5G CITY

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

D3.2: 5GCity Virtualization Infrastructure Interim Release

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: **5GCity**	Project title: **5GCity**

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

Work package: **WP3**

Deliverable title: **5GCity Virtualization Infrastructure Interim Release**

Start date of the project: 01/06/2017 (duration 30 months)	Contractual delivery date: Month 24	Actual delivery date: 31/05/2019

Editor name:
Felipe Huici (NEC)

List of Contributors

Participant	Short Name	Contributor
Virtual Open Systems	VOSYS	Teodora Sechkova, Michele Paolino
Nextworks	NXW	Elian Kraja, Leonardo Agueci, Gino Carrozzo
Accelleran	XLRN	Antonio Garcia, Trevor Moore
I2CAT	I2CAT	August Betzler
University of Bristol	UNIVBRIS	Carlos Colman Meixner
NEC	NEC	Felipe Huici
Ubiwhere	UW	Pedro Diogo
betevé	BTV	Jordi Colom

List of Reviewers

Participant	Short Name	Contributor
Italtel	ITL	Antonino Albanese
ADLINK	ADLINK	Gabriele Baldoni

Change History

Version	Date	Partners	Description/Comments
1.1	24/04/2019	VOSYS	Added EdgeVIM Contribution
1.2	07/05/2019	I2CAT, NXW, XLRN, UW	Added RAN virtualization, VNF data models
1.3	10/05/2019	NEC	Added Unikraft contribution
1.4	15/05/2019	NEC, UNIVBRIS XLRN	Added Intro, Conclusion. Small updates on all sections Moved vEPC/vL3 to generic VNF/PNF section
1.5	20/05/2019	VOSYS	Integrated contributions from NEC, i2CAT, XLRN
2.0	23/05/2019	VOSYS	Integrated second round of contributions
2.1	24/05/2019	VOSYS	Consolidated version for internal review
2.2	27/05/2019	ITL	First review
2.3	28/05/2019	ADLINK	Second Review
3.0	30/05/2019	VOSYS	Final Version

DISCLAIMER OF WARRANTIES

This document has been prepared by 5GCity project partners as an account of work carried out within the framework of the contract no 761508.

Neither Project Coordinator, nor any signatory party of 5GCity Project Consortium Agreement, nor any person acting on behalf of any of them:

- makes any warranty or representation whatsoever, express or implied,
  - with respect to the use of any information, apparatus, method, process, or similar item disclosed in this document, including merchantability and fitness for a particular purpose, or
  - that such use does not infringe on or interfere with privately owned rights, including any party's intellectual property, or
- that this document is suitable to any particular user's circumstance; or
- assumes responsibility for any damages or other liability whatsoever (including any consequential damages, even if Project Coordinator or any representative of a signatory party of the 5GCity Project Consortium Agreement, has been advised of the possibility of such damages) resulting from your selection or use of this document or any information, apparatus, method, process, or similar item disclosed in this document.

5GCity has received funding from the European Union's Horizon 2020 research and innovation programme 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 Contents

Figures

Tables

`

Executive Summary

This deliverable describes the 5GCity's virtualization infrastructure intermediate prototype released at the end of the second Project Year. The document is organized to describe release components in a bottom up approach, starting with base technologies and moving up the stack.

First, we begin with Unikraft, a unikernel common code base and automated build system. Unikraft is able to produce extremely lean and efficient virtual machines and containers, ideal for the types of resource-constrained devices that constitute the project's platforms for edge and far edge network sections.

Next, we describe our results on RAN virtualization (i.e., in LTE and Wi-Fi), a key component to enable sharing in the 5GCity smart city infrastructure. In this deliverable we provide a full description of EdgeVIM, a set of OpenStack extensions leveraging Trusted Execution Environment (TEE) functionality to allow to remotely ensure the trustworthiness of the underlying 5GCity hardware infrastructure; in addition to Unikraft's isolation properties, EdgeVIM helps in giving third-parties and even higher degree of confidence in the project's resources.

Then, the work on modelling and packaging various types of Virtual Network Functions (VNF) and the description of their main characteristics is reported, detailing generic VNFs and Physical Network Function (PNF) in use in the project, as well as use case specific applications to be onboarded and orchestrated via 5GCity platform.

Each chapter in this document describes the role played by a particular technology in the 5GCity ecosystem as well as plans for integration and deployment of the specific technologies.

1. Introduction

This document gives an overview of the 5GCity virtualization platform and infrastructure that forms the basis for the implementation of the project's use cases and deployment. For explanatory purposes, a bottom-up approach is followed, thus describing lower-level technologies first and working our way to orchestration frameworks and integration into 5GCity dashboard. Each of the chapters has sub-sections describing how each of the technologies (a) plays a direct role in implementing the main project's concepts as well as its use cases and (b) how the technology will be deployed and integrated in order to make those use cases become a reality.

In greater detail, we begin, in chapter 2, by describing Unikraft, a unikernel build system able to produce extremely lean, efficient and strongly secure/isolated images. Unikraft supports all of 5GCity's chosen hardware architectures (x86, ARM) and virtualization/isolation technologies (KVM[1], containers[2]), and so is one of the main components of the neutral host concept. In addition, the project has, and will continue to, add and prioritize functionality needed for the project's use cases (e.g., the Click modular router[3] for NFV-oriented use cases, or potentially machine-learning frameworks for the ML-based unauthorized waste dumping prevention use case). In addition to virtualizing computation, another one of the key components of the 5GCity neutral host concept is the ability to virtualize the Radio Access Network (RAN). In other words, taking for example LTE and Wi-Fi resources and allowing for them to be added to a (virtualized) slice. Chapter 3 describes the technology and management mechanisms to make this happen in full. Chapter 4 then moves further up the functionality stack and describes EdgeVIM, a set of OpenStack extensions that add Trusted Execution Environment (TEE) capabilities to the 5GCity infrastructure, in particular leveraging ARM's TrustZone technology. Given that 5GCity is supposed to offer infrastructure to third-parties, it is important for those third-parties to be able to ensure that the code that they deploy on those platforms is the one they provided, and that no sensitive data are ever leaked, even if they can never physically verify the trustworthiness of the underlying hardware infrastructure or of the entity operating it. Such mechanism should reassure potential users of 5GCity and hopefully increase its user base. Finally, chapter 5 reports on the Network and Non-Network Function modelling and packaging which completes the set of virtualization offerings from 5GCity. In particular, a set of VNFs and PNFs is presented, which refers to the group of generic functions and/or use-case specific application functions to be deployed in the project pilots. These VNF packages are used to populate the 5GCity functions catalogue, implemented through the 5G App & Service Catalogue modules. 5GCity users can pick and choose (and deploy) the necessary functionality to run one of the six project use cases.

[1] https://www.linux-kvm.org/page/Main_Page
[2] https://www.docker.com/
[3] https://github.com/kohler/click

2. Unikraft

2.1. Overview

One key enabling technology in 5GCity, and in particular on the platform level and its neutral host concept, is the use of extremely lean and efficient unikernels. Unikernels are specialized images targeted at a specific application; for instance, in the case of a web server, there is no need to have USB drivers or a real-time scheduler, so unikernels are a way to build a system, including the operating system that is tailored to the needs of specific applications. Because of this extreme specialization, and the fact that there is a single memory address space (no user-space/kernel space division since isolation is provided by the underlying hypervisor), unikernels are able to provide impressive performance numbers, some examples are:

- 10-40 Gb/s throughput on a single CPU core
- Python-based scripting using only hundreds of KBs
- capability to run thousands of unikernels on a single commodity x86 server

While unikernels have been around for a few years now, they have never seen significant deployment due to the complexity behind actually creating them: because they are application-specific, in principle expert time must be spent to create one for each application. To address this, and given that 5GCity's platform should be, as much as possible, agnostic to the applications running over it, the project has focused on developing Unikraft.

Unikraft is an incubation project under the Xen Project[4], hosted by the Linux Foundation, focused on easing the creation of unikernels. As containers increasingly become the way cloud applications are built, there is a need to drive even more efficiency into the way these workloads run. The ultra-lightweight and small trusted compute base nature of unikernels make them ideal not only for cloud applications, but also for fields where resources may be constrained or safety is critical.

Unikraft tackles one of the fundamental downsides of unikernels: despite their clear potential, building them is often manual, time-consuming work carried out by experts. Worse, the work, or at least chunks of it, often needs to be redone for each target applications. Unikraft's goal is to provide an automated build system where non-experts can easily and quickly generate extremely efficient and secure unikernels without having to touch a single line of code. Further, Unikraft explicitly supports multiple target platforms: not only virtual machines for Xen and KVM, but also OCI[5]-compliant containers and bare metal images for various CPU architectures.

In greater detail, 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 1):

- **Library pools** contain libraries that the user of Unikraft can select 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 functionality to build the unikernel from. This last library includes drivers (both virtual such as netback/netfront and physical

[4] Please see: https://xenproject.org/
[5] Please see: https://www.opencontainers.org/

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.

