5G CITY

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

D3.3: 5GCity Virtualization Infrastructure Final 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: **VOSYS**	Document version: **V1.6**

Work package: **WP3**

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

Start date of the project: 01/06/2017 (duration 30 months)	Contractual delivery date: Month 28	Actual delivery date: 30/09/2019

Editor name: Michele Paolino (VOSYS)

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	09/09/2019	NEC	Section 2 updated
1.2	11/09/2019	I2CAT	Section 3 updated
1.3	17/09/2019	VOSYS	Section 4 updated
1.4	19/09/2019	VOSYS	First revision
1.5	23/09/2019	ITL, ADLINK, i2CAT, NXW	Second revision
1.6	30/09/2019	VOSYS	Final version ready for submission

DISCLAIMER OF WARRANTIES

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 virtualization infrastructure in its final release. It is built on top of Deliverable D3.2 [2], which marked the release of the intermediate works by WP3 at M24. In this document, the description of each major area of activity on 5GCity virtualization infrastructure has continue to be mapped into specific sections. The structure of the document follows the presentation of contents delivered in D3.2 and contains updated information about latest development and deployments activities. Functional and performance measurements have been added into this deliverable to mark the completion of a functional validation stage of the produced virtualization software.

This deliverable D3.3 is organized in bottom-up mode. We start with base technologies and move up the stack to address VNF models and packages.

At first, we begin with Unikraft[1], a unikernel[2] common code base and automated build system. Unikraft is able to produce extremely lean and efficient virtual machines and containers, ideal for resource-constrained devices that constitute the project's platforms for edge and far edge network sections.

Next, we describe RAN virtualization (i.e., in LTE and Wi-Fi), a key component in being able to share 5GCity wireless infrastructure.

Then, we provide a 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.

The work on modelling and packaging various types of Virtual Network Functions (VNF) and the description of their main characteristics concludes this document. We describe the 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.

It is worth mentioning that each chapter in this document covers in a dedicated section the role that specific technology plays in the 5GCity architecture. Another section discusses the plans for integration and deployment into the 5GCity infrastructures in Barcelona, Bristol and Lucca.

[1] Please see: https://xenproject.org/developers/teams/unikraft/
[2] Please see: http://unikernel.org/

1. Introduction

This deliverable gives an overview of the 5GCity's virtualization platform and infrastructure that forms the basis for the implementation of the project's use cases and deployment. For explanatory purposes, the chapters in this deliverable are organized from the bottom up, describing lower-level technologies first and working our way to orchestration frameworks and integration into 5GCity dashboard. Each chapter 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
b) will be deployed and integrated in order to make those use cases become a reality
c) performs under functional tests.

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[3], containers[4]). Therefore, it is one of the main components of the Neutral Host concept.

In addition to virtualizing computation, another key component of the 5GCity Neutral Host architecture 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.

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[5] 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 they deploy on platforms is exactly the one they provided, and that no sensitive data are 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.

[3] https://www.linux-kvm.org/page/Main_Page
[4] https://www.docker.com/
[5] https://developer.arm.com/ip-products/security-ip/trustzone

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; therefore, unikernels are a way to build a system, including the operating system that is tailored to the needs of specific applications. Due to 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[6]-based scripting using only hundreds of KBs of memory
- The 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 the 5GCity platform should be, as much as possible, agnostic to the applications running over it, the consortium has focused on developing Unikraft.

Unikraft is an incubation project under the Xen Project[7], 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 and time-consuming. Worse, the work, or at least parts of it, often needs to be redone for each target applications. Unikraft's goal is to provide an automated build system that can quickly generate extremely efficient and secure unikernels. Further, Unikraft explicitly supports multiple target platforms: not only virtual machines for Xen and KVM[8], but also OCI[9]-compliant containers and bare metal images for various CPU architectures.

In greater detail, Unikraft *decomposes* operating systems into elementary pieces called micro-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. 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, ARM64 or MIPS)

[6] Please see: https://www.python.org/
[7] Please see: https://xenproject.org/
[8] Please see: https://www.linux-kvm.org/
[9] Please see: https://www.opencontainers.org/

2) the *platform tool*, where target platforms can be Xen, KVM, bare metal (i.e. no virtualization), user-space Linux and even containers

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., KVM 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.

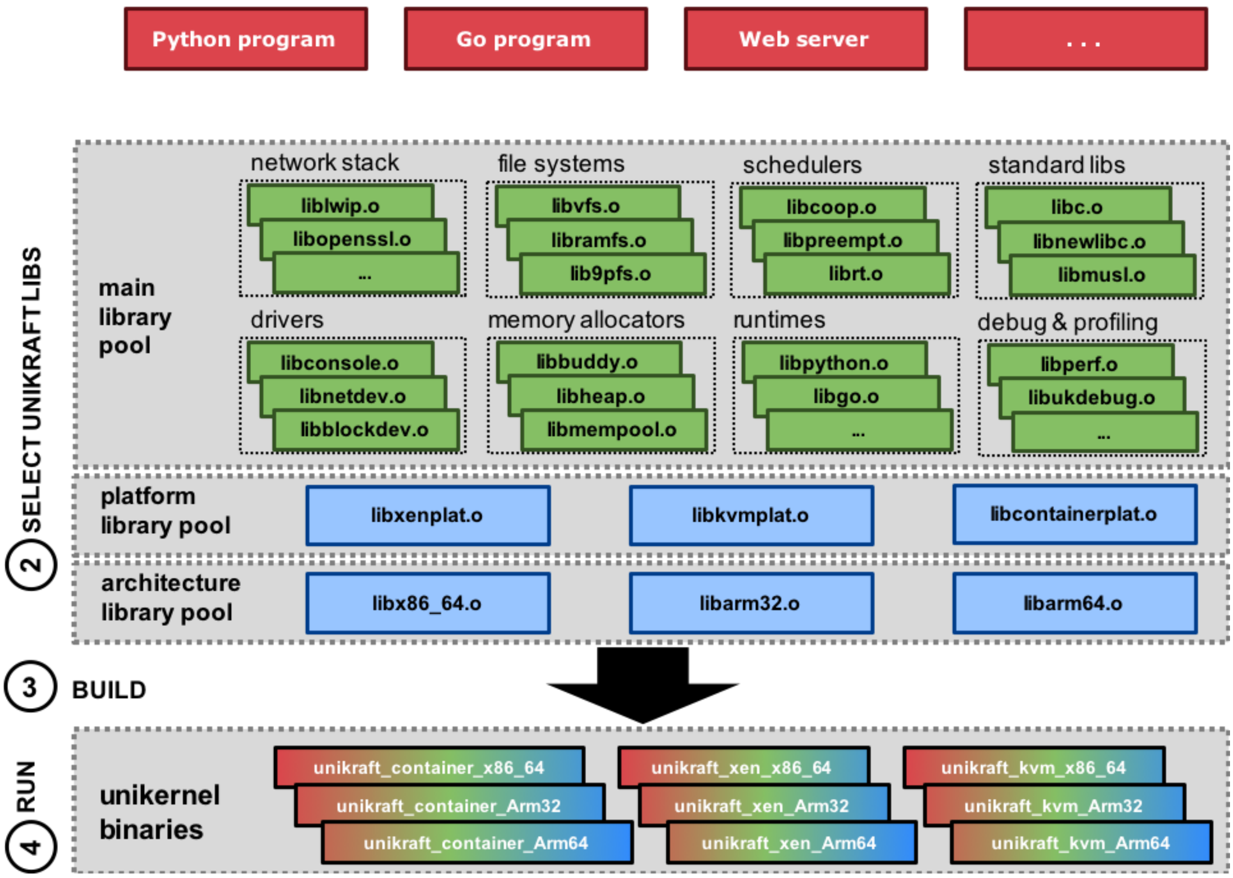

Figure 1. Unikraft micro-library architecture and workflow.

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 and hardware 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.

