online]. Available: https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-design-tools/2016-4.html

[43] SDNet Packet Processor User Guide, Feb. 2018. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug1018-sdnet-installation.pdf

[44] OpenStack Architecture Design Guide, Feb. 2018. [Online]. Available: https://docs.openstack.org/arch-design/

[45] OpenStack Magnum’s Developer Documentation, Feb. 2018. [Online]. Available: https://docs.openstack.org/magnum/latest/

[46] ETSI GS MEC 001, “Mobile-Edge Computing (MEC); Terminology”, Mar. 2016, available online at http://www.etsi.org/deliver/etsi_gs/MEC/001_099/001/01.01.01_60/gs_MEC001v010101p.pdf

[47] Juju, Feb. 2018. [Online]. Available: https://Jujucharms.com/

[48] Juju for Telcos and Service Providers Pt. 2, Feb. 2018. [Online]. Available: https://insights.ubuntu.com/2015/07/23/Juju-for-telcos-and-service-providers-pt-2

[49] Juju Charm Store, Feb. 2018. [Online]. Available: https://Jujucharms.com/store

[50] Open Baton, Feb. 2018. [Online]. Available: https://openbaton.github.io/

[51] K. Katsalis, N. Nikaein, and A. Huang, “JOX: an event-driven orchestrator for 5G network slicing”, in Proc. IEEE/IFIP Network Operations and Management Symposium, 2018.

[52] OSM, Feb. 2018. [Online]. Available: https://osm.etsi.org/

[53] OSM Release THREE, Feb. 2018. [Online]. Available: https://osm.etsi.org/images/OSM-Whitepaper-TechContent-ReleaseTHREE-FINAL.pdf

[End of document]