H2020-ICT-2018-2-825377

UNICORE

UNICORE: A Common Code Base and Toolkit for Deployment of Applications to Secure and Reliable Virtual Execution Environments

Horizon 2020 - Research and Innovation Framework Programme

D5.1 Deployment plan, requirements and business cases

Due date of deliverable: 29 February 2020

Actual submission date: 28 February 2020

Start date of project	1 January 2019
Duration	36 months
Lead contractor for this deliverable	Orange Romania
Version	1.0
Confidentiality status	"Public"

Abstract

This deliverable marks the milestone fifth version of the UNICORE Deployment Plan, Requirements and Business Cases document. The goal of the EU-funded UNICORE project is to develop a common code-base and toolchain that will enable software developers to rapidly create secure, portable, scalable, high-performance solutions starting from existing applications. Key to this is to compile an application into very lightweight virtual machines - known as unikernels - where there is no traditional operating system, only the specific bits of operating system functionality that the application needs. The resulting unikernels can then be deployed and run on standard high-volume servers or cloud computing infrastructure.

The technology developed by the project will be evaluated in a number of trials, spanning several application domains. This document details the target deployment plans for the Unikernels, the requirements for each of the use-cases to be tested during the validation phase of UNICORE, the associated business models and business cases for each use-case. This document details and finishes with a conclusion section.

Target Audience

The target audience for this document is the general public interested in UNICORE solutions and use-case validations.

Disclaimer

Impressum

Full project title	UNICORE: A Common Code Base and Toolkit for Deployment of Applications to Secure and Reliable Virtual Execution Environments
Title of the work package	WP5 – Unikernels in Practice
Editor	Orange Romania
Project Coordinator	Emil Slusanschi, UPB
Technical Manager	Felipe Huici, NEC
Copyright notice	© 2019 Participants in project UNICORE

Executive Summary

The Work Package 5 within the UNICORE project is responsible for the validation of UNICORE krafting solutions in specific application areas. This Report contains the Deployment Plan, Requirements and Business Cases related to the selected UNICORE use cases initially identified in deliverable D2.1 and in DoA.. This is the first version of the UNICORE deployment plan which describes four deployment targets and the corresponding use-cases initially specified in deliverables D2.1 and D2.3. In this report we briefly recall the UNICORE use-cases, describe in detail the infrastructure required for their deployment, the related and specific UNIKERNEL requirements and the Business case and Business Models.

UNICORE provisions four different deployment targets consisting of seven use-cases. The deployments targets are Serverless computing, Network Function Virtualization (NFV), Home Automation and Smart Contracts. The NFV targets additional four specific use-cases in order to validate the use of unikernels across different scenarios such as Universal Consumer's Premise Equipment, run by NEC; Broadband Network Gateway for Wired Internet Access, run by ORO; Wireless 5G virtual Radio Access Network NFV Clusters, run by Accelleran; and Customer Premise Equipment NFV run by OA.

This document complements the use-cases description presented in Deliverables 2.1 and D2.3 with detailed information on requirements for UNIKERNEL, including the capabilities needed for the deployment in the test-beds, the management plane service requirements, network libraries and protocols that UNIKERNEL should support, API requirements and finally, scalability, security and isolation requirements. Orchestration and Management Integration details are also provided in terms of requirements for service instantiation and resource configuration on the deployed infrastructure as well as the tools and methods to be used for orchestration capabilities.

The business case and business model description for the use-cases, where applicable, provides a high-level overview of the implementation of UNIKERNELS needed to provide customer services, the specific optimization compared to current implementation and the way forward towards possible applications stemming from using UNIKERNELS.

The reports also presents an initial deployment plan strategy which refers to various best practices and State-of-The-Art on deployments methodologies in comparison with UNIKERNEL deployments.

Through this deliverable we provide an initial deployment plan for UNIKERNELS, the strategies and timeline of activities and a way forward as this report can be referenced in the following deliverables of Work Package 5.

List of Authors

Authors	Ioan Constantin (ORO), Marius Iordache (ORO), Cristian Patachia (ORO) Franck Messaoudi (OA), Thierry Masson (OA) , Stephen Parker (XLRN), Jesús Martín (CSUC), Xavier Peralta (CSUC), Matteo Pardi (NXW), Gino Carrozzo (NXW), Cristina Basescu (EPFL), Gaylor Bosson (EPFL), Felipe Huici (NEC), Radu Stoenescu (CNW), Razvan Deaconescu (UPB), Emil Slusanschi (UPB)
Participants	ORO, CSUC, OA, NXW, EPFL, NEC, UPB, CNW
Work Package	WP5 - Unikernels in practice
Security	Public
Nature	R
Version	1.0
Total number of pages	55

Contents

List of Figures

Acronyms

5GC 5G Core Network

ABI Application Binary Interface

API Application Programming Interface

AMQP Advanced Message Queuing Protocol

ARM Advanced RISC Machines

ASLR Address Space Layout Randomisation

AWS Amazon Web Services

BPF Berkeley Packet Filter

BMS Building Management System

BNG Broadband Network Gateway

CLI Command Line Interface

CPE Customer Premises Equipment

CPU Central Processing Unit

CNW Correct Networks SRL

CSUC Consorci de Serveis Universitaris de Catalunya

CU Central Unit

CVE Common Vulnerabilities and Exposures

DALI Digital Addressable Lighting Interface

DEDIS Decentralized and Distributed Systems

DHCP Dynamic Host Configuration Protocol

DMA Direct Memory Access

DOA Description of Action

DOS Denial Of Service

DPDK Data Plane Development Kit

dRIC dRAX RAN Intelligent Controller

DSL Digital Subscriber Line

DU Distributed Unit

DUT Device Under Test

EAD Ethernet Access Devices

eBPF extended Berkeley Packet Filter

ELF Executable and Linkable Format

EPC Evolved Packet Core

EPFL Ecole Polytechnique Fédérale de Lausanne

EVM Ethereum Virtual Machine

FPU Floating Point Unit

GDOI Group Domain of Interpretation

GPU Graphics Processing Unit

HA High Availability

HAL Hardware Abstraction Layer

HVAC Heating, Ventilation, and Air Conditioning

IoT Internet of Things

IP Internet Protocol

IPSec IP security

ISP Internet Service Provider

KPI Key Performance Indicator

KVM Kernel-based Virtual Machine

LTE Long Term Evolution

MANO Management and Orchestration

MCAPI Multicore Communications API

MPLS Multiprotocol Label Switching

MSAR Multi-Service Access Routers

MQTT Message Queuing Telemetry Transport

NAT Network Address Translation

NATS Neural Autonomic Transport System

NIC Network Interface Card

NF Network Function

NFV Network Function Virtualisation

OCI Open Containers Initiative

ODM Original Design Manufacturer

ONAP Open Network Automation Protocol

ONVIF Open Network Video Interface Forum

OPC Open Platform Communications

OPC-UA OPC Unified Architecture

OS Operating System

PBFT Practical Byzantine Fault Tolerance

pCPE physical CPE

PLR Packet Loss Ratio

PNF Physical Network Function

POS Performance Oriented Scheduler

PTZ Pan Tilt Zoom

QoS Quality of Service

RAM Random Access Memory

RAN Radio Access Network

Redis REmote Dictionary Server

REST REpresentational State Transfer

RGB Red Green Blue

RRD Round Robin Database

RRU Remote Radio Unit

RTU Remote Terminal Unit

S3 Simple Storage Service

SCF Smart Contract File

SCTP Stream Control Transmission Protocol

SDN Software Defined Networking

SDWAN Software Defined Networking in a Wide Area Network

SIP Session Initiation Protocol

SNMP Simple Network Management Protocol

SQL Structured Query Language

TCP Transmission Control Protocol

TRL Technology Readiness Level

UDP User Datagram Protocol

UE User Equipment

UI User Interface

UIT Universitat Internacional de Catalunya

vCPE virtual CPE

vCPU virtual CPU

VM Virtual Machine

VMM Virtual Machine Monitor

VoD Video on Demand

VoIP Voice Over Internet Protocol

VNF Virtual Network Function

vRAN virtualized Radio Access Network

XDP eXpressive Data Path

1 Introduction

This document is the deliverable D5.1 of the UNICORE project. Its purpose is to provide a detailed deployment plan for the four different deployment targets in UNICORE, including the high level description of each deployment target and the corresponding use-cases, the infrastructure description, the unikernel requirements for each particular use-case and any orchestration and management integration requirements. Further, this report will provide an analysis and description of the business cases for each of the use cases, pointing out what the business models will be.

This report is structured in four chapters, which are:

Chapter 1 provides an introduction to the rest of the document.

Chapter 2 offers a description of the deployment plan and business model for UNICORE, including an analysis of the tools, methodologies and best practices used for software deployment, applicable to UNICORE.