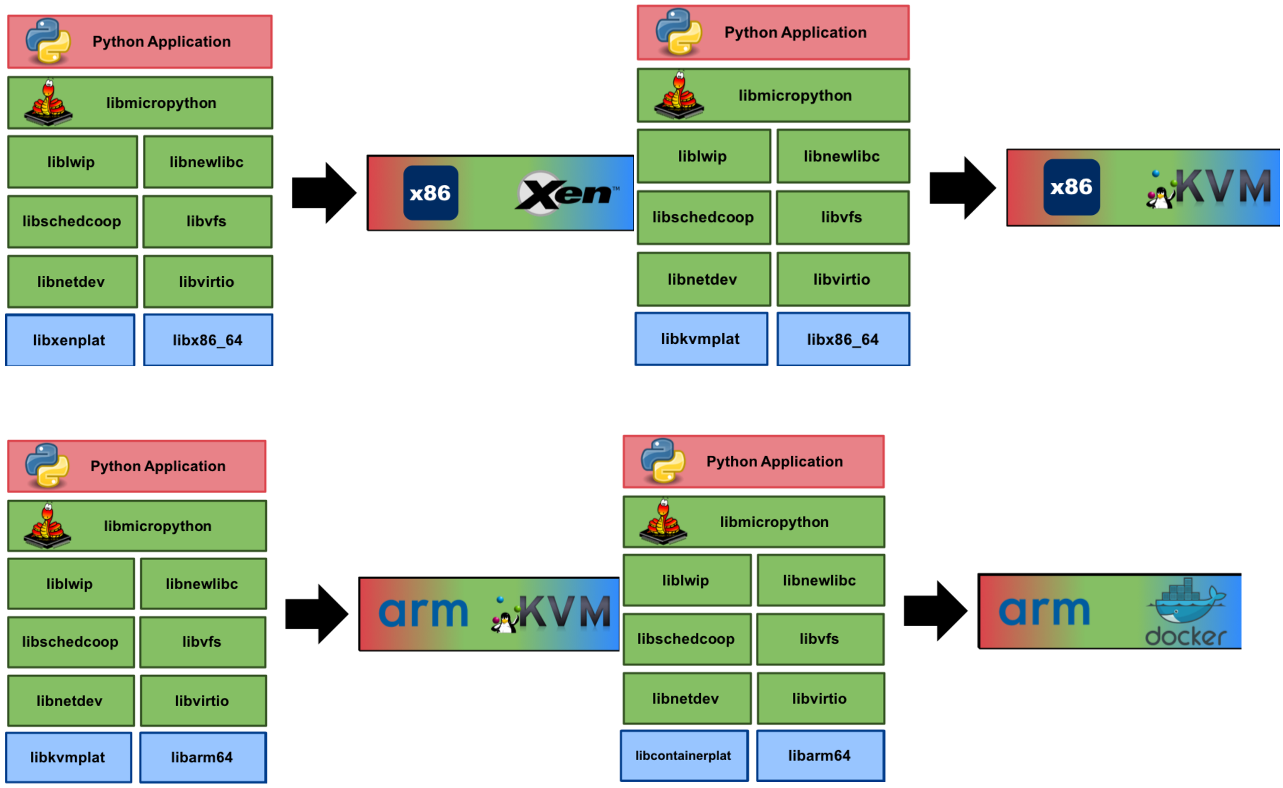

Figure 2. Unikraft can output images for different combinations of platforms and hardware architectures

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, Virtual File System (VFS) support (needed for filesystems), support for ARM, and support for virtio[10] (to be able to do networking on KVM, the project's preferred virtualization platform).

Since then, a large set of micro-libraries and functionality has been released or is in the process of being upstreamed, with a target release date for a v.0.4 sometime in the Q4 of 2019. The set of functionality includes, but is not limited to:

- External standard libraries: musl, libuv, zlib, openssl, libunwind, libaxtls (TLS), etc.
- Language environments: Javascript (v8), Python, Ruby[11], C++
- Frameworks: Node.js[12], PyTorch[13], Intel DPDK[14]
- Applications: Click[15] modular router, NGINX[16], SQLite[17], Redis[18], etc.

It is worth nothing that many 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://github.com/unikraft .

[10] Please see: https://wiki.libvirt.org/page/Virtio
[11] Details at: https://www.ruby-lang.org/
[12] Please see: https://nodejs.org/
[13] Please see: https://pytorch.org/
[14] Please see: https://www.dpdk.org/
[15] Code at: https://github.com/kohler/click
[16]Please see: https://www.nginx.com/
[17] Details at: https://www.sqlite.org/index.html
[18] 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[19] 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[20] 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[21] 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.

2.3. Deployment and integration in 5GCity

In terms of deployment, Unikraft acts 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

[19] https://ubuntu.com/
[20] https://www.docker.com/
[21] 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 have deployed 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 consolidating PyTorch[22] 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 have finished, and have upstreamed, 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 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 have already added ELF (Extensible Linking Format) 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.

2.4. Functional test results

Unikraft is under heavy development, and there will be lots of new functionality coming online especially in Q3 and Q4 of 2019 (the plan is to have all functionality related to 5GCity ready by then, though most of it is ready now or will soon be). Having said that, lots of functionality, as mentioned in previous sections, is already in place. From the bottom of the stack and moving up, the following micro-libraries exist, have been tested through a standard, rigorous open-source review process, and have been upstreamed or will be soon:

- **CPU architecture libraries**: X86_64 works and is upstream. This covers all 5GCity deployments, although we are also working on supporting ARM64, which should be ready in a couple of months (basic ARM32 support is already upstream).
- **Platform libraries**: support for KVM, including network drivers is in place, with early versions of the block driver on the public mailing list (KVM being the main 5GCity virtualization platform). Support for Xen is in place, with network and block drivers. There is also basic support for solo5[23] and Amazon's Firecracker[24]. In addition, within a month there should be upstream support for OCI containers.
- **Language environments**: we currently support C and C++, with support for Micropython, Python v2/v3, Ruby, Lua, WASM/WAMR, JavaScript, Go, Rust and Java (OpenJDK) likely to be upstream in Q3/Q4. In particular, for 5GCity, C++ is important since it is needed for the illegal waste use case

[22] Please see: https://pytorch.org/
[23] Code at: https://github.com/Solo5/solo5
[24] https://firecracker-microvm.github.io/

where the software used for the detection is based on that language (as is the Click/NFV software, also highly relevant to the project).

- **Frameworks and Applications**: a port of DPDK, Intels' high networking performance framework and another software package that can be used as basis for NFV, should be ready in Q4. In terms of applications, we have working and tested ports of nginx, SQLite[25] and Redis, with ongoing ports of memcached[26], lighttpd[27] and other popular software packages.

In terms of the actual functional tests, these are highly dependent on both the micro-library in question and the reviewer involved in the open source process. Having said that, generally speaking, where a particular library/software package comes with a test suite, the Unikraft review process leverages it and considers upstreaming of such a port if and when the tests, or most of them, pass (in many cases, certain tests, e.g., OS-specific ones, don't apply to Unikraft). As an example, as part of the current Python3 porting process we run its full test suite as a functional test. As another example, for the Click modular router, we ran a number of different NFV configurations, ensuring their correctness. In all cases, the tests are pass/fail, except for the few, as explained earlier, that do not apply.

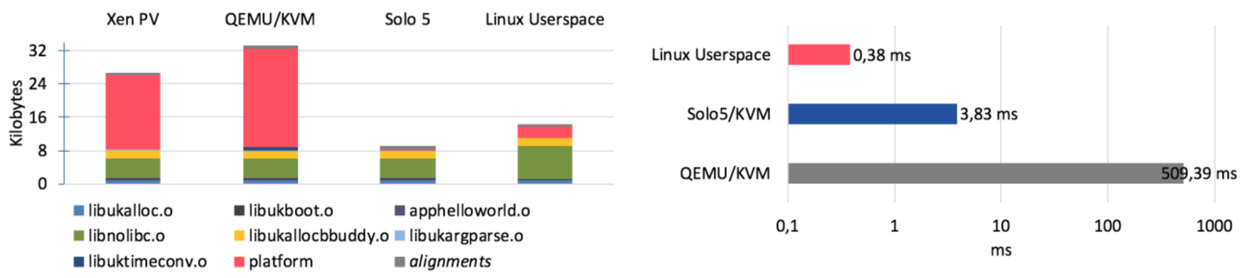

Image breakdowns: Contribution of minimal micro-libraries set to *.text*, *.rodata* and *.data* sections for a simple "Hello World!" guest (uncompressed).