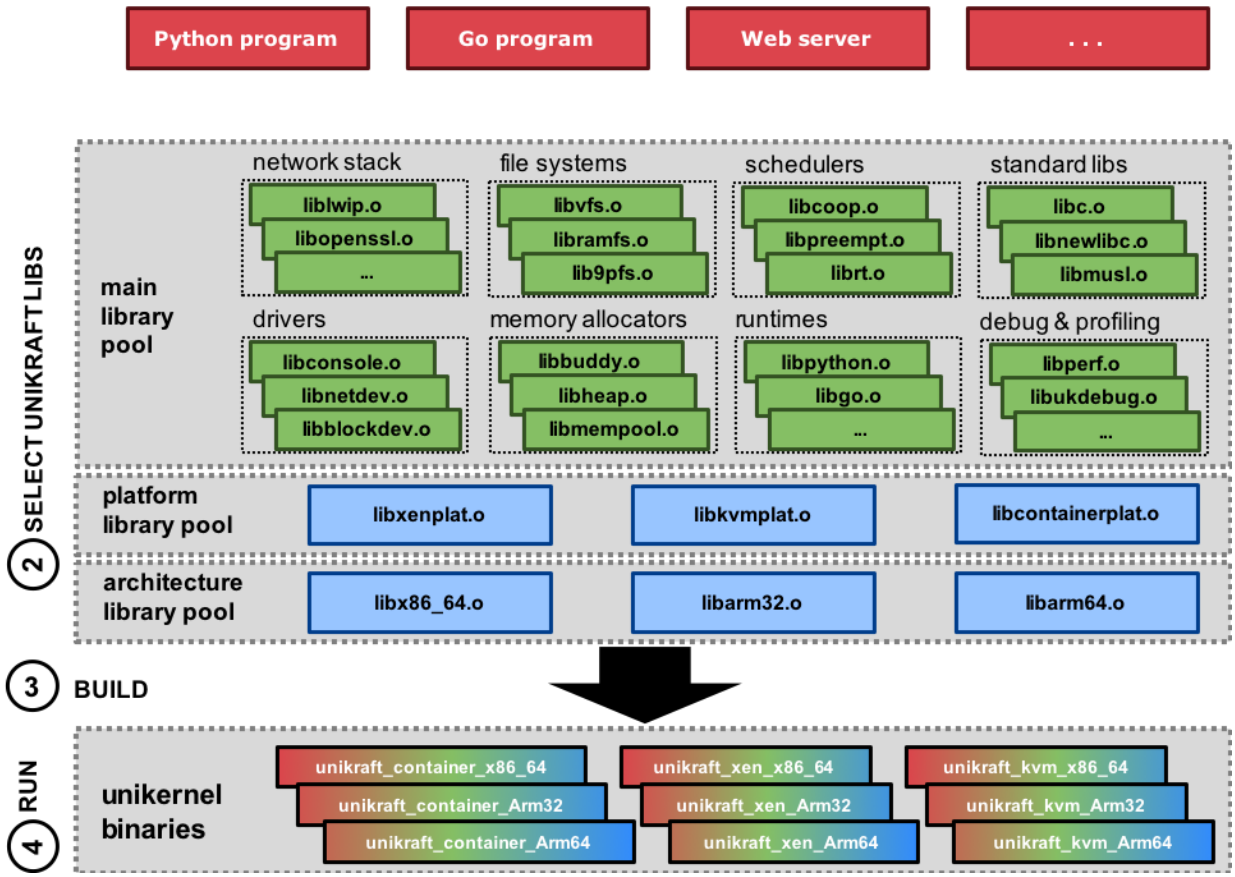

Figure 1. Unikraft micro-library architecture and workflow.

As shown in Figure 2, one critical feature missing from many other unikernel projects is Unikraft's ability to build images that can run on multiple target platforms (e.g., Xen, KVM, OCI containers) and hardware architectures (X86-64, ARM32 and ARM64 as of this writing). This means that the user of Unikraft needs to do no additional work in order to deploy their target application on a heterogeneous set of virtualization platforms. In 5GCity this feature is essential to being able to painlessly deploy application on edge devices, which are often ARM-based, as well as on more traditional, x86-based servers.

Over the last year, the lead team at NEC Laboratories Europe (along with external contributors) have made great strides in developing and testing Unikraft's base functionality, including support for a number of CPU architectures, platforms, and operating system primitives.

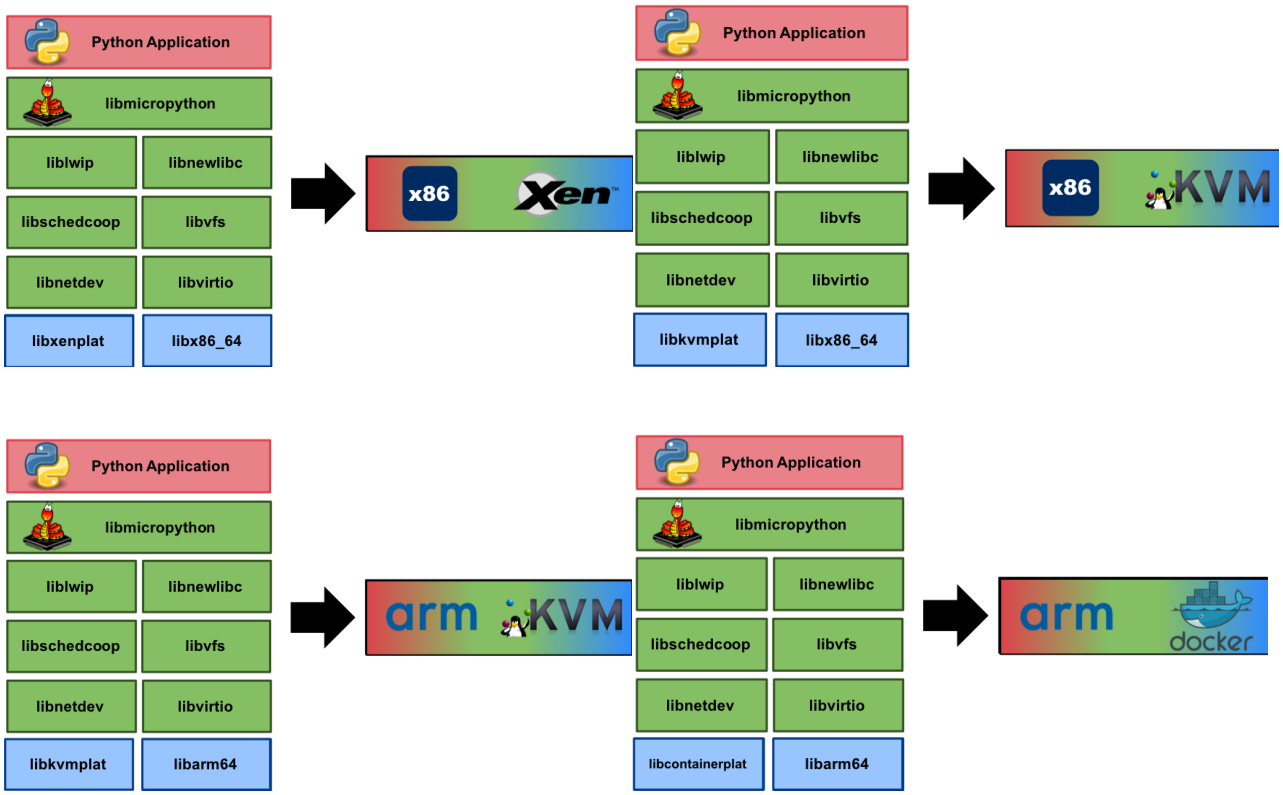

Figure 2. Unikraft can output images for different combinations of platforms and hardware architectures

In terms of roadmap, Unikraft v0.3 was released in February 2019 and contained a number of components relevant to 5GCity. This included networking support, including a lightweight TCP/IP stack, VFS support (needed for filesystems), support for ARM, and support for virtio[6] (to be able to do networking on KVM, the project's preferred virtualization platform).

Going forward, the Unikraft project will focus on automating the build process and supporting higher-layer functionality and applications:

- External standard libraries: musl, libuv, zlib, openssl, libunwind, libaxtls (TLS), etc.
- Language environments: JavaScript (v8), Python[7], Ruby[8], C++
- Frameworks: Node.js[9], PyTorch[10], Intel DPDK[11]
- Applications: Click modular router, lighttpd, NGINX, SQLite[12], Redis[13], etc.

Again, the majority of the items above are needed to support 5GCity scenarios (e.g., Python in order to support Smart City/IoT use cases). In addition, the coming period will further focus on benchmarking Unikraft to quantify the performance benefits it can provide.

The current, upstreamed code of Unikraft can be found at:
https://xenproject.org/developers/teams/unikraft/ .

