5G CITY

Grant Agreement No.761508 5GCITY/H2020-ICT-2016-2017/H2020-ICT-2016-2

D3.1: 5GCity Edge Virtualization Infrastructure Design

Dissemination Level			
	☒	PU:	Public
	☐	PP:	Restricted to other programme participants (including the Commission Services)
	☐	RE:	Restricted to a group specified by the consortium (including the Commission
	☐	CO:	Confidential, only for members of the consortium (including the Commission Services)

Grant Agreement no: **761508**	Project Acronym: **5G CITY**	Project title: **5G CITY**

Lead Beneficiary: **VOSYS**	Document version: **V3.0**

Work package: **3**

Deliverable title: 5GCity Edge Virtualization Infrastructure Design

Start date of the project: 01/06/2017 (duration 30 months)	Contractual delivery date: M12	Actual delivery date: **31/05/2018**

Editor name: Michele Paolino (VOSYS)

List of Contributors

Participant	Short Name	Contributor
Virtual Open Systems	VOSYS	Michele Paolino, Teodora Sechkova, Daniel Raho
Nextworks s.r.l.	NXW	Nicola Ciulli, Paolo Cruschelli, Elian Kraja, Elio Francesconi
NEC Europe ltd.	NEC	Felipe Huici, Sharan Santhanam, Nicolas Weber
Fundació i2CAT	I2CAT	August Betzler, Joan Josep Aleixendri, Alfonso Egio
ADLINK TECHNOLOGY SARL	ADLINK	Gabriele Baldoni
Italtel	ITL	Antonino Albanese, Viscardo Costa
Accelleran	Accelleran	Trevor Moore, Simon Pryor

List of Reviewers

Participant	Short Name	Contributor
Fundació i2CAT	I2CAT	Shuaib Siddiqui
ITALTEL	ITL	Antonino Albanese, Viscardo Costa

Change History

Version	Date	Partners	Description/Comments
0.1	30/03/2018	VOSYS	First Table of content
1.0	27/04/2018	VOSYS, NXW, NEC, I2CAT	First contributions
1.8	11/05/2018	VOSYS, ITL, NXW, XLRN, ADLINK	First revision from contributors, added missing sections
2.0	14/05/2018	VOSYS	VOSYS review with comments for the partners
2.1	14/05/2018	ADLINK, ITL, NXW, I2CAT, XLRN	Comments addressed from partners
2.4	17/05/2018	VOSYS	Consolidated version for review
2.5	18/05/2018	VOSYS, NXW	Comments from NXW integrated in the consolidated version
2.7	24/05/2018	VOSYS, ITL, I2CAT, NXW, ADLINK	ITL and I2CAT reviews integrated with comments addressed
3.0	30/05/2018	VOSYS, I2CAT, NEC, ADLINK, NXW	Addressed comments from the project coordinator and technical manager

DISCLAIMER OF WARRANTIES

5GCity has received funding from the European Union’s Horizon 2020 research and innovation program under grant agreement No 761508. The content of this deliverable does not reflect the official opinion of the European Union. Responsibility for the information and views expressed in the deliverable lies entirely with the author(s).

Table of Content

Figures

Tables

Executive Summary

This deliverable presents the characteristics of the 5GCity virtualization platform and MEC node, which aims at building the city of the future 5G ICT service infrastructure by combining NFV and MEC concepts, extending state of the art virtualization techniques for the execution of virtual machines and for their networking features.

In the first part of the document, the project's approach towards the coexistence of ETSI MEC and NFV standardization activities is explained. The key differences between them are presented, together with the planned activities to make them coexist in the 5GCity infrastructure. Then in Section 3 and Section 4, the computing and the networking virtualization features of the 5GCity infrastructure are detailed. Both Sections start with a description of what is the state of the art of each technology today, in order to introduce and clarify the advancements that will be developed in 5GCity, which are detailed respectively in Sections 3.2 and 4.2.

In greater detail, 5GCity computing extensions include unikernels and VMs developments to improve performance, efficiency (Unikraft, KVM) and security (EdgeNFVI, EdgeVIM). On the other hand, wireless slicing (RAN and Wi-Fi virtualization), vSwitch acceleration (VOSYSwitch) and MEC service function chaining are part of the project's networking enhancements presented in this document.

1. Introduction

Smart cities have the objective of providing multiple added value services to citizens, companies or other entities while minimizing the infrastructure costs in a way to address performance requirements (computing power, bandwidth, power consumption, security, etc.) with the highest efficiency. These services can be very different from each other, and must be properly isolated to guarantee privacy, programmability and infrastructure openness.

In this context, virtualization is a technology of pivotal importance because it abstracts computing and networking infrastructure resources to provide application with useable logic instances (e.g., virtual machines, unikernels[1], network slices, etc.) that are fundamental for NFV and MEC platforms.

However, the high number of heterogeneous devices interconnected to build the city infrastructure together with the geographically scattered nature of cabinets, smart gateways, lampposts and the mobility of smart devices represent a challenge for virtualization. Moreover, these new technologies will enhance current smart cities services by delivering almost real time services, which will allow the emergence of a new generation of applications.

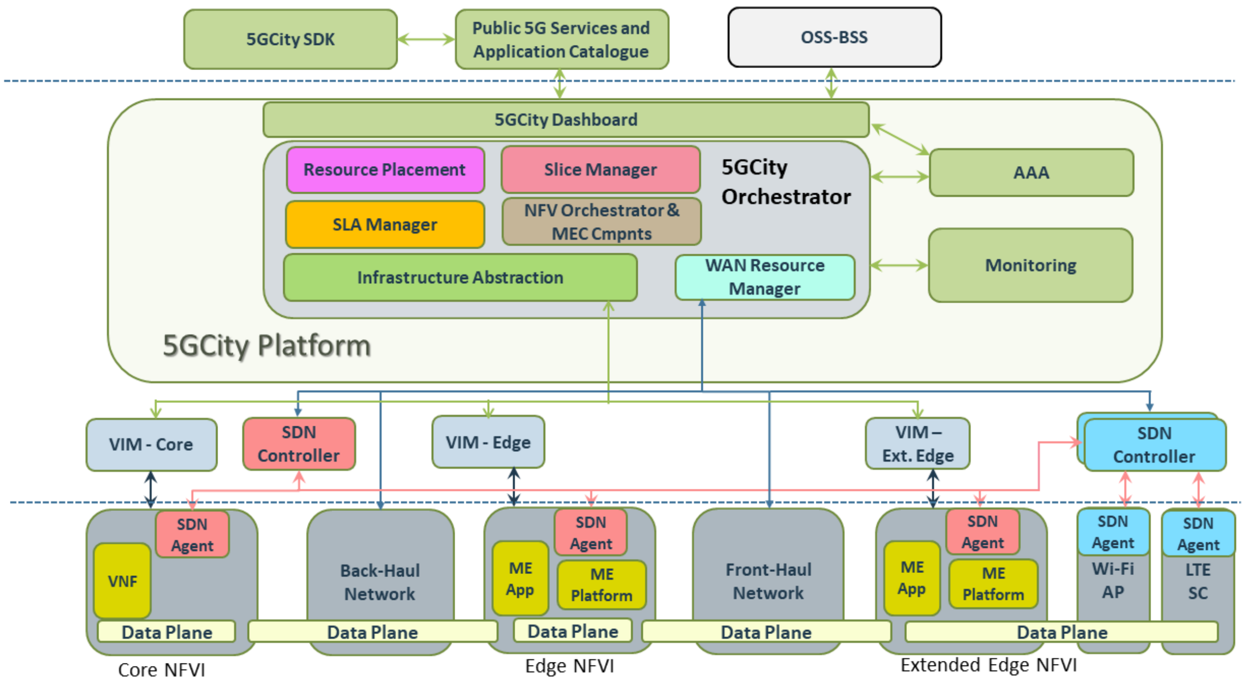

Figure 1. The 5GCity architecture, with the Infrastructure Layer at the bottom

This deliverable documents the activities developed within WP3, which has the objective of developing the project virtualization platform by:

i) Optimizing virtualization technologies for heterogeneous and resource constrained devices,

ii) Implementing network virtualization targeting efficient software switches and wireless virtualization,

iii) Creating specific VNFs data models and systems for MEC nodes.

[1] Please see: http://unikernel.org/

WP3 activities focus on the lowest layer of the project architecture depicted in Figure 1, (Infrastructure Layer) and together with the VIMs set the ground for the execution of the 5GCity Platform.

In this document, the project positioning with respect to the ETSI MEC and NFV standard is detailed in Section 2. Differences between the two specification groups activities are shown, followed by a description of both the implementation of existing components and those that the 5GCity project aims to develop.

Afterwards, the virtualization of computing resources to run multiple applications, isolated in a multi-tenant environment, are detailed in Section 3. The state of the art of existing virtualization solutions (KVM[2], unikernels and VOSYSmonitor) is also described together with the current virtualization approach towards trusted computing and security. These details introduce the 5GCity enhancements that are explained later on in Section 3.2, where Unikraft[3], the Edge NFVI and the EdgeVIM design and implementations are also described.

Section 4 uses a similar structure to describe the project activities in the direction of networking virtualization. OVS-DPDK, VOSYSwitch, RAN slicing, service function chaining, Wireless and eNB virtualization state of the art are described in Section 4.1 while the project enhancements are highlighted in Section 4.2.

[2] Please see: https://www.linux-kvm.org/

[3] More info at: https://www.xenproject.org/developers/teams/unikraft.html

2. Applicability of ETSI MEC architecture to 5GCITY

While ETSI NFV (Network Function Virtualisation) has been around for a while now, ETSI MEC (Multi access Edge Computing) is much newer. It has changed its scope during its lifecycle by shifting its focus towards an architecture which can encompass multi-access technology. NFV has grown in popularity because it enables service providers to replace network appliances with software running on servers, enabling cost reduction, service innovation and deployment acceleration. While ETSI MEC is tailored to edge computing coupled with mobile access technology, it takes the same principles that drive NFV and optimizes them for the mobile environment.

In this section, ETSI NFV and ETSI MEC similarities and differences are described to detail the 5GCity approach towards the coexistence of both within the project's infrastructure (Section 2.3).

2.1 ETSI MEC - NFV architectures mapping

In this section, we will provide a rationale which describes a potential mapping between NFV and MEC reference architectures. Their similarities are described below

- **Standard platform.** similar to NFV, MEC is built on top of a stack of standard components, including a well-defined compute platform and virtualization layer.
- **Open environment.** MEC is designed to promote innovation through openness and interoperability, just like NFV.
- **Software-focused.** While both NFV and MEC need hardware, the emphasis is on moving functionality to software. Doing so brings benefits in terms of scalability, commercial models, and speed of innovation and deployment.

NFV and MEC have similarities and a common heritage, but clearly show some differences. The main differences are those related to the type, location and scope of the applications they target.

- **Application Type**: NFV can address a wide variety of existing network functions and applications, including routing, VPNs, firewalls, security, voice applications including IMS and SBC and so on. Each of these is independent and uses NFVI for basic hosting and networking functions. In contrast, MEC provides a more focused MEC application platform specifically designed for supporting services associated with radio access. The MEC application platform provides an abstracted way to interface with the complexity of the radio network, enabling new applications. MEC applications depend on a set of middleware services which are hosted on a MEC server:
  - Service registry;
  - Radio Network Information Services (RNIS);
  - Traffic Offload Function (TOF).
- **Application Scope**: MEC is designed to support high-level mobility applications. With MEC, wireless operators, over-the-top providers and enterprises can quickly build advanced mobility applications that are small and portable. NFV addresses a much broader set of arbitrary network applications.
- **Application Location**: Consumers are insatiable in their desire for quick access to bandwidth-intensive applications. In addition, emerging requirements for 5G deployments will stipulate bandwidth and latency requirements. MEC is designed to be implemented in the access part of the

network (at the micro or macro-cell or the first aggregation point) to help answer this demand by maximizing bandwidth and minimizing latency. In contrast, NFV is targeted at deployments throughout the network.

A specific ETSI MEC working group has been created to design a solution which maps ETSI MEC to an ETSI NFV architecture. The ETSI report **[1]** which it produced contains:

- A set of initial assumptions;
- A potential solution which provides a mapping of MEC to NFV architecture;
- An ordered list of open issues, which are raised by the proposed mapping.

The following assumptions have been done:

- The mobile edge platform is deployed as a VNF. For that purpose, the procedures defined by ETSI NFV is used. It is not expected that these procedures need to be modified for use with ETSI MEC.
- The mobile edge applications appear like VNFs towards the ETSI NFV MANO components. This allows re-use of ETSI NFV MANO functionality. It is however expected that ETSI MEC might not use the full set of MANO functionality, and require certain additional functionality. Such a specific mobile edge application is denoted by the name "ME app VNF" in the remainder of the present document.
- The virtualisation infrastructure is deployed as a NFVI and its virtualised resources are managed by the VIM. For that purpose, the procedures defined by ETSI NFV Infrastructure specifications, i.e. ETSI GS NFV INF-003 [2], ETSI GS NFV INF-004 [3], ETSI GS NFV INF-005 [4], can be used. It is not expected that these procedures need to be modified when used with the ETSI MEC.

The solution envisioned in [1], and described in Figure 2 encompasses all the reference points described in ETSI NFV (green lines) and ETSI MEC (blue line), while further suggesting a new category of cross-area reference point (red lines). The new reference points (Mv1, Mv2 and Mv3) are introduced between elements of the ETSI MEC architecture and the ETSI NFV architecture to support the management of ME app VNFs. These are related to existing NFV reference points, but it is expected that only a subset of the needed functionalities will be used for ETSI MEC, while some extensions may be necessary:

- **Mv1**: This reference point connects the mobile edge application orchestrator (MEAO) and the NFVO. It is related to the Os-Ma-nfvo reference point as defined in ETSI NFV.
- **Mv2**: This reference point connects the VNF Manager of that performs the LCM of the ME app VNFs with the MEPM-V to allow Lifecycle Management related notifications to be exchanged between these entities. It is related to the Ve-Vnfm-em reference point as defined in ETSI NFV, but will possibly include additions, and might not use all functionality offered by Ve-Vnfm-em.
- **Mv3**: This reference point connects the VNF Manager with the ME app VNF instance, to allow the exchange of messages (e.g., related to mobile edge application lifecycle management or initial deployment-specific configuration). It is related to the Ve-Vnfm-vnf reference point as defined in ETSI NFV, but will possibly include additions, and might not use all functionality offered by Ve Vnfm-vnf.

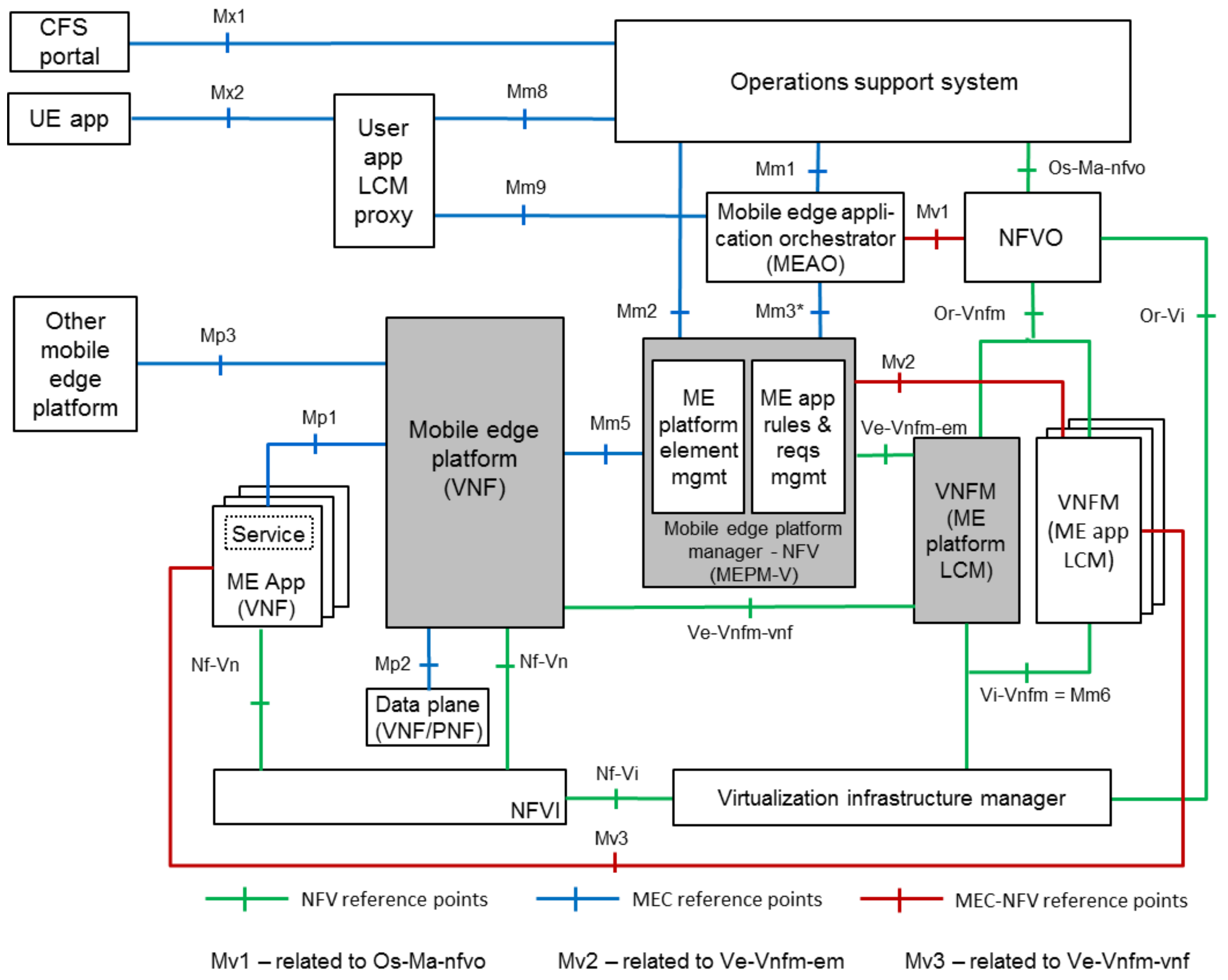

Figure 2. Mapping between ETSI MEC and ETSI NFV architectures [1]

The mapping described in [1], as depicted in Figure 2, leaves a number of open issues, identified in Table 1. For each one of the identified issues a tentative solution and an evaluation of the impact in the actual NFV landscape have been provided by [1].