Guest lifetimes: Total time to create, boot, greet, shutdown, and destroy the "Hello world!" guest on different platforms.
Tested on: Debian 9; Linux 4.19.0; Intel® Xeon® E3-1231v3 3.40GHz, 4MB RAM for guests

Figure 3. Unikernel performances (image breakdowns and guest lifetimes).

Finally, we report on some preliminary performance numbers from these functional tests, as further proof that the technology is fully implemented and works. In the graphs above we show (left) the rather tiny image sizes that Unikraft can produce (e.g., only 8KB for a Solo5 KVM virtual machine, orders of magnitude smaller than an equivalent hello world image using a standard Ubuntu VM); and (right) how long it takes to create, boot, print a message and shutdown different VMs. Again, a Unikraft-build Solo5 KVM VM gets through all of these operations in less than 4 milliseconds, considerably faster than the second or seconds a standard VM might take. To better understand the results, it is worth explaining that Solo5 is an optimized KVM "toolstack" from IBM: a toolstack is the set of libraries that contain the functionality needed to, among other things, create and destroy virtual machines. If the KPI we are interested in is boot times, then the two main contributors to that time are the toolstack and the guest (i.e., the unikernel) itself. We rely on solo5 for some of our measurements to ensure that an otherwise heavy-weight toolstack does not "hide" the fast boot times of our guests (this negative effect can be clearly seen in the QEMU/KVM line on the graph on the right).

Other performance indicators and KPIs will be reported in planned deliverables reporting on 5GCity use cases by WP5.

[25] https://www.sqlite.org/index.html
[26] https://memcached.org/
[27] https://www.lighttpd.net/

3. RAN virtualization

3.1. Overview

With the final release of the RAN controller and its integration with the 5GCity platform, as well as the final versions of the radio virtualization agents and software components, the 5GCity solution can now fully control and manage RAN resources to apply slicing. Satisfying the RAN requirements of all 5GCity use cases, the RAN virtualization is now ready to be used for the validation and demonstration phase in the three cities. Further, with the release of the infrastructure abstraction layer and a driver module for supporting Ruckus[28] hardware, the 5GCity RAN virtualization solution now overall supports Small Cells (Accelleran) and two types of Wi-Fi solutions (i2CAT nodes, Ruckus nodes), being open to further additions. This allows creation of slices with any combination of these radio technologies, giving a higher degree of flexibility to tenants when choosing which technology to enable in their slice and also adding a larger amount of hardware supported from the infrastructure provider's point of view, broadening the services they can offer.

In this Section, we describe the latest updates to the RAN virtualization framework, in the form of an update to the contents provided in the previous WP3 deliverables (D3.1[1], and D3.2, [2]).

3.2. Role in 5GCity

The RAN virtualization plays a crucial role in 5GCity, as it enables the slicing of the RAN and thus the different use cases that are being demonstrated in the project. Each use case has different requirements for the RAN, varying in the number of users and technology to be supported. These requirements now can be satisfied in any of the three cities, thanks to the common, fully integrated RAN solution. As detailed in deliverable D3.2, the main components that form the RAN virtualization solution are:

- **RAN controller** (Racoon): as the core component of the RAN virtualization framework, it exposes a series of Northbound and Southbound APIs to enable the configuration of LTE and Wi-Fi resources to form part of 5GCity slices. Further, it polls statistics from the radio equipment that are reported to the Prometheus data base and are exposed to the 5GCity platform. The RAN controller also hosts the software module for the Wi-Fi scheduler that allows to assign shares of airtime to slices that rely on Wi-Fi to provide connectivity to users.
- **Infrastructure abstraction module**: to enable the support of different radio technologies, the infrastructure providers can implement drivers within the infrastructure abstraction layer to support the configuration and management API offered by the RAN controller. The so-called infrastructure abstraction module acts as a proxy to the 5GCity slice manager, directing calls to the 5GCity RAN controller or any other RAN controller for which a driver has been developed (e.g. the Ruckus solution hosted in Bristol).
- **Wi-Fi virtualization agents**: a set of software components that expose a Northbound API to the RAN controller to allow the setup, configuration and monitoring of the Wi-Fi access points provided by i2CAT for the 5GCity project. Please note that the virtualization of the Ruckus Wi-Fi APs is handled by the Ruckus controller.
- **LTE virtualization** (Physical eNodeB + vL3 Control Plane): the LTE virtualization is enabled by two main components that implement eNodeB (RAN) functions: the physical eNodeB L2/L1 functions running on the deployed small cells in the urban furniture) and, as part of the 5GCity platform. the virtualized vL3 Control Plane functionality of the eNodeB implementing Neutral Host functionality.

[28] Please see: https://www.ruckuswireless.com/

The latter component is the one that exposes a Northbound API to the RAN controller and allows the configuration and management of a cluster of eNodeBs deployed in the urban furniture. Since small cell cluster control and near real-time functions are managed by this vL3, the vL3 is the only virtualized component that is strongly suggested to run in the Edge. However, with appropriate transport and latency capabilities this vL3 could equally be deployed at the Data Centre. This vL3 implements the typical Central Unit (CU) functions that would be available in a 5G architecture based on the split between CU, Distributed Units (DU) and Radio Units (RU). In this context vL3 corresponds to CU and the physical small cell functionality corresponds to an integrated DU/RU.

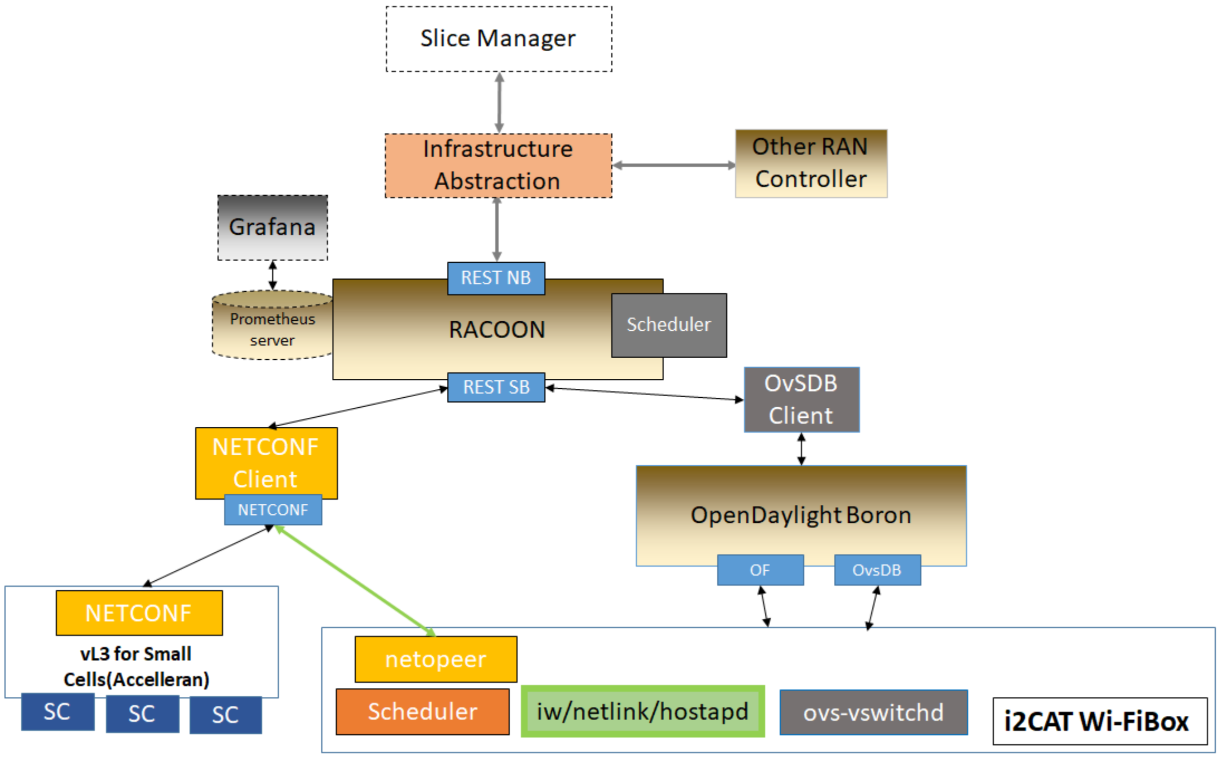

Figure 4: Overview of the software components forming the RAN virtualization framework.