[6] Please see: https://wiki.libvirt.org/page/Virtio
[7] Please see: https://www.python.org/
[8] Details at: https://www.ruby-lang.org/
[9] Please see: https://nodejs.org/
[10] Please see: https://pytorch.org/
[11] Please see: https://www.dpdk.org/
[12] Details at: https://www.sqlite.org/index.html
[13] Please see: https://redis.io/

2.2. Role in 5GCity

Unikraft generates lightweight, efficient, secure and isolated images that form one of the main components of the project's neutral host concept: providing platforms, often running on resource-constrained hardware, that are virtualized and multi-tenant capable. To make this feasible, and given the limited resources of the underlying hardware, Unikraft's unikernels enable the ability to efficiently run a relatively large number of instances, much larger than what would be possible with conventional virtualization technologies. For instance, a standard KVM virtual machine running Ubuntu distribution might be 100MBs or even GBs in size, whereas a Unikraft-built (Micro)python KVM VM (Python being ideal for a large number of IoT/Smart City use cases) may consist of only a few MBs.

Likewise, Unikraft can also generate extremely lean, OCI-compliant containers; in this case, Unikraft outputs Micropython containers that are only hundreds of KBs in size, multiple orders of magnitude when compared to standard virtual machine or even standard Docker containers.

With this in place, the "only" remaining task is to make sure Unikraft supports functionality relevant to 5GCity. To this end, we have been working on the following components:

- **KVM support**: this is the project's preferred virtualization technology
- **Container support**: in order to be able to run Unikraft images on hardware platforms at the very edge of a deployment. Such devices may not support hardware virtualization extensions (i.e., we could not run KVM/virtual machines), so container support provides a nice fall-back that still gives isolation between the different 5GCity tenants.
- **X86 and ARM support**: all edge (and server) devices envisioned by the project have either an x86 or ARM processor.
- **newlib/musl**: these are leaner/more modern versions of the standard libc/glibc, needed by practically all applications.
- **Networking support, including the lwIP (lightweight IP) network stack**: Practically all 5GCity applications need communications via a TCP/IP network stack.
- **Block and filesystem support**: Practically all 5GCity applications need a filesystem and block support to function.
- **Click modular router**: a mature, open source, modular packet processing software from MIT[14] that makes it easy to implement an extremely large array of different (virtualized) network functions. For instance, creating a load balancer would be as simple as providing a Click configuration file (a text file) specifying the modules and connections needed to create a load balancer VNF. Each individual Unikraft-built Click instance can run a different NVF based on different configuration files.
- **Python support**: The language of choice for IoT and Smart City scenarios, also needed by other functionality listed below.
- **PyTorch support:** Perhaps the most utilized machine-learning framework, we are envisioning to use it to implement the ML-based detection algorithm for illegal waste dumping (i.e., 5GCity use case 1).

2.3. Deployment and integration in 5GCity

In terms of deployment, Unikraft will act as one of the base technologies for implementing the neutral host concept and use case. As mentioned, Unikraft supports ARM, x86, KVM and containers, so its generated images should run seamlessly on the various devices envisioned by the project for deployment. In terms of

[14] Massachusetts Institute of Technology

orchestration, some level of work might be needed to be able to boot a Unikraft image using existing frameworks (e.g., OpenStack, which might need some minor work to support direct kernel booting).

In terms of other use cases and functionality, we are in the process of deploying a GPU-capable server in the city of Lucca as part of the implementation and deployment of the illegal waste detection use case. The actual automated detection algorithm is based on machine learning, and more specifically neural networks. Along these lines, we are currently in the process of adding PyTorch support to Unikraft, since this framework is not only extremely popular but also supports neural networks. The idea here is to be able to run the illegal dump detection algorithm within a Unikraft unikernel running in a 5GCity, city of Lucca device. Towards the implementation of NFV-oriented use cases, we are close to finishing the port of the Click modular router software. Click is a modular packet processing framework, where each of the modules implements a small packet processing function, for example decreasing the TTL. Because of this, it is really simple to implement a huge array of network functions by simply writing a configuration file that defines which modules to use, and how packets should flow between them. The project is working towards providing the ability to run Click within virtualized, well isolated and extremely efficient Unikraft images running on 5GCity devices.

Finally, we are further looking into the possibility of supporting the remaining, and potentially other, future, 5GCity use cases. Porting all of the necessary libraries and software to Unikraft represents a significant amount of effort that might not be feasible within the lifetime of the project (and sometimes code might be proprietary, so sources may not be available at all). To address this, we are looking into binary re-writing techniques, where – essentially – we take a compiled binary (built for instance using a standard Linux environment) and re-write the calls to the Linux ABI with calls to Unikraft functions. These additional levels of indirection would result in somewhat degraded performance, but would significantly aid in the implementation of use cases. As initial steps towards this, we are currently in the process of adding ELF parsing (ELF being the native Linux binary format) and processing support to Unikraft in order to be able to handle the binaries we are building in a Linux environment.

3. RAN virtualization

3.1. Overview

5GCity extends the concept of slicing to the RAN, i.e. LTE and Wi-Fi resources are virtualized, and they can be added to a slice, similarly to the way it is done with compute resources. A slice that includes any of the two radio resources provides radio access connectivity for users. In LTE, isolation of slices is achieved by emitting a dedicated PLMNID on the small cells and associating it to a dedicated EPC. The EPC can be provisioned in different ways: it can be a vEPC instance hosted by 5GCity, as part of the deployment of a service, or it can be a remote EPC that is not controlled by 5GCity. This choice is made by the tenant during the setup of a slice, allowing him to pick the model that seems better suited. In Wi-Fi, each slice is represented by a set of virtual Access Points (vAPs) that are instantiated on each physical radio interface that belongs to the slice. Each group of vAPs associated to a slice is configured separately, allowing the tenant to use different SSIDs and security setting for each of them. Additionally, in Wi-Fi, each slice is assigned an airtime ratio, i.e. a percentage of the overall available airtime.

The instantiation, management and control of the radio interfaces in 5GCity is handled by the RAN controller. Like the Slice Manager talks to VIMs to manage compute resources, to manage radio resources it interacts with the RAN controller. Since in 5GCity a variety of radio resources are virtualized, potentially there can be multiple instances of RAN controllers capable to slice LTE and Wi-Fi radio resources, or even other technologies, if supported. Each RAN controller instance is packed in a VM that needs to be instantiated as part of a 5GCity platform deployment.

In the following section, we introduce the RAN controller used in 5GCity by describing its integration within the 5GCity ecosystem.

3.2. Role in 5GCity

Each 5GCity platform deployment requires specific software elements to support RAN slicing that can be split into two categories:

1. the software running on the radio equipment that enables its remote management and configuration
2. one or more RAN controller instances that translate any RAN slice creation or configuration request coming from the 5GCity slice manager to a protocol that is understood by the radio devices.

In the following subsections, we will describe the role of each component in detail.

3.2.1. 5GCity RAN Controller

The RAN controller is the entity responsible for managing the RAN devices, specially Accelleran Small Cells and i2CAT Wi-Fi nodes. Like other platform components, the RAN controller requires a single instance to be hosted in the 5GCity infrastructure. Like the VIM(s), the RAN controller sits underneath the Slice Manager, while the RAN devices, like the NFVI, sit below the RAN controller.

The RAN controller is composed by a set of services that provide control over the RAN infrastructure (see 3.3.1). All components are packed into a single VM that can be managed by OpenStack. A REST API [7] is used

for the communication between the slice manager and the RAN controller. The protocols used by the i2CAT RAN controller to talk to the radio devices are NETCONF, OvSDB[15] and OpenFlow[16] (OF).

3.2.2. Info about the LTE software components

The software components deliver an LTE-based logical RAN architecture for 5GCity (Figure 3) based on enhancements in the area of RAN functional disaggregation, RAN and Network Slicing with SDN control and RAN function virtualization.

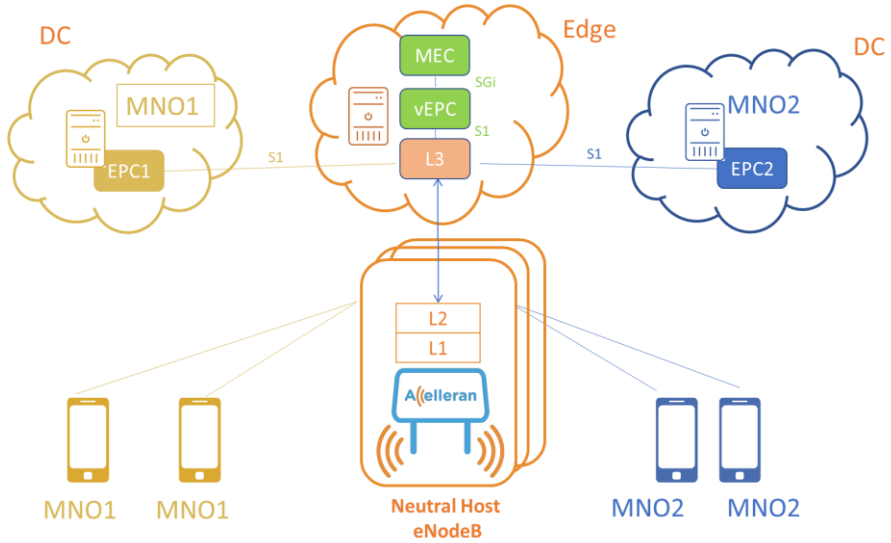

Figure 3. 5GCity RAN network architecture overview

Network functions of the delivered software components 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 VNF which runs in the Edge NFVI. 5GCity L3 supports network slicing and connection to multiple EPC instances (one per slice).
- **vEPC**: EPC 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.