ISSUE#ID	ISSUE NAME	Proposal	Impacted area	Impact
ISSUE#1	Mapping of ME app VNFs to Network Services	It is suggested that the MEAO arranges with the NFVO via Mv1 to manage the ME app VNF instances as part of one or more NSs	Interfaces Mv1 (MEAO-NFVO)	HIGH
ISSUE#2	Usage of NFV Network Service	Use the concept of NSs to represent the set of ME app VNFs and ME platform VNFs and their interconnection/dependency.	information model	HIGH
ISSUE#3	Communication between MEAO and NFVO via Mv1	Under analysis by ETSI MEC WG.	Interfaces Mv1 (MEAO-NFVO)	HIGH
ISSUE#4	Communication between VNFM and MEPM-V via Mv2	Certain functionalities as provided by the Ve-Vnfm-em reference point will be used between the MEPM-V and the VNFM that manages the lifecycle of the ME app VNFs, as discussed in clause 6.4.3	Interfaces Mv2 (VNFM, MEPM-V)	MEDIUM

ISSUE#5	Communication between VNFM and ME app instance via Mv3	Under analysis by ETSI MEC WG.	Interfaces Mv3 (VNFM-MeApp)	MEDIUM
ISSUE#6	AppD vs. VNFD for ME app VNFs	Under analysis by ETSI MEC WG.	information model	MEDIUM
ISSUE#7	VNF Package vs. MEC application package	Identify with ETSI NFV whether an extension mechanism for VNF Packages exists that can be re-used, and what are the rules for re-use. If it does not exist, suggest to ETSI NFV to specify such a mechanism	information model	MEDIUM
ISSUE#8	VNF package onboarding	**Solution 1:** If the MEAO is the master, the ME app package would first be provided by the OSS to the MEAO via Mm1, and onboarded to the NFVO by the MEAO via Mv1, using procedures defined in ETSI GS NFV-IFA 013 [5]. In that case, the MEC specific extensions of the VNF package are directly available to the MEAO, as the package passes through the MEAO. **Solution 2:** If the NFVO is the master, the ME app package would be onboarded directly into the NFVO by the OSS via Os-Ma-nfvo. Via Mv1, the MEAO would be notified about package onboarding, and would be able to subsequently fetch whole packages or the needed package parts (so called package artifacts), using procedures defined in ETSI GS NFV-IFA 013 [5]. This would allow the MEAO to access the MEC specific extensions of the VNF package	VNF Lyfe Cicle management	HIGH
ISSUE#9	Managing traffic redirection	The Data Plane in a MEC in NFV deployment may be realised by means outside the scope of the ETSI MEC specifications (solution 1). It may also be realised based on the NFP mechanism defined in ETSI NFV (solution 2).	MEC dataplane configuration	LOW
ISSUE#10	Comparison of AppD and VNFD data structures	Under analysis by ETSI MEC WG.	Information model	MEDIUM
ISSUE#11	NFV construct that corresponds to Mobile Edge Host	ETSI ISG NFV has defined several constructs to structure an NFVI, such as NFVI-PoP (basically, a data center) and Zone (a set of co-located and well-connected physical resources which is a subset of an NFVI-PoP).	Mobile edge host definition NFVI definition	LOW
ISSUE#12	ME App VNF Instance Relocation	Under analysis by ETSI MEC WG.	VNF Lyfe Cicle management	LOW
ISSUE#13	Application instantiation	Under analysis by ETSI MEC WG.	VNF Lyfe Cicle management	LOW
ISSUE#14	Application instance termination	Under analysis by ETSI MEC WG.	VNF Lyfe Cicle management	LOW

Table 1. List of Issue raised from the proposed ETSI MEC to ETSI NFV mapping [1]

Note that these 14 issues originate from one of the main differences between ETSI MEC and ETSI NFV, which is linked with the information model needed to describe MEC Applications.

Attribute name	Qualifier	Cardinality	Data type	Description
appDId	Mandatory	1	String	Identifier of this MEC application descriptor. This attribute shall be globally unique. See note 1.
appName	Mandatory	1	String	Name to identify the MEC application.
appProvider	Mandatory	1	String	Provider of the application and of the AppD.
appSoftVersion	Mandatory	1	String	Identifies the version of software of the MEC application.
appDVersion	Mandatory	1	String	Identifies the version of the application descriptor.
mecVersion	Mandatory	1..N	String	Identifies version(s) of MEC system compatible with the MEC application described in this version of the AppD.
appInfoName	Mandatory	0..1	String	Human readable name for the MEC application product. May change during the MEC application product lifetime.
appDescription	Mandatory	1	String	Human readable description of the MEC application.
virtualComputeDescriptor	Mandatory	1	VirtualComputeDescription	Describes CPU, Memory and acceleration requirements of the Virtualisation machine.
swImageDescriptor	Mandatory	1	SwImageDescriptor	Describes the software image which is directly loaded on the virtualisation machine instantiating this Application.
virtualStorageDescriptor	Mandatory	0..N	VirtualStorageDescriptor	Defines descriptors of virtual storage resources to be used by the MEC application.
appExtCpd	Mandatory	0..N	AppExternalCpd	Describes external interface(s) exposed by this MEC application.
appServiceRequired	Mandatory	0..N	ServiceDependency	Describes services a MEC application requires to run.
appServiceOptional	Mandatory	0..N	ServiceDependency	Describes services a MEC application may use if available.
appServiceProduced	Mandatory	0..N	ServiceDescriptor	Describes services a MEC application is able to produce to the platform or other MEC applications. Only relevant for service-producing apps.
appFeatureRequired	Mandatory	0..N	FeatureDependency	Describes features a MEC application requires to run.
appFeatureOptional	Mandatory	0..N	FeatureDependency	Describes features a MEC application may use if available.
transportDependencies	Mandatory	0..N	TransportDependency	Transports, if any, that this application requires to be provided by the platform. These transports will be used by the application to deliver services provided by this application. Only relevant for service-producing apps. See note 2.
appTrafficRule	Mandatory	0..N	TrafficRuleDescriptor	Describes traffic rules the MEC application requires.
appDNSRule	Mandatory	0..N	DNSRuleDescriptor	Describes DNS rules the MEC application requires.
appLatency	Mandatory	0..1	LatencyDescriptor	Describes the maximum latency tolerated by the MEC application.
terminateAppInstanceOpConfig	Mandatory	0..1	TerminateAppInstanceOpConfig	Configuration parameters for the Terminate application instance operation.
changeAppInstanceStateOpConfig	Mandatory	0..1	ChangeAppInstanceStateOpConfig	Configuration parameters for the change application instance state operation.
NOTE 1: The appDId shall be used as the unique identifier of the application package that contains this AppD. NOTE 2: This attribute indicates groups of transport bindings which a service-producing MEC application requires to be supported by the platform in order to be able to produce its services. At least one of the indicated groups needs to be supported to fulfil the requirements. NOTE3: The "Qualifier" column indicates whether the support of the attribute is mandatory, optional or conditional. NOTE4: The "Cardinality" column contains the minimum and maximum cardinality of this information element (e.g. 1, 2, 0..N, 1..N).				

Table 2. Attributes of the AppD descriptor [6]

As stated in **[6]**, an application Descriptor (AppD) is a part of the application package and describes application requirements, and rules, required by the application provider to be able to deploy a MEC

Application. Table 2 provides the full attributes which composes the AppD descriptor, while Table 3. provides a preliminary analysis of the gap between AppD and VNFD deployment templates **[1]**.

VNFD attribute	AppD attribute
vnfdId	appDId
vnfProvider	appProvider
vnfProductName	appName
vnfSoftwareVersion	appSoftVersion
vnfdVersion	appDVersion
	mecVersion
vnfProductInfoName	appInfoName
vnfProductInfoDescription	appDescription
vnfmInfo	
localizationLanguage	
defaultLocalizationLanguage	
vdu	
>swImageDescriptor	swImageDescriptor
virtualComputeDesc	virtualComputeDescriptor
virtualStorageDesc	virtualStorageDescriptor
intVirtualLinkDesc	
vnfExtCpd	appExtCpd
	appServiceRequired
	appServiceOptional
	appServiceProduced
	appFeatureRequired
	appFeatureOptional
	transportDependencies
	appTrafficRule
	appDNSRule
	appLatency
deploymentFlavour	
>vnfLcmOperationsConfiguration	
	terminateAppInstanceOpConfig
	changeAppInstanceStateOpConfig
configurableProperties	
modifiableAttributes	
lifeCycleManagementScript	
elementGroup	
vnfIndicator	
autoScale	

Table 3. High-level comparison of VNFD and AppD descriptors [1]

It is clear that the main difference is composed by a set of additional attributes of the AppD descriptor which are needed to describe the dependencies between the MEC application and a set of services provided by the MEC Platform.

2.2 Existing MEC Solutions

Open source software is expected to play an important role in future 5G systems and MEC solutions as well. The ever-increasing number of new use cases and the need for devices with different form factors are a challenge for the entire industrial landscape. While a small number of leading smartphone manufacturers already provide state-of-the-art mobile devices, 5G is likely to shift production to a large number of small operators specializing in specific niche markets. These industries are expected to rely on open source development and access kits in order to adapt their respective products to selected markets. Naturally, the main players will provide commercial mass markets.

To date, we have knowledge of only one open source software implementation of ETSI MEC, which is LL-MEC (within Mosaic5G ecosystem [7], [8]), briefly explained in the following paragraph.

2.2.1 A Low Latency Multi-access Edge Computing Platform for Software-Defined Mobile Networks

The open source LL-MEC implements two main parts: the LL-MEC platform (Figure 3) and the data control APIs. LL-MEC is responsible for providing two main services: native IP service endpoints and real-time radio network information to MEC applications per user and service. It can be connected to a number of underlying RAN and CN gateways. The data plane APIs act as a layer of abstraction between the RAN and CN data plane and the LL-MEC platform. The OpenFlow[4] and FlexRAN[5] protocols facilitate communication between the LL-MEC platform and the underlying RAN and CN. With LL-MEC, it is possible to develop RAN and CN coordinate network applications using LL-MEC and FlexRAN SDK that allow monitoring and controlling not only traffic but also the status of the network infrastructure. These applications can range from elastic applications to obtain statistics on user traffic for applications to low-latency applications that redirect user traffic (local breakout) by applying criteria for setting the data path. All RAN and CN product data and APIs are open for use by other apps and third parties.

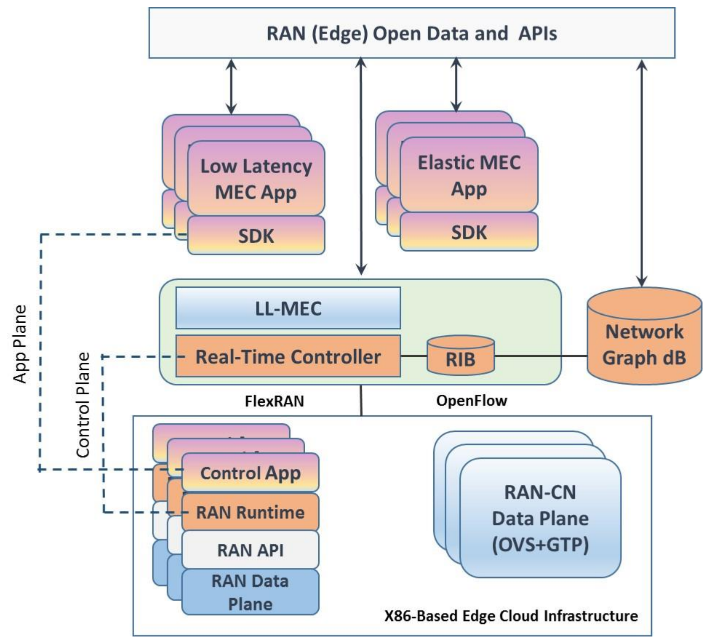

Figure 3. LL-MEC Platform [9]

To make the topic easier, Figure 4 shows the high-level diagram of the LL-MEC, mainly composed of a three-layer design:

- Abstraction Layer
- MEC Platform
- MEC Application

This platform runs on software-defined mobile network consisting of multiple LTE eNodeBs and SDN-enabled switches, whether it is physical or software, and fully separates the data plane from control functions. Furthermore, the agent acts as a local controller on behalf of RAN or SDN-enabled switches. The entities and interfaces implemented in this platform follow the ETSI MEC Specifications supporting the functionalities defined by Mp1 and Mp2 interface, keeping at the same time the 3GPP compatibility. Mp1 is the interface

[4] More details at: https://www.opennetworking.org/
[5] Please see: http://networks.inf.ed.ac.uk/flexran/

between mobile edge platform and applications while the Mp2 is the interface between mobile edge platform and the abstracted data plane as specified by ETSI.

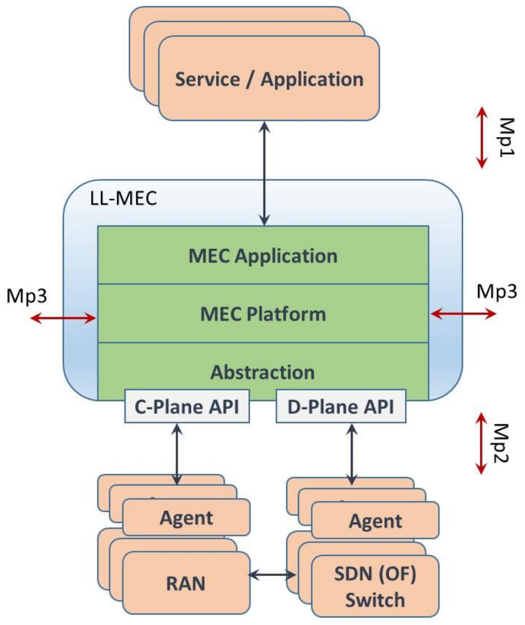

Figure 4. LLMEC High Level Schematic [9]

2.2.1.1. Abstraction Layer

The abstraction layer includes both the Radio-API entity and the Data-Plane-API entity and has the role of abstraction for the control-plane and for the data plane respectively providing only the information necessary for the development of the MEC Applications:

- The Radio-API has been designed to give an abstract view of the radio network status measuring the parameters of interest from the RAN. Moreover, provides the possibility to modify the state of the underlying network;
- The Data-Plane-API essentially provides the Mp2 interfaces for Edge Packet Services (EPS) within the MEC platform to control the data plane of the core network. EPS will pass the required rules to OpenFlow enabled switches through the Data-Plane-API.

2.2.1.2. MEC Platform

The MEC platform is in a Mobile Edge Host as a middleware (or core entity) between the MEC applications and the real network elements. It gives application developers the possibility to focus on the specific application rather than on the functionalities of the underlying RAN. The MEC platform is the brain of the LL-MEC: it controls the main services as events trigger and register, providing library integration and low latency support. Moreover, the MEC platform provides the necessary building blocks to realize MEC applications. It has to be noted that the current implementation of the LL-MEC does not support the Mp3 reference point used for the communication with the other MEC platforms.

The MEC platform is composed of the following components:

1) Radio Network Information Service
2) Service Registry
3) Edge Packet Service
4) Event Manager

2.2.1.3 MEC Application

One of the main benefits resulting from the separation of the control plane and data plane is that the applications on the top of the platform can be developed without deep knowledge of the underlying network.

Applications communicate with the MEC Platform through the Mp1 interface (northbound interfaces - MEC APP API); by means of the Mp1 reference point the MEC applications have access to the network information or delegate the control decision towards network. The Mp1 includes REST-API, messages bus, and local API. Another key feature of LL-MEC is that the application can be implemented in different scheduling ways such as round robin, first-in-first-out or deadline scheduler to have different time scales and priorities when running the tasks. In particular, the RAN-related applications can benefit from this feature to avoid further delays during interaction with the radio network.
Applications can not only interact with the MEC platform through the APIs to use and provide mobile edge services, but they can also provide services that provide information and messages useful for other applications.

2.3. 5GCity Approach

ETSI NFV and MEC share the same principles and can be combined in a single infrastructure. However, ETSI MEC is a young standardisation activity, partially still under definition. For this reason, some components of the 5GCity architecture need to be adapted and/or extended to support both ETSI NFV specific components (e.g., the NFVO) and MEC descriptors and deltas. Similarly, the following new MEC specific components need to be implemented:

- The Multi-access Edge Application Orchestrator (MEAO)
- The Multi-access Edge Platform Manager – NFV (MEPM-V),
- The Multi-access Edge Platform (ME platform).

This section presents the 5GCity approach towards the integration between ETSI MEC and NFV together with fog05, the open source project which will be extended to implement such integration.

2.3.1 fog05

fog05 is an IaaS software that can harvest compute power from low end devices as well as mobile devices and expose this computing power to the city infrastructure. fog05 leverages on a communication protocol that can work with poor network connectivity (intermittent or low bandwidth) and has a fully distributed control plane that allow runtime discovery of new compute nodes.

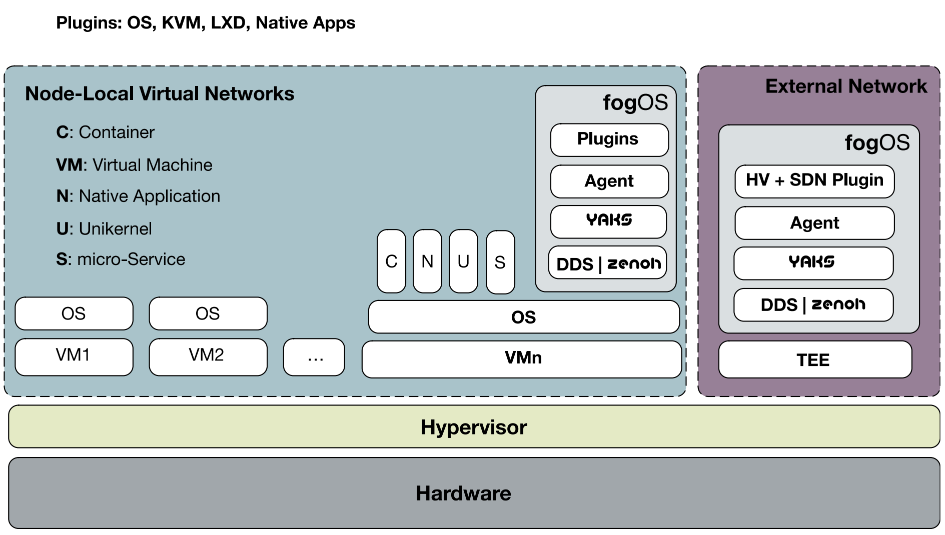

Figure 5 fog05 high level architecture

fog05 has a full plugin architecture (Figure 5), that ease the support of new hypervisors, SDN controllers, deployable units or MANO algorithms. This functionality will be used to enable to the ETSI MEC support in 5GCity.