The Figure above shows an overview of the aforementioned components and the interactions between them. Please note that for the LTE virtualization, every LTE-enabled slice requires the eNodeBs that form part of it to be connected to a dedicated EPC. In 5GCity we allow tenants to generate slices using different types of vEPC. For the testing and validation, we use two vEPCs: an Attocore vEPC[29] 1 license per city) and the nextEPC[30] open source solution. These vEPCs are not part of the 5GCity platform but are treated as VNFs instantiated by tenants for mobile/LTE connectivity services established within a Neutral Host slice.

3.3. Deployment and integration in 5GCity

In deliverable D3.2 the details for each component that forms part of the RAN virtualization framework were given and their basic interactions were described. With the final steps in the development and integration work taken, in the following we can provide a compact overview of how the components are deployed and integrated as part of the 5GCity platform.

[29] https://www.attocore.com/products/attoepc/
[30] https://nextepc.org/

Component	Resource requirements	Interacts with
RAN Controller	• 16 GB RAM • 8 vCPUs • 50 GB of storage	Slice Manager / Infrastructure Abstraction (NB), virtualized i2CAT Wi-Fi nodes (SB), vL3 (SB)
Infrastructure Abstraction Layer	• 4 GB RAM • 2 vCPUs • 20 GB of storage	Slice Manager (NB), RAN Controller (SB)
vL3	• 2 GB RAM • 2 vCPUs • 20 GB of storage	RAN Controller (NB), Virtualized eNodeBs (SB)

Table 1. RAN Virtualization components profiling.

Apart from these platform elements, vEPCs are necessary to enable LTE for a slice that incorporate small cells. The following table give the requirements for the two types of vEPCs that have been integrated and tested in 5GCity.

vEPC Type	Resource requirements
AttoCore vEPC	• 4 GB RAM • 2 vCPUs • 20 GB of storage
nextEPC[31]	• 2 GB RAM • 2 vCPUs • 10 GB of storage

Table 2. Virtual EPC package profiling for RAN virtualization tests.

Finally, we would like to highlith that the virtualization agent software running on the i2CAT Wi-Fi and Accelleran Small Cells has been updated during the testing phase, enhancing functionality and improve their performance.

3.4. Functional test results

On the way to its final release, the RAN virtualization framework has undergone a series of functional tests to validate the operation and integration of all of its components. Both the interaction of the RAN controller with the upper layers of the platform (infrastructure abstraction and slice manager), as well as the interaction with the RAN equipment and virtualization components has been confirmed. In this sense, all API calls exposed by the NB RAN controller API were tested and validated, leading to consequential API call towards the Wi-Fi nodes and the vL3 component. To validate the functions of the RAN controller, the following tests were performed in particular:

[31] https://nextepc.org/

- Setting up wireless slices including both LTE and/or Wi-Fi, as a result of a tenant picking Small Cells and Wi-Fi nodes to form part of its slice
- Creating services that include presence at a Small Cell or Wi-Fi AP:
    - For LTE, this means configuring a new PLMNID via the vL3 component and attaching it to a vEPC.
    - For Wi-Fi this means setting up a new virtual AP on top of a physical AP
- Using the infrastructure abstraction module in Bristol to validate that calls towards the 5GCity RAN Controller and calls towards the Ruckus Wi-Fi controller are correctly distinguished and applied.

Apart from these tests that involve the interactions between platform components and interactions with the radio devices, a series of more atomic tests have been performed. Some of these tests are “natural” subsets of the tests mentioned above, such as instantiating, and deleting vAPs in consecutive manner, using automated test procedures to iteratively test the NB APIs of the different software components against the hardware (up to 1000 repetitions), etc.

In order to validate the Wi-Fi virtualization software, a series of on-street experiments were performed. In Barcelona there are three Wi-Fi lampposts in total deployed in 5GCity, each equipped with two wireless cards using different chipsets: one ath10k card and one ath9k card. The map in Figure 5 shows the positions of the three Wi-Fi node, along with the names of the SSIDs for the ath10k cards[32] (dark blue) and ath9k cards[33] (light blue). For the tests, all wireless cards were set to use channel 36 with 40 MHz bandwidth.

For these performance evaluations, we defined a series of tests that need to be performed. All require a UE, in particular a linux-based laptop, to be connected to the different lampposts deployed in the district 22@ and to use the iperf tool that requires a client and server instance to run in two devices and it allows to exchange data streams to be exchanged between the devices. For the 5GCity performance evaluation, the iperf instances ran on the linux-enabled laptops and on the Wi-Fi nodes (which are also linux-based).

An exchange of data between these two devices results in an over-the-air communication using their respective virtual Wi-Fi interfaces instantiated on top of the physical ones. For the evaluations, we set up the client and server on the devices and measure how much throughput could be achieved with a continuous stream of UDP packets that saturated the wireless channel. Such data flows were executed in both up – and downstream, always one at a time. We repeated each experiment 3 times for a duration of 1 minute each.

Since the wireless channel depends on the signal quality between the UEs and the Wi-Fi nodes installed in the lamp posts, there is a whole range of different results to be expected as devices are closer/farther away from each other. A simple way to measure the “quality” of the channel between two wireless devices is to look at the signal strength measured between them. For the set of experiments performed in 5GCity, we defined two ranges of signal strength; the first one, with a signal strength above -60 dBm we considered to be the “close” range, with a good signal. The second one, with a signal strength of -80 dBm or below, was the “far” range, with a medium-bad signal.

[32] https://www.qualcomm.com/products/qca9888
[33] https://compex.com.sg/shop/Wi-Fi-module/802-11n/wle200nx/

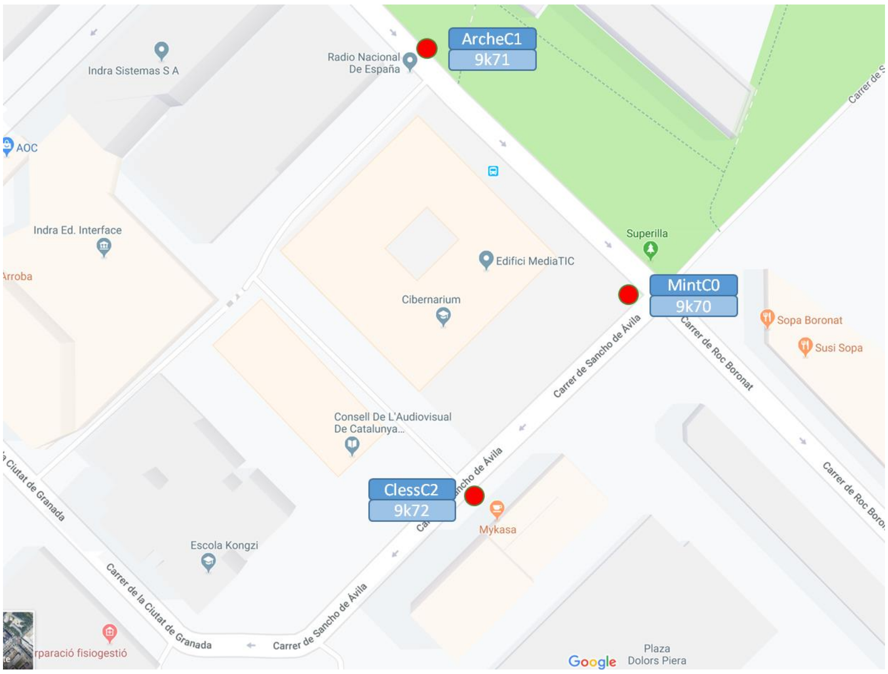

Figure 5: 5GCity Wi-Fi deployment with names of SSIDs (ath10k / ath9k)

The exemplary results shown in the following report on the performance measured with a Dell laptop used as UE. The results obtained for the ath10k (Figure 6) and ath9k (Figure 7) show that the interfaces are operating correctly. Differences in throughput can be explained by different distances (close/far) and the use of different chipsets (ath9k, ath10k). With a "close" distance we refer to positioning a UE in such a way that the signal strength is above -60 dBm, while with far we refer to any positioning of a UE that results in a signal strength below -80 dBm. In that sense all results are coherent and validate that the virtualized Wi-Fi is working correctly, giving sufficient performance for the UCs to be executed.