The virtualised L3 control plane supports network slicing based on connectivity to several vEPC-s (whether local at the edge or remote in the data centers of the participating MNOs) as per 3GPP Multiple Operator Core Network and the configuration and management of the Small Cell components from the RAN Controller via a Netconf Server in each of the vL3 instances managing their respective Physical Small Cells embedding L2/L1 functionality.

[15] Please see: http://docs.openvswitch.org/en/latest/ref/ovsdb.7/
[16] Details at: http://www.opennetworking.org/wp-content/uploads/2013/05/TR-535_ONF_SDN_Evolution.pdf

For simplicity and flexibility of the automatic small cell discovery logic, the virtualised L3 functionality is delivered as a single VNF that contains:

- Kubernetes[17] for the lifecycle management of L3 Docker containers
- REDIS and NATS[18] logic for the automatic discovery of the Physical Small Cells added to the network
- the image of the L3 Docker container for which an instance will be spawned to control each Physical Small Cell (L2/L1) added into the system.

The L3 Docker container contains all needed L3 functionality and includes a NETCONF Server instance that will be used to control the respective Physical Small Cell from the RAN Controller NETCONF Client.

The VNF could itself be managed via OpenStack if delivered as a KVM image, however, since the per-Small Cell L3 instantiation and discovery is already organised within the VNF via internal Kubernetes and REDIS/NATS logic not visible outside of the VNF, it is deemed a better approach to deploy the virtualised L3 as a normal VM.

The following Figure 4 shows the details of the virtualized L3 Small Cell VNF approach:

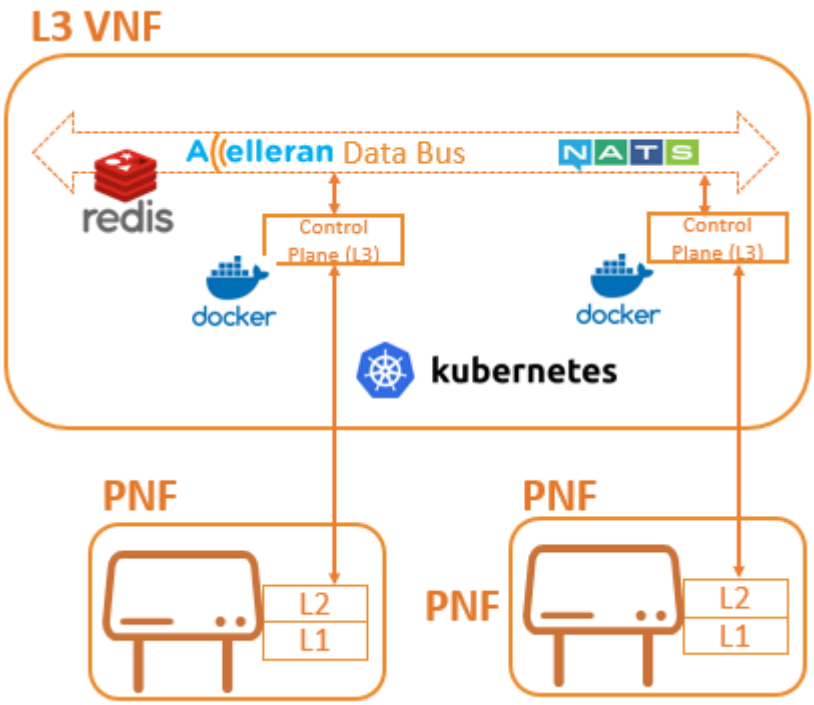

Figure 4. Accelleran virtualized L3 overview

Management of L3 and Core VNFs

The L3 VNF provide API's towards the Edge NFVI SDN controller for slice management and also provide interfaces for classic FCAPS [6] management of the static configuration (e.g. operating frequency, global cell parameters, etc) which are configured on installation.

- **eNB FCAPS**: General RAN FCAPS management is supported in virtualised L3 via NETCONF and CLI. One of these mechanisms is required to initially configure each cell for service in the network.

[17] Please see: https://kubernetes.io/
[18] Please see. https://nats.io/

- **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 data model which is formally defined in YANG[19]. 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. 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 VM and provides WebGUI and/or CLI based configuration interfaces, however the vEPC is not controlled via NETCONF. 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.

3.2.3. Info about i2CAT Wi-Fi node software components

The i2CAT custom built Wi-Fi nodes are Single-Board Computers equipped with two wireless interfaces (ath9k[20], ath10k[21]) that run a series of services that allow their remote management and configuration. The two main elements are a Netopeer[22] server that enables the devices to be configured remotely via NETCONF and the OpenvSwitch[23] environment. Latter is set up at initialization time, setting up the configuration plane connectivity to the RAN controller and configuring the internal virtual switches required for data and management plane data steering.

After this bootstrapping process, the virtual switch environment is completely managed by the RAN controller, using the OpenFlow and OvSDB protocols to configure the nodes.

When new slices are created, a combination of NETCONF, OpenFlow and OvSDB are used to:

- Set up new virtual access points on the nodes
- Set up new bridges in the OvS environment
- Stitch the virtual access points and the bridge to the already existing virtual switch structure
- Configure the bridges with the rules to set up the data plane, so that any traffic from the slice will be correctly handed over from/to the virtual access points (using the service VLAN ID).

This process is executed for every node on which the slice has requested presence and as an outcome all the virtual access points are interconnected over the same service VLAN ID that also connects to any VNF being hosted in the edge or main Data Center.

3.3. Deployment and integration in 5GCity

3.3.1. I2CAT RAN Controller

I2CAT's RAN controller is composed by a set of services that provide control over the RAN infrastructure. These services include but are not limited to:

[19] Please see: https://tools.ietf.org/html/rfc6020
[20] Please see: https://wireless.wiki.kernel.org/en/users/Drivers/ath9k
[21] Please see: https://wireless.wiki.kernel.org/en/users/drivers/ath10k
[22] Please see: https://github.com/CESNET/netopeer/wiki/NetopeerServer
[23] Please see: https://www.openvswitch.org/

- An instance of OpenDaylight Boron[24] (SR3)
- A java application called NETCONF[25]-manager based on Spring Boot[26]
- Monitoring tools such as Prometheus[27]

These services are hosted within one single VM. The compute resource requirements mainly come from the OpenDaylight instance, which requires to deploy the RAN controller in an environment that guarantees the following:

- 16 GB of RAM
- 8 vCPU
- 50 GB hard disc.

If installed in an OpenStack environment, the following ports have to be unfiltered on the RAN controller VMs security rules:

- TCP:22 for SSH access
- TCP:830 for NETCONF port
- TCP:3000 for Monitoring
- TCP:8000 for NETCONF RAN REST component
- TCP:8008 for RAN Chunk administration REST component
- TCP:8080 for RAN Topology component
- TCP:8181 for RAN OpenDayLight component
- TCP:9696 for RAN Topology administration

3.3.2. I2CAT Wi-Fi nodes

Wi-Fi nodes provided by i2CAT are standalone solutions that come with the required hardware and software components. Nevertheless, we want to point out that RAN controller makes use of OpenFlow, NETCONF and OvSDB protocols to administer the i2CAT-WiFi devices meaning that it requires a very specific software deployment.

OpenFlow and OvSDB dependencies can be met by installing OpenvSwitch. For compatibility, I2CAT's RAN controller does not require any specific version to run on the Wi-Fi devices, but version 2.7.9 or greater are used in the Wi-Fi nodes, since 802.1q Q-in-Q feature was introduced on that version.

After the installation process a virtual switch (bridge) must be created on the device and the control path to the controller must be set using

```
`ovs-vsctl BRIDGE_NAME set-controller ptcp:CONTROLLER_IP:OPENFLOW_PORT`
```

to cover the OpenFlow dependency and set its manager using

```
`ovs-vsctl set-manager ptcp:CONTROLLER_IP:OVSDB_PORT`
```

to cover the OvSDB dependency.

[24] Please see: https://www.opendaylight.org/what-we-do/current-release/boron
[25] Please see: https://tools.ietf.org/html/rfc6241
[26] Details at: https://spring.io/projects/spring-boot
[27] Please see: https://prometheus.io/

Moving to the NETCONF dependency, Netopeer server is being used as NETCONF toolset for the I2CAT's managed devices. Note that Netopeer depends on various libraries, such as pyang[28], libnetconf[29] and libssh[30]; which have to also be installed on the device. The models implemented have been tested and deployed on NetopeerV1, which was marked as deprecated since June 2017 but the models are not V2 ready yet.

The supported models have been provided by i2CAT and Accelleran and it is currently a proprietary software.

3.3.3. Accelleran LTE nodes

The LTE nodes are composed of a virtualised L3 Control Plane component running at the edge server and implementing the typical Small Cell VNF functions and the physical small cells implementing the L2/L1 Small Cell PNF functions (see Section 3.2.2).