A more detailed description of fog05 can be found in D2.2 Section 3.7.3.

2.3.2 Applicability of ETSI MEC architecture in 5GCity

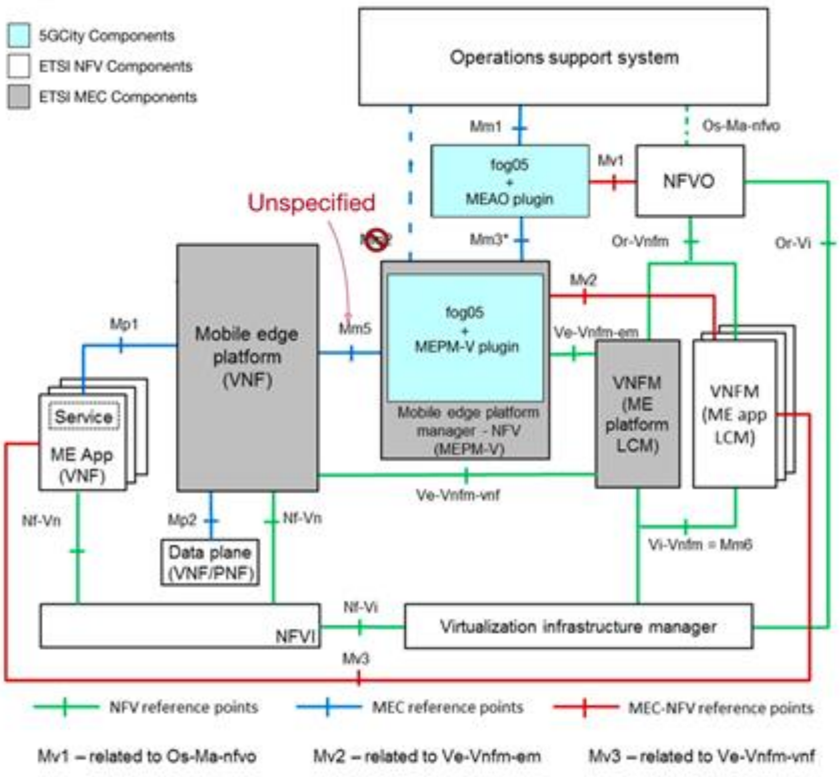

Figure 6. 5GCity MEC Mapping

Figure 6depicts the updated architecture for MEC in NFV that will be used as a base for the integration between MEC and NFV in 5GCity. Some reference points have been removed (the one highlighted using dashed lines and as well as federation reference points) while the MEAO and MEMP-V will use fog05 extended with specifically implemented plugins for both MEMP-V and MEAO as described in (D4.1 – Section 2.1 [10]).

Moreover, in the 5GCity architecture the **NFVO** will act as a master with respect to onboarding packages, while the final decision regarding placement will be always taken by the 5G service placement algorithms. This means that the MEAO and the NFVO should collaborate when it is time to instantiate or migrate an ME application/service, while this collaboration should go through another interface which is not the one used to send management information.

Another key component of the MEC architecture is the **MEPM-V,** which is responsible for Life Cycle Management (LCM) and Performance Monitoring (PM) for the Mobile Edge Platform. This component will act as an Element Manager from an NFV point of view, and will interact with the MEAO through the Mm3* reference point to retrieve configuration for the ME platform coming from MEAO or to notify something to it. This reference point is different from ETSI MEC Mm3 because in our case LCM will go directly through the VNFM. Interface Mm5 (not yet specified in ETSI MEC) will be used for sending configurations (DNS rules, configuration of persistence storage and so on) to the ME platform as well as receiving notifications or request from this one, regarding cardinality one MEAO can manage different MEPM-V, but one MEPM-V can be managed only by one MEAO.
Then we have the **Mobile Edge platform** that allows ME app to register and access the different platform services, as well as managing DNS rules and other useful information, such as RNIS. In 5GCity, an ME platform will be developed as PoC to demonstrate in some use cases how MEC applications can communicate. Regarding the management of traffic redirection, it will be triggered by the MEAO under request of the MEMP-V, and will be put in place by the NFVO mainly for three reasons:

1) The MEAO does not have any direct connection with NFVI and Data plane.
2) It makes more sense that all the actual configuration is done by only one orchestrator, the NFVO.
3) It gives us the possibility to remove the Mp2 interface between ME platform and the Data Plane.

For the mapping of a ME Host in an NFV concept, we will use NFVI-PoP and zones. As a consequence, we consider a single ME platform and a MEMP-V for each zone, solving the problem of allocation. The problem of reallocation (e.g. migrating a ME app between different zones) is a gap in ETSI NFV that still needs to be solved.

For each issue identified by ETSI MEC (**Table 1**), we identified the 5GCity solution in **Table 4**:

Issue	Description	5GCity Solution	Comment
ISSUE#1	Mapping between ME app VNFs and NS	The MEAO will have a map between ME app VNFs inside NSs	The map comes when the MEAO takes the descriptors form NFVO
ISSUE#2	NSD should express eventual dependency to other NSs.	5GCity information model will address this issue.	Extending NSD with MEC relevant fields.
ISSUE#3	Communication between MEAO and NFVO.	Communication goes through Mv1 ~ Os-Ma-nfvo, OSM REST API	
ISSUE#4	Communication between MEMP-V and VNFM.	Communication goes through Mv2 ~ Ve-Vnfm-em. MEPM-V act as Element Manager for the Mobile Edge Platform need to keep track of LCM operation	Ve-Vnfm-em is not exposed in OSM, we have to implement this interface for the MEMP-V.

		initiated by the NFVO, it also needs to access to PM counters for the virtualized resources in which ME apps VNFs related to the ME platform that is managed by the MEPM-V. PM information uses OSM MON tool.	
ISSUE#5	Communication between VNFM and ME App VNFs.	As Mv3 ~ Ve-vnfm-vnf no changes are needed.	MEC doesn't cover this part, we can use the NFV approach.
ISSUE#6	MEC AppD vs NFV VFND.	5GCity information model will take in account both descriptors. MEAO stores only MEC information. NFVO stores only NFV information.	Need to extend out Information Model to cover relevant MEC fields.
ISSUE#7	Packages of ME Apps vs VNFs.	5GCity packages will contains files related to NFV and MEC.	MEAO will store only MEC part, and NFVO only NFV part.
ISSUE#8	NS/ME app onboarding.	NFVO is the master, and MEAO is the slave, this means that the onboarding comes first to NFVO that validate eventual MEC information, store the mapping between ME app and NS, and send MEC information to MEAO and NFV information will be used by the NFVO.	
ISSUE#9	Management of traffic redirection.	The ME platform ask traffic redirection through Mm5(which is an unspecified reference point) then this information goes to MEAO through Mm3* and the MEAO create a NFP based on the new traffic rules and uses Mv1 to ask the NFVO to instantiate.	The MEAO is the trigger for traffic redirection, then the actual configuration is done by NFVO for the NFV part and by the ME platform for the MEC related part.
ISSUE#10	Comparison between AppD and VNFD data structures	See issue 6.	
ISSUE#11	Multi-access Edge Host in NFV.	ME Host can be the NFVI in a cabinet. MEAO need to be able to ask NFVO to deploy in specific cabinets. Each NFVI-PoP can be a ME Host.	MEC should be able to reuse such as NFVI-PoP (basically, a data centre) and Zone (a set of co-located and well-connected physical resources which is a subset of an NFVI-PoP). This can be mapped to well known definition of availability-zone in an OpenStack deployment.
ISSUE#12	ME App VNF Instance Relocation	The MEAO and NFVO should collaborate when is time to relocate a ME App instance, this communication goes through a reference point separate from Mv1	Relocation is triggered by MEAO based on information coming from MEPM-V
ISSUE#13	Application instantiation	Same as issue#12	Instantiation is triggered by MEAO

ISSUE#14	Application instance termination	Same as issue#12	Termination in triggered by MEAO based on information coming from MEPM-V

Table 4. 5GCity approach – issues and solutions

3 Computing Virtualization

Computing virtualization consists in abstracting the system hardware resources to run multiple independent instances of an application. Different techniques exist today to implement such abstraction: Hypervisors, Containers and Unikernels are the most important ones (Figure 7).

In this section, the 5GCity enhancements to computing virtualization techniques are detailed after a presentation of these techniques as state of the art.

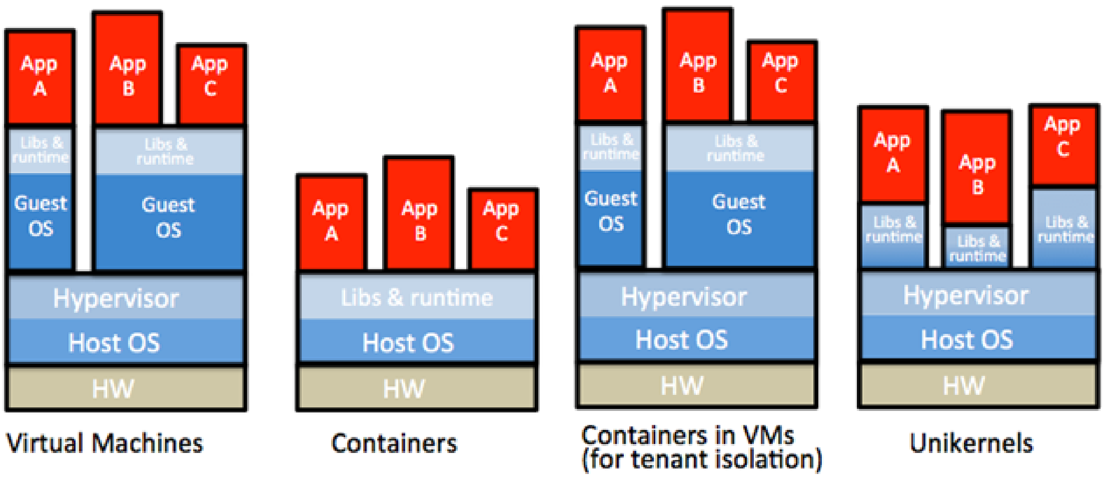

Figure 7. Virtualization solutions architectures [11]

A hypervisor (e.g., Kernel-based Virtual Machine – KVM [12] and XEN [13]) is a software layer able to create virtual instances of hardware resources such as CPUs, memory, devices, etc. It enables the execution of multiple operating systems (virtual machines) on the same hardware. The virtualisation provided by type-1 (e.g., Xen) hypervisors is implemented at the **operating system level**, meaning that two virtual machines (VMs) running on the same host share the hypervisor but do no not share the OS implementation, nor the libraries, the runtime and any other higher-level component.

Containers (e.g., Docker[6]), on the other hand, use operating systems features to package applications together with all their dependencies (libraries, binaries, etc.). In the case of Linux, control groups (cgroups) and namespaces are leveraged to provide, respectively, resource management and isolation between the container instances. Therefore, virtualisation is implemented in this case at the **libraries and runtime level**. These shared components are read only, while each container has its own specific access point for writing them. Containers can be instantiated faster than virtual machines, but do not have the same level of isolation provided by the hypervisor. For this reason, to enable multi-tenancy, containers are usually isolated inside virtual machines, as shown in Figure 7.

Finally, unikernels (e.g., Unikraft, RumpRun, etc.) are specialised, single-address-space virtual machine images virtualized at the **library operating system level**, in the sense that the operating system in this case is seen as a library from which the application can select only the needed components in a modular way. They shrink the attack surface and resource footprint of cloud services. Unikernels share the isolation properties of the hypervisors and the very low instantiation time of containers because of their smaller image/footprint.

[6] Please see: https://www.docker.com/

3.1 State of the Art

3.1.1 KVM

KVM is an open source hypervisor included in the Linux kernel, available for different CPU architectures (e.g., x86, ARM, s390, etc.) and implemented as a kernel module which is accessed by a standard IOCTL (input-output control) interface. It exploits CPU Virtualization Extensions to execute guest's instructions directly on the host processor(s) and to provide VMs with an execution environment almost identical to the real hardware.

KVM borrows directly from the Linux kernel functions such as memory management and CPU scheduling. As a consequence, the hypervisor codebase is light and simple compared with other solutions. Additionally, it relies on external user space components to execute virtual machines. In fact, KVM doesn't offer itself machine or device models abstractions (bios, devices, etc.), but uses Quick Emulator (QEMU[7]) for emulating guest hardware devices and instantiating guests. For example, QEMU is able to emulate a specific network interface card (E1000 Intel NIC, etc.), as well as a specific machine model (ARMv7 A15, x86 with q35 chip, etc.). Other external components, such as the C library for Virtualization libvirt[8], are used to remotely manage the hypervisor and to connect it to Virtualized Infrastructure Managers like OpenStack.

In the KVM paradigm guests are seen by the host as normal POSIX (Portable Operating System Interface for Unix) processes, with QEMU residing in the host userspace and utilizing KVM to take advantage of the hardware virtualization extensions. QEMU and KVM are able to run unmodified guests using emulation. However, since emulation is reputed to add significant overhead, KVM also supports Input Output (IO) para-virtualization through Virtio, a standard abstraction solution for different hypervisors IO drivers. Virtio driver/device for KVM are today available for different IO types: e.g., network, disk (block), random number generator and balloon (for memory over commitment).

3.1.2 VOSYSmonitor

An important challenge for virtualization today is to address consolidation while keeping separated different levels of criticality on a common hardware platform. For example, an operating system providing security, critical or real-time services can be executed together with other virtualized OSes which provide streaming, gaming and social network services. This is very important in use cases like those of smart cities, Internet of Things (IoT) and automotive, where the virtualized applications interaction with Cyber Physical Systems (CPS) needs to address specific certification, real-time or security requirements.

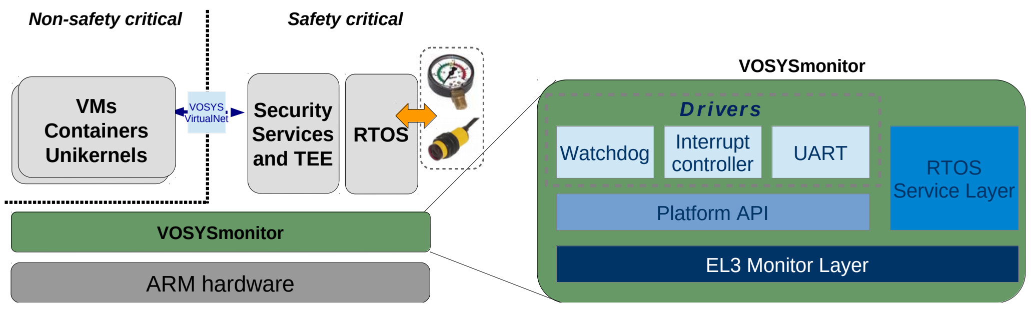

Figure 8. VOSYSmonitor architecture

VOSYSmonitor, presented in Figure 8, is the Virtual Open Systems proprietary solution that addresses this challenge on ARMv8 processors, the low power architecture widely used in mobile, embedded and edge systems, and today also in the server side by System on Chip (SoC) makers such as Qualcomm [14] and Cavium. VOSYSmonitor enables the native concurrent execution of a safety/security critical OS (e.g., a Trusted

[7] Please see; https://www.qemu.org/
[8] Details at: https://libvirt.org/

Execution environment or a Real Time Operating System - RTOS) along with another operating system which supports the execution on hypervisors, containers and unikernels. Communication between the two OSes is implemented through a virtual Ethernet connection, VOSYS VirtualNet.

VOSYSmonitor runs in the monitor layer, the most secure operating mode available on ARM processors with the TrustZone [15] hardware security extension, which manages the interaction between two execution worlds and guarantees peripherals and memory isolation between critical and non-critical OSes. It is built with low level programming languages to provide highest performance and programmability. The most important component of the architecture is the Exception Level 3 (EL3) Monitor layer, which handles exceptions and context switching operations between the safety critical and non-safety critical environments. Moreover, drivers to access low level peripherals are implemented on top of the Platform API, which abstract driver function calls for the Monitor Layer. Finally, a specific programmable interface (Service Layer), is needed to dispatch secure services and handle interrupts in the real time safety critical environment.

3.1.3 Unikraft

In recent years, several papers and projects dedicated to specialized OSes, network stacks, protocols and unikernels have shown the immense potential for performance gains that these have. For instance, in [16] the authors specialize the network stack and use domain-knowledge to pre-prepare packets in order to significantly speed up video delivery. Likewise, by leveraging specialization and the use of minimalistic OSes, unikernels are able to yield impressive numbers. MirageOS [17] uses Mini-OS [18], a minimalistic operating system that is part of the Xen ecosystem, to build OCaml-based, tiny unikernels able to boot in tens of milliseconds and thus provide just-in-time virtualized services, instantiated as the first packet in a flow arrives at a host. Erlang on Xen [19] provides a small memory footprint Erlang unikernel that can execute mixed workloads. Further, ClickOS [20] also uses Mini-OS as its basis by building a unikernel that includes the Click Modular Router software [21], along with multiple optimizations to the network sub-system, to build a NFV unikernel able to perform at high rates of 10+Gb/s. Along those lines, Minicache [22] provides a CDN cache node that can service high-definition video content at rates of up to 40Gb/s while assigning a single CPU core to the unikernel. Multiple other unikernel or virtualized operating systems exist that can be used as basis for building specialized OSes: OSv, Solo5 and IncludeOS [23], [24], [25] are but a few examples. In terms of memory footprints, it is not uncommon for such unikernels to require as little as hundreds of KBs or a few MBs to run web servers or other functionality [17], [26].

The fundamental drawback behind specialization, and unikernels in particular, is that they require that applications be manually ported to the underlying minimalistic OS (e.g., having to port nginx, Python, mysql or memcached to MiniOS or OSv); worse, once this process is done, optimization and tweaking the resulting image is time-consuming, manual work, where an expert developer has to perform multiple cycles consisting of measurement, programming improvements, rebuilding and measuring again.

By providing an automated tool for unikernel creation, Unikraft, an open source project under the auspices of the Xen Project and the Linux Foundation, seeks to drastically reduce the amount of expert time needed to implement specialized images. In addition, the fact that such build process will be automated will allow Unikraft to fully automate the optimization/tweaking cycle mentioned above, once again removing the time-consuming and expensive effort from an expert from the equation. Finally, Unikraft will take specialization to the extreme, by allowing users to easily choose which features from all layers of the software stack, including the operating system, they would like to have in support of their application.