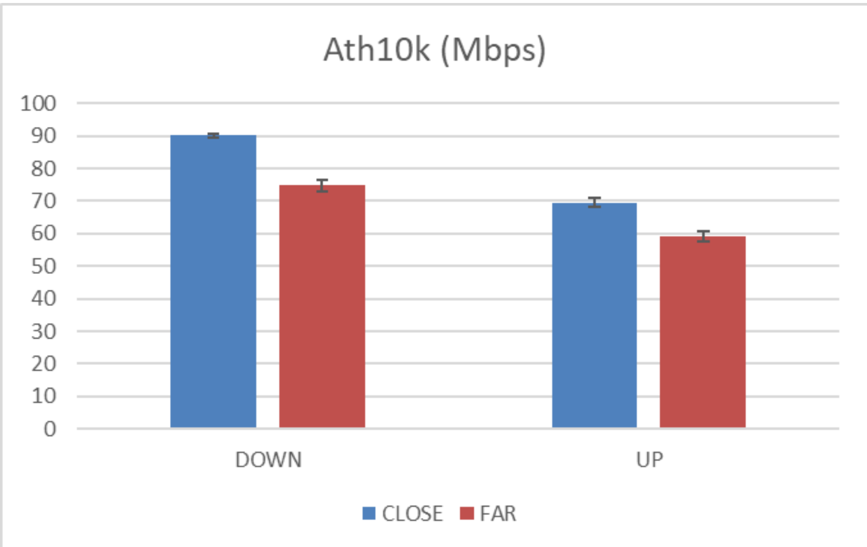

Figure 6: Wi-Fi performance over a 5GCity-controlled ath10k interface

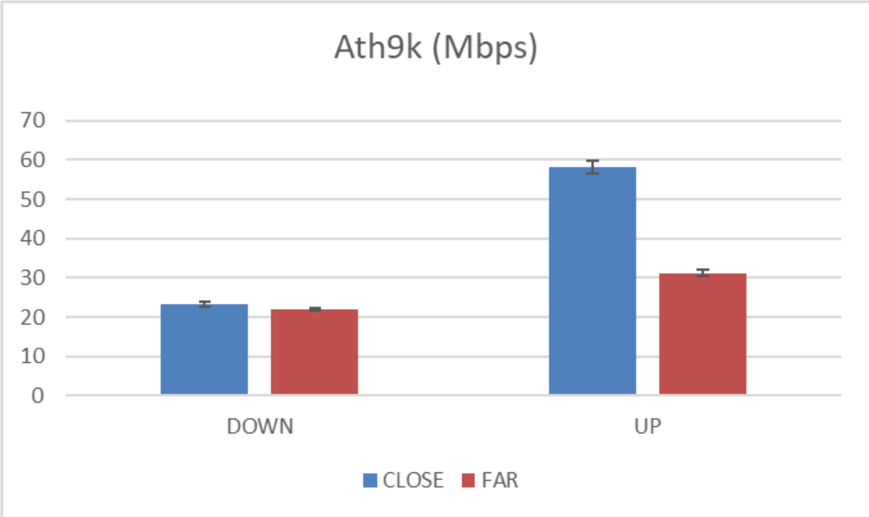

Figure 7: Wi-Fi performance over a 5GCity-controlled ath9k interface

Another important aspect of the virtualization of the Wi-Fi is the isolation and correct airtime ratio application that is handled by the Wi-Fi scheduling system developed for 5GCity. Via the APIs exposed by the nodes, it is possible to set the airtime share that each slice (i.e. virtual APs belonging to a tenant) will get. A basic test with 3 slices running on top of a single AP deployed in the lab and 3 UEs (one for each slice) show how the different airtimes are applied correctly for each slice when setting the slices to get 50% (STA-256), 12.5% (STA-192) and 37.5% (STA-64), respectively.

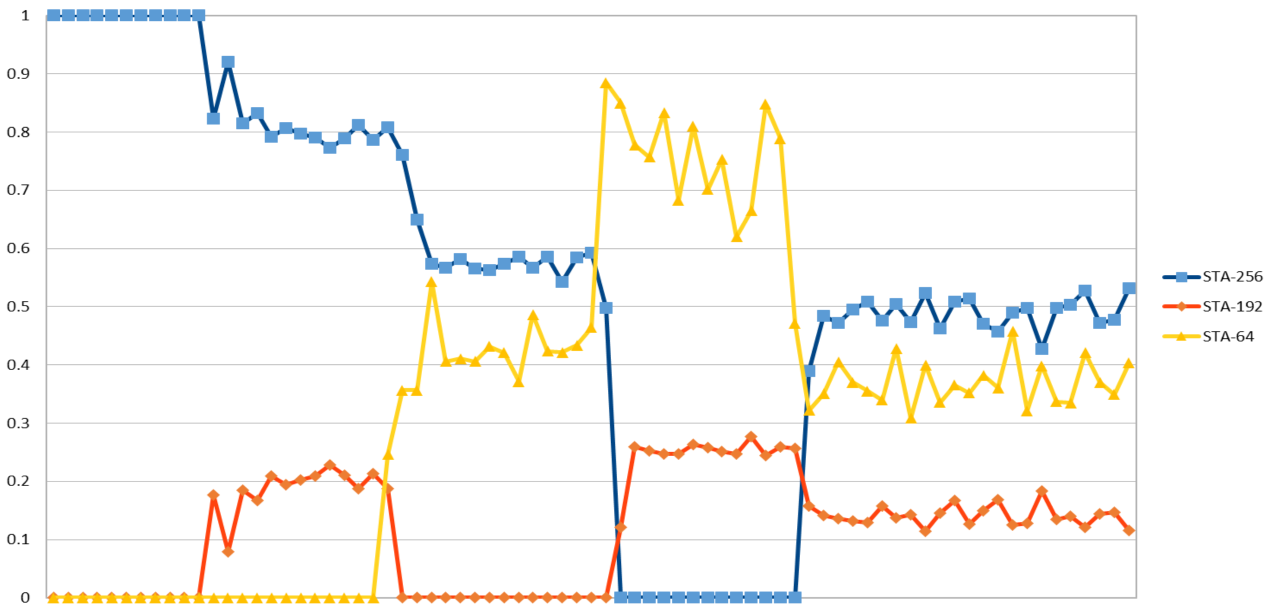

Figure 8: Results showing the airtime consumed by each UE when applying the Wi-Fi scheduler with a setting of 50% (STA-256), 12.5% (STA-192) and 37.5% (STA-64)

Figure 8 shows the consumed airtime from 0 to 1 (0 – 100%) on the AP, when different UEs (STAs) are active. Please note that whenever airtime is not consumed (i.e. the sum of the airtime assigned to the active slices is not 100%), the remaining airtime is proportionally shared among the active slices, since the scheduler operates work conserving. The results show that the different ratios for each slice are applied correctly. Only minor fluctuations are introduced due to channel quality variations and interference as a result from the test environment, where other radios are active.

We would also like to highlight that we have successfully tested the creation and management of LTE slices in the three cities that use the same hardware but with different configurations (different radio bands such as B7, B38 and B42), the use of different Wi-Fi hardware (Ath10k, ath9k, Ruckus Wi-Fi) and three different vEPCs: the AttoCore vEPC, the nextEPC (release 13) and the open5GS (nextEPC release 14).

In order to validate the initial Small Cell deployment and most particularly verify the initial coverage and throughput provided by each of the three lampposts with Small Cells in the city of Barcelona, a series of drive tests experiments were performed. In Barcelona, there are three Small Cell lampposts. Figure 9 shows the positions of the three Small Cells, along with the different parameters used initially.