3.3.3.1. vL3 Control Plane

The virtualised L3 functionality is delivered as a single VNF which contains Kubernetes for the lifecycle management of L3 Docker containers, REDIS and NATS logic for the automatic discovery of the Physical Small Cells added to the network, and the image of the L3 Docker container for which an instance will be spawned to control each Physical Small Cell (L2/L1) that is added into the system.

The delivered VNF is a qcow2[31] image, which requires a single ethernet interface on the 10.10.201.0/24 subnet. It is configured with a static IP address of 10.10.201.15.

SSH access is available on port 22. The username is "ad", and the password is provided by Accelleran.

Netconf configuration

Each vL3 container includes a Netconf Server for remote configuration by the RAN Controller Netconf Client. However, as the vL3 containers are in a private IP subnet, their Netconf servers cannot be directly reached on that internal IP address. This is solved by mapping a unique port on the VNF's external IP address to each vL3 instance. The port number is calculated using a unique identifier, called Instance ID, which is shared between each vL3 and Physical Small Cell pair. The Instance ID is set on the Physical Small Cell and is of the form "5gcity-XX", where XX is replaced by a unique (across the network of Small Cells) two-digit number. The Netconf Server for this vL3 is then available on port 305XX.

3.3.3.2. Physical Small Cell (L2/L1)

The L2/L1 PNF functions of the Small Cell are delivered as an Accelleran update file, which can be applied using the WebGUI interface of the Small Cell. In order to access the WebGUI interface (Figure 5) of the Small Cell, access http://XX.XX.XX.XX where XX.XX.XX.XX is the IP address of the Small Cell management interface. For further information see the Small Cell user guide.

Once updated, the installer must check that the vL3/Small Cell pair identifier is configured. Connect to the Small Cell via SSH on port 22, and update /mnt/app/bootstrap:

```
redis.hostname:10.10.201.15
redis.port:32000
instance.filter:5gcity-XX
```

[28] See: https://github.com/mbj4668/pyang
[29] See: https://github.com/CESNET/libnetconf
[30] See: https://git.libssh.org/projects/libssh.git/
[31] See: https://git.qemu.org/?p=qemu.git;a=blob;f=docs/interop/qcow2.txt

The instance.filter field is the same Instance ID described in section 3.2.3.1, and should be unique across the network of Physical Small Cells.

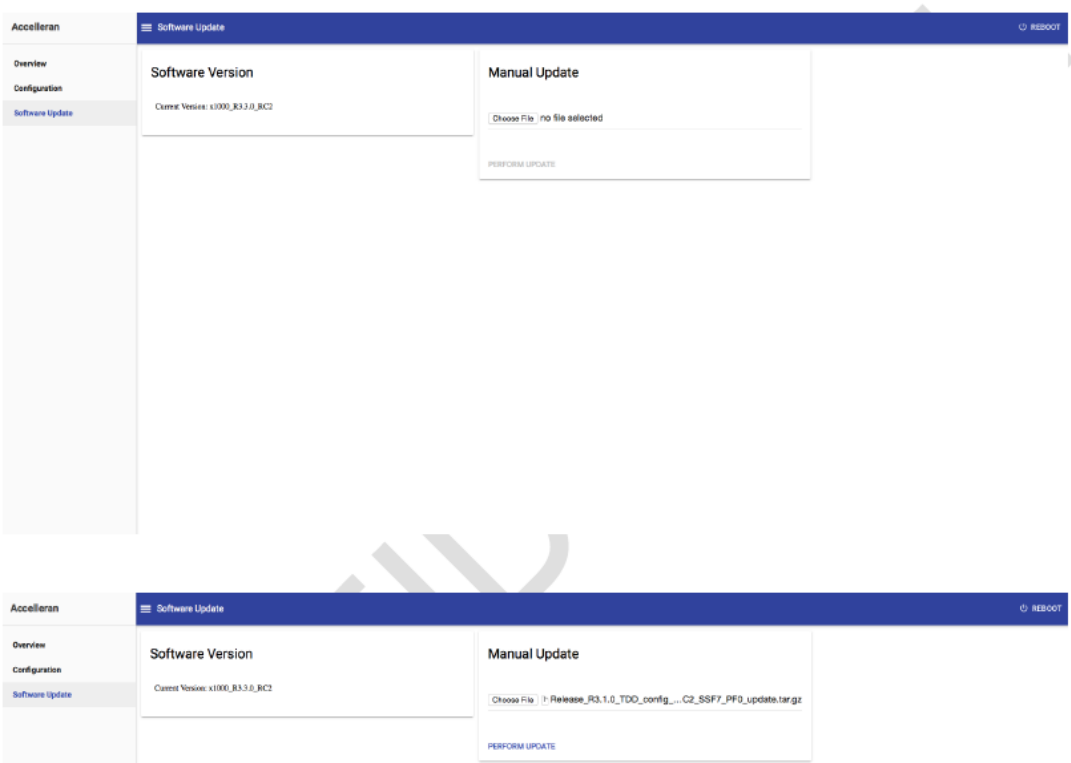

Figure 5. The Small Cell WebGUI interface

4. EdgeVIM

4.1. Overview

EdgeVIM is a collection of OpenStack extensions developed during the 5GCity project, adding trust into the edge infrastructure by leveraging ARM TrustZone technology. Its goal is to harden and protect the edge compute infrastructure and to integrate the security checks into the existing cloud and edge managing tools.

- The EdgeVIM consists of two parts: extensions of OpenStack and Trusted applications (see Figure 6). The OpenStack extensions consist of Attestation Filter added to the Open Stack Compute (Nova[32]) scheduler and an attestation agent running on each compute node. Trusted apps are implemented inside a Trusted Execution Environment provided by ARM TrustZone and OP-TEE[33]. They perform node authentication, kernel integrity verification and geo-fencing.

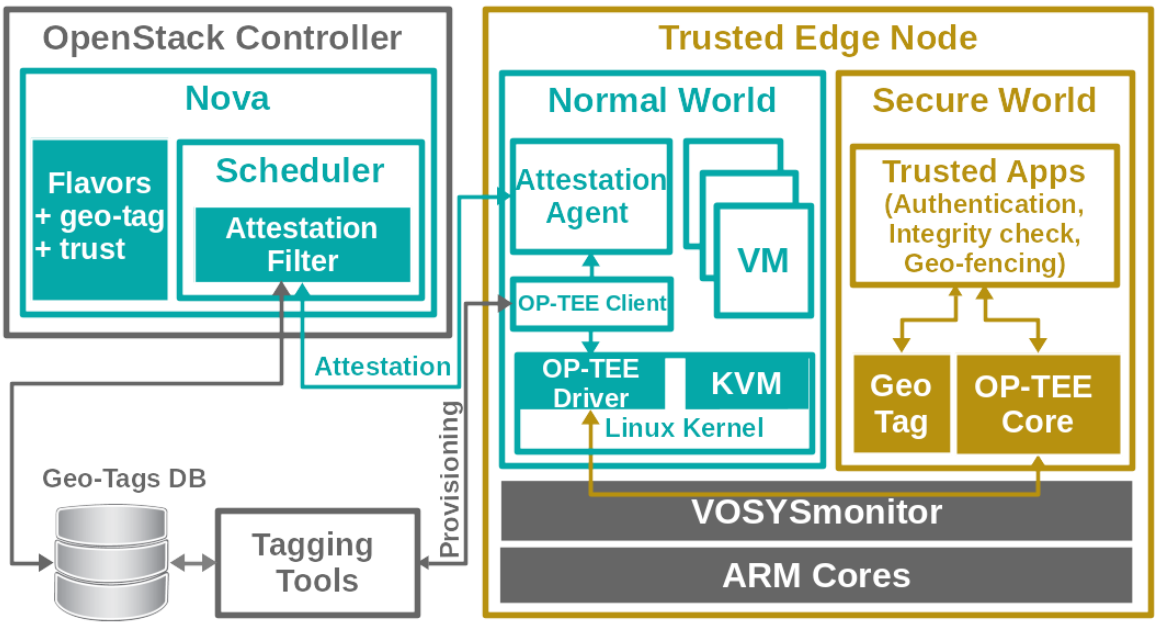

Figure 6. EdgeVIM overview

EdgeVIM is developed as a proprietary solution and the source code is not shared openly. Information about it can be found on the company's website:

http://www.virtualopensystems.com/en/products/vosystrustedvim/

4.2. Role in 5GCity

The edge computing infrastructure of the smart cities is remote, dispersed and vulnerable to man-in-the-middle and device tampering attacks. Additional measures surpassing the standard cloud security practices have to be taken to trust the edge devices. Furthermore, workloads and services can carry restrictions of the

[32] Detail at: https://github.com/openstack/nova
[33] Please see: https://www.op-tee.org/

geographic location where they can be placed. Complying with such requirements is nowadays important for a Neutral Host platform.

In the 5GCity architecture, EdgeVIM plays the role of a VIM managing the resources of the EdgeNFVI. It reuses the capabilities of OpenStack and enhances them with security and location-awareness. Moreover, it adds trust into the EdgeNFVI by relying on ARM edge nodes with TrustZone.

4.2.1. Attestation

EdgeVIM provides trust into the edge compute infrastructure on top of which the VNFs will be deployed. Since the VNFs in 5GCity are deployed as VMs, the role of the EdgeVIM is attesting each edge node and verifying its identity and integrity before the VM is deployed on it. In case none of the nodes passes the attestation, a security policy should exist, stating whether the security requirements can be lowered or the VM placement should be aborted. In the case of successful VM deployment, the EdgeVIM is able to check periodically the kernel integrity in order to detect signs of malicious software. If such are detected the VM can be migrated to another trusted compute node.

4.2.2. Location-awareness

EdgeNFVI provides a trusted environment to securely store location information inside each compute node. Any 5GCity service or workload (VM) can be tagged with location requirements and EdgeVIM attestation functionality verifies that the edge node matches them. Similar to the security case, a location policy is needed to state the strictness of the conditions.

The security and location-awareness are orthogonal and applicable to all 5GCity use cases. The Neutral Host, as well as the media and surveillance verticals benefit from the added functionality.

4.3. Deployment and integration in 5GCity

EdgeVIM can be deployed as a stand-alone VIM or as an extension of an existing OpenStack deployment.

4.3.1. Integration into an existing OpenStack deployment

In order to integrate EdgeVIM into an existing OpenStack deployment, two main steps need to be performed:

1. Add one additional “Attestation” filter to the Nova Scheduler in the existing deployment. The filter implementation is provided as part of the EdgeVIM. The configuration remains the same as described in the OpenStack documentation:
    a. https://docs.openstack.org/ocata/config-reference/compute/schedulers.html
2. Trusted Environment installation on the ARM compute nodes. The low-level installation instructions and proprietary binary files are provided by VOSYS as part of the EdgeVIM. The supported hardware platform is Xilinx Zynq MPSoC ZCU102[34].

4.3.2. Stand-alone deployment

When deployed as a stand-alone VIM a standard OpenStack deployment is performed. The attestation filter is part of the OpenStack controller. The installation of Trusted Environment on the ARM compute nodes remains the same as in the previous case.

In order to keep the compatibility with existing orchestrators, the EdgeVIM Northbound API reuses the one of OpenStack.

[34] Documentation available at: https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html

5. VNFs Data models

Virtual Network Functions in 5GCity are stored on the 5G App & Service Catalogue and are differentiated in two categories based on the implemented functionality:

- Type 1 VNFs provide pure basic networking functions, e.g. firewalling, routing, NAT and DNS, etc. These are generic functions that can be used across different use cases.
- Type 2 VNFs provide specific use case functions and are related to the application workflow of the scenario to be implemented. For example, a media controller VNF is used for the video acquisition and production use case.

This section is organized to reflect this distinction and reports key summary information on the VNFs developed for Type 1 and Type 2 to implement the various 5GCity use cases (UCs).

5.1. Generic VNFs and PNFs

The list of generic VNFs is reported in Table 1. These VNFs can be also used as Physical Network Functions in case it is possible to share the functionality across various network services (e.g. DNS for different network services and/or slices).

The Accelleran vEPC and vRAN (L3) VNFs are needed to support all the LTE RAN use cases, including the Neutral Host or any other Use Cases based on LTE RAN nodes with Neutral Host support.

VNF Name	Package details
vFirewall	**Description:** Security Front/Back End VNFs to protect users and service providers data **VNF name:** vFirewall • **Memory:** 512 MB • **CPU:** 1 vCPU • **Storage:** 3 GB • **Image:** vFirewall-v3.qcow2 • **Format:** QCOW2 **Provider:** NXW
vDNS	**Description:** Associate domain names to VNFs to make more user friendly the interactions among UEs and VNFs **VNF name:** vDNS • **Memory:** 512 MB • **CPU:** 1 vCPU • **Storage:** 3 GB • **Image:** vDNS.qcow2 • **Format:** QCOW2 **Provider:** NXW
vLB	**Description:** Balancing the requests from the UEs to the services running on the VNFs **VNF name:** vLB • **Memory:** 512 MB • **CPU:** 1 vCPU • **Storage:** 3 GB

	• **Image:** vLoadBalancer.qcow2 • **Format:** QCOW2 **Provider:** NXW
vEPC	**Description:** EPC core functions instantiated and running at the Edge Server. **VNF name:** vEPC • **Memory**: 8192MB • **CPU:** 4 vCPU (i5 performance or better for 1Gbps of traffic) • **Storage:** 5.5 GB (min for Desktop Linux, although the vEPC only requires 0.1 GB) • **Image:** vpec-snapshot-atto-0.raw • **Format:** QCOW2 **Provider:** Accelleran
vL3	**Description:** Virtual L3 Small Cell (including Netconf) and ancillary virtualisation and discovery functions (Kubernetes, REDIS, NATS). **VNF name:** vL3 • **Memory:** 4096 MB • **CPU:** 4 vCPU • **Storage:** 16 GB (includes Kubernetes, REDIS, NATS and 1GB L3 image). • **Image:** 5GCity.qcow2 • **Format:** QCOW2 **Provider:** Accelleran

Table 1: List generic VNFs and PNFs

5.1.1. vFirewall

The vFirewall VNF is based on VyOS [1], an open source GNU/Linux-based operating system extended with network routing and firewall software suitable for being deployed in VNFs in the form of Virtual Machines (VMs). The vFirewall provides several network functionalities:

- Routing
  - Static routing: manually configured routing entries. They do not change on network changes or reconfigurations
  - Dynamic routing: the routing table is adapted on the conditions of the network.
- NAT
  - Source NAT: allow internal users to access internet
  - Destination NAT: allow access to internal users from the internet through port forwarding
- Firewall:
  - Stateful firewall: the firewall is able to keep track of the active connections and only packets matching those active connections are allowed to pass it.
  - Zone-based firewall: different networks can be associated to different security zones to control the traffic among them
- VPN
  - OpenVPN[35]
  - Site to site IPSec
  - L2TP/IPsec and PPTP remote access VPN

[35] Please see: https://openvpn.net/

- VTI (Virtual Tunnel Interfaces)

The vFirewall offers two different configuration tools. The CLI can be used for actuating configurations directly on the VNF, or a bash script may run the configurations, giving the opportunity to the operator to run remote configurations. The changes are saved on the running configuration and persistency is guaranteed only by committing the changes on the start-up configuration. After each commit, the previous configuration is archived and versioned.

5.1.2. vDNS

The vDNS VNF is based on BIND9 [2], based on Linux-based operating system. It is able to play both DNS roles:

- Authoritative name server on one or more specific domains
- Non-authoritative name server, acting as a recursive resolver

The vDNS is configurable via CLI, applying the modifications to the configuration files or via REST API.

5.1.3. vLB

The vLB is a load balancer, based on HA-Proxy [3]. It runs on a GNU/Linux-based operating system, providing high availability load balancer for TCP and HTTP-based applications. It can be used as a:

- TCP proxy
- HTTP reverse-proxy
- Load balancer
- Traffic regulator

The configuration of the load balancer is done by triggering REST API on the vLB VNF.

5.1.4. vEPC

This implements the standard EPC core functions instantiated and running at the Edge Server.

5.1.5. vL3

This implements the virtual L3 Small Cell (including Netconf) functions and ancillary virtualisation and discovery functions (Kubernetes, REDIS, NATS).

5.2. Use Case specific VNFs and PNFs

5.2.1. UC1: Unauthorized Waste Dumping Prevention

Table 2 reports the list of Network Functions related to the UC1: Unauthorized Waste Dumping Prevention. It consists of two Network Services:

1. The purpose of the first Network Service, *Edge Video Processing*, is to identify a possible infringement and send it to a Backend Service (PNF) for further analysis. This Network Service consists of two VNFs: a vFirewall and a Garbage Infringement Recognition system.

2. The purpose of the second Network Service, *Edge Notification*, is to notify the infringement to the Police. It allows Police Officers to view the infringement and react immediately. This Network Service consists of two VNFs: a vFirewall and a miniCache.

VNF Name	Package details
Garbage Infringement Recognition	**Description:** Performs the recognition of a possible infringement and sends it to the Backend Service **VNF name:** GarbageInfringementRecognition • **Memory:** 2 GB • **CPU:** 2 vCPU • **Storage:** 20 GB • **Image:** garbage-infringement-v01.qcow2 • **Format:** QCOW2 **Provider:** NXW
miniCache	**Description:** Permits to Police Officers to view infringement photos **VNF name:** miniCache • **Memory:** 64 MB • **CPU:** 1 vCPUs • **Storage:** 0 GB • **Image:** 5GCity-minicache-v3.qcow2 • **Format:** Unikernel **Provider**: NXW
Backend Service	**Description:** Collects the infringements and notifies police officers when new infringements are present. **PNF name:** BackendService • **Memory:** 2 GB • **CPU:** 2 vCPUs • **Storage:** 10 GB • **Image:** backend_service.qcow2 • **Format:** QCOW2 **Provider:** NXW

Table 2. List of UC1 specific VNFs and PNFs

5.2.1.1. Garbage Infringement Recognition

The Garbage Infringement Recognition VNF is based on OpenCV [4], an open source computer vision and machine learning software library. The library has several optimized algorithms that includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects etc. The software of this Network Function is written in Python and has two main features:

1. identify possible infringements
2. send the possible infringements to the Backend Service for further analysis.