3.1.4 Virtualization security and trust

Security and trust are particularly important in smart cities environments because of their distributed architecture and for the importance of the data they use. In fact, citizen’s data (coming from cameras, mobility services, health, etc.) need to be well protected to avoid data leakages that can be sold or used for retaliations by attackers. Moreover, in an architecture where Edge devices are scattered throughout the city and possibly connected through wireless technologies like 5G, the risk of man in the middle, device identity stolen and fake requests is important. To make things worse, edge devices are often in a position difficult to secure from being stolen/tampered, and it might be easy for an attacker to replace/add/tamper devices making them not trustworthy. 5GCity will provide a virtualization-based security and trust infrastructure that can be used by developers to enhance security, authenticate devices and secure citizens data.

These points are of course well known to ETSI, which within the GS NFV-SEC 003 document (also known as NFV Security and Trust Guidance), defines Trust as "confidence in the integrity of an entity for reliance on that entity to fulfil specific responsibilities". To achieve Trust, NFV-SEC 003 identifies attribution, attestation, non-repudiation and identity as elements that have to be combined together [27].

The concept of Trust in NFV needs to be implemented at all layers, starting from the hardware up to the higher levels of the software architecture. In the next sections, Trust challenges for NFVI, VNFs and VIM are detailed while the plans for development in 5GCity are detailed in section 3.2.2.

3.1.4.1 NFVI Trust

At the bottom layer of the NFV architecture, the NFVI component runs directly on the hardware and creates the abstraction that enables portability for the VNFs. The key technology to know the state of the platform and to implement Trust in hardware is the Trusted Platform Module (TPM), a device standardized by the Trusted Computing Group [28]. TPMs implement cryptographic functions needed to enforce specific behaviours and protect the system against unauthorized changes and attacks. TPMs are used for secure storage, disk encryption as well as platform integrity verification. Another technology in the same direction is the Trusted Execution Environment (TEE) by standardized by GlobalPlatrofm [29]. The TEE is a secure area of the main processor which provides an isolated and trusted environment. Main implementations existing today include:

- **Intel TXT** is a hardware technology from Intel which aims to provide root of trust and verify the integrity of platform by relying on TPM [30]. During the boot time measurement, the cryptographic hash of platform components (such as BIOS, OS, hypervisor) are calculated and are verified against known good measurement values.
- **Intel SGX** or Intel Software Guard Extensions is an extension to Intel processors' architecture which enables the use of protected areas of execution in memory, called enclaves. It makes an application code executing within the enclave protected even when the BIOS or operating system are compromised [31]**.**
- **Arm TrustZone** technology implements TEE as a system-wide approach to security. TrustZone is hardware-based security built into SoCs by semiconductor chip designers who want to provide secure end points and a device root of trust [32].

These hardware technologies need to be made available to hypervisors and exposed to the VIM to extend the concept of Trust also to the higher levels of the NFV architecture. Extensions to the NFVI system (operating systems, boot procedures, hypervisor, driver libraries, agents) needs to be developed today to address this challenge.

3.1.4.2 VNF Trust

Trust-enabled NFVI systems can use trust to verify the integrity and reliance of the VNFs that are running on top of it [33]. In fact, without a proper integrity verification procedure, a corrupted VNF image can be selected by the hypervisor and threatening the entire system (included the other VNFs). OpenStack has had an image signing feature since its release Mitaka, but it does not consider scenarios where the image server is compromised.

To establish trust in VNF one should address the challenges of encrypting or signing a VNF, verifying the integrity of a VNF, preventing insider attacks. This can be done only by relying on a Trust-enabled NFVI and extending the components that take part to the VNF launch procedure.

Additionally, Trust capabilities exposed by the NFVI can be also used inside the guests, to enhance the safety and the security of the VNFs. These challenges can be met by the virtualization of the Trusted Platform Module. Including a virtual TPM as a part of the NFVI will allow the VNFs to benefit from the hardware TPM's secure storage and cryptographic functions. Each VNF should be provided with a dedicated TMP functionality, each emulating the functions of a hardware TMP [34].

Virtualizing the TPM is a complicated task which is addressed by the Trusted Computing Group with the foundation of the Virtualized Platform Work Group and the release of the Virtualized Trusted Platform Architecture Specification [35].

3.1.4.3 VIM Trust

One of the challenges in resource management is the task of identifying a host to launch the VNFs. The host selection process is performed based on resource criteria (RAM, vCPU, NICs, etc) while launching an instance. In a trusted environment the criteria is extended with the additional requirement that the host is functional and it is trusted. Resource selection can also be performed by specifying the location details or the geographic boundaries where the VNFs must be launched. The geo-location trust can be verified with the help of a component in the TPM, which stores the geo-tagging index.

The hardware-based security, combined with an external stand-alone, web-based remote attestation server, ensures that the compute node runs only software with verified measurements and a secure cloud stack.

3.1.4.4 OpenStack Trusted Compute Pools feature

With the goal of meeting some of the VIM Trust challenges, OpenStack provides a Trusted Compute Pools feature [36] which today relies only on the Intel TXT technology. There is not work done yet incorporating different TPM implementations, like ARM TrustZone.

The trusted pools consist of compute nodes with Intel TXT technology enabled, verified by a remote attestation server. The nova-scheduler component of OpenStack Compute queries the attestation server to collect the list of trusted hosts and places workloads and VMs into the trusted servers (Figure 9).

The verification steps performed by the attestation server are described as [36]:

- Compute nodes boot with Intel TXT technology enabled.
- The compute node BIOS, hypervisor, and operating system are measured.
- When the attestation server challenges the compute node, the measured data is sent to the attestation server.
- The attestation server verifies the measurements against a known good database to determine node trustworthiness.

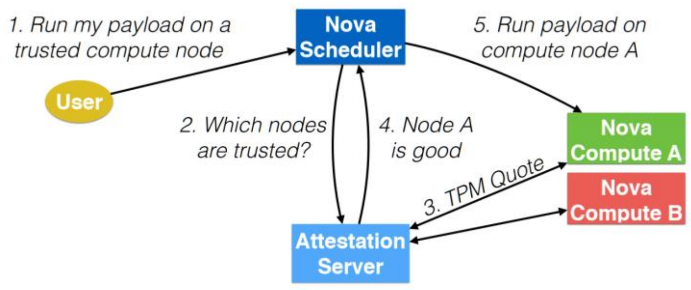
1. Run my payload on a trusted compute node
User
Nova Scheduler
5. Run payload on compute node A
2. Which nodes are trusted?
4. Node A is good
Nova Compute A
3. TPM Quote
Nova Compute B
Attestation Server

Figure 9. OpenStack attestation with Trusted Compute Pools [37]

With Intel TXT enabled in the compute nodes, the measured data is sent to the attestation server in the form of a TCG-standard TPM Quote, a signed report of the current Platform Configuration Registers (PCRs) values [38]. A detailed representation of this process is shown in Figure 10.

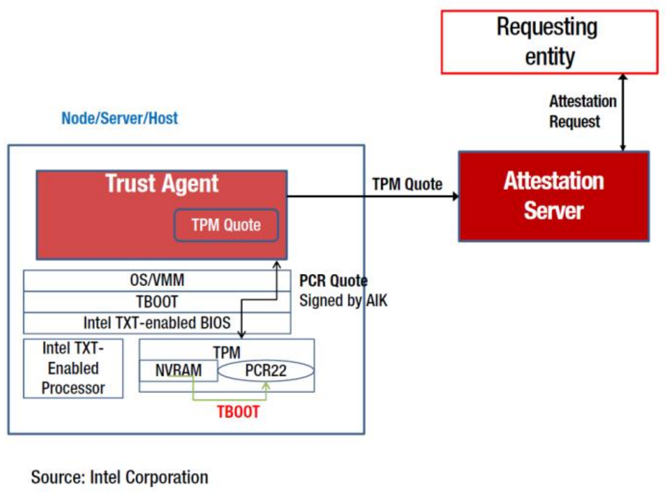

Figure 10. TMP Quote collection **[39]**

Once received by the attestation sever the TPM Quote is verified against stored whitelist values. OpenStack does not provide a description of setting-up the attestation service but provide a reference to the OpenAttestation project [40] . It is an Intel-maintained open-source project that is a software development kit (SDK) based on a server-client architecture for managing host integrity verification [39]. The client is highly dependent on the HW and has the following requirements: *"Client system must have TPM 1.2 compliant device with driver installed, and TPM/TXT enabled in BIOS".* The SDK architecture is shown in Figure 11 and some of its main features include:

- Support of major Linux-hosted operating systems and the associated hypervisors (Xen, KVM)
- Java-based privacy certificate authority
- Java-based host agent that accesses the platform TPM through the open source TSS (TrouSerS) trusted computing software stack
- RESTful-based simple Query API
- Basic whitelist service and API, with whitelist management capabilities

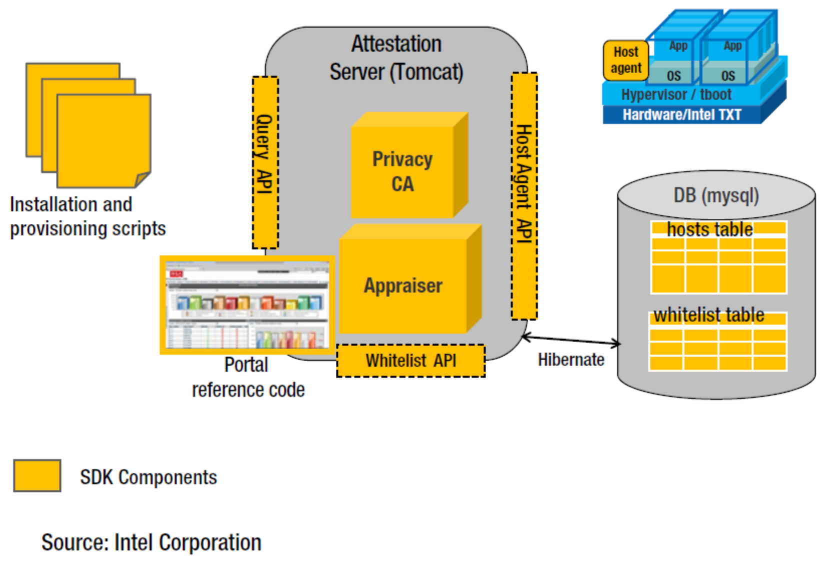

Figure 11. OpenAttestation SDK architecture overview **[39]**

3.2 5GCity Enhancements in computing virtualization

3.2.1 Unikraft Context and Enhancements

Quickly developing, upgrading and deploying applications is the core function of the IT industry: online content providers, network operators, CDNs, business-to-business providers, and even Internet of Things providers need to be able to quickly rollout software releases to enhance their product offerings while reducing developer time and increasing customer satisfaction. Such software is typically deployed and runs on shared hardware hosted either in public (e.g. EC2), private or mobile-edge clouds or other federated infrastructures (such as CDNs or customer premises equipment). The huge success of public clouds and the ongoing mobile-edge cloud deployments is testament to the huge benefits of sharing hardware among different entities which lowers costs and provides the ability to scale resources on demand.

Running software on shared hardware massively boosts efficiency but also reduces isolation. When public clouds appeared, the standard unit of deployment was the virtual machine; indeed, running VMs on the same machine reduces isolation a bit (e.g. various forms of covert channels are possible), but the risk was deemed acceptable and this lead to a huge uptake of cloud computing. Traditional virtual machines, however, are heavyweight as they require a full operating system image to run; this implies that running many of them on the same hardware requires a lot of RAM and CPU cycles and can reduce performance: memory and disk space is wasted and starting / stopping VMs takes tens of seconds in the best case, and often much longer.

For these reasons, the software industry has embraced containers as a replacement to VMs for a wide variety of applications, with the goal of further improving performance of shared hardware, reducing dev-ops costs and speeding-up software deployment. Containers are as cheap as traditional operating system (OS) processes which means that starting, stopping or migrating them can be done in well under one second; they share the OS kernel thus reducing the memory wasted by duplicating OS functionality across VMs. Finally, tool stacks such as Docker allow to easily create containers starting from existing ones or predefined templates.

Despite their efficiency, containers offer poor isolation as shown by their many vulnerabilities. Additionally, the Meltdown[9] attack permitted any container to read the memory of any other container on the same machine, thus evading isolation altogether. Meltdown has been solved in the meantime, at the cost of a severe loss in performance.

At this point, the software world appears stuck with inherently insecure and not-so-efficient containers, because virtual machines are deemed too expensive to use in many scenarios. This is especially troublesome in scenarios where critical infrastructure, as is the case in municipalities in general and for 5GCity in particular, is in play: such infrastructure needs to be shared but needs to be shared with strong isolation and security but also efficiency.

Within this context, Unikraft seeks to solve this problem by enabling smart city software developers to easily build and quickly deploy lightweight virtual machines starting from existing applications. Unikraft is targeting the development of tools that will enable lightweight VM development to be as easy as compiling an app for an existing OS, enabling EU players, and smart cities in particular, to lead the next generation of cloud computing services and technology.

Lightweight virtual machines are VMs that include only the minimum functionality to achieve the task of the VM, and so are ideal for smart city deployments where resource-constrained devices are often the norm. As most VMs run a single (or a small number of) app such as a web or database server, by creating a VM that includes only the minimal amount of software needed to make the target application run, we can reduce the

[9] Plaese see: https://meltdownattack.com/

memory footprint of the VM and its boot time by orders of magnitude; e.g. the image of a lightweight VM containing a python interpreter can be as small as 4MB, which is to be contrasted with, for instance, a standard Ubuntu VM that is typically around 1GB in size (which again, would be too large to run in many, if not most, smart city deployment sites such as lamp posts or street cabinets).

Unikernels are the smallest lightweight VMs one can create: they are VMs where there is no traditional operating system running underneath the application; instead, the application is compiled against bits of OS functionality that it needs, resulting in a very small app+OS bundle. Many unikernels have been developed already such as ClickOS, MiniCache, Mirage, Minipython, Solo5, OSv, Erlang on Xen, HalVM; they all offer great performance and low memory footprint for their chosen task. For instance, LightVM [26] has shown that one can run 8000 unikernels on the same hardware (more than containers), while still achieving very good performance.

Despite their advantages, developing applications with unikernels is a manual process today requiring significant expert resources, which prevents them from being widely used by the software industry.

Within the 5GCity project, Unikraft aims to enable standard developers and dev-ops engineers to create, maintain and deploy smart city-focused unikernels with ease. It will achieve this goal by developing an open-source toolchain that will enable secure and portable unikernel development. Developing unikernel based applications will be reduced to slight changes in the app Makefile, choosing from a menu of available implementations for the required system functionality, and compiling the app.

Unikraft decomposes operating systems into elementary pieces called libraries (e.g., schedulers, memory allocators, drivers, filesystems, network stacks, etc.) that users can then pick and choose from, using a menu, to quickly build images tailored to the needs of specific applications. In greater detail, Unikraft consists of two basic components (see Figure 12 below):

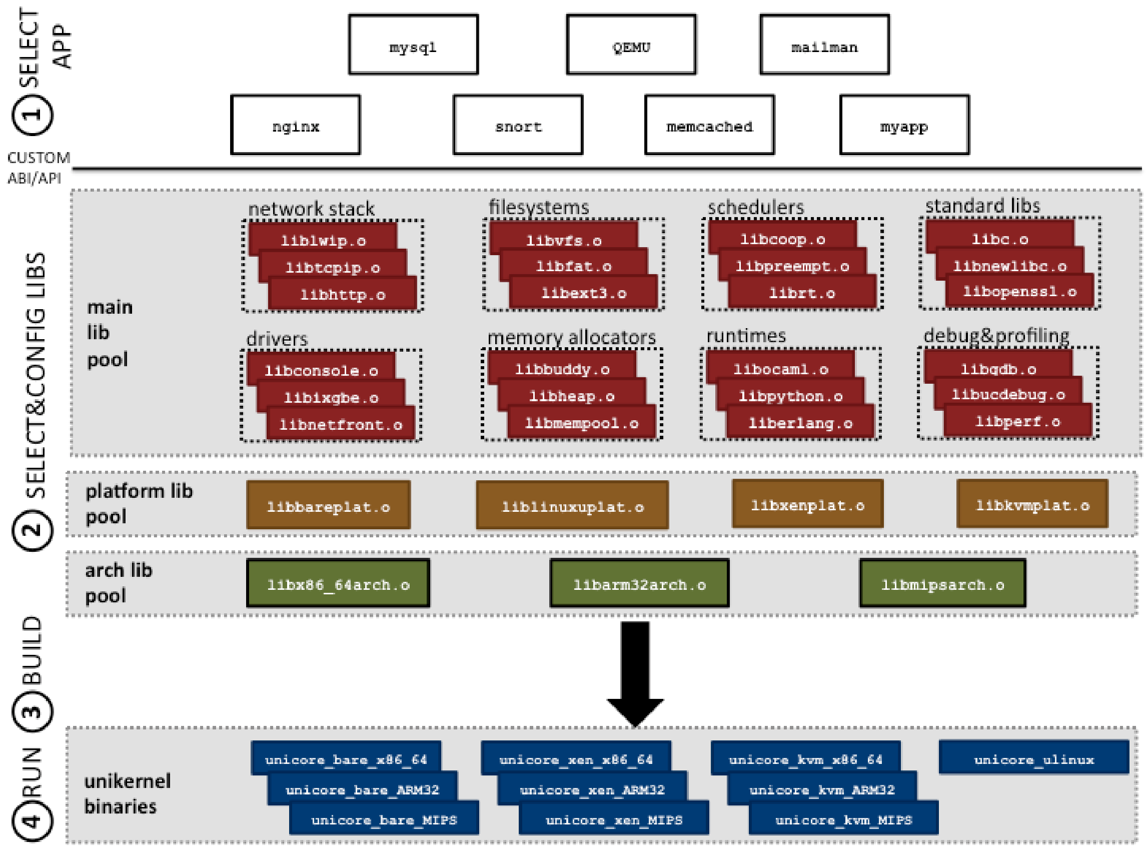

Figure 12. Unikraft components

Library pools contain libraries that the user of Unikraft can select from to create the unikernel. From the bottom up, library pools are organized into (1) the architecture library tool, containing libraries specific to a computer architecture (e.g., x86_64, ARM32 or MIPS); (2) the platform tool, where target platforms can be Xen, KVM, bare metal (i.e. no virtualization), user-space Linux and potentially even containers; and (3) the main library pool, containing a rich set of functionalities to build the unikernel from. This last library includes drivers (both virtual such as netback/netfront and physical such as ixgbe), filesystems, memory allocators, schedulers, network stacks, standard libs (e.g. libc, openssl, etc.), runtimes (e.g. a Python interpreter and debugging and profiling tools. These pools of libraries constitute a code base for creating unikernels. As shown, a library can be relatively large (e.g libc) or quite small (a scheduler), which should allow for a fair amount of customization for the unikernel.