Figure 9: 5GCity Small Cells deployment in Barcelona

For the initial functional tests, a combination of SpeedTest with the use of GNetTracker Android application in Essential smartphone was used leading to some preliminary data as summarised in Figure 10. Figure 11 shows the screenshots of actual data gathered by GNetTracker and SpeedTest for UE in Position 1 of the map.

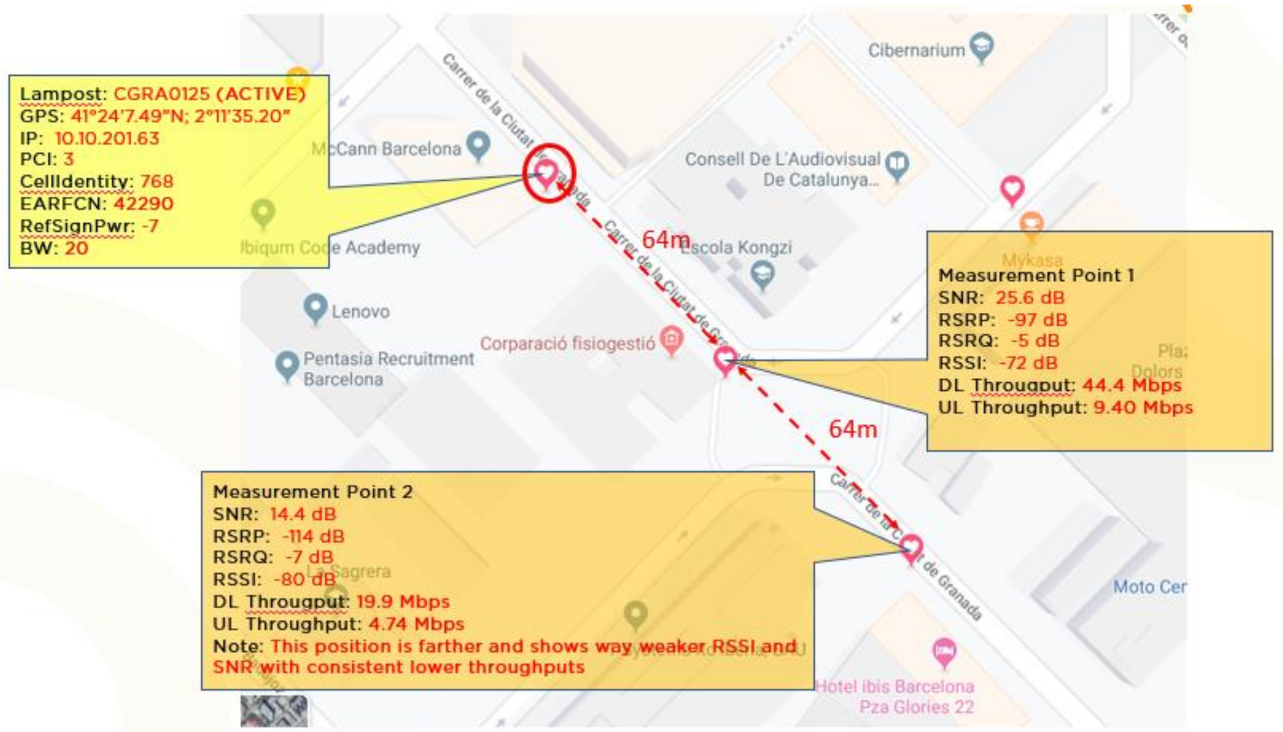

Figure 10: 5GCity Small Cell CGRA0125 initial throughput at range validation

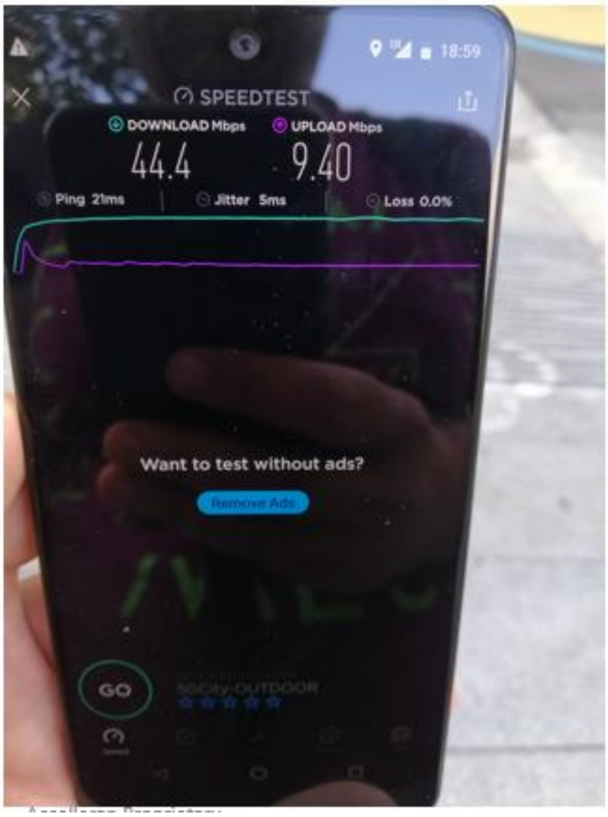

Figure 11: 5GCity GNetTracker and SpeedTest data for CGRA0125 lamppost with UE in Position 1

A further set of drive tests were performed in order to obtain a map of measurement records in Google Earth using GNetTracker Android application with the Essential phone against Small Cell lamppost CGRASANA with different Tx power configurations in order to gather SNR, RSRP, RSRQ field data that could be used to characterise different radio conditions (cell center, mid cell, cell edge) in order to calculate practical inter-site distances (or suitable configured Tx power for the available 60m inter-site distance in the deployment). **Error! Reference source not found.** shows one example of the generated maps with the SNR radio measurements for the maximum Tx power configured in the Small Cell CGRASANA.

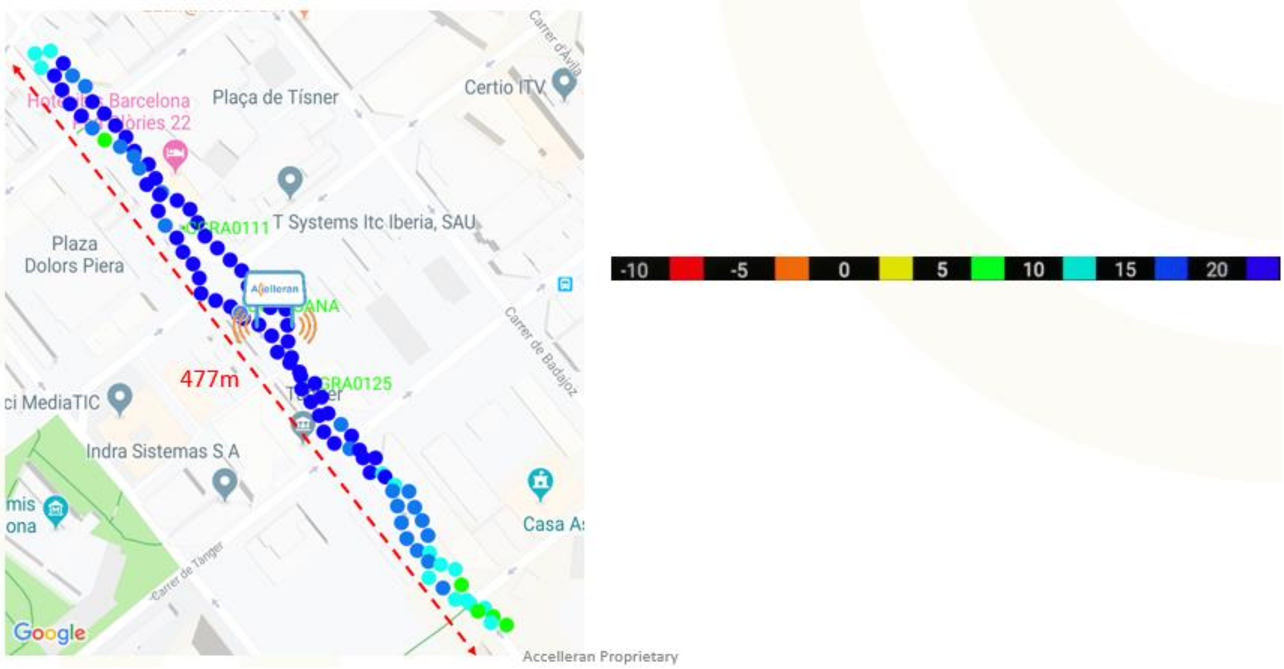

Figure 12: 5GCity CGRASANA Small Cell SNR

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 13. The OpenStack extensions consist of Attestation Filter added to the Open Stack Compute (Nova[34]) 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[35]. They perform node authentication, kernel integrity verification and geo-fencing.