It acquires a video stream from an IP camera and tries to detect when a pedestrian is in the proximity of the garbage bins. When the pedestrian is detected on the detection window, it takes a snapshot of the initial state on the zone of interest and starts storing frames with a configurable frequency. When the person is no longer detected on the detection window, it stops storing frames and takes a snapshot of the final state on the zone of interest. At this point, the VNF computes the differences between initial and final state and if there are differences, it concludes that there has been an infringement. If the infringement is detected, it sends the stored frames to the Backend Service, otherwise the stored data is deleted.

5.2.1.2. miniCache

The minicache VNF is a unikernel which includes an http server that permits the police officers to view the images of the infringement. The http server gets the data from the Backend Service and store them until the deletion of the infringement from an operator. The usage of unikernels decrease the instantiation time of the service, permitting the Municipality to have an immediate reaction on committed offense.

5.2.1.3. Backend Service

The Backend Service VNF is built on top of a GNU/Linux-based operating system. The main purpose of the Backend Service is storing the infringements sent by the Garbage Infringement Recognition VNF. The service is written in Python and offers a REST API to interact with the Garbage Infringement Recognition VNF.

The features of the Backend Service are the following:

- Store infringement data: Backend Service receives two requests from Garbage Infringement Recognition VNF
    - POST infringement: a list of metadata is received to identify the infringement and the location where it happened.
    - PUT infringement data: A zipped file containing the images of the infringement is received. File is unzipped and stored in the filesystem for presentation at GUI.
- Validation of the infringement: an operator checks the list of infringements and validates them manually.
- Notification to the police officers: the police officers will receive via email the infringement details and an URL to the infringement images on their smartphones.

5.2.2. UC2: Neutral Host

This UC2 is the underlying use case on which the rest of the use cases rely and therefore do not have any use case specific VNF. The Accelleran vEPC and vL3 VNFs needed to support the LTE RAN use cases, including the Neutral Host or any other Use Cases based on LTE RAN nodes with Neutral Host support are included in section 5.1.

5.2.3. UC3: Video Acquisition and Community media engagement in live events

Table 3 reports the list of Network Functions related to the UC3: Video Acquisition and Community media engagement in live events.

A first Video Processing Network Service is configured for aggregation and transmission of the video acquired. This Network Service consists of two VNFs: a media controller and a media switcher.

A second Video Acquisition Network Service is configured to acquire and transcode the video from the users. This Network Service consists of two VNFs: one vFirewall and a list of media transcoders.

VNF Name	Package details
Media Controller	**Description:** Media Controller, responsible for all the business logic and to feed a GUI to the users and for the video director. **VNF name:** MediaController • **Memory:** 2 GB • **CPU:** 2 vCPU • **Storage:** 2 GB • **Image:** mediacontroller-v01.qcow2 • **Format:** QCOW2

	Provider: MOG
Media Switcher	**Description:** Media engine capable of receive multiples RTMP streams simultaneously and choose one to be selected as output. **VNF name:** MediaSwitcher • **Memory:** 4 GB • **CPU:** 2 vCPUs • **Storage:** 2 GB • **Image:** mediaswitcher-v1.qcow2 • **Format:** QCOW2 **Provider**: MOG
Media Transcoder	**Description:** Media engine for transcode WebRTC streams to RTMP stream with H.264 video codec and AAC audio coding. **VNF name:** MediaTranscoder • **Memory:** 2 GB • **CPU:** 2 vCPUs • **Storage:** 2 GB • **Image:** backend_service.qcow2 • **Format:** QCOW2 **Provider:** MOG

Table 3. List of UC2 specific VNFs and PNFs

5.2.3.1. Media Controller

Media Controller, responsible for all the business logic and to feed a GUI to the users and for the video director. Also, responsible to orchestrate all the media flows.

5.2.3.2. Media Switcher

Media engine capable of receive multiples RTMP streams simultaneously and choose one to be selected as output. Also, can provide four monitoring streams to be used for control purposes and feed the video switcher GUI.

5.2.3.3. Media Transcoder

Media engine for transcode WebRTC streams to RTMP stream with H.264 video codec and AAC audio coding.

5.2.4. UC4: UHD Video Distribution – Immersive Services

Table 4 reports the list of Network Functions related to the UC4: UHD Video Distribution – Immersive Services.

The purpose of the first Network Service, Video Distribution, is the distribution of UHD video content. This Network Service consists of two VNFs and two PNFs. The two VNFs are a load balancer and the RAI ON-DEMAND VNF. The PNFs composing the network service are a DNS server, used to redirect the user on the on-demand VNF with less load, and the RAI-CONTENT PNF that contains all the immersive video content accessed by the RAI on-demand VNFs.

The purpose of the second Network Service, Immersive and Augmented Reality, is to provide to the users connected through HoloLens[36] the augmented reality experience or through the Orah 360° Camera[37]an immersive experience. This Network Service consists of two VNFs: the RAI-HOLO that identifies the objects

[36] Please see: https://www.microsoft.com/en-us/hololens
[37] Please see: https://360camreview.com/orah-4i-vr-camera/

captured by the HoloLens and the RAI-LIVE-STREAM that is in charge to transcode and transmit the live feed acquired from the Orah camera.

VNF Name	Package details
RAI-ONDEMAND	**Description:** see below **VNF name:** RAI-OnDemand • **Memory:** 4 GB • **CPU:** 2 vCPU • **Storage:** 20 GB • **Image:** rai-ondemand.qcow2 • **Format:** QCOW2 **Provider:** RAI
RAI-HOLO	**Description:** see below **VNF name:** Rai-Hololens • **Memory:** 2 GB • **CPU:** 2 vCPUs • **Storage:** 50 GB • **Image:** rai-hololens-v1.qcow2 • **Format:** QCOW2 **Provider**: RAI
RAI-LIVE-STREAM	**Description:** see below **VNF name:** Rai-Live-Stream • **Memory:** 4 GB • **CPU:** 2 vCPUs • **Storage:** 20 GB • **Image:** rai-stream.qcow2 • **Format:** QCOW2 **Provider:** RAI
RAI-CONTENT	**Description:** see below **PNF name:** Rai-Content • **Memory:** 2 GB • **CPU:** 2 vCPUs • **Storage:** 50 GB • **Image:** rai-CONTENT.qcow2 • **Format:** QCOW2 **Provider:** RAI

Table 4. List of UC4 specific VNFs and PNFs

5.2.4.1. Rai-OnDemand

The use case basically provides a website where the users can watch videos.

This VM includes an httpd used to access and stream the immersive 360° content (virtual tour in the Puccini museum and the Puccini house).

5.2.4.2. Rai-HoloLens

Microsoft HoloLens is self-contained, holographic computer, enabling you to engage with your digital content and interact with holograms in the world around you.

- In 5GCity, HoloLens device is used to send image streams to a remote image-recognition server, specifically when a user watches to a point of interest like a monument.

- In the VNF, a Tomcat[38] servlet receives the image and triggers the image recognition service. The image recognition service detects the monument and produces an HTTP response with the id of the monument detected. Once the HoloLens software receives the monument identification, it loads the correct hologram on display.

The software developed for this particular service runs both on a VM (for the image recognition part) and into the HoloLens device. The VM runs Apache Tomcat v8.0.32 with a Java-Spring API application. The image recognition system is based on MPEG CDVS[39]. Onboarded on HoloLens we have a Rai software, developed mainly with Unity Game Engine 2017.4[40], with Holo-toolkit SDK[41] and Vuforia library[42].

5.2.4.3. Rai-Live Steam

In order to distribute through the 5GCity infrastructure a 360° source of video, an Orah 4i Live Spherical Live-Streams 4K 360 Degree VR Camera is used. The stream can be configured on Orah camera setting menu. The current settings are 4k 25000 kbps with H264 high profile encoding. The service developed is based on ffmpeg[43] open source software.

Mandatory: setup a DHCP service for the Orah camera.

5.2.4.4. Rai-Content

Content for Video-on-Demand (VoD).

These are videos produced by the production centre of Turin having the following specs:

- The Puccini videos: Puccini evocation UHD, Museum tour 4K and the virtual museum Casa Puccini tour were realized with the Orah 4i camera, with definition (horizontal) UHD-TV is 3840/1920/25p, coded in H264 high profile at around 10-20 Mb/s.
- The videos linked to the map of Lucca were made with the Nikon KeyMission 360 action camera[44]. Also, in this case the equirectangular image format (generated by the camera in real time) is 3840/1920/25p; this image is coded H264 at about 10-12 Mb/s.

NFS server is required to mount the network folder from Rai-Content and load the videos from it.

5.2.5. UC5: Mobile Backpack Unit for Real-Time Transmission

Table 5 reports the list of Network Functions related to the UC5: Mobile Backpack Unit for real time Transmission, and video production at the edge. The aim of this UC is to produce a video program signal with the feeds of up to 3 cameras, the mixer control to be done anywhere in the network, and the final result to be sent to the TV studios.

VNF Name	Package details
Video Production	**Description:** This Linux VM (CENTOS 7), will provide a video production mixer **VNF name:** TBD • **Memory:** 32 GB • **CPU:** 16 vCPUs • **Storage:** 250 GB

[38] See: http://tomcat.apache.org/
[39] See: https://mpeg.chiariglione.org/tags/cdvs
[40] See: https://unity.com/
[41] See: https://github.com/Microsoft/MixedRealityToolkit-Unity/releases
[42] See: https://library.vuforia.com/getting-started/overview.html
[43] See: https://ffmpeg.org/
[44] Please see: https://imaging.nikon.com/lineup/action/keymission_360/spec.htm

	• **Image:** TBD • **Format:** QCOW2 **Provider:** betevé

Table 5. List of UC5 specific VNFs and PNFs

5.2.5.1. VNF brief description

The requirements on the table are the ones needed to use the Whatchity[45] system, this a high-quality cloud-based production system that is currently being customised for betevé's system in order to be deployed in the 5GCity edge infrastructure. In case this system is not available, the MOG system from the UC3 will be used (see 5.2.3.2 Media Switcher).

5.2.6. UC6: Cooperative, Connected and Automated Mobility

The goal of this UC is to showcase how ubiquitous 5G and the need for low latency and reliable networks (uRLLC - ultra Reliable and Low Latency Communications) are key enablers of autonomous driving. This use case intends to leverage distributed nodes to:

- Provide the means to efficiently and effectively collect common C-ITS[46] standard messages at the edge, through a distributed Edge-based cache
- Relay safety-related messages to connected vehicles (moving nodes, connecting to Small Cells), with the lowest possible latency (leverage the Small Cell co-located edge cache)
- Distribute acquired data at the edge, replicating data for simpler high-availability (no clustering nor failover mechanisms)

Table 6 lists the different services that have been designed and implemented in order to support the different use cases related to CCAM.

MEC application	Package details
Smart Centralized Broker (uw_dds_broker)	**Description**: This Linux Container provides the pub/sub broker (DDS-based) which gathers and publishes messages sent by and to vehicles and a Redis instance configured as Master of the replica set. **Requirements:** • **Host CPU arch**: ARMv7 • **Memory:** 512 Mb • **CPU:** 2 vCPUs • **Storage:** 5 GB **Provider**: Ubiwhere
Caching System (uw-redism)	**Description** A Redis Linux Container which shall be used in the first node, as "Slave" of the Replica Set **Requirements:** • **Host CPU arch**: ARMv7 • **Memory:** 512 Mb • **CPU:** 1 vCPU • **Storage:** 5 GB

[45] See: https://www.watchity.com/
[46] See: https://www.itsstandards.eu/cits

	Provider: Ubiwhere
Caching System (uw-rediss)	**Description** A Redis Linux Container which shall be used in the first node, as "Master" of the Replica Set **Requirements:** • **Host CPU arch**: ARMv7 • **Memory:** 512 Mb • **CPU:** 1 vCPU • **Storage:** 5 GB **Provider**: Ubiwhere

Table 6. List of UC6 specific VNFs and PNFs

5.2.6.1. MEC application brief description

5.2.6.1.1. Smart Centralized Broker

This is the Edge-level broker service, which acts as a pub/sub (Publish/Subscribe) data broker for all CAM/DENM messages (C-ITS stack) coming to and from Vehicles and other street-level furniture such as streetlight poles. This service adopts the highly-reliable, performant and scalable DDS (Data Distribution Service [47] using different open source solutions [48]), so that different Vehicles can publish and subscribe to relevant events such as geo-localized road hazards warnings and temporary speed limits, information about vulnerable road users and any other relevant data which may prevent vehicle accidents, when acted upon accordingly in the right moment.

In this particular use case, this Broker is responsible for acquiring vehicle-reported information related to GPS and real-time speed (taken from OBD-2 interface [49]) in order to determine if a warning shall be sent to the connected vehicles. Regardless of the radio interface (LTE or Wi-Fi), this service will then alert the Vehicle that:

- it is within a macroblock (where the speed-limit is extremely low, and all vehicles must respect this)
- there is a road hazard in the vicinity, which has been previously reported by any other vehicle.

As this Broker's job is to process incoming data and reply accordingly in the most fast and reliable way (uRLLC type of slice), it passes such relevant information to a Redis instance, whenever possible (as to minimize the time it would take to reply to other vehicles). This Redis instance is detailed next.

5.2.6.1.2. Resilient Caching System

This component is the one that provides the needed data persistency. This instance of Redis is configured as Master/Slave, so there are two different instances running in two isolated nodes. The master is co-located alongside uw_dds_broker, meaning that both are running in the same Edge node; while the slave is running in another Edge node that is actually working also as a Wi-Fi node. This allows us to have a scalable and fault-tolerant Caching system. This sort of Edge-based caching mechanisms is crucial to save on costs (OPEX) associated to the required bandwidth to broadcast all Vehicle-originated data all the way to cloud services. At scale, with millions of connected vehicles broadcasting data, the stress on the backhaul network would be enormous; with this design, Cloud-based services would collect only the most important data whenever needed, in bulk and most likely compressed.

[47] https://www.dds-foundation.org/omg-dds-standard/
[48] https://projects.eclipse.org/proposals/eclipse-cyclone-dds
[49] http://www.obdii.com/background.html

6. Conclusion

In this deliverable, we presented all of the components of the 5GCity virtualization infrastructure, including a description of their core functionalities implemented by the second year of the project (M24).

The document presents how these functionalities complement each other, and how they work in order to implement and deploy the various 5GCity various use cases. The deliverable further describes how each of the technologies will be actually deployed, and how they will specifically contribute to the use cases. The development and consolidation works are progressing on the technologies targeted by this deliverable, as the Consortium plans to add more functionalities and fine tune integration of the various components together in order to have full implementations and for the benefit of deployments of the 5GCity use cases.

These consolidation works are progressing on the Unikraft technology, where support for more functionalities (e.g. PyTorch) is ongoing, as well as RAN virtualization, where we are working on RAN controller to enable the integration of additional RAN systems (e.g. of the kind in use at Bristol pilot infrastructure). This addition will define a common API between the Slice Manager and the RAN controllers, allowing the 5GCity platform to be extensible with regards to RAN resources.

EdgeVIM is also being updated with new features which will increase the security of the hypervisor and the VMs. At the same time, we are exposing these capabilities to the upper layers of the 5GCity architecture and the users.

The work on VNF modelling will continue by fine tuning the initial VNF packages produced for the various use case functions in use, and adding the missing ones which will emerge to implement either functional splitting of macro applications or new application behaviours.

The target for completion of all these works is the next deliverable D3.3, due by M28 (Sept-2019), which will contain the final 5GCity Virtualization Infrastructure Implementation.

Abbreviations and Definitions

AP	Access Point
ABI	Application Binary Interface
CPU	Central Processing Unit
CLI	Command Line Interface
DNS	Domain Name System Service
DSP	Digital Signal Processing
EPC	Evolved Packet Core
FFT	Fast Fourier Transform
GPS	Global Positioning System
GUI	Graphical User Interface
IPsec	Internet Protocol Security
L2TP	Layer 2 Tunnelling Protocol
KVM	Kernel Virtual Machine
LTE	Long Term Evolution
MAC	Medium Access Control
MME	Mobility Management Entity
MNO	Mobile network operator
NAT	Network Address Translation
NFVI	Network Functions Virtualization Infrastructure
OCI	Open Containers Initiative
OF	OpenFlow
OPEX	Operating Expenses
OP-TEE	Open Portable Trusted Execution Environment
PLMNID	Public Land Mobile Network Identifier
PNF	Physical Network Function
PNF	Physical Network Function
PPTP	Point-to-Point Tunnelling Protocol
RAN	Radio Access Network
RLC	Radio Link Control
RTMP	Real Time Messaging Protocol
SDN	Software-Defined Networking
SSID	Service Set Identifier
TTL	Time To Live
vAP	virtual Access Point
VIM	Virtualized Infrastructure Manager
VM	Virtual Machine
VNF	Virtual Network Function
VPN	Virtual Private Network
UHD	Ultra-High Definition

References

[1] VyOS - an Open Source Linux-based Network OS, https://vyos.io/

[2] BIND 9 - Versatile, Classic, Complete Name Server Software, https://www.isc.org/downloads/bind/

[3] HA-Proxy - The Reliable, High Performance TCP/HTTP Load Balancer, http://www.haproxy.org/

[4] OpenCV (Open Source Computer Vision Library), https://opencv.org/

[5] MiniCache, a Minimalistic, Virtualized Content Caches, http://sysml.neclab.eu/projects/minicache/

[6] FCAPS - Fault-management, Configuration, Accounting, Performance, and Security, http://marco.uminho.pt/~dias/MIECOM/GR/Projs/P2/fcaps-wp.pdf

[7] REST - Representational State Transfer, https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

<END OF DOCUMENT>