The Unikraft build tool is in charge of compiling the application and the selected libraries together to create a binary for a specific platform and architecture (e.g., Xen on x86_64). The tool is currently inspired by Linux's kconfig system and consists of a set of Makefiles. It allows users to select libraries, to configure them, and to warn them when library dependencies are not met. In addition, the tool can also simultaneously generate binaries for multiple platforms.

Beyond these important and basic capabilities, we will use Unikraft to implement the neutral host use case, in particular running various applications in lean, efficient unikernels at the edge of the smart city networks. In addition, we will be investigating the possibility of supporting the other 5GCity use cases [41] with Unikraft, for instance by creating a ML prediction unikernel to support the illegal waste dump detection use case.

3.2.2 NFVI and VIM Trusted computing extensions

In 5GCity trusted computing extensions for the NFVI and VIM components will be developed, enhancing several open source implementations of NFV components such as KVM, libvirt and OpenStack to build an infrastructure capable of addressing the challenges detailed in Section 3.1.4.

3.2.2.1 TPM Virtualization through VOSYSmonitor

The first basic technology that will be implemented in 5GCity is TPM virtualization for ARM devices, aiming to enable each VNF to have a virtual TPM (vTPM) instance to be used in the guest to secure applications processing, as shown in Figure 13. To do this, VOSYSmonitor, libvirt and KVM will be extended in a way that guests can access securely to their own TPM implementation.

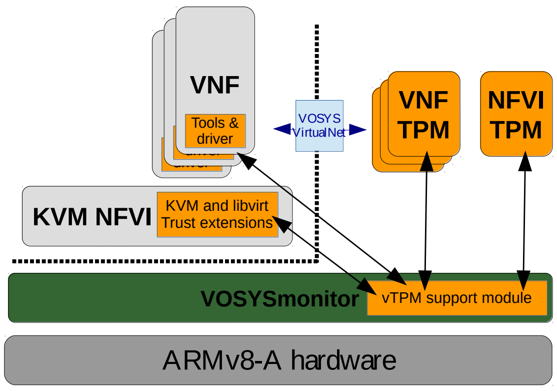

Figure 13. 5GCity vTPM extensions

At the lowest layer of the system architecture, VOSYSmonitor will be enhanced with a vTPM support module, which will support the concurrent execution of multiple TPM binaries in the ARM TrustZone Secure World. Each of them will serve a different virtual machine or VNF, with an additional TPM allocated exclusively to the NFVI. The OPTEE [42] open source TEE project will be used as a reference implementation.

As a consequence, both the NFVI and the guests will be extended to access TPM functions. In the case of the NFVI, VOSYS will focus on extending KVM and its OpenStack driver libvirt to expose TPM capabilities to the higher components of the NFV architecture and to enable both NFVI authentication (useful for geo-tagging) and VNF verification. In the guests, tools and drivers from the OPTEE project will be extended to support the 5GCity vTPM solution.

3.2.2.2 5GCity EdgeVIM: OpenStack Trusted Compute Pools for ARM

In 5GCity an EdgeVIM based on OpenStack with a Trusted Computing Pools feature and Arm support will be developed leveraging on VOSYSmonitor, libvirt, KVM extensions (explained in the previous section) and on the OpenStack Filter scheduler's TrustedFilter. One of the key objectives of this work is to solve the geo-fencing issues that the 5GCity municipalities might have.

To achieve the goal of porting Trusted Compute Pools on ARM there is a need to a) extend the existing attestation service and b) enhance the existing Nova scheduler to support Trusted ARM compute nodes.

For the attestation service, the current implementation is highly dependent on Intel TXT enabled compute nodes. In order add support for ARM compute nodes there is a need of an attestation server capable of verifying the integrity of ARM TrustZone enabled computes. To do this, both the creation of an ARM TEE attestation client and the possibility to develop a simplified attestation service (maybe integrated in the Nova Scheduler) will be explored.

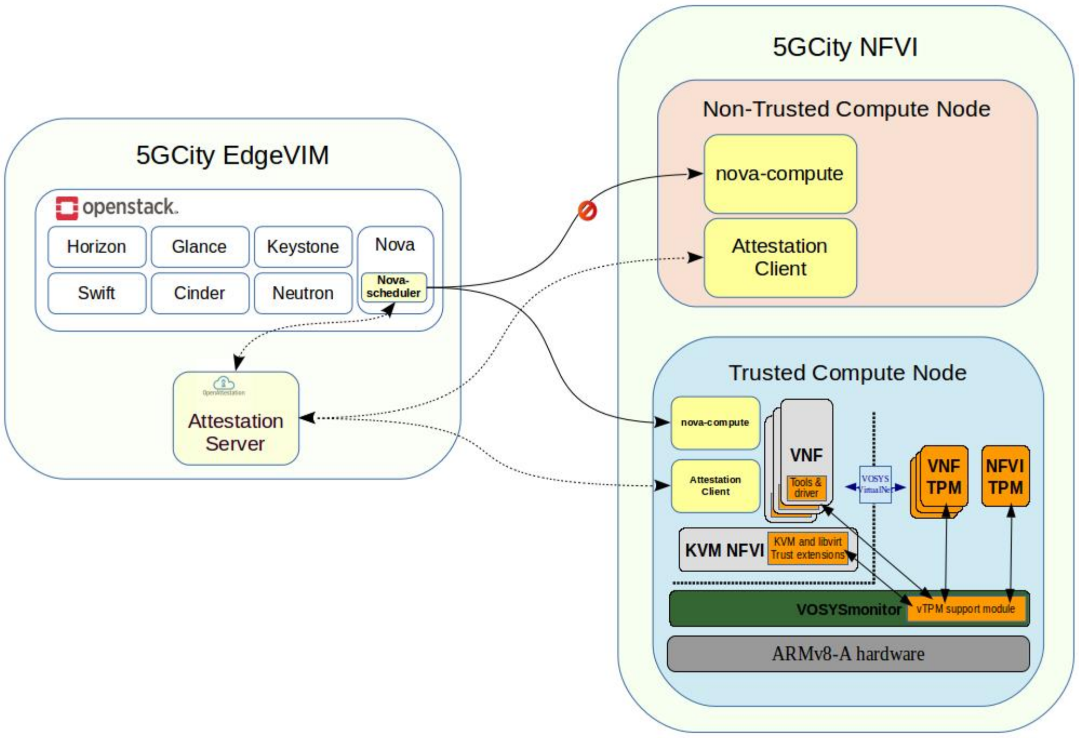

Figure 14. 5GCity EdgeVIM architecture

Regarding the OpenStack scheduling, currently the nova-scheduler service is configured by default as a filter scheduler. It supports a variety of compute filters as well as the addition of custom filters' implementations. VOSYS will leverage and extend the functionality provided by the TrustedFilter to filter hosts that do not meet the trust requirements. The request to run a VM on a trusted host will be specified as an additional property of the OpenStack flavor.

Figure 14 represents the overall EdgeVIM architecture and the interaction between the VIM and the 5GCity NFVI during the verification of a compute node as trusted or not.

4 Network Virtualization

Networking virtualization is the process of combining hardware and software network resources/functions into a single software-based administrative entity: the software defined virtual network.

In this section, the 5GCity networking virtualization enhancements are detailed after a presentation of the related state of the art. The state of the art description is limited only to technologies and components of interest for 5GCity (e.g., only OVS-DPDK and VOSYSwitch are presented as virtual switch solutions because they will be extended during the project).

4.1 State of the Art

4.1.1 OVS-DPDK

Open vSwitch (OvS) is an open source implementation of a virtual switch, used to connect virtual machines between them and with the external network. When taking decisions on how to forward data packets, native OvS uses kernel correlations of matches and actions to determine how to process a packet. The decisions are taken based on a flow table that is located in the kernel space. When a packet is received, it enters this kernel space and if there exists a rule that matches the packets, the corresponding actions are performed, e.g. sending the packet over a specific interface. This is the so called fast path, where a packet can quickly be processed by the kernel space process. If no rule matches the packet, the packet is handed over to the user space daemon, that contains all the programmed rules, and it is processed there. The check performed in the user space is slower, which is why this sort of packet switching is called slowpath. When a slowpath packet hits one of the switching rules, the daemon inserts this rule in kernel space table, so that further incoming packets with the same match can be processed via fastpath.

The performance of native OvS depends on the performance of the Linux network stack, which may reach is limits when dealing with heavy traffic, as it can be the case in 5GCity use cases, where heavy traffic loads from media use cases or telco traffic needs to be handled. A set of user space libraries allow to eliminate the bottleneck caused by the necessity of switching packets through the kernel space: the Data Plane Development Kit (DPDK[10]).

The DPDK libraries allow to bypass the kernel, connecting the user space daemon directly with the network interfaces. This shifts the fastpath from the kernel space directly to the user space. This incurs in a noticeable switching speed increase of up to around 15 times the speed achieved with native OvS. A major drawback of using DPDK is the additional software overhead: DPDK needs modified code in the applications to work, therefor requiring an additional effort when programming the software.

4.1.2 VOSYSwitch virtual switch

VOSYSwitch is a user-space, modular and NFV-ready virtual switch based on the open source Snabb [43] NFV framework (Figure 15). It is developed in Lua and provides better performance than OVS-DPDK [44] thanks to the acceleration provided by the LuaJIT[11] (Lua Just in Time compiler). In fact, by leveraging on LuaJIT, VOSYSwitch can benefit of a well-engineered trace based just in time (JIT) compiler [45], which relies on profiling execution information collected at runtime to detect and compile performance critical application fragments. Unlike most just in time compilers that operate at the method level, trace-based compilers delve in deeper into the control-flow of a method by profiling the execution of program paths. The ultimate goal is to capture the smallest set of execution traces that are representative of the dynamic behavior of the

[10] Please see: https://dpdk.org/
[11] Please see: http://luajit.org/

application. Doing so, a trace-based compiler can focus its entire optimization budget on a tiny, yet very important part of an application [46]. This concept, well known in compilers literature, is applied to networking Virtualisation by VOSYSwitch. As a result, the optimized machine code of this virtual switch is reflecting the actual network traffic which is passing through it.

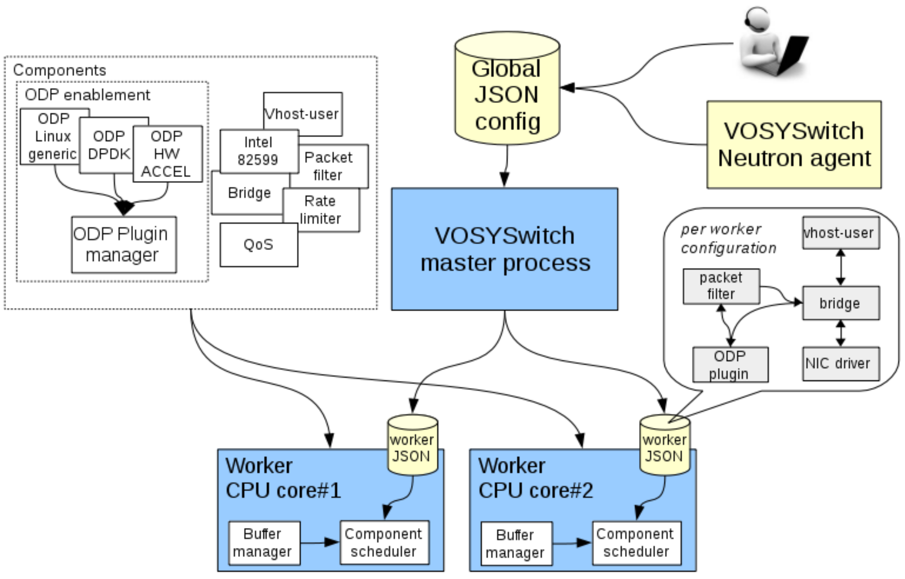

Figure 15. VOSYSwitch virtual switch architecture

VOSYSwitch can be executed on Intel and ARMv8 server architectures. It can be enriched with modules that implement specific functions (e.g., rate limiter, firewall, Open Data Plane, etc.) and it is configured through a JSON (JavaScript Object Notation) file which defines the switch components and their links in the form of network forwarding graph. The configuration file can be edited by the network administrator or by the OpenStack Neutron agent. Furthermore, the switch architecture implements a master-worker multiprocessing scenario where workers are configured and controlled via shared memory communication.

4.1.3 Wireless Virtualization

5G will integrate different types of radio technologies, such as evolution of LTE, the 5G New Radio (NR) and Wi-Fi based technologies. In 5GCity, the targeted dense edge deployments are composed of a potentially large number of wireless LTE and Wi-Fi links. Network slicing is one of the core mechanism of the design of such 5G networks, requiring the instantiation of multiple virtual networks over a single, shared physical infrastructure. In wireless network mediums, virtualization, which is required to enable network slicing, can be performed in different ways.

When speaking in general terms of wireless virtualization, techniques like time-hopping or TDMA enable slicing in the wireless medium. Such mechanisms require tight synchronization among wireless devices. In Wi-Fi this requires infrastructure mode, where a single access point is responsible for synchronizing the rest of the network. In 5GCity we are looking at other ways to implement network virtualization with the same goals: to provide network slicing and isolation to support the neutral host case.

A different and simpler way to virtualize wireless interfaces is to share a wireless interface among a set of tenants or series. For example, in LTE, for each tenant, a public land mobile network (PLMN) ID can be instantiated on the same carrier to differentiate between the tenants. In the following sections the approaches followed by the Wi-Fi and LTE-based solutions deployed in 5GCITY are discussed.

4.1.3.1 Wi-Fi Virtualization

In Wi-Fi, virtualization is implemented in user space by instantiating virtual wireless interfaces on top of physical interfaces that run on top of the mac80211 kernel module. Finally, the hardware drivers bridge the mac80211 kernel module with the physical NIC, as shown in Figure 16. Wi-Fi supports a series of different types of virtual interfaces (vifs): virtual access points, virtual mesh interfaces, etc. For the RAN, the most relevant of these options is the one that allows to instantiate virtual access points. In practice, each virtual access point has its own SSID that is announced with dedicated beacons, as a physical access point would do. This type of virtualization allows, for example, to instantiate dedicated SSIDs for particular tenants or services, along with specific settings for critical concepts as security (WPE / WPA /WPA2, etc.).

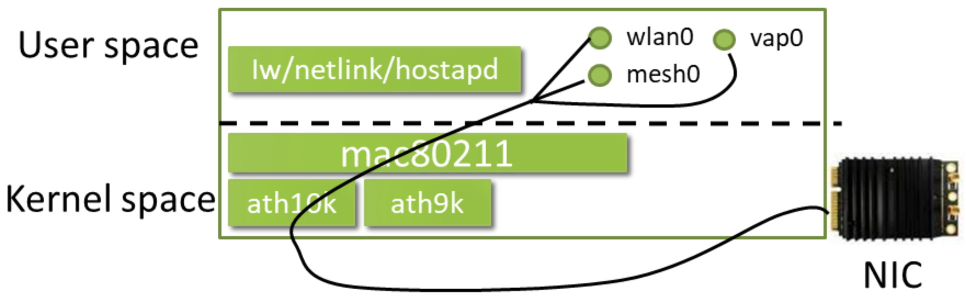

Figure 16. Physical wireless interface (NIC) virtualization in Linux

Once a physical interface has been virtualized with one or multiple virtual interfaces running on top of it, it is possible to use SDN software elements to generate network slices. The basic mechanism consists in adding the vifs to virtual software switches like OvS or VOSYSwitch, adding them effectively to the data plane of the SDN-based solution. An SDN controller then can handle and configure the configuration of these virtual switches in such a way that virtual access points belonging to a tenant or service can be added to a network slice, e.g. integrating the vif into a layer 2 subnet, connecting it to other elements of the network slices, e.g. wired backhaul, VNFs, etc.

4.1.3.2 LTE-based virtualization

RAN virtualisation and RAN slicing are subjects of intense interest in the scope of 5G research, however in terms of real-world deployment the subject is still in very early stages. Part of the reason for this is that 4G networks do not provide explicit support for slicing and most 4G technical solutions were not conceived to support virtual networking.

Much debate has occurred in vRAN about the possible functional partitioning of the LTE stack between datacentre, edge and radio units, with at least 8 possible partitions being defined by 3GPP (Figure 17).

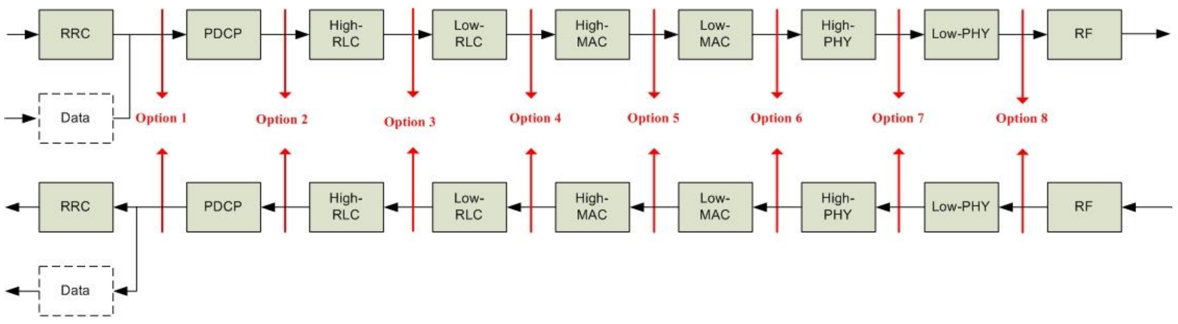

Figure 17. LTE stack partitioning options

The consensus emerging today is that a combination of Option 2 and Option 7 will be adopted generally – option 2 allowing for a high degree of centralisation of functions which are less time-sensitive, while option 7 allows for a less global degree of centralisation at the network edge where network timing and latencies can be more tightly controlled. 3GPP are in the process of standardizing an Option 2 split for 5G – known as the F1 interface – and the Xran organization recently published a proposed specification for an Option 7 fronthaul interface [47].

5GCity will deploy, in realistic city-based scenarios, implementations of virtualized RAN, Multi-access Edge Computing networking support and RAN slicing for Neutral Host based on existing 4G technology, with a clear upgrade path to 5G NR.

4.1.4 Service Function Chaining in an NFV enabled environment

The delivery of end-to-end services often requires various service functions. These include traditional network service functions such as firewalls and traditional IP Network Address Translators (NATs), as well as application-specific functions. The definition and instantiation of an ordered set of service functions and subsequent ’steering’ of traffic through them is termed Service Function Chaining (SFC).

This section describes an architecture used for the creation and ongoing maintenance of Service Function Chains (SFC) in a network. It includes architectural concepts, principles, and components, with a focus on those to be standardized in the IETF [48]. It also contains the description of SFC framework in an NFV enabled infrastructure [49] and the high-level description of the SFC implementation state of the art within the OpenStack [50], considered as the de-facto standard framework which implements VIM functionality Service function chains, enabling composite services that are constructed from one or more service functions.

An overview of the issues associated with the deployment of end-to-end service function chains, abstract sets of service functions and their ordering constraints that create a composite service and the subsequent "steering" of traffic flows through said service functions, is described in in IETF [48]. The current service function deployment models are relatively static, coupled to network topology and physical resources, greatly reducing or eliminating the ability of an operator to introduce new services or dynamically create service function chains. This architecture presents a model addressing the problematic aspects of existing service deployments, including SFC architecture composed by the following elements:

- **Network Service:** An offering provided by an operator that is delivered using one or more service functions. This may also be referred to as a composite service. The term "service" is used to denote a "network service" in the context of this document. For example, to some a service is an offering composed of several elements within the operator’s network, whereas for others a service, or more specifically a network service, is a discrete element such as a "firewall". Traditionally, such services (in the latter sense) host a set of service functions and have a network locator where the service is hosted.
- **Classification:** Locally instantiated matching of traffic flows against policy for subsequent application of the required set of network service functions. The policy may be customer/network/ service specific. This network Service definition clearly is totally coherent with the ETSI-NFV definition of Network Service. The relevance of SFC framework to ETSI NFV architecture will be discussed later in this section.
- **Classifier**: An element that performs Classification. Service Function Chain (SFC): A service function chain defines an ordered set of abstract service functions (SFs) and ordering constraints that must be applied to packets and/or frames and/or flows selected as a result of classification. An example of an abstract service function is "a firewall". The implied order may not be a linear progression as the architecture allows for SFCs that copy to more than one branch, and also allows for cases where

there is flexibility in the order in which service functions need to be applied. The term service chain is often used as shorthand for service function chain.

- **Service Function (SF)**: A function that is responsible for specific treatment of received packets. A Service Function can act at various layers of a protocol stack (e.g., at the network layer or other OSI layers). As a logical component, a Service Function can be realized as a virtual element or be embedded in a physical network element. One or more Service Functions can be embedded in the same network element. Multiple occurrences of the Service Function can exist in the same administrative domain. One or more Service Functions can be involved in the delivery of added-value services. An SF may be SFC encapsulation aware, that is it receives and acts on information in the SFC encapsulation, or unaware, in which case data forwarded to the SF does not contain the SFC encapsulation.
- **Service Function Forwarder (SFF):** A service function forwarder is responsible for forwarding traffic to one or more connected service functions according to information carried in the SFC encapsulation, as well as handling traffic coming back from the SF. Additionally, a service function forwarder is responsible for delivering traffic to a classifier when needed and supported, transporting traffic to another SFF (in the same or different type of overlay), and terminating the SFP.
- **Metadata**: provides the ability to exchange context information between classifiers and SFs and among SFs.
- **Service Function Path (SFP)**: The SFP provides a level of indirection between the fully abstract notion of service chain as a sequence of abstract service functions to be delivered, and the fully specified notion of exactly which SFF/SFs the packet will visit when it actually traverses the network. By allowing the control components to specify this level of indirection, the operator may control the degree of SFF/SF selection authority that is delegated to the network.
- **SFC Encapsulation**: The SFC Encapsulation provides at a minimum SFP identification, and is used by the SFC-aware functions, such as the SFF and SFC-aware SFs. The SFC Encapsulation is not used for network packet forwarding. In addition to SFP identification, the SFC encapsulation carries metadata including data plane context information.
- **Rendered Service Path (RSP)**: The Service Function Path is a constrained specification of where packets assigned to a certain service function path must go. While it may be so constrained as to identify the exact locations, it can also be less specific. Packets themselves are of course transmitted from and to specific places in the network, visiting a specific sequence of SFFs and SFs. This sequence of actual visits by a packet to specific SFFs and SFs in the network is known as the Rendered Service Path (RSP). This definition is included here for use by later documents, such as when solutions may need to discuss the actual sequence of locations the packets visit.
- **SFC-enabled Domain**: A network or region of a network that implements SFC. An SFC-enabled Domain is limited to a single network administrative domain.
- **SFC Proxy:** Removes and inserts SFC encapsulation on behalf of an SFC-unaware service function. SFC proxies are logical elements.

ETSI-NFV and IETF-SFC architectures are complementary in the sense that the first provides a set of tools to ensure virtual function lifecycle management, while the second provide a framework to ensure traffic steering across network functions. The two architectures show a clear contact point in the definition of the path that data follows across the end-to-end service.

- **Service Function Chain (SFC)**: A service function chain defines an ordered set of abstract service functions (SFs) and ordering constraints that must be applied to packets and/or frames and/or flows selected as a result of classification. An example of an abstract service function is "a firewall". The implied order may not be a linear progression as the architecture allows for SFCs that copy to more than one branch, and also allows for cases where there is flexibility in the order in which service functions need to be applied. The term service chain is often used as shorthand for service function chain.

- A **VNF Forwarding Graph Descriptor (VNFFGD)** describes a topology of the NS or a portion of the NS, by referencing a pool of connection points and service access points, the descriptors of its constituent VNFs, PNFs and of the VLs that connect them. It may also contain one or more Network Forwarding Path (NFP) descriptors.

In the last part of this section we will provide a high-level description of state-of-art of a SFC framework implementation in OpenStack, which is the de-facto standard implementation of VIM functionality of the ETSI-NFV architecture.

OpenStack has its own architecture to realize Service Functioning Chain topics. Such architecture is based on an extension of the standard Neutron project (Figure 18) with the following elements:

- Neutron API Service Chain which is a restful API used which grants external and programmatic configuration of the SFC primitives within OpenStack
- A neutron Service Chain Plugin which provides the level of abstraction towards resources
- Service Chain drivers that provide technology specific primitives for the underlying layer configuration.

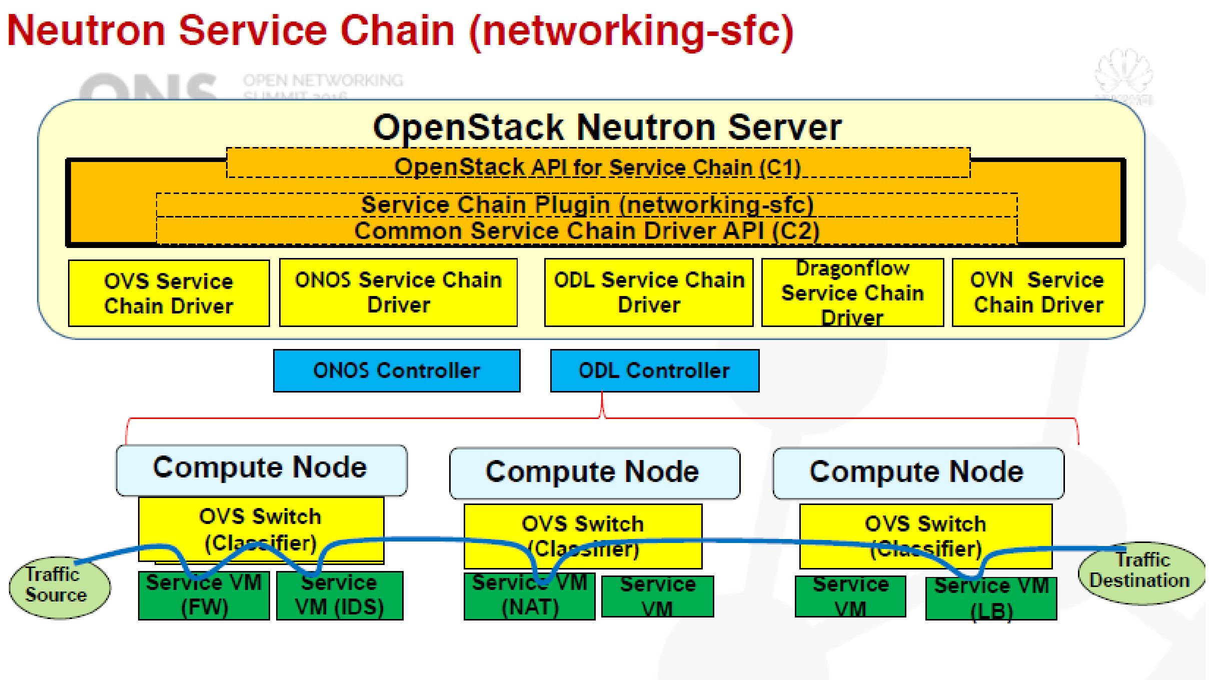

Figure 18. OpenStack SFC architecture [50]

The typical OpenStack deployment foresees that all OpenStack Networking services and OpenStack Compute instances connect to a virtual network via ports making it possible to create a traffic steering model for service chaining using only ports. Including these ports in a port chain enables steering of traffic through one or more instances providing service functions.

A port chain, or service function path, consists of the following:

- A set of ports that define the sequence of service functions.
- A set of flow classifiers that specify the classified traffic flows entering the chain.

If a service function involves a pair of ports, the first port acts as the ingress port of the service function and the second port acts as the egress port. If both ports use the same value, they function as a single virtual bidirectional port.

A port chain is a unidirectional service chain. The first port acts as the head of the service function chain and the second port acts as the tail of the service function chain. A bidirectional service function chain consists of two unidirectional port chains.

A flow classifier can only belong to one port chain to prevent ambiguity as to which chain should handle packets in the flow. A check prevents such ambiguity. However, you can associate multiple flow classifiers with a port chain because multiple flows can request the same service function path.

OpenStack provides an architecture which is flexible enough to provide support to SFC in different deployment scenario. Beside the typical case, where SFC-enabled framework is casted over an SDN enabled architecture (where the data-plane configuration is done by means of wide range SDN controllers), there is a case where SFC-enabled framework is casted over a non-SDN enabled architecture, where the configuration of data-plane is done directly by neutron server component residing over OpenStack controller node.

The current OpenStack implementation of IETF-SFC architecture, suffers of the following limitations:

- OpenStack does not provide support to encapsulation component of the IETF-SFC architecture
- OpenStack offers a limited range of capabilities in terms of traffic classification (L2, L3 parameters)
- The model used, allows to steer traffic only between ports which are associated to a VM, with no proxy functionality enabled. This model is not flexible enough to provide support for nodes (like MEC nodes) which are inline nodes between RAN devices and data-centre devices.

4.2 5GCity Enhancements in Networking virtualization

4.2.1 VOSYSwitch

VOSYSwitch is today a virtual switch technology available for ARM and Intel processors. During 5GCity, specific VOSYSwitch extensions will be developed to improve switching performance at the edge.

First, a lightweight version of VOSYSwitch will be created aiming at improving system security with a smaller attack surface and achieving a smaller overhead.

In addition, hardware accelerators will be used to enhance processing capabilities of the virtual switch, targeting to achieve higher performance. Different types of hardware accelerators can be used to offload the system general purpose processor, e.g., Application-Specific Integrated Circuits (ASICs), network processors, Field-programmable gate array (FPGAs), Graphics Processing Unit (GPU), etc. These are today considered an important part of the NFVI and are defined in the Interfaces and Architecture ETSI GS NFV-IFA 001 document [51] where accelerators life-cycle and operations are specified (life-cycle management, feature discovery and fault tolerance, etc.). Among these solutions, VOSYS will particularly focus on FPGAs, which provide the best performance per watt trade-off and are for this reason particularly fitting the edge of the network for their reconfigurability and power consumption. More in detail, VOSYSwitch will be enhanced with specific Lua modules to enable packet processing directly in the FPGA.

4.2.2 Wireless Virtualization

One main feature required to implement network slicing for the intrinsic neutral host use case of 5GCity is the virtualization of the wireless RAN. The achievements discussed in this section emerge from the work performed to extend the network slicing features presented in Section 4.1.3, so VNFs and wired backhaul can be integrated with RAN elements. The extensions need to provide means to:

- Use a standard control and management plane to set up and configure wireless interfaces;
- Provide network isolation and data rate control for the wireless interfaces;
- Integrate virtual wireless resources with traditional, SDN-based network slices and the rest of the 5GCity architecture.

In the following we present the enhancements developed for the Wi-Fi and LTE-based solutions.

4.2.2.1 Wi-Fi Enhancements

In order to achieve these features, the wireless RAN virtualization in 5GCity is implemented as a set of software modules that range from Linux kernel and user space software modules running on the wireless nodes, over a NETCONF-based control and management plane, to the main SDN controller. The latter is responsible for configuring and managing the RAN and that acts as interface to the rest of the 5GCity architecture.

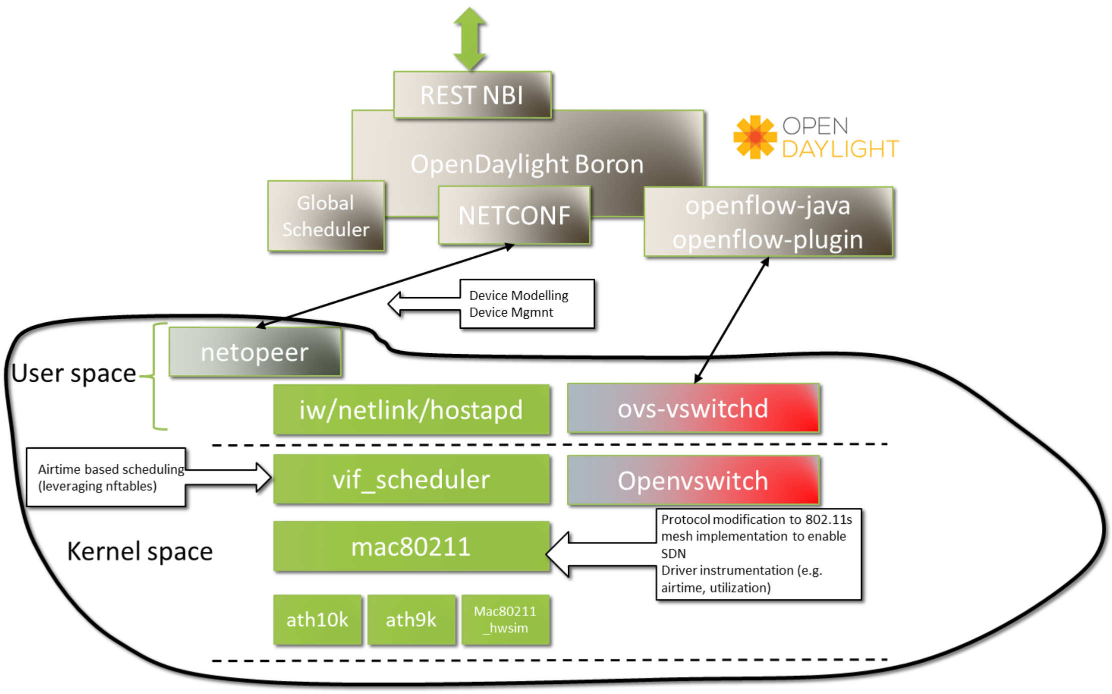

Figure 19. SDN-based wireless virtualization software components

Figure 19 depicts an overview of the software components running in the wireless nodes hosting the wireless network interfaces (bottom) and the SDN controller in charge of the management and configuration of the wireless interfaces (top). We refer to the software running in the Wi-Fi nodes as Agent software, a composition of services that enables the network slicing in the Wi-Fi RAN. The agent software is tightly coupled to the functions provided by core software modules hosted in the SDN controller. In the following, each of these components are explained in detail.

Agent software for wireless *virtualization*

For the Wi-Fi virtualization two crucial software elements are being developed in order to allow the slicing of the RAN: the vif scheduler and the netopeer agent. The vif scheduler is responsible for applying isolation in form of airtime slicing of the available radio resources, whereas the netopeer agent hosts a NETCONF server that is used to set up and configure the vifs on top of the physical network interfaces installed in a wireless node.

Isolation and data rate adaption: vif scheduler

A basic slicing feature for networks is the assignment of shares of the available data rate of network connections to a specific tenant or service in order to provide QoS. In wired connections, where fix data rates are provided (e.g. 1 Gbps), slicing can be easily implemented by using queuing, packet scheduling or other, similar mechanism to control the amount of traffic that is sent over interfaces belonging to a specific tenant.

In wireless communications, while some upper data rate limits can be determined from the hardware specification, the actual available data rate of a wireless link can very heavily. In particular, each user equipment connected to a Wi-Fi access point can have a different nominal data rate than other users attached at the same time due to the position of the equipment, obstacles or even mobility. Further, different user equipment may not support all data rates offered by an access point. Another consideration for availability of wireless resources is that as more users get connected to an access point, the actual data rate decreases more and more due to the CSMA/CA accessing scheme implemented in Wi-Fi. Thus, it is not possible to guarantee specific data rates to a tenant as part of their network slice.

In 5GCity therefore a Wi-Fi RAN slice is defined as the assignment of a ratio of the actually available radio resources, in terms of airtime. We define the airtime to be the real time the transmission of a packet occupies the radio medium. In order to implement this type of slicing, we design our so called *vif scheduler*. This scheduler is composed of two parts:

1 **The local scheduler**, an agent software running in the wireless nodes. This software is implemented as a dynamically loadable kernel module that sits on top of the mac80211 module. The scheduler can be configured to apply specific airtime ratios for any underlying virtual access points in the downstream traffic.

2 **The global scheduler**, a software module that forms part of the SDN controller. The global scheduler is responsible for configuring the airtime ratios of the local schedulers and for monitoring them in order to detect whether the ratios are correctly applied.

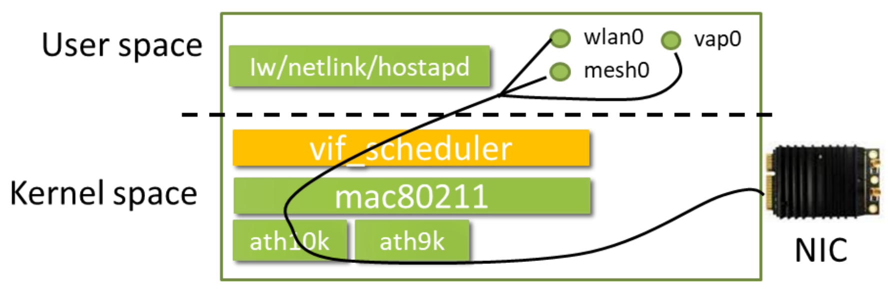

Figure 20. Vif scheduler as part of the agent software running on the Wi-Fi nodes

The local scheduler, a kernel module integrated into the network stack as shown in Figure 20, processes packets to be transmitted from the user space. The local scheduler component needs to fulfill the following two basic requirements: First, it needs to be work conserving, i.e., the amount of unused airtime shall be minimized by reassigning unused resources to vifs in need of airtime for packet transmissions. Second, it needs to be independent from any underlying Wi-Fi drivers and hardware in order to achieve a high degree of interoperability and portability.

Slicing is performed in the time domain: each user, e.g. a tenant or wireless operator represented as a vif on top of a physical radio, is assigned a specific ratio of the available airtime. This share represents a guaranteed minimum of the available airtime during a certain time interval, but not a maximum. The ratio assignment is done by the SDN controller that takes this input via its Northbound API and configures it on the local schedulers running in the nodes via NETCONF. The percentage of time the radio channel can be used by the tenant reflects the SLA signed by the tenant.

The local scheduler keeps track of the usage of each vif and only allows a packet to be handed over to the underlying layers if the airtime limit of the tenant has not yet been reached during a periodic interval. Since the actual transmission duration of a packet and therefore its airtime is not known a priori to the local scheduler, it estimates the transmission duration based on calculations that take into account the current degree of congestion of the radio medium, the size of the packet and the currently active modulation coding scheme.

Once the packet has been transmitted, the actually used airtime (which may variate from the expected one) is measured. This airtime is discounted from the available airtime of the tenant and the deviation from the expected one is used to determine the degree of congestion. If after discounting the credit no more airtime is available during the current time slot, no further packets may be transmitted by the tenant and the next transmission is delayed until the next cycle.

The local vif scheduler performs local optimization of the available airtime for a tenant in a work conserving manner: any unused airtime during a specific time slot is assigned to the active tenants in order to improve the performance.

SDN Controller: Global Scheduler

Since the local scheduler optimizes locally and takes into account only transmitted packets into its calculations but not received ones, at some points the SLAs might not be fulfilled correctly: in the presence of other Wi-Fi nodes operating on the same channel and using the local scheduler, the assigned ratios might not be applied correctly. Figure 21 shows a case where two physical interfaces operating in the same frequency and in range of each other run the local vif scheduler. Two tenants, A and B, are present and are configured to apply an airtime ratio of 30% and 70%, respectively. In the case that tenant A only transmits on the first node and tenant B only transmits on the second node, each local scheduler will try to assign the full airtime (i.e. 100%) to the vifs. However, since the nodes compete for the channel, the actual resulting transmission ratio is 50% both for tenant A and B, as they try to access both the medium in the same way. This, however, violates the ratios of 30%/70% accorded in the SLA.

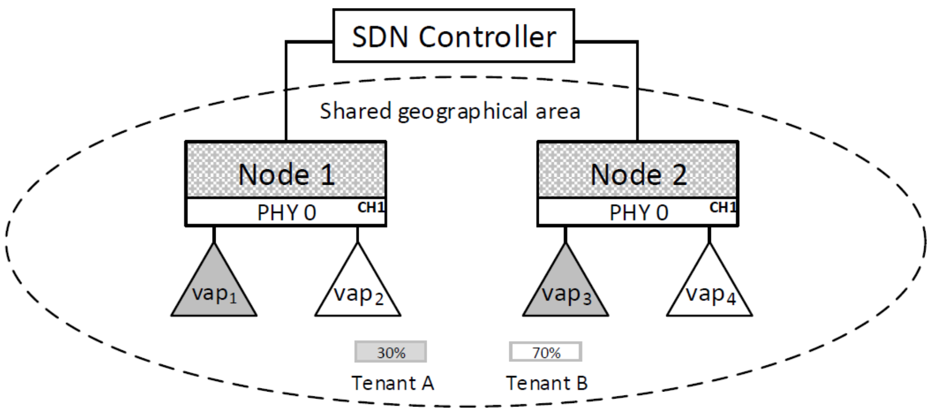

Figure 21. Local schedulers competing for the medium can lead to SLA violations.

In order to avoid such situations, the global scheduler, running in the SDN controller plays a crucial role: the global scheduler module periodically collects information on the actual transmission ratios from the local schedulers. Based on the actual airtime usage it can detect SLA violations. In case of an SLA violation, the SDN controller reconfigures the local schedulers in such a way that the desired SLA ratios are applied. For this type of reconfiguration of the local schedulers, but also for the instantiation and configurations of vif in general, the SDN controller uses a NETCONF client that talks with the NETCONF servers running on each of the nodes. The overall architecture including the software elements of the SDN controller and the Wi-Fi nodes in an example with three tenants is shown in Figure 22. Take into account that the same NETCONF interface is intended to be used to control and configure the Small Cells.

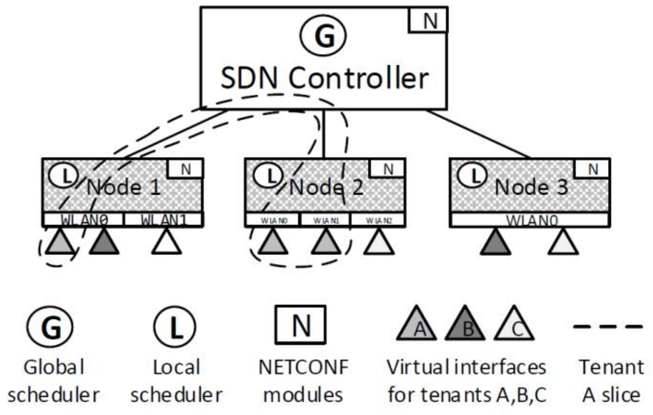

Figure 22. Example of the scheduler architecture with the global and local schedulers and three tenants

4.2.2.2 LTE based virtualization enhancements

The LTE-based RAN architecture for 5GCity (Figure 23) includes several innovations in the area of RAN functional disaggregation, RAN and Network Slicing with SDN control and RAN function virtualization.

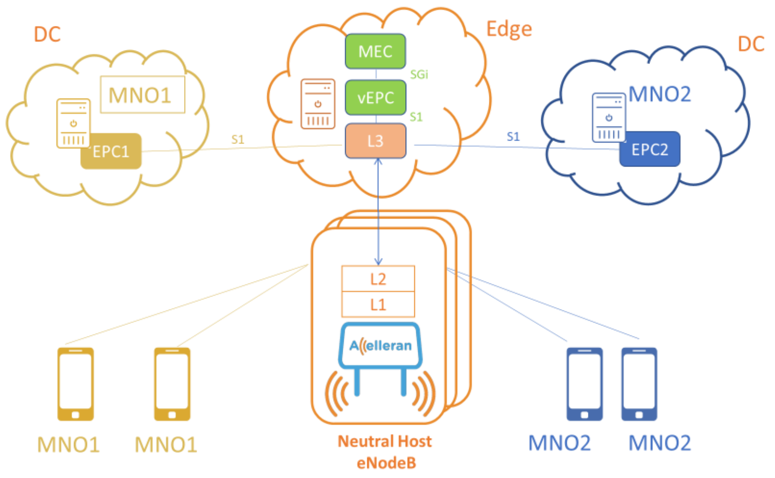

Figure 23. 5GCity RAN Network Architecture Overview

Network functions in the architecture are:

- LTE Layer 1: The physical layer functions of the LTE air interface require specialised processing acceleration for DSP functions such as FFT, Turbo coding, etc and execute in each radio head on specialised DSP silicon. This is not a virtualised function in 5GCity and is common to all network slices.
- LTE Layer 2: The RLC and MAC functions of the LTE air interface are required to meet real-time schedules and are closely coupled to the LTE physical layer 1 implementation. These functions also execute in each radio head. RLC/MAC are not virtualised functions in 5GCity and are common to all network slices.

- LTE Layer 3: The layer 3 (control plane) function of the LTE air interface is implemented as a virtual network function which runs in the Edge NFVI. 5GCity L3 supports network slicing and connection to multiple EPC (MME) instances (one per slice).
- vEPC: EPC (packet core network) is deployed at the network edge to support MEC access and low latency applications. Each instance of vEPC supports a network slice offering MEC application access.
- Datacenter EPC: The RAN functions support connectivity to EPC functions implemented at the data center. The neutral host use case assumes multiple EPCs and network slices to support access provision to multiple tenant service providers.

Management and Orchestration of RAN VNFs

The RAN VNFs provide API's towards the Edge NFVI SDN controller for dynamic slice management and also provide interfaces for classic FCAPS management [52] of the static configuration (e.g. operating frequency, global cell parameters, etc) which are configured on installation and are independent from the slice LCM.

- eNB FCAPS: RAN layer 3 is deployed as a Docker container. General RAN FCAPS management is supported via multiple interfaces including BBWF TR069 [53], webGUI and CLI. One of these mechanisms is required to initially configure each cell for service in the network.
- eNB SDN control: RAN L3 provides an SDN control interface supporting the following functions:
  - Initialisation
  - Slice Profile Creation
  - Slice Profile Modification
  - Slice Profile Deletion

  SDN control is implemented using the NETCONF protocol via a datamodel which is formally defined in YANG. Figure 24 illustrates the RAN orchestration model as implemented in NETCONF / YANG.

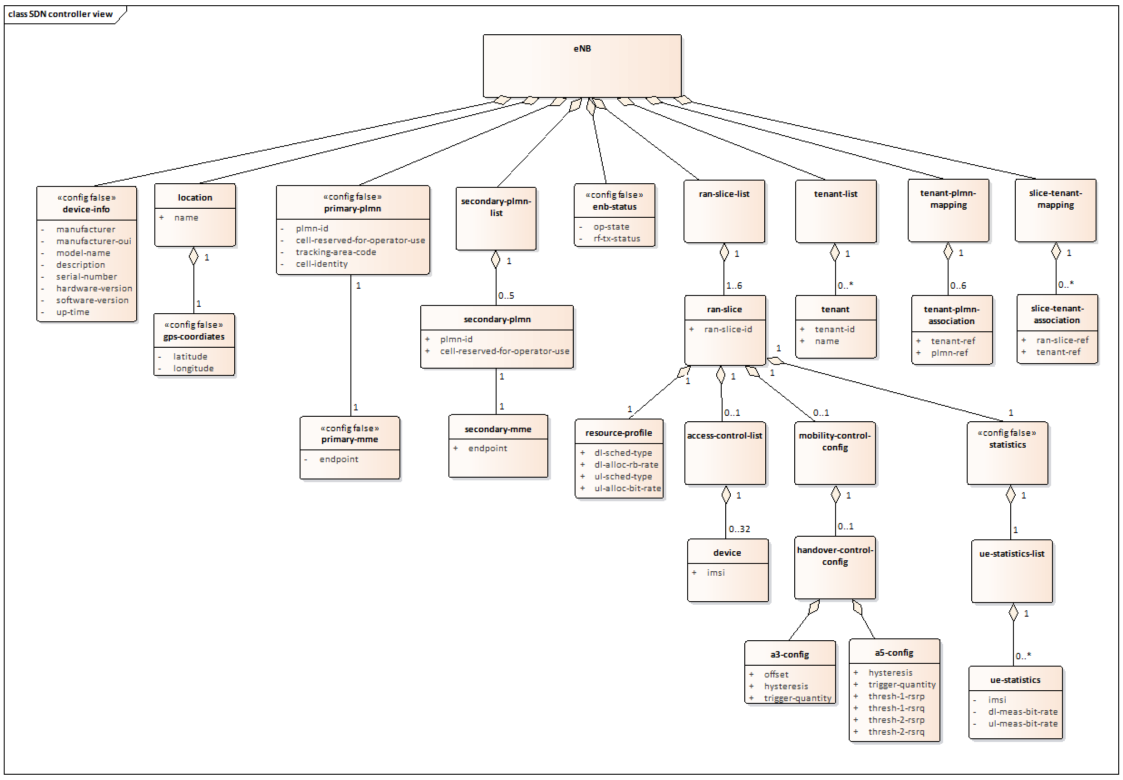

Figure 24. RAN SDN Control Interface Model

The data model has been aligned with the approach taken for Wi-Fi orchestration. Each slice corresponds to a 3GPP PLMNId (network identifier) and there can be up to six slices created. Additionally, per slice, an access control list can be instantiated to control which specific users (IMSI identifiers) are permitted access to the slice. Each RAN slice is associated with an MME and EPC via a standard S1 connection. This EPC can be either at Edge or Datacentre level.

- vEPC: The vEPC for edge deployment is also deployed as a Docker container and provides WebGUI and/or CLI based configuration interfaces. Traffic is presented to the MEC applications via a standard SGi interface. Therefore, termination of GTPu tunnelling protocols is handled within the EPC and is of no concern to the Mobile Edge.

4.2.4 Local traffic steering in MEC node

In this section we will concentrate on a specific aspect of MEC node design, which is related to how the traffic steering among MEC node entities is accomplished. At the current state of the art, there is still no consolidate ETSI standard design which describe a solution for this problem. ETSI MEC WG is working on early draft document [54] which is more focused on lawful interception and cover only some aspects of traffic offloading topics. Also, there is no specific software solution, which is compliant with ETSI MEC in an ETSI NFV compliant architecture

The problem of local traffic handling in an ETSI MEC enabled architecture is central to 5GCity architecture which consists in a three-tier architecture with pool of resources available at data-center level, edge level and far-edge level (collocated with radio access devices). The acronym MEC stands for Multi-access Edge Computing, implying a pool of constrained resources, located at the edge of the cloud administrative domain, which acts as bump in the wire, handling traffic coming from multiple radio access technology (LTE, WiFi, VOLTE, UMTS, GPRS). Traffic handling at edge node can be divided in 3 different scenarios:

- Scenario1: traffic received from RAN devices and transparently forwarded to Core resources.
- Scenario2: traffic received from RAN devices has to be served locally by MEC application which acts as end-point. No traffic for these flows is sent to the core resources.
- Scenario3: traffic received from RAN devices has to be first served locally by MEC application and then re-inserted of its normal path to core resources.

For all the afore-mentioned scenarios, traffic steering at edge node imply that MEC nodes data and control plane must be enabled with full awareness of the protocols flowing across MEC nodes data plane. 5GCity radio access frameworks encompass two different access technologies: Wi-Fi and LTE.

A part of WP3 activities, the project will investigate a software solution which fully provides traffic handling at MEC node level. This design solution need to be realized by specifying:

- Data-plane which is able to dynamically select traffic according user-plane protocol characteristics
- Control plane design which is able to dynamically program the data-plane to steer traffic among MEC local entities.

An initial solution is depicted in Figure 25. where the different entities of MEC architecture which comes into play to realize traffic offloading are depicted. MEC platform will implement the control plane functionality and will program via mp2 interface MEC host data-plane on the basis of the basis of the joined commands received by the MEC orchestration layers (MEPM-V and MEAO) and NFV orchestration layer (NFVO, WAN Resource manager).

The final result is that MEC host data-plane is able to steer traffic across MEC local entities (MEC applications, and MEC specific services exposed by MEC platform). The functional block called LGW, is part of the LTE stack and enables local termination of GTP-u tunnels sessions. Final solution should also be investigated against Issue#9 as described in Section 2.1.

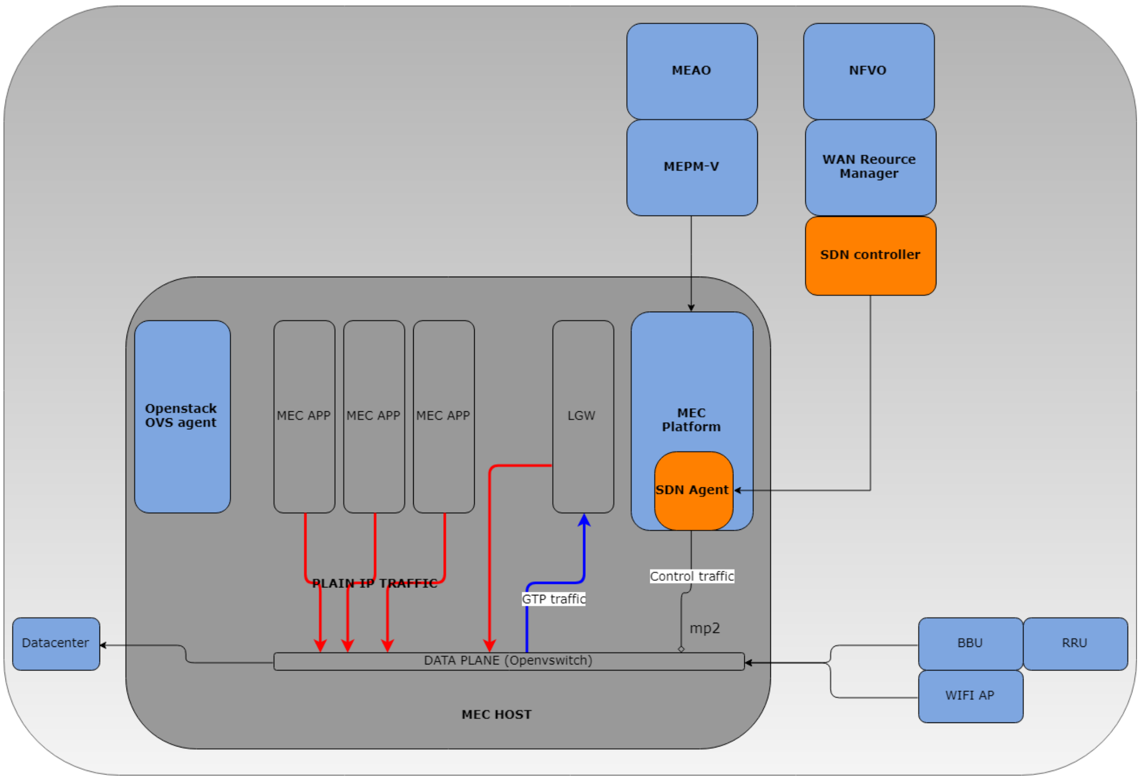

Figure 25. Local traffic handling in ETSI MEC architecture

Given the different scenarios of traffic handling and the different combination of radio access technologies and related protocols, a set of requirements for MEC nodes data plane can be derived. The MEC node requirements, presented in Table 5, are divided in a first set of requirements, which are pertaining to data-plane and in a second set which are related to control-plane.

REQ_ID	Type	Description
REQ_DP_1	MUST	MEC node data-plane must be able to classify traffic entering the MEC node from the normal path by operate a filtering according to the following parameters - Wi-Fi traffic (MAC addresses, IP Address, VLAN tags) - LTE traffic (GTP-U parameters (TEID, Type), IP addresses
REQ_DP_2	MUST	MEC node data-plane must be able to forward traffic to selected output destination port according to the following parameters - MAC addresses - IP addresses - VLAN tags GTP-U parameters (TEID, dest_ip_addr)
REQ_DP_3	MUST	MEC node data-plane must be able to steer traffic according to a logic which is SFC oriented. The SFC traffic steering involves only MEC applications
REQ_CP_1	MUST	MEC node data-plane must be able to be configured by means of a dedicated control plane, embodied by MEC platform
REQ_CP_2	MUST	MEC node must be equipped with VNF (L-GW) which is able to terminate the GTP-u traffic and extract plain IP traffic to be served by MEC applications

Table 5. Requirement list for MEC node data plane and control plane.

In the remaining part of this section, we will provide a quick state of the art regarding the awareness of 5GCity radio access protocols of most common software data-plane implementation as required in REQ_DP_1. Our investigation covers only the part related to GTP-u protocol, since full awareness of plain IPv4 protocol is already in place in most common data-plane software solutions.

The current implementation of Openvswitch (v2.9.0) provides support for functionality related to REQ_DP_1 and REQ_DP_2 related only when in a scenario where Wi-Fi is the only radio access technology. When the Radio Access technology is LTE enabled, Openvswitch does not offer support to GTP-u protocol in terms of being able to configure dataflows entry for GTP-u parameters selection and GTP-u header manipulation (GTP header push and pop operations). There are some non-main stream projects which offers patches which seems to be a promising approach for the enabling of GTP-u protocol awareness and are actually under evaluation.

The current implementation of VOSYS virtual switch does not provide GTP-u protocol awareness, and there are no plans or roadmap to enable the related feature in the next period.

In addition, it is worth observing that the current Openflow version (v1.5.0) lacks support for custom header fields and nesting headers. Potential evolution of Openflow protocol is under analysis by OFN [55].

In this section we have provided a rationale which describe a specific problem (local traffic steering in a MEC node), we have provided requirements which describe a potential solution and an initial solution which integrates MEC host data-plane and control-plane in the wider landscape of 5GCity orchestration platform. Next steps to be taken are organized according two main key items:

- Analysis of the patches to Openvswitch to enable MEC data-plane with full GTP-u protocol awareness
- Design of an SDN enabled control plane solution integrated with 5GCity orchestration platform which is able to interact with MEC Platform entity which operates the configuration of MEC data-plane.

5 Conclusions

Virtualization is a key technology for the 5GCity infrastructure because it provides support for multitenancy, flexibility and isolation which are must have features for any smart city.

This deliverable shows that existing virtualization technologies (both for the virtualization of the computing resources in Section 3.1 and network resources in Section 4.1) need to be extended to address smart cities challenges and provides documentation on how these challenges will be tacked in 5GCity (Sections 3.2 and 4.2). Moreover, it documents how 5GCity plans to fully cover the convergence of ETSI NFV and MEC (Section 2).

The following table recaps the WP3 objectives presented in Section 1 and shows how these will be achieved during the WP3 activities.

	5GCity Virtualization objective	How we address it
1	Optimizing virtualization technologies for heterogeneous and resource constrained devices	**Performance** (efficiency, scalability, computing power) is addressed with specific developments on unikernels (unikraft) and virtual machines (KVM). Thanks to these extensions a higher number of more performant guests will be executed on the 5Gcity infrastructure. **Security and trusted computing** is added at the infrastructure level through the EdgeNFVI and EdgeVIM solutions based on Arm TrustZone. These extensions will enable VNFs geo-fencing and Trusted Computing features in guests.
2	Implementing network virtualization targeting efficient software switches and wireless virtualization	**Wireless (Wi-Fi and RAN) virtualization** is addressed standardazing the control/management plane to configure wireless interfaces and adding support to data rate control and isolation. This will enable efficient and performant wireless medium slicing. **Performance** will be addressed by optimizing VOSYSwitch for resource contrained devices. Moreover, existing Service Functions Chaining limitations (e.g., flexibility, support for encapsulation, traffic classification) will be addressed focusing on MEC nodes. This will enable higher performance/efficiency and traffic steering among MEC nodes.
3	Creating specific VNFs solutions for MEC nodes	**MEC is considered as part of 5Gcity**. Its integration in the project architecture has been designed in this document and will be developed by extending the open source project fog05. Thanks to this work, MEC applications will be able to run on the 5Gcity infrastructure. Further developments in this direction will come from Task 3.3 (which will start at M13) and will be documented in D3.2.

Table 6 5GCity WP3 objectives and solutions

References

[1] ETSI, “ETSI GR MEC 017 V1.1.1,” February 2018. [Online]. Available: http://www.etsi.org/deliver/etsi_gr/MEC/001_099/017/01.01.01_60/gr_MEC017v010101p.pdf.

[2] ETSI, “ETSI GS NFV-INF 003 V1.1.1,” 12 2014. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV-INF/001_099/003/01.01.01_60/gs_NFV-INF003v010101p.pdf.

[3] ETSI, “ETSI GS NFV-INF 004 V1.1.1,” January 2015. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV-INF/001_099/004/01.01.01_60/gs_nfv-inf004v010101p.pdf.

[4] ETSI, “ETSI GS NFV-INF 005 V1.1.1,” 12 2014. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV-INF/001_099/005/01.01.01_60/gs_NFV-INF005v010101p.pdf.

[5] ETSI, “ETSI GS NFV-IFA 013 V2.1.1,” October 2016. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/013/02.01.01_60/gs_NFV-IFA013v020101p.pdf.

[6] ETSI, “ETSI GS MEC 010-2 V1.2.2,” 2018.

[7] “LL-MEC,” [Online]. Available: http://mosaic-5g.io/ll-mec/. [Accessed May 2018].

[8] N. N. Anta Huang, “Low Latency MEC Framework for SDN-based LTE/LTE-A Networks,” in *IEEE International Conference on Communications (ICC)*, Paris, France, 2017.

[9] N. Nikaein, “Agile Network service delivery,” 7-8th November 2017. [Online]. Available: https://www.openairinterface.org/docs/workshop/4_OAI_Workshop_20171107/Talks/NIKAEIN_mosaic5g_OAI_WS.pdf. [Accessed May 2018].

[10] 5GCity Project, *D4.1 - Orchestrator Design, Service Programming, and Machine-learning Models,* 2018.

[11] “Unikernels meet NFV,” June 2016. [Online]. Available: https://www.ericsson.com/research-blog/unikernels-meet-nfv/.

[12] Y. K. ,. D. L. ,. U. L. ,. A. L. Avi Kivity, “KVM: the linux virtual machine monitor,” in *Linux symposium*, Ottawa, 2007.

[13] B. D. K. F. S. H. T. H. A. H. R. N. I. P. a. A. W. Paul Barham, “Xen and the art of virtualization,” in *The nineteenth ACM symposium on Operating systems principles*, New York, USA, 2003.

[14] “Introducing the Qualcomm Falkor CPU core purpose-built for cloud workloads,” [Online]. Available: https://www.qualcomm.com/news/onq/2017/08/20/introducing-qualcomm-falkor-cpu-core-purpose-built-cloud-workloads.. [Accessed August 2017].

[15] D. F. Tiago Alves, “TrustZone: Integrated Hardware and Software Security-Enabling Trusted Computing in Embedded Systems,” 2004.

[16] R. N. W. a. M. H. Ilias Marinos, “Network stack specialization for performance,” in *ACM conference on SIGCOMM*, 2014.

[17] T. L. M. S. T. G. D. S. D. S. R. M. A. C. B. S. J. L. J. C. a. I. L. Anil Madhavapeddy, “Jitsu: Just-In-Time Summoning of Unikernels,” in *12th USENIX Symposium on Networked Systems Design and Implementation*, 2015.

[18] “Mini-OS,” [Online]. Available: https://wiki.xenproject.org/wiki/Mini-OS.. [Accessed May 2018].

[19] “Erlang on Xen,” 2012. [Online]. Available: http://erlangonxen.org/. [Accessed May 2018].

[20] M. A. C. R. V. O. M. H. R. B. a. F. H. Joao Martins, “ClickOS and the Art of Network Function Virtualization,” in *11th USENIX Symposium on Networked Systems Design and Implementation*, 2014.

[21] “The Click modular router: fast modular packet processing and analysis,” [Online]. Available: https://github.com/kohler/click.

[22] A. I. F. M. J. M. Y. V. F. S. K. Y. M. H. a. F. H. Simon Kuenzer, “Unikernels Everywhere: The Case for Elastic CDNs,” in *13th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, (VEE '17)*, 2017.

[23] D. L. G. C. P. E. N. H. D. M. a. V. Z. Avi Kivity, “OSv—Optimizing the Operating System for Virtual Machines,” in *2014 USENIX Annual Technical Conference (USENIX ATC ’14)*, 2014.

[24] “The Solo5 unikernel,” [Online]. Available: https://github.com/Solo5/solo5. [Accessed May 2018].

[25] “A minimal, resource efficient unikernel for cloud services,” [Online]. Available: https://github.com/hioa-cs/IncludeOS. [Accessed May 2018].

[26] C. L. F. S. J. M. S. K. S. S. K. Y. C. R. a. F. H. Filipe Manco, “My VM is Lighter (and Safer) than your Container,” in *26th Symposium on Operating Systems Principles (SOSP '17)*, 2017.

[27] ETSI, “ETSI GS NFV-SEC 003 V1.1.1,” 12 2014. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV-SEC/001_099/003/01.01.01_60/gs_NFV-SEC003v010101p.pdf. [Accessed May 2018].

[28] “Trusted Computing Group,” [Online]. Available: https://trustedcomputinggroup.org/trusted-computing/. [Accessed May 2018].

[29] “GlobalPlatform TEE,” [Online]. Available: https://www.globalplatform.org/mediaguidetee.asp. [Accessed May 2018].

[30] “Intel Trusted Execution Technology,” [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/trusted-execution-technology/trusted-execution-technology-security-paper.html. [Accessed May 2018].

[31] Intel, “Intel SGX,” [Online]. Available: https://software.intel.com/en-us/sgx. [Accessed May 2018].

[32] "ARM TrustZone," [Online]. Available: https://www.arm.com/products/security-on-arm/trustzone. [Accessed May 2018].

[33] S. Ravidas, "Incorporating Trust in Network Function Virtualization," 2016.

[34] R. R. S. a. L. v. D. Perez, "vTPM: virtualizing the trusted platform module," in *15th Conf. on USENIX Security Symposium*, 2006.

[35] TCG, "Virtualized Trusted Platform Architecture Specification," September 2011. [Online]. Available: https://www.trustedcomputinggroup.org/wp-content/uploads/TCG_VPWG_Architecture_V1-0_R0-26_FINAL.pdf. [Accessed May 2018].

[36] "OpenStackOpenStack wiki," [Online]. Available: https://docs.openstackOpenStack.org/nova/pike/admin/security.html. [Accessed May 2018].

[37] S. Weis, "Trusted Computing & OpenStack," PrivateCore, July 2014. [Online]. Available: https://pdfs.semanticscholar.org/presentation/762c/447a11ee258d313fb87c24dab0b5939cd14c.pdf . [Accessed May 2018].

[38] Trusted Computing Group, Incorporated. , "TPM Main Part 2 TPM Structures," July 2007. [Online]. Available: https://trustedcomputinggroup.org/wp-content/uploads/mainP2Structrev103.pdf. [Accessed May 2018].

[39] J. G. William Futral, Intel Trusted Execution Technology for Server Platforms: A Guide to More Secure Datacenters, Apress, 2013.

[40] "OpenAttestation Project," [Online]. Available: https://github.com/OpenAttestation/OpenAttestation. [Accessed May 2018].

[41] 5GCity Project, *D2.1 - 5GCity System Requirements and Use Cases.*

[42] " Open Portable Trusted Execution Environment," [Online]. Available: https://www.op-tee.org/index.html.

[43] "Snabb framework," [Online]. Available: http://snabb.co/. [Accessed May 2018].

[44] J. F. Michele Paolino, "Turning an open source project into a carrier grade vswitch for NFV: Vosyswitch challenges & results," in *Network Infrastructure and Digital Content (IC-NIDC), 2016 IEEE International Conference on*, Beijing, China, 2016.

[45] "Optimizations performed by LuaJIT," [Online]. Available: http://wiki.luajit.org/Optimizations. [Accessed May 2018].

[46] M. M. C. G. W. A. G. C. W. a. M. F. Bebenita, "Trace-based compilation in execution environments without interpreters," in *the 8th International Conference on the Principles and Practice of Programming in Java*, Vienna, Austria, 2010.

[47] "xRAN," [Online]. Available: http://www.xran.org/resources/. [Accessed May 2018].

[48] Internet Engineering Task Force (IETF), "RFC 7665: Service Function Chaining (SFC) Architecture," [Online]. Available: https://datatracker.ietf.org/doc/rfc7665/.

[49] European Telecommunications Standards Institute , “ETSI GS NFV-EVE 005 V1.1.1,” December 2015. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV-EVE/001_099/005/01.01.01_60/gs_NFV-EVE005v010101p.pdf. [Accessed May 2018].

[50] “Service Function Chaining Extension for OpenStack Networking,” [Online]. Available: https://docs.openstack.org/networking-sfc/latest/. [Accessed May 2018].

[51] European Telecommunications Standards Institute, “ETSI GS NFV-IFA 001 V1.1.1,” December 2015. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/001/01.01.01_60/gs_nfv-ifa001v010101p.pdf. [Accessed May 2018].

[52] “ISO/IEC 7498-4: Information processing systems -- Open Systems Interconnection -- Basic Reference Model -- Part 4: Management framework,” 1989. [Online]. Available: http://standards.iso.org/ittf/PubliclyAvailableStandards/s014258_ISO_IEC_7498-4_1989(E).zip.

[53] The Broadband Forum, “TR-069: CPE WAN Management Protocol,” March 2018. [Online]. Available: https://www.broadband-forum.org/technical/download/TR-069.pdf.

[54] ETSI MEC WG, “Multi-access Edge Computing (MEC); Support for regulatory requirements, ETSI MEC 026 V1.2.2,” 2018.

[55] “Open Networking Foundation,” [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2013/05/TR-535_ONF_SDN_Evolution.pdf. [Accessed May 2018].

Abbreviations and Definitions

Abbreviations

3GPP	3rd Generation Partnership Project
API	Application Programming Interface
AppD	Application Descriptor
ASIC	Application Specific Integrated Circuits
BBWF	Broadband World Forum
CDN	Content Delivery Network
CN	Core Network
CPS	Cyber Physical Systems
CPU	Central Processing Unit
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
DPDK	Data Plane Development Kit
DSP	Digital Signal Processing
EL3	Exception Level 3
eNodeB, eNB	Evolved Node B
EPC	Evolved Packet Core
EPS	Edge Packet Services
ETSI	European Telecommunication Standard Institute
EU	European Union
FCAPS	fault, configuration, accounting, performance, security
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
GPRS	General Packet Radio Service
GPU	Graphical Processing Unit
GTP-U	GPRS tunneling protocol user plane
ICT	Information Communication Technology
IETF	Internet Engineering Task Force
IMSI	international mobile subscriber identity
IOCTL	Input-Output Control
IoT	Internet of Things
IoT	Internet of Things
IP	Internet Protocol
JIT	Just in time
JSON	JavaScript Object Notation
KVM	Kernel-based Virtual Machine
LCM	Life Cycle Management
LGW	Local Gateway
LTE	Long Term Evolution
LTE-A	Long Term Evolution Advanced
MAC	Media Access Control address
MANO	Management and Orchestration
MEAO	Multi-access Edge Application Orchestrator
MEC	Multi access Edge Computing
MEPM	Multi-access Edge Platform Manager
MEPM-V	Multi-access Edge Platform Manager - NFV
MME	Mobility Management Entity

NAT	Network Address Translator
NETCONF	Network Configuration Protocol
NFV	Network Function Virtualization
NFVI	Network Function Virtualization Infrastructure
NFVO	Network Function Virtualization Orchestration
NIC	network interface controller
NR	New Radio
ODP	Open Data Plane
OPTEE	Open Platform Trusted Execution Environment
OS	Operating System
OSM	Open Source MANO
OSS	Operations support System
OVS	Open Virtual Switch
PCR	Platform Configuration Registers
PLMN	Public Land Mobile Network
POSIX	Portable Operating Systems Interface for Unix
QEMU	Quick Emulator
QoS	Quality of Service
RAM	Random Access Memory
RAN	Radio Access Network
REST	Representational State Transfer
RLCC/MAC?	Radio Link Control
RNIS	Radio Network Information Services
SDK	Software Development Kit
SDN	Software Defined Network
SF	Service Function
SFC	Service Function Chaining
SFF	Service Function Forwarder
SFP	Service Function Path
SLA	Service Level Agreement
SoC	System on Chip
SSID	Service Set Identifier
TCG	Trusted Computing Group
TDMA	Time-Division Multiple Access
TEE	Trusted Execution Environment
TOF	Traffic offload function
TPM	Trusted Platform Module
TSS	TrouSerS open source project
UMTS	Universal Mobile Telecommunications System
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VNF	Virtual Network Function
VNFFGD	VNF Forwarding Graph Descriptor
VNFM	Virtual Network Function Manager
VOLTE	Voice Over LTE
vRAN	Virtual Radio Access Network
vTPM	Virtual Trusted Platform Module
YANG	Yet Another Next Generation
3GPP	3rd Generation Partnership Project
API	Application Programming Interface
AppD	Application Descriptor

ASIC	Application Specific Integrated Circuits

<END OF DOCUMENT>