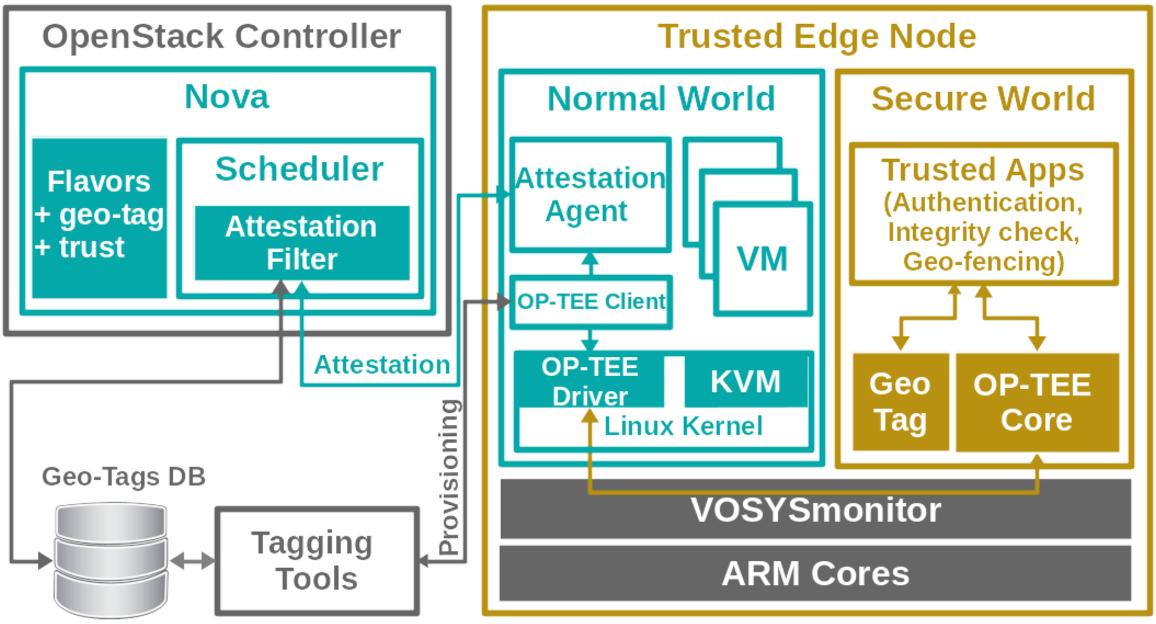

Figure 13. EdgeVIM overview

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

[34] Detail at: https://github.com/openstack/nova
[35] 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:

- 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:

  https://docs.openstack.org/ocata/config-reference/compute/schedulers.html

- 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[36].

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.

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

EdgeVIM is being deployed as part of the 5GCity infrastructure in Bristol where the Trusted environment will include one OpenStack controller and two trusted ARM compute nodes (Xilinx ZCU102 MPSoC).

4.4. Functional test results

EdgeVIM and EdgeNFVI have been tested and benchmarked. In this section both functional and performance tests are detailed.

4.4.1.Functional tests

The functional tests of EdgeVIM are designed in respect of its main features in 5GCity, which as described in Section 4.2,are edge nodes attestation and location awareness. Two scenarios are created, one for evaluating and measuring the attestation capabilities and another adding on top of that location-awareness.

Scenario 1 (Attestation): A user executes a request for the creation of a VM with the additional requirement of trusted compute environment. The EdgeVIM attests each compute node, verifies its identity and the integrity of the operating systems and filters out the non-trusted ones. Afterwards the standard VM scheduling process continues.

Scenario 2 (Location-awareness): A user executes a request for the creation of a VM on a trusted environment with an exact location. The EdgeVIM attests each compute node, verifies its identity and the integrity of the operating systems and filters out the non-trusted ones and the ones that do not match the requested location. Afterwards the standard VM scheduling process continues.

4.4.2.Performance tests

Performance tests are also executed with the goal of evaluating the computational overhead of the added security in comparison of a vanilla OpenStack. We repeat 50 times a creation and deletion of a VM with three different EdgeVIM configurations:

- **Configuration 1**: Acting as a vanilla OpenStack (no added features)
- **Configuration 2**: With node attestation enabled
- **Configuration 3**: With node attestation and location-awareness enabled

OpenStack (release Pike) is deployed with Devstack[37] and consists of:

- One x86 controller node: Intel(R) Xeon(R) CPU E5-2623 v4 @ 2.60GHz, 32GB memory, Ubuntu 16.04.4 LTS, KVM-enabled 4.4.0-128 Linux kernel
- One ARM64 compute node: Xilinx Zynq UltraScale+ MPSoC ZCU102 with a quad-core ARM Cortex-A53, 4GB memory, Ubuntu 18.04.4 LTS, KVM-enabled 4.14.0 Linux kernel

The hardware configuration of the VMs is based on the default OpenStack m1.tiny flavor (1 VCPU, 1GB Disk, 512MB RAM) and the booted guest OS is a CirrOS cloud image. For the custom set up, we create a flavor derived from m1.tiny extended with trust and location properties that have to be matched during the VM scheduling process. The measurements are performed by Rally[38].

[37] Please see: https://docs.openstack.org/devstack/
[38] See: https://docs.openstack.org/rally/latest/overview/index.html

Action	Min (sec)	Median (sec)	90%ile (sec)	Max (sec)	Avg (sec)	Count
VM create	12.816	12.939	13.084	13.4	12.97	50
VM delete	2.35	2.366	2.39	2.55	2.378	50

Table 3. VM create and delete times, no added features (Configuration 1)

Action	Min (sec)	Median (sec)	90%ile (sec)	Max (sec)	Avg (sec)	Count
VM create	12.844	12.986	13.324	17.216	13.233	50
VM delete	2.356	2.371	2.395	2.62	2.388	50

Table 4. VM create and delete times with node attestation (Configuration 2)

Action	Min (sec)	Median (sec)	90%ile (sec)	Max (sec)	Avg (sec)	Count
VM create	12.821	12.986	15.129	15.211	13.549	50
VM delete	2.353	2.372	2.556	2.595	2.414	50

Table 5. VM create and delete times with node attestation and location-awareness (Configuration 3)

The results show ~2% average overhead during VM creation in configuration 2 and ~4% overhead in configuration 3 while the VM deletion process is not affected. On the other hand, the median values stay almost unchanged which shows sporadic deviations in the times caused by the network speed. This can be improved in future implementations.

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 functionality they implement:

- 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 instead 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 6. 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. for DNS workloads).

The Attocore vEPC, nextEPC and Accelleran's vRAN (virtual L3) VNFs are packaged to support all the use cases deploying LTE RAN, starting from Use Case 2 – Neutral Host and including also other Use Cases relying on LTE RAN nodes with Network Slicing support.

VNF Name	Package details
vFirewall	**Description:** Security Front/Back End VNFs to protect users and service providers data **VNF name:** vFirewall • **Memory:** 1024 MB • **CPU:** 1 vCPU • **Storage:** 5 GB • **Image:** vFW-node_exporter-v1.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
vDNS+vLB	**Description:** Associate domain names to VNFs to make more user friendly the interactions among UEs and VNFs. Balance the requests from the UEs to the services running on the VNFs. Configurable via ReST APIs **VNF name:** vDNS-vLB • **Memory:** 512 MB • **CPU:** 1 vCPU • **Storage:** 3 GB • **Image:** vDNS-vLB.qcow2 • **Format:** QCOW2 **Provider:** NXW
Attocore vEPC	**Description:** EPC core functions instantiated and running at the Edge Server. The EPC software is produced and licensed by Attocore. **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
nextEPC	**Description:** EPC core functions instantiated and running at the Core Server. The EPC software is free open source software and licensed under GNU Affero General Public License v3.0. **VNF name:** nextEPC • **Memory**: 4096MB • **CPU:** 4 vCPU • **Storage:** 20 GB • **Image:** nextEPC-Rest.qcow2 • **Format:** QCOW2 **Provider:** NXW
vL3	**Description:** Virtual L3 Small Cell (including Netconf) and ancillary virtualisation and discovery functions (Kubernetes[39], REDIS[40], NATS[41]). **VNF name:** vL3

[39] Please see: https://kubernetes.io/
[40] Please see: https://redis.io/
[41] Please see: https://nats.io/

	• **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 6: List generic VNFs and PNFs

5.1.1.vFirewall

The vFirewall VNF is based on VyOS [3], 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[42]
  - Site to site IPSec
  - L2TP/IPsec and PPTP remote access VPN
  - 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.

A wide variety of hardware- and kernel-related metrics for monitoring purposes are exposed from this VNF via the Prometheus Node Exporter integrated in the VNF package.

5.1.2.vDNS

The vDNS VNF is based on BIND9 [4], 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

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

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 [5]. 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. vDNS+vLB

vDNS and vLB are placed together in an effort to have a single set of modified ReST APIs for configuring both services. The key API methods exposed by this VNF to program the core functionalities described in the previous sections are listed in the table below.

Method	Endpoint	Headers	Body	Description
POST	*http://<vDns-ip-address>:9999/dns*	*'Content-Type: application/json', 'X-Api-Key: secret'*	*{"hostname": "host.example.com", "ip": "X.X.X.X"}*	Through this ReST API it is possible to add a type A DNS record matching the specified hostname with the specified IP address. Moreover, the vLB will be configured accordingly.
DELETE	*http://<vDns-ip-address>:9999/dns*	*'Content-Type: application/json', 'X-Api-Key: secret'*	*{"hostname": "host.example.com"}*	Through this ReST API it is possible to delete the type A DNS record matching the specified hostname. Moreover, the vLB will be configured accordingly.

Table 7: ReST API of vDNS+vLB VNF.

Three Prometheus exporters are installed for monitoring purposes in this merged VNF:

- Node Exporter, to expose system level metrics
- HA Proxy Exporter, to expose LB metrics
- Bind Exporter, to expose DNS metrics

5.1.5. vEPC

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

5.1.6.nextEPC

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

NextEPC[43] is an open source implementation of the Evolved Packet Core of LTE networks. It is aligned to 3GPP Release 14, its version 1.0.0 has been released in May 2019 and there are plans to release first versions of 5G Core (5GC) software in mid-2020.

The nextEPC VNF is configurable applying the modifications to the configuration files or via ReST APIs as described the table below:

Method	Endpoint	Headers	Body	Description
POST	*http://<nextEPC-ip-address>:8898/put*	*'Content-Type: application/json'*	*{ "PLMN_ID_MCC" : "plmn_id_mcc" , "PLMN_ID_MNC" : "plmn_id_mcc" , "TAC_ADDRESS" : "tac_address" , "NET_NAME" : "net_name" , "UE_POOL" : "X.X.X.X/X" , "PRIMARY_DNS" : "X.X.X.X" , "SECONDARY_DNS" : "X.X.X.X"}*	Through this ReST API it is possible to change seven parameters in mme.conf and pgw.conf, and the pgwtun interface will be configured accordingly be configured accordingly.

Table 8: ReST API of vEPC based on NextEPC.

A wide variety of hardware- and kernel-related metrics for monitoring purposes are exposed via the Prometheus Node Exporter.

5.1.7.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

This section is unmodified with respect to the previous deliverable D3.2, [2], issued in May 2019, since fixes and updates to the VNF mostly consisted in fixing to VNF configuration, without major changes to their core

[43] https://nextepc.com

functionalities. The contents previously reported in Deliverable D3.2 are here replicated to provide the reader with a consistent and self-contained document.

5.2.1. UC1: Unauthorized Waste Dumping Prevention

Table 9 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 9. List of UC1 specific VNFs and PNFs

5.2.1.1. Garbage Infringement Recognition

The Garbage Infringement Recognition VNF is based on OpenCV [6], 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 the 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, nextEPC 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 10 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: a 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 10. 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

Deliverable D3.2 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[44] the augmented reality experience or through the Orah 360° Camera[45] an immersive experience. This Network Service consists of two VNFs: the RAI-HOLO that identifies the objects 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	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 11. 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.

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

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[46] 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[47]. Onboarded on HoloLens we have a Rai software, developed mainly with Unity Game Engine 2017.4[48], with Holo-toolkit SDK[49] and Vuforia library[50].

5.2.4.3. Rai-Live Stream

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[51] 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[52]. 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 12 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

---

46 See: http://tomcat.apache.org/
47 See: https://mpeg.chiariglione.org/tags/cdvs
48 See: https://unity.com/
49 See: https://github.com/Microsoft/MixedRealityToolkit-Unity/releases
50 See: https://library.vuforia.com/getting-started/overview.html
51 See: https://ffmpeg.org/
52 Please see: https://imaging.nikon.com/lineup/action/keymission_360/spec.htm

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

Table 12. 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[53] 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[54] 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 13 lists the different services that have been designed and implemented in order to support the different use cases related to CCAM.

[53] See: https://www.watchity.com/
[54] See: https://www.itsstandards.eu/cits

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 **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 13. 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 of the Cooperative Intelligent Transport Systems system developed by Ubiwhere (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 [55] using different open source solutions [56]), 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 [57]) in order to determine if a warning shall be sent to

[55] https://www.dds-foundation.org/omg-dds-standard/ https://www.dds-foundation.org/omg-dds-standard/
[56] https://projects.eclipse.org/proposals/eclipse-cyclone-dds https://projects.eclipse.org/proposals/eclipse-cyclone-dds
[57] http://www.obdii.com/background.html http://www.obdii.com/background.html

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, the Data Distribution Broker developed by Ubiwhere, 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 are 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.

5.3. Final packages

The VNF packages described in this chapter are available at a 5GCity repository hosted on GitHub.

https://github.com/5GCity/5GCity-descriptors

VNF descriptors and Network Service Descriptors, together with related 5GCity SDK originating services and functions will be stored in this repository and kept under revision control. VNF Images will be also stored in GitHub but referenced for download in the wiki associated to the same repository because there typically do not require git revision control and consist of large files.

In the first place, both descriptors and images will be organized by type: Generic and Use case specific.

Use Case specific descriptors and images will be further organized according to the use case to which they belong. Images protected by proprietary and/or restricted license schemes will not be uploaded in the public repository but directly uploaded by use case owners on the target VIMs of the 5GCity infrastructures in which the use case experiments are planned to run.

6. Conclusions

In this deliverable, we present the final release of the 5GCity virtualization infrastructure components implemented in WP3. Since the previous version (D3.2, 5GCity Virtualization Infrastructure Intermediate Deliverable), the content has been enriched with an update on the developed features and with functional and performance tests where possible. In all cases, results show the high quality of the implemented components, with further margin of improvement in some specific cases.

The document also clearly presents how the different functionalities complement each other, enhancing the 5GCity platform from the points of view of wireless networks (LTE and Wi-Fi) and computing virtualization (Unikraft, EdgeVIM, VNF models). All of them are strongly connected 5GCity, with resource constrained devices and more in general with a high performance, efficient and secure smart city infrastructure.

Finally, the deliverable describes how each of the technologies will be actually deployed, how they will contribute to the use cases. The partners activities on these components will continue aiming at optimizing each for the project use cases.

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
NB	North Bound
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
SB	South Bound
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
VTI	Virtual Tunnel Interfaces
UHD	Ultra-High Definition

References

[1] 5GCity Project: Deliverable D3.1: “5GCity Edge Virtualization Infrastructure Design”

[2] 5GCity Project: Deliverable D3.2: “5GCity Virtualization Infrastructure Interim Release”

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

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

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

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

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

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

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

<END OF DOCUMENT>