Chapter 3 provides the description of the four deployment targets including infrastructure description, unikernel and orchestration and management requirements and business case descriptions, where applicable.

Chapter 4 summarizes what has been achieved in this document, identifying shortcomings and proposing possible solutions for the future.

2 Deployment Plan

The deployment of software artifacts consists of a set of activities that make the whole system available validated for use. Starting from the development phase, it is common practice to pass through a refinement of the implementation, testing, integration and when everything is completed, the final application software is packed in a final deliverable and installed on the target.

Every software system is unique, therefore the precise processes or procedures within each activity can hardly be defined in general terms. In fact, how to deploy software and test it in target conditions really depends on the scale it has to operate at, the type of hosting servers in use, the type of application to be deployed, the level of security required, the infrastructure in use, etc. Therefore, deployment should be interpreted as a general process that has to be customized according to specific requirements or characteristics.

The UNICORE project is no exception to this. Each use case has its own targets and rules and will follow a specific procedure of deployment, according to its needs. Nevertheless, because the process starts from the development and all of the targets relies on Unikraft framework, some common path can be identified.

Software repository. The Unikraft project resides on GitHub's public repositories at https://github.com/unikraft. GitHub is a Git repository-hosting environment that includes all of the distributed revision control and source code management (SCM) functionality of Git, as well as adding its own features. This allows the use case owners (and developers) to have a single and verified source, where to get the foundations to build their applications.

Testing. Integration testing and functional testing give the software developer a chance to catch a bug before it reaches production. This has a huge importance, because frees people from the worry of breaking something on the production servers. Secondly, it protects the software every time an update is released. This has much importance in Unicore, because it directly reflects on the updates that are constantly added to the Unikraft framework.

CI / CD. Continuous Integration is a practice in modern software that focuses on making preparing a release easier. CD, here mentioned as an acronym, can either mean Continuous Delivery or Continuous Deployment, and while those two practices have a lot in common, they also have a significant difference that can have critical consequences for a business[1].

Continuous Delivery focuses an organization on building a streamlined, automated software release process. The core of this release process is an iterative feedback loop. It revolves around delivery of software to the end user as quickly as possible, learning from their hands-on experience, and then incorporating that feedback into the next release. This means that on top of having automated your testing, you also have automated your release process and you can then deploy your application at any point of time by clicking on a button. Continuous Deployment goes one step further, automatically releasing every change that passes all stages of your production pipeline directly to the final destination. There's no human intervention, and only a failed test will prevent a new change to be deployed to production.

As already stated before, CI / CD processes can hardly be defined the same for all of the use cases, although some of them can be applied to the Unikraft framework releases. However, due to the experimental nature of the artefacts, Continuous Delivery is probably the highest level of automation that the project can target, and after this a simple script (manually triggered) might update the production deployment.

For UNICORE, mechanisms of Continuous integration based on Jenkins software are being configured for the parts related to core UNIKRAFT elements. Extension of Jenkins functionalities to some use cases is also under consideration, above all in cases of free open source software to be built and used in the deployment scenarios. Due to the different nature of the target deployment scenarios for the four target use areas of validation, we plan to use no o automation for delivery, hence leaving to the use case tester to deploy the krafted unikernels in the target platforms for test.
As a general plan cross use case targets, the UNICORE consortium is following a phased approach linked to the major milestones and deliverables planned for the project.

- **Phase-0 (M6-M12): initial experiences with UNIKRAFT.** During this initial phase the various use case teams have experimented UNIKRAFT to port initial functions. The aim of this phase is to make initial hands-on experience with UNIKRAFT in order to familiarize with the krafting tool, the procedures to identify and resolve software dependencies with a more focused scope deriving from the use case scenarios.

- **Phase-1 (M13-M14): use case deployment planning,** which has led to the delivery of this document.
- **Phase-2 (M15-M20): Porting and deployment of core use case functions to unikernels.** This phase will select initial core functions for each use case and will implement krafting via UNIKRAFT and testing of the resulting application binaries.
- **Phase-3 (M21-M36): Completion of porting and deployment of core use case functions with unikernels.** This phase will complete the porting and validation of the various functions identified for each use case which might benefit from being implemented via unikernels.

The upcoming Chapter 3 details each of UNICORE's use-cases, the UNIKERNEL and integration requirements, and the corresponding business cases for each considered scenario.

3 Deployment Targets

3.1 Serverless Computing

3.1.1 Description of the Use-Case

CSUC has worked for more than 20 years hosting, developing and implementing different digital repositories focused on digital content for the University Community, in concrete this kind of repositories is called institutional repositories, adding different services to these repositories. Each institutional repository stores their own documentation related to teaching, research and institutional documents: PhDs, final year projects, teaching material used in different subjects, etc. Concrete examples in this scenario are two university repositories: IRTA PubPro[2] and UIC Open Access Archive[3].

These institutional repositories are based on an open source software called DSpace[4], that provides tools for the management of digital collections and it is adapted to the norms, standards, and good international practices for this kind of digital material.

The different institutional repository admins can upload documents in these repositories that can consist in: pdfs, videos, images, etc.

In our Use-Case we are focusing on the digital images conversion that are uploaded to repositories, thus they have large size and in order to visualize them is mandatory to convert them to a light-weight image.

3.1.2 Description of the Infrastructure

The CSUC current infrastructure is formed by a cluster of kubernetes for each repository. Each repository cluster is managed by Rancher, a PaaS platform which allows to threat the kubernetes clusters as a whole. The figure 3.1.1 shows the architecture of the different components to run a repository which consists mainly in:

- Bastion host which secures the internal work
- 3 Rancher Master nodes in High Availability
- 3 Kubernetes Master Nodes in High Availability
- Kubernetes worker nodes (Repository)

- Media Converter

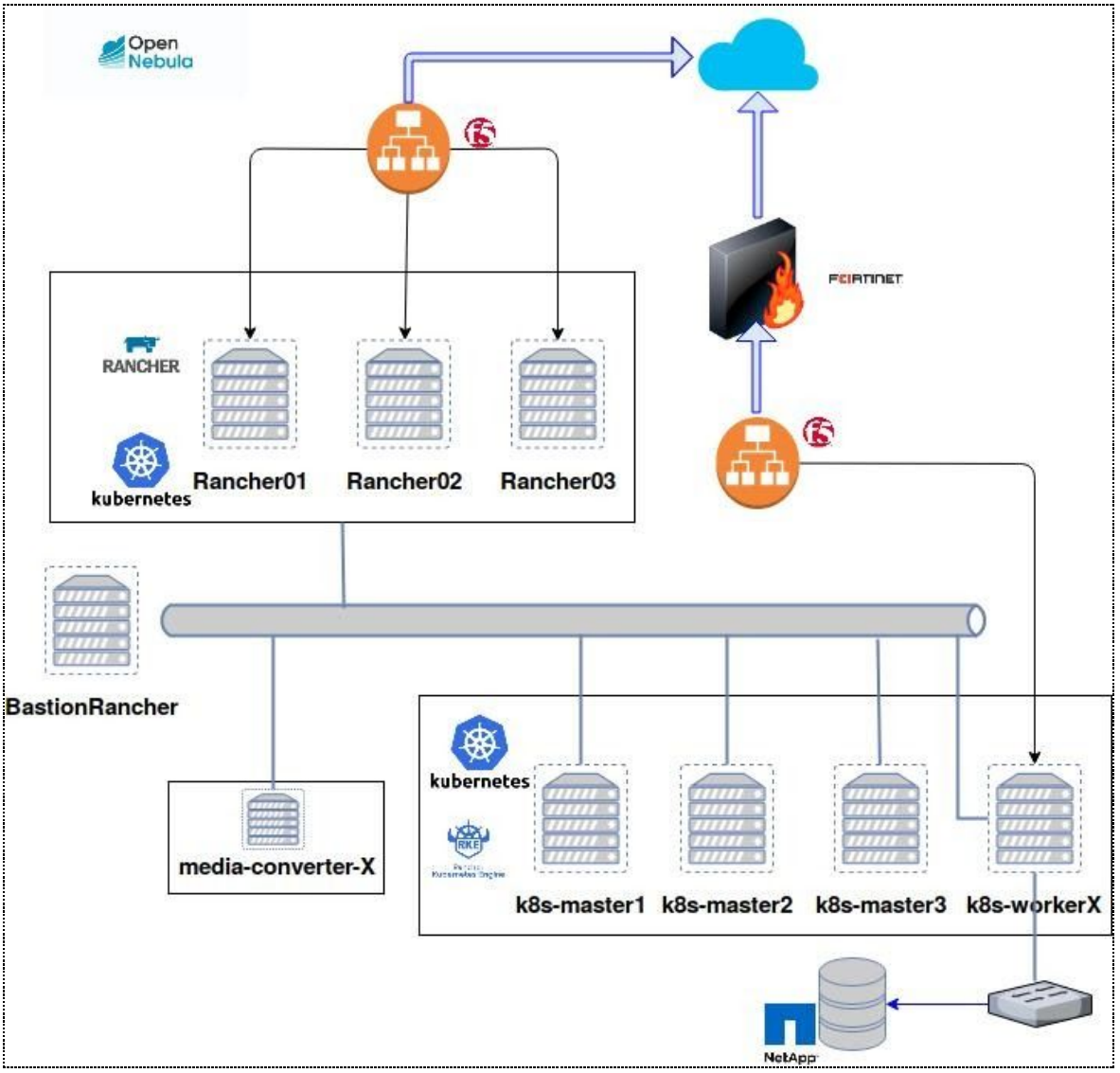

Figure 3.1.1: Current architecture

The most important component in this use case is the media converter. There is a unique media converter for each repository. This media converter is a virtual machine running over the KVM hypervisor and managed by OpenNebula. The purpose of the media converter is to change the characteristics of the files the repository admin uploads to the repository. Thus, for example applies to an image repository which needs to resize images in order to make their visualization easier, so, the media converter runs a cron job that checks every night if there are any images to convert, then starts to run a script to resize and upload them to the repository.

Following the previous scenario CSUC use case will focus on changing how the images that have to be uploaded to the repository are converted. Thus, the use case aims to change the media converter virtual machine by an unikernel serverless solution. The new proposed scenario based on unikernels is shown at figure 3.1.2. This new solution will change the image conversion behaviour and add some components in order to automate the task. The main idea is to upload the images to be converted to an S3 bucket (input), once the image is uploaded a system will notice and will call the orchestrator to instantiate a function as a service based on unikernels which will take the image, convert it upload it to another S3 bucket (output). Each unikernel will be executed over a KVM hypervisor to ensure the isolation between them and acting as if they were virtual machines and it will be running one unikernel per image.

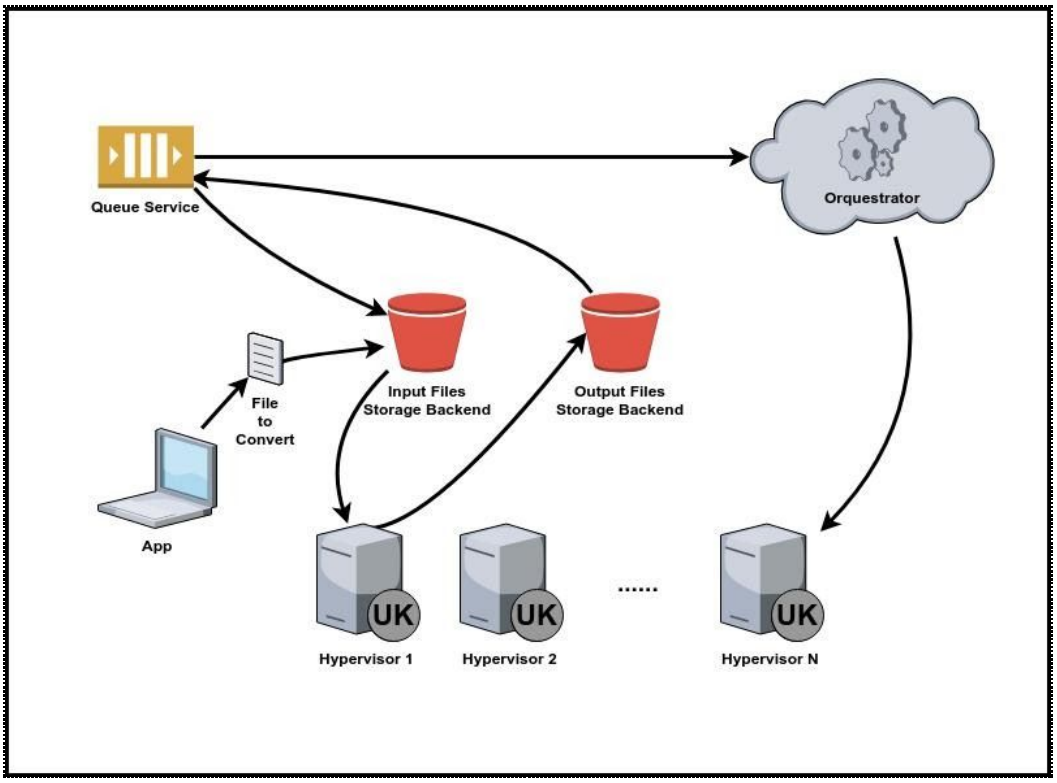

Figure 3.1.2: Possible final architecture

The queue service will monitor the conversion tasks to be started and also will send this information to the orchestrator and will check if the task is done. The orchestrator will receive the commands from the queue service to start as many unikernels processes as files would be in the input storage backend to convert it. The hypervisor, KVM, will run the

unikernels and assure the isolation between them, considering unikernel processes as if they were virtual machines.

3.1.3 UNIKERNEL Requirements

The target solution has to improve the behavior of virtual machines by leveraging unikernel mainly characteristics which are:

- Low deployment time
- Lifetime limited
- High number of instances per node
- Low resources consumption
- High performance

For the image conversion a script written in Python will be developed using the Pillow library and boto3 a python native library to manage the S3-like object storage protocol to upload and download the images to be or already converted.
The Pillow library requirements are the following:

- python3
- libwebpdemux2
- python3:any
- libwebpmux3
- mime-support
- zlib1g
- python3-pil.imagetk
- python3-tk
- libc6
- libgcc1
- libfreetype6
- libpng16-16
- libjpeg8
- liblcms2-2
- libjpeg-turbo8
- libtiff5
- libjbig0
- libwebp6
- liblzma

CSUC use-cases will use all the components developed by unicore toolkit in order to create the unikernel image which better fits on the purpose.

The most important components would be the Automatic Build Tool in case we can achieve a way to build image on instantiation time, the verification tool and the performance optimization to provide more information about how it performs the used image.

3.1.4 Orchestration and Management Integration Requirements

Regarding CSUC orchestration requirements the final solution based on unikernels should be deployed by Kubernetes or OpenNebula similar at openFaaS. This final solution should be run on a x84 architecture. In any case these requirements are independent from the functional requirements, so, it won't affect the use-case.

3.1.5 Description of Business Case

The purpose of this use case is to change the behaviour on how CSUC converts the images before uploading them to the different repositories. This new behaviour has to allow to manage the images in a lightweight mode leveraging the unikernels performance and also changing different actors in the whole workflow in order to accomplish a more automatic interaction between all the components.

3.2 Network Function Virtualization

3.2.1 Broadband Network Gateway for wired Internet Access

3.2.1.1 Description of the Use-Cases

Orange's target for UNICORE work is determined by the definition of a novel approach to the implementation of Broadband Network Gateways (BNGs). Different implementation models are considered:

- *An evolution from the current physical monolithic implementation to a virtualized monolithic implementation.*
- *A further evolution to a virtualized deployment using unikernel VMs.*

The main objective for ORANGE is to define and implement the BNG unikernel application in a virtualized environment, within the use-case life cycle approach for service requirements analysis, design, system requirements, overall architecture for implementation, including

control and management for use case service and infrastructure resources. The technology developed should enable the seamless creation and deployment of any unikernel Unicore application with due consideration of performance, scalability, security, isolation, efficiency.

The entire system is based on an efficient NFV, relying on Unicore to develop the BNG lightweight network function for improved performance from the end-user perspective and efficient resource optimizations from the service-provider perspective. The entire system will run on dedicated lightweight VMs, instantiated in an automatic manner and orchestrated on a per customer application basis.

Today's ORANGE BNG implementation is based on several physical BNGs (Nokia 7750 hardware), deployed in different locations in the network, in ORANGE Data Centers. Customers from different regions are connected to their dedicated BNGs, for clarity, by defining a region, Region A, the clients from that specific region will connect to Region A BNGs, as shown in figure 3.2.1

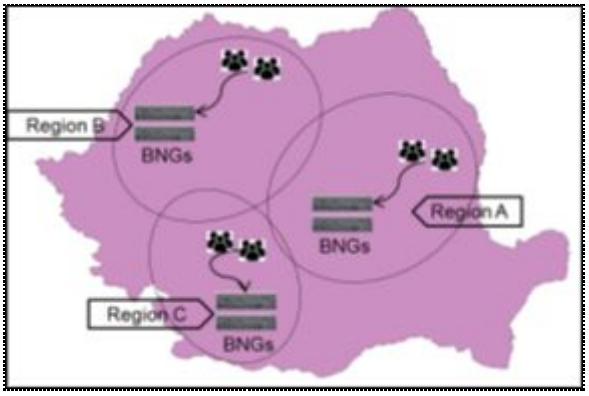

Figure 3.2.1: Orange BNG Use Case Scenario

Currently, a resilient system is deployed at each site location. This consists of two (active/standby) BNGs being deployed for High Availability (HA) purposes. This pair of the BNGs supports all traffic from the region, for multiple services (Internet, VoIP, etc). The network provides connectivity for different customer applications with a capacity of up to 1Gbps along with QoS and user monitoring. The entire system is designed to cope with an estimated traffic load and number of customers, including specific authentication access and service allocation resources. Deployments using this monolithic approach can take up to several months, regardless of the number of customers and specific service requirements and

needs.

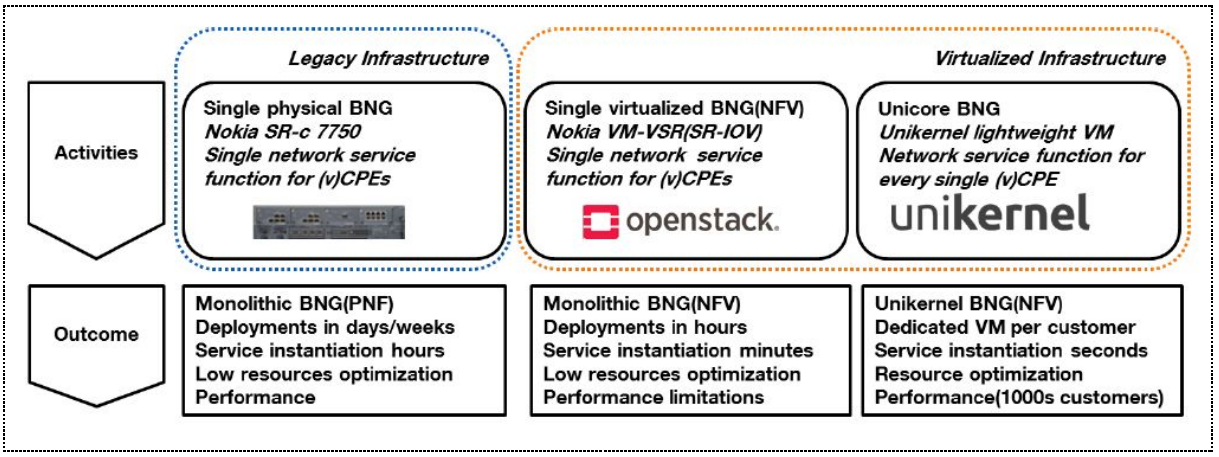

Figure 3.2.2: BNG Implementation Evolution

The next phase implementation will consist of replacing the physical BNGs (PNFs) in the service provider Data Centers, with virtualized infrastructure in an NFV/VNF deployment scenario. This approach of replacing the PNFs with VNFs is primarily designed to improve the deployment time and resource utilisation of the BNGs.

From a service perspective the functionality of both deployment approaches is the same, but there are obvious improvements in deployment times and resource usage. However, the VNF/NFV approach does leave some questions open regarding VM performance, resource utilisation and security. This evolution path is shown in figure 3.2.2

3.2.1.2 Description of the Infrastructure

The legacy architecture, as shown in figure 3.2.3 is a compact physical platform which enables service delivery, high performance, dense interfaces and high user capacity equipment, with comprehensive features and different network functions, using common system modules, a network hardware architecture containing control processing modules, I/O modules for traffic forwarding programming, RIB APIs, filtering capabilities and media adapters modules providing physical interface connectivity. On top of the architecture, several features and protocols are supported, such as IP L2/L3 and MPLS features, Segment Routing and SDN, control interface, management and configuration interfaces as CLI,

OpenFlow, Netconf, YANG models, SNMP, OAM fault and performance operation.

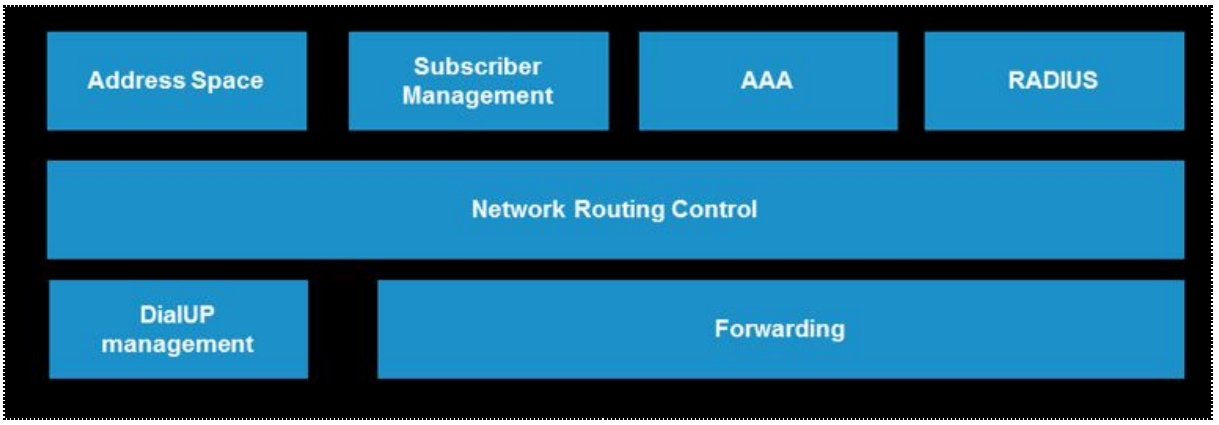

Figure 3.2.3: Legacy BNG Architecture

The first version of BNGs were based on physical equipment handling all control, management and network functions. As mentioned previously, there are several drawbacks to this approach, such as cost a resource allocation, the time taken to deploy equipment, poor scalability and other general drawbacks of this monolithic solution approach.

The BNG virtualized architecture using NFV and VNFS, will enable the decoupling of the software from the underlying hardware. This allows for the separation of the Control Plane (CP) and User Plane (UP). In this scenario, the CP takes responsibility for the user control management component and the UP is responsible for policy implementation and data forwarding. This virtualized BNG will centralise management, is scalable for the management of subscribers and will enable flexible network resource allocation. The functional components, as described in Figure 3.2.4, are implemented as Virtual Network functions (VNFs) hosted in Network Function Virtualization Infrastructure (NFVI). The proposed NFVI model is ETSI MANO NFV. ETSI MANO identifies Service (1), Control (2) and Management (3) interfaces and details the interactions between the CP and UP. This deployment is shown in figure 3.2.5.

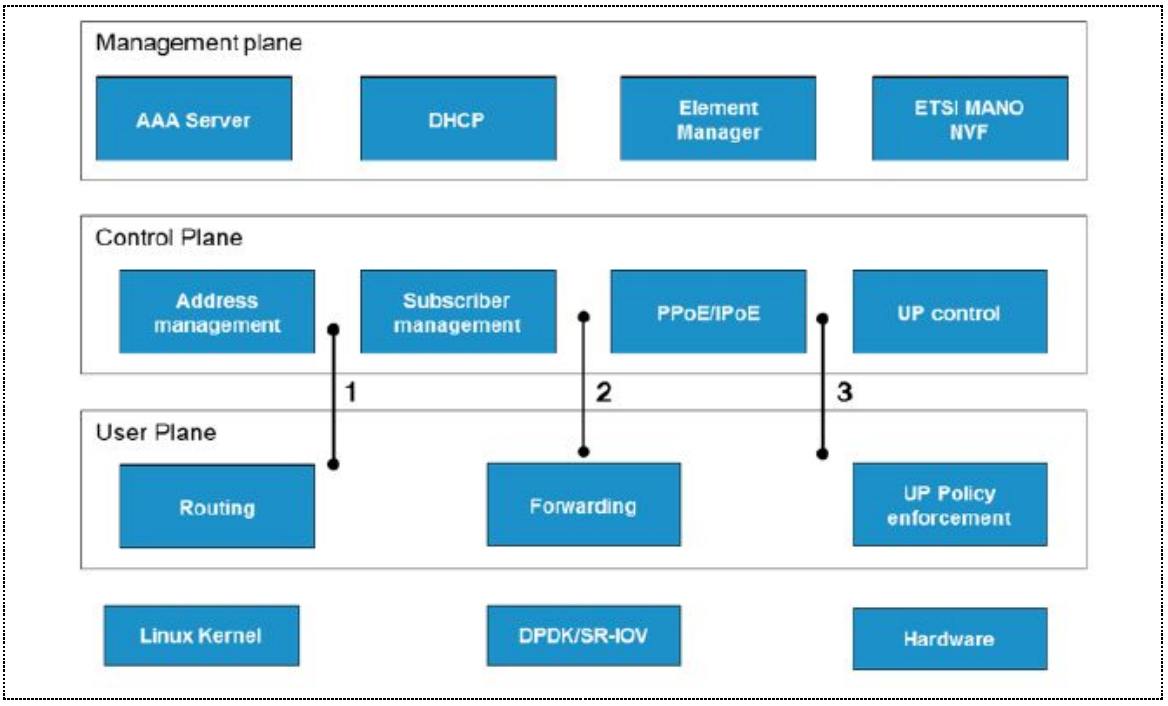

Figure 3.2.4 - BNG Control and User Plane Separation

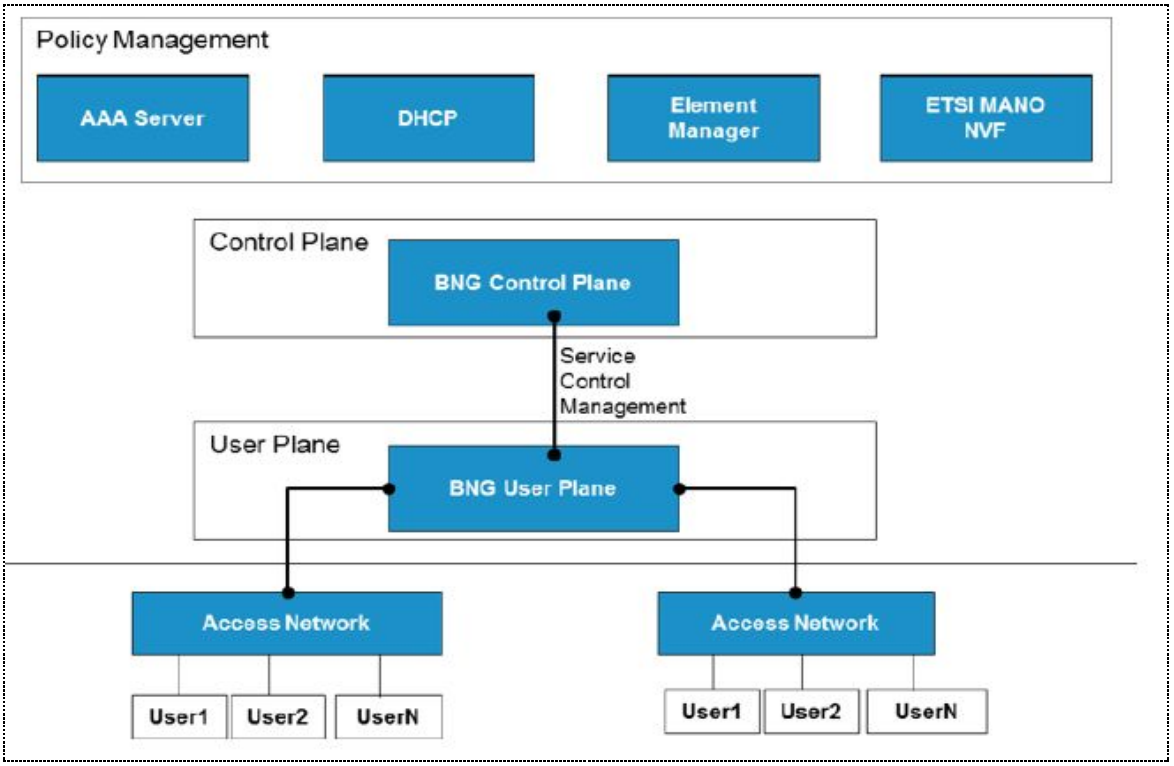

Figure 3.2.5 - Virtualized BNG Deployment Scenario

A migration to a micro-services architecture facilitates more fine-grained distribution of BNG services and allows for more flexible deployment scenarios. In the industry, there is a

movement away from the current use of VM and container technologies towards the introduction of smaller and more scalable NFVs with automation and orchestration and seamless scaling. BNGs deployed in Unikernels would lead to a smaller,faster approach to deploying the BNG application in a cloud infrastructure. As shown in figure 3.2.6, deploying BNG functionality as lightweight applications in unikernels will overcome the limitations outlined in deploying physical infrastructure.

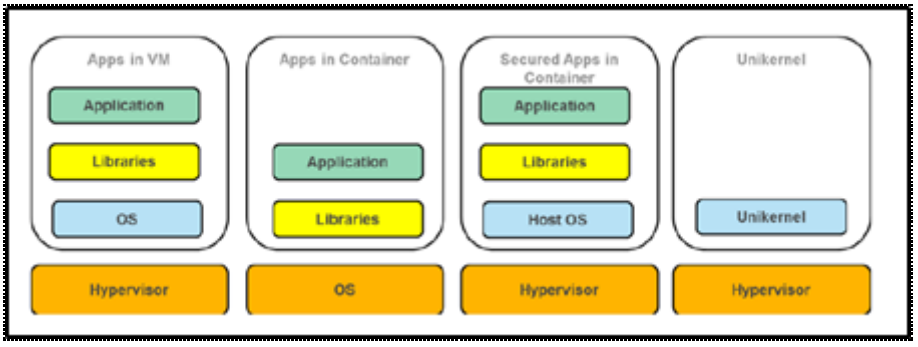

Figure 3.2.6 - BNG Transformation Apps for UNIKERNEL Implementation

3.2.1.3 UNIKERNEL Requirements

From a high level perspective, the Unicore unikernel implementations should provide the possibility of:

- Deployment of lightweight, dedicated VMs deployments (per customer or per CPE).
- Automated fast service instantiation.
- Scalability capabilities for up to tens of thousands of instances.
- Per region service provisioning.
- Tools for monitoring resource allocation, service KPIs and metrics performance.

The entire Unicore system is expected to be supported on a virtualized environment, VNFs with VMs Openstack (with an KVM Hypervisor) based on containerised infrastructure, with the possibility of introducing several open tools for orchestration and service instantiation. Infrastructure platform deployments will support specific integration of lightweight BNG's VMs. The scenario should be extended with the purpose of supporting the testing and use case validation of more than 1000s (v)CPEs, instantiated and connecting to the proper light BNG, into an end-to-end scenario. The performance of this proposed system should respect customer service KPIs and network KPIs, in terms of bandwidth, delay, provisioning time,

capacity, resource consumption and efficiency.

Deployment capabilities: Unikernels should support Kubernetes or Openstack orchestration and 'bare-metal' deployments on X86 Hardware.

Management plane service, resource and service orchestration OSMv5/ONAP, in Openstack/ KVM context.

User Plane forwarding services: deploy on the top of infrastructure up to 1000 vBNGs, per instance an average traffic of 100Mbps, per session flow, 24.000 packets/s.

Unikernel based services on top: UNIKERNEL should, at a minimum, support running of Data Plane Management Kit (DPDK) with performance similar to running on 'bare-metal' x86 architecture. UNIKERNEL should support the latest implementations of the OpenSSL toolkit,inclusive of all dependencies and libraries.

Protocols: Orange's Unikernel implementation shall support, at minimum the following list of network communications and management protocols:

- BGP;
- OSPF;
- ISIS;
- Static Routing;
- AAA Server;
- DHCP

Scalability: A successful overall implementation of unikernel applications should be evaluated when deploying more than 1000s Unicore vBNGs (vCPEs instances), defining several service capabilities and characteristics, providing ranges of speed and QoS profiles, authentication mechanism and traffic restrictions for user, into an isolated and secured light process.

Security and isolation requirements: The ORANGE BNG use case, as implemented on Unikernels, requires advanced security features, at the BNG application level, to minimize the exposure to different security attacks (Unicore by design). Hardware and platform security level are more relevant for a telco operator and this is achieved through the decomposition of a monolithic application into smaller building blocks which can be executed in isolated environments.

As per the requirements documented in Deliverable 2.1, the unikernels used for ORO's use-cases should have a specialized code base, should not accept system call, should be immutable, should provide Address Space Layout Randomization and Stack overflow protection.

3.2.1.4 Orchestration and Management Integration Requirements

For service instantiation and resource configuration on the deployed infrastructure, orchestration tools are mandatory to be used, tools used from the open tool community, such as OSMv5 for resources orchestration, ONAP for resources and services orchestration, in a virtualized (Openstack based) or containerised (Docker based) scenario. The use case implementation is not limited to any orchestration tool and can be integrated to any other component, adopting also service orchestration capabilities. The orchestration process is intended to be adapted and integrated accordingly into the testbed infrastructure, automation and software programmability of the system being seen as a mandatory resource apps block and different APIs implementation for control, management and service instantiation, with some measurable cost reduction in case of development.

3.2.1.5 Description of Business Case

The Unicore unikernels implementation assumes the decomposition of monolithic BNGs into a number of unikernels, with one unikernel per customer. The decomposition starts from the commercial monolithic BNGs service approach, based on the unikernel BNG provided by NEC. The Unicore lightweight BNGs VMs should provide, per customer, at least the same performance as the monolithic one. The change is provided through the physical separation of client's application domains, providing the capability and flexibility to provide customer specific implementation and resource allocation whilst allowing for optimization and fast deployment of the services.

This implementation should allow Orange to deploy virtualized BNGs for each customer, be it B2B or B2C and improve on the current deployment state of performance, power efficiency, security and isolation and management. This can, in turn, drive the creation and operation of new products and services stemming from the decomposed BNG virtualized in UNIKERNELS, as this approach will have better scalability, both horizontal and vertical and

will benefit ORO and its customers from increased flexibility, thus, allowing ORO to derive specific services and products based only on the functionality required by each customer.

Description of Business Model

Orange delivers communications services: internet, fixed voice and TV for both residential and business customers over the fiber optics infrastructure (FTTH) deployed in urban areas and gateway and STBs / CAMs installed by Orange inside the customer's buildings:

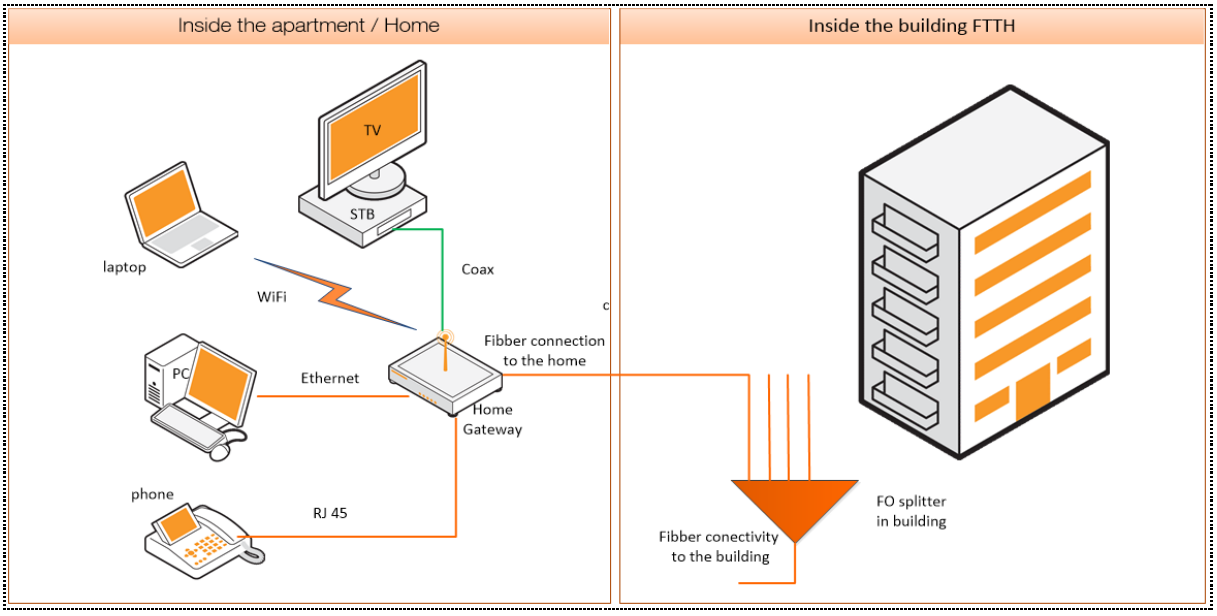

Figure 3.2.7 - FTTH/FTTB Delivery Model of Orange Romania

A complex portfolio Internet services is available, providing both best effort (up to 1 Gbps downlink), and guaranteed bandwidth. Based on the service configuration, customers receive either a dynamic or static IP address. Customers can connect their devices (desktop PCs, laptops, tablets) using either Ethernet or Wi-Fi LAN interfaces.

Fixed voice services are based on VoIP technology, enabling customers to make calls to and receive calls from any destination. New customers are able to port their existing fixed number to Orange, or receive a new number based on their address (county). The service is delivered using a dedicated voice (FXS) port, where the customer can plug a fixed phone. For business customers, the service can also be delivered using IP phones, or a SIP trunk. Additionally, Orange uses fixed Internet as an enablement layer for value-added IT services, such as SD-WAN, Security, Business Wi-Fi.

A per customer unikernel VM is a completely different deployment model. It will enable per customer and network monitoring and resource control, improved capabilities and capacity for customers and the possible application of several alternate subscription models.

3.2.2 Wireless 5G vRAN NFV Clusters

3.2.2.1 Description of the Use-Case

Accelleran has traditionally specialised in developing software for small cells (base stations) for 4G and more recently, 5G mobile networks. Although this software has always been architected for independence from hardware, operating system and third party protocol stacks, to date commercial releases have always been in the form of embedded software running on specialised ODM hardware.

The overall telecom industry trend is to move away from monolithic software running on specialised hardware towards cloud native applications running on COTS or cloud servers such as AWS, GCP etc. These new deployment options, apart from bringing improvements in computing power, scalability and redundancy also enable new types of services such as Radio Access Network (RAN) slicing, improved network manageability, AI driven automation and edge computing.

Many of these NFV and RAN disaggregation concepts, that are foundational to 5G network deployments are also applicable in 4G environments.

Accelleran has taken its existing embedded software and demonstrated that it can be adapted to run in virtual computing environments with relative ease, leading to its planned dRAX™ product offering. The outline dRAX architecture is shown in figure 3.2.8 The initial dRAX™ solution is composed of the following services:

- dRAX™ RAN Intelligent Controller (dRIC)
- dRAX™ Information Base
- dRAX™ Data Bus
- Cell Control Plane

All of which can run on a general compute server.

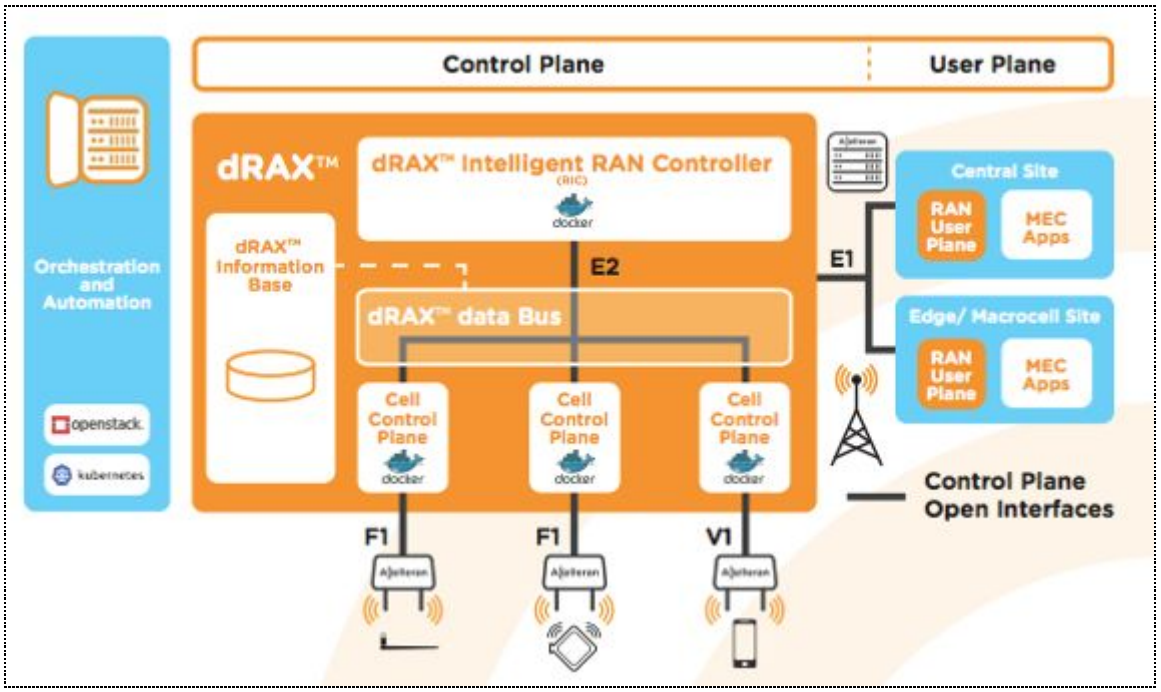

Figure 3.2.8 - Outline dRAX Architecture

Accelleran has identified two Use Cases where the use of unikernels is warranted. The first Use Case revolves around the Intelligent RAN Controller (RIC) which is shown in figure 3.2.8 above. The RIC is responsible for network manageability and can is responsible for aspects such as Admission Control, Handover, and Self Organising Networks (dynamic network configuration without the need for human intervention). Historically such features were implemented using static algorithms which were compiled into the deployed software. In an NFV deployment, where software components can be deployed at a much more granular level, it becomes possible to deploy these algorithms in a more dynamic manner e.g. as standalone applications running in Unikernels. The ability to develop Unikernel based standalone network management applications and potentially hot-swap them into a running network raises several new possibilities that would not have been possible in a traditional monolithic software deployment. For example, in the case of handover, the base software could be deployed with a fairly standard handover algorithm. A premium handover algorithm could be licenced with, say to focus on overall system load-balancing or average user throughput. This algorithm (Unikernel application) could be hot-swapped into a running network without the need for a software upgrade or any network outage. Such a deployment model also allows for proprietary algorithm development e.g. a network operator could decide to implement and deploy a tailored AI based SON or network interference mitigation

algorithm and again the Unikernel deployment model allows for such an algorithm to be hot-swapped or trialled into an operation network.

The second Use Case revolves around software scalability. As mentioned above, traditionally RAN software was deployed in a monolithic way on ODM hardware. In such a deployment scenario, the granularity and dimensioning of individual components is known and can be hard-coded and configured in advance e.g. a 4G small cell might support 64 parallel active users and the software should be tailored accordingly so as not to exceed hardware limitations. As software is virtualised it is no longer reasonable to hard-code such limitations. A typical 4G or 5G (or combined) network deployment might consist of one core network, a number of base stations and a number of users. Historically, such a network deployment would be planned carefully in advance and a poorly planned deployment could lead to network congestion on the one hand or deployed but under-utilised (expensive) equipment on the other. Modern network deployments are expected to be much more dynamic; It should be possible to add new network components on demand and the software components controlling these components should also scale dynamically.

Accelleran has been rearchitecting its software with this type of scalability of deployment in mind. Our software needs to be able to handle the dynamic addition (or removal) of Core Network connections, base station control software and per user context handling software blocks. In this Use Case Accelleran plans to deploy Unikernel based software modules/microservices to handle this dynamic scalability.

3.2.2.2 Description of the Infrastructure

Figure 3.2.8. shows the 3GPP reference RAN architecture for 5G network deployments. The complete architectural description can be found in [5]. Although the figure refers to 5G, conceptually this architecture is also applicable to 4G deployments (although in the case of 4G interface names would be different)

The key components shown are:

- 5GC: 5G Core Network
- gNB-CU: gNB Central Unit
- gNB-DU: gNB Distributed Unit

The gNB-CU and gNB-DU are connected via a logical F1 Interface. Figure 3.2.9 does not show the Remote Radio Unit (RRU) which provides the fronthaul to the network. The RRU connects to the gNB-DU via a CPRI/eCPRI interface.

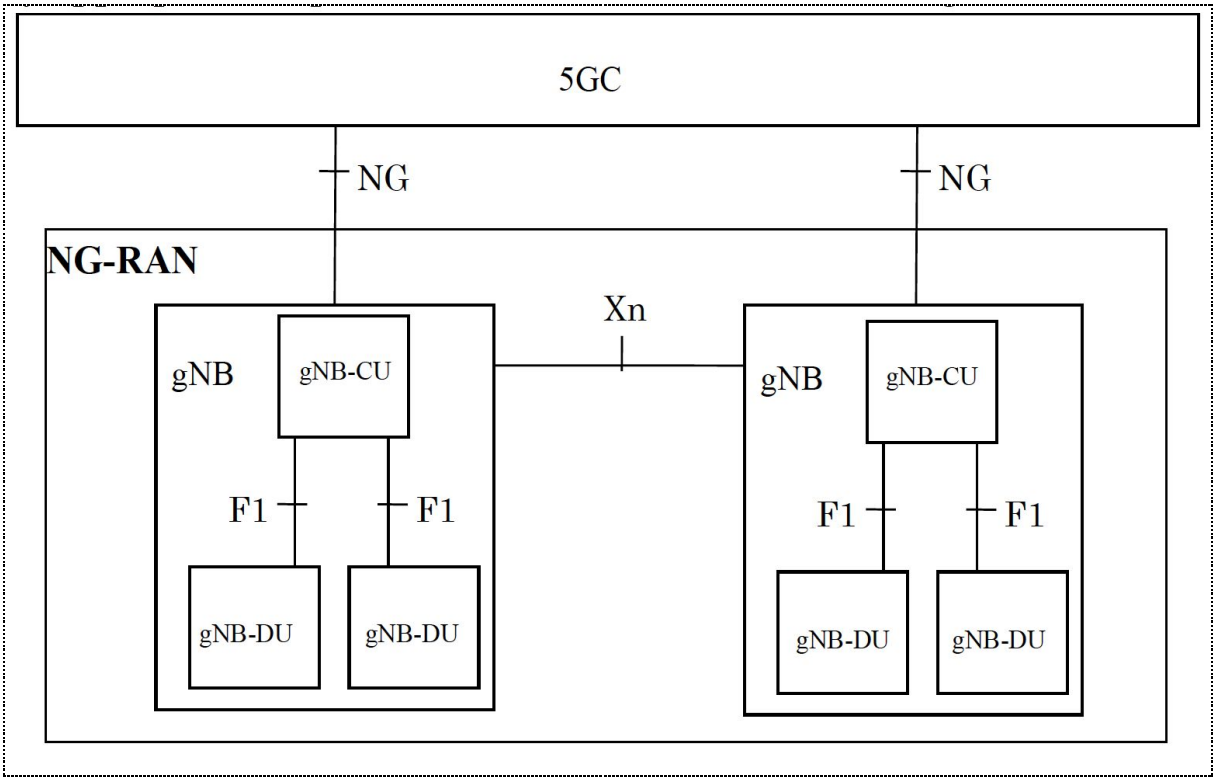

Figure 3.2.9 - 3GPP Reference 5G Network Architecture

In a typical 4G deployment the CU, DU and RRU functionality would be co-deployed on proprietary ODM hardware. In the architecture shown in figure 3.2.9, the RRU and DU and CU can can still be co-located or they can be deployed separately e.g. the RRU and DU can be separated by up to 10 kilometers and it is foreseen that the DU and CU can be deployed up to 40 kilometers apart. The 5GC (or EPC in 4G terminology) is typically located separately. Accelleran's primary focus is on CU software.

3.2.2.3 UNIKERNEL Requirements

The requirements in Unikernels can be split into both functional and non-functional requirements. The functional requirements ensure that the behaviour of the software is the same when running in a Unikernel as when it is deployed on ODM hardware or in a Docker container.

Functional requirements:

- Openssl
- Libc

- Libpthread
- Netconf Client
- Zlog
- Python 3
- NATS
- Redis client

Non-functional requirements:

- **Performance**: Performance should be no worse than is achieved when the software executes on ODM hardware i.e. no degradation in performance
- **Scalability:** The Accelleran CU software can run on standard x86 or ARM based processors so it would be anticipated that we could execute Unikernels based on either architecture. As such, scalability should really only be limited in terms of the number of physical (or virtual) servers that can be deployed. Typically the entities that may need to be scaled in a given deployment would be Core network connections (up to 6), Cells (up to 50) and UEs (up to 512 per Cell). It is difficult to state without doing some performance analysis, what number for servers would be required to meet these scaling requirements. For the purpose of scaling there is no requirement that scalable entities are co-located on the same server i.e. if necessary UEs, Cells and network connections can be load-balanced across different servers.
- **On demand deployment:** Ideally, it will be possible to spin up Unikernels on demand i.e. there would be no need to pre-configure a number of Unikernels in advance of them being actually required. In reality in a RAN network deployment, timing requirements become tougher the closer one gets to the radio interface (RRU). In practice, when spinning up new Core Network connections or Cells there can be a bit of tolerance in the start up time i.e. some 10s of additional milliseconds will not have a significant impact on the system. However, UE connections need to be handled in different timescales. When a UE signals that a connection is required then such a new connection should be handled quickly e.g. a delay of milliseconds could be tolerated but a delay of 10s of milliseconds could not. Again, it would be necessary to analyse Unikernel deployment times to determine whether on demand deployments per UE are possible or whether a preconfiguration of a pool of UE handling Unikernels would be required.

3.2.2.4 Orchestration and Management Integration Requirements

In a typical network deployment, Accelleran will not be in a position to dictate or proscribe a particular orchestration or management system. Software will typically be deployed in operator networks on operater-owned server hardware. As such, we would need to support any industry-mandated orchestration mechanism. Currently, Accelleran are using Kubernetes for orchestration

3.2.2.5 Description of Business Case

The Accelleran NFV use cases involve the disaggregation of a relatively monolithic software block into a number of Unicore unikernels. Two use cases are considered; applications interfacing with the RIC and Unikernels specific to aspects of the RAN CU networking. As mentioned previously, the overall telecom industry trend is to move away from monolithic software running on proprietary hardware to disaggregated software running on COTS hardware.

In reality the current state of the art is still relatively large monolithic software blocks running in virtual machines or, in some cases, docker containers orchestrated by Kubernetes. NFV through the use of virtual machines is not really a viable option as it does not scale and docker deployments have well-known security issues which are unlikely to find much traction in an industry that is very security conscious.

Unikernels should allow Accelleran to deploy software at a level of granularity that bypasses the current state of the art while also addressing industry security concerns.

3.2.3 EKINOPS NFV

Ekinops has distinguished two use cases in the context of Unicore project known as the SDWAN Controller Key Server as unikernel and the unikernel based vCPE.

3.2.3.1 Description of the Use-Cases

The first use case introduces a unikernel Key Server in a Software-Defined WAN (SDWAN) infrastructure composed of three layers namely:

- ***Director:*** A centralized web portal offering multi-access and allowing Service Providers (SPs), partners and customers to run and operate an SDWAN network. The Director can be managed on its North-Bound Interface (NBI) by a third-party via REST-API on one hand. On the other hand, it manages the *SDWAN Edge* via the *Controller* on its South-Bound Interface (SBI).
- ***Controller:*** Responsible about the Edge devices management (authentication, activation, IP config, IPSec key management, and traffic policy distribution). It interacts with the Director on its NBI APIs based on Open Netconf. This Controller is natively managed by the Director and can also be managed by a third-party management system. Regarding its architectural design, the Controller is composed of several µ-services, called µ-controllers, each of which is responsible for a specific management function (e.g., Route Reflector, Key Server, and Bootstrap Server). Our interest will be hence, on the Key Server µ-controller.
- ***SDWAN EDGE:*** where the SDWAN tunnels are initiated/terminated. Creates and terminates secured (encrypted) tunnels over wired or wireless networks (e.g., DSL, Fiber, LTE, MPLS).

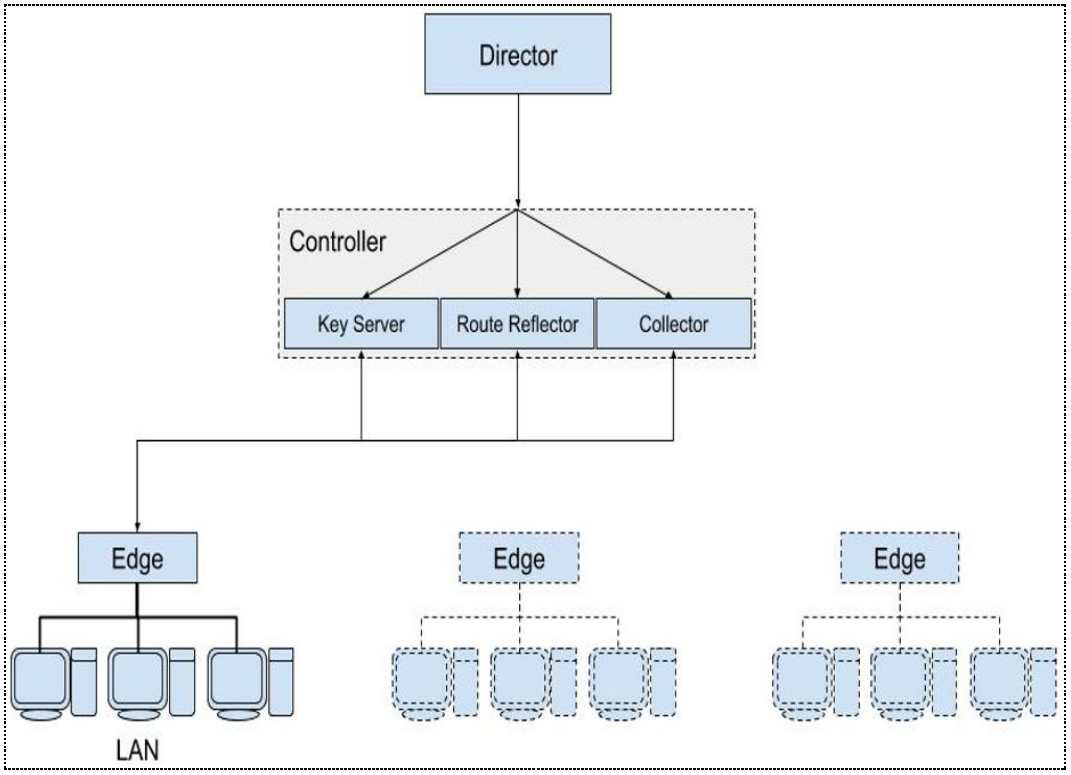

Figure 3.2.10 - (Simplified) view of the Ekinops' SDWAN Solution

The second use case that Ekinops is working on is unikernel based vCPE.

The imminent arrival of 5G will drive changes in communications service provider networks, leveraging on Network Function Virtualization (NFV) and Software Defined Networking (SDN) technologies. The upcoming 5G ecosystem will address vertical markets to give rise to a plethora of novel services with different requirements, such as Ultra-Reliable Low-Latency Communication (URLLC), machine Massive Type Communications (mMTC), and enhanced Mobile Broadband(eMBB). To efficiently accommodate all these needs, nowadays' networks require architectural enhancements. Routers are one element that Ekinops is working on in order to provide better performances that meet the KPIs defined by the 5G Infrastructure Public Private Partnership (5GPPP) and other SDOs. In this regard, Ekinops has developed OneOS6, a virtualized version of their router. Even if this solution is scalable, on-demand deployable as VNF, Ekinops needs to push their solution farther to be more competitive. Unikernels is the right solution since a unikernel version of OneOS6 will save the cost of deployment, decrease the resource consumption, and highly increase the scalability.
OneOS6 design is composed of three planes; the management plane, the control plane and the data plane. In the management plane, we find a conf server that handles Command Line Interface (CLI), and Graphical User Interface (GUI). The control plane is composed of several daemons for routing, security, and netflow. This plane receives configuration from the netconf server via the Transmission Control Protocol (TCP) channel (i.e., socket). The communication between the control plane and the data plane is based on a shared memory and handled by a standard MultiCore Communications API (MCAPI) library. The data plane has several processes, each of them being a Data Plane Development Kit (DPDK) based running as legacy Linux threads pinned to an isolated CPU. Figure 3.2.11 depicts the design of the OneOS6 architecture.

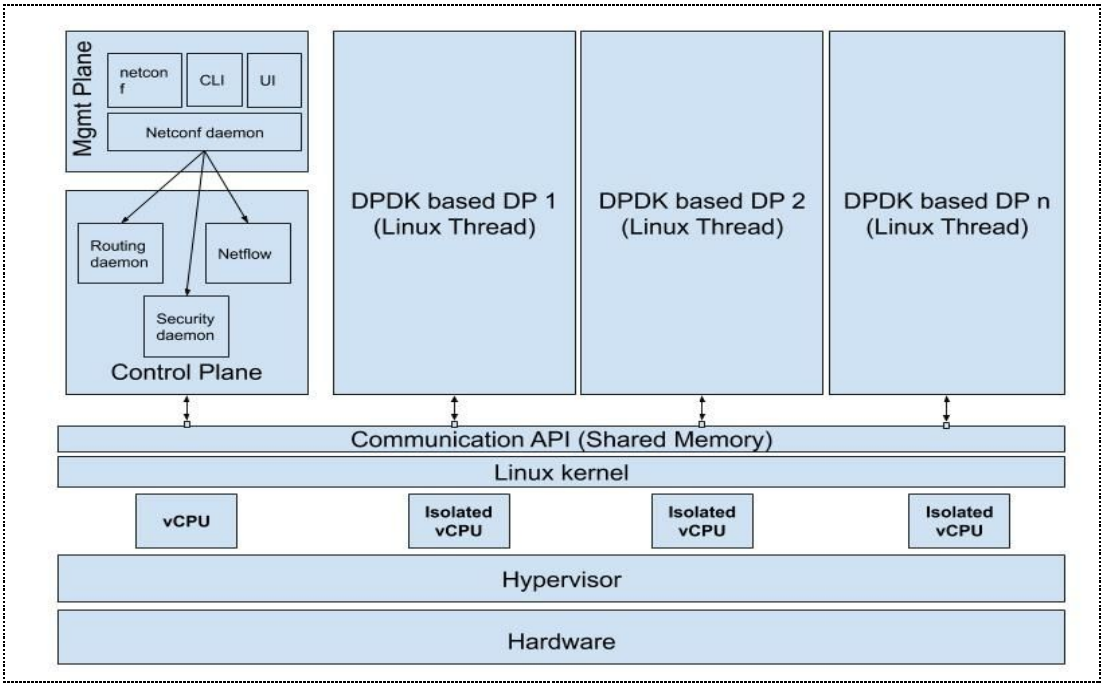

Figure 3.2.11 - vCPE architecture design

3.2.3.2 Description of the Infrastructure

For the SDWAN use case, the infrastructure consists of physical CPEs, a Gateway between the Internet and MPLS network, and an SD-VPN controller. The purpose of the use case implementation is to include a unikernel key server in the system to establish VPN tunnels between CPEs on the Internet and the MPLS network in an automated way. For this purpose the VPN Controller manages the encryption keys to be used when new VPN tunnels need to be instantiated between CPEs or between CPEs and the Gateway. When a new CPE is added to the network it first establishes a secure and authenticated connection to the VPN controller. The controller provides a common encryption key to be used for the creation of VPNs. The Gateway advertises this routing information with BGP to the rest of the MPLS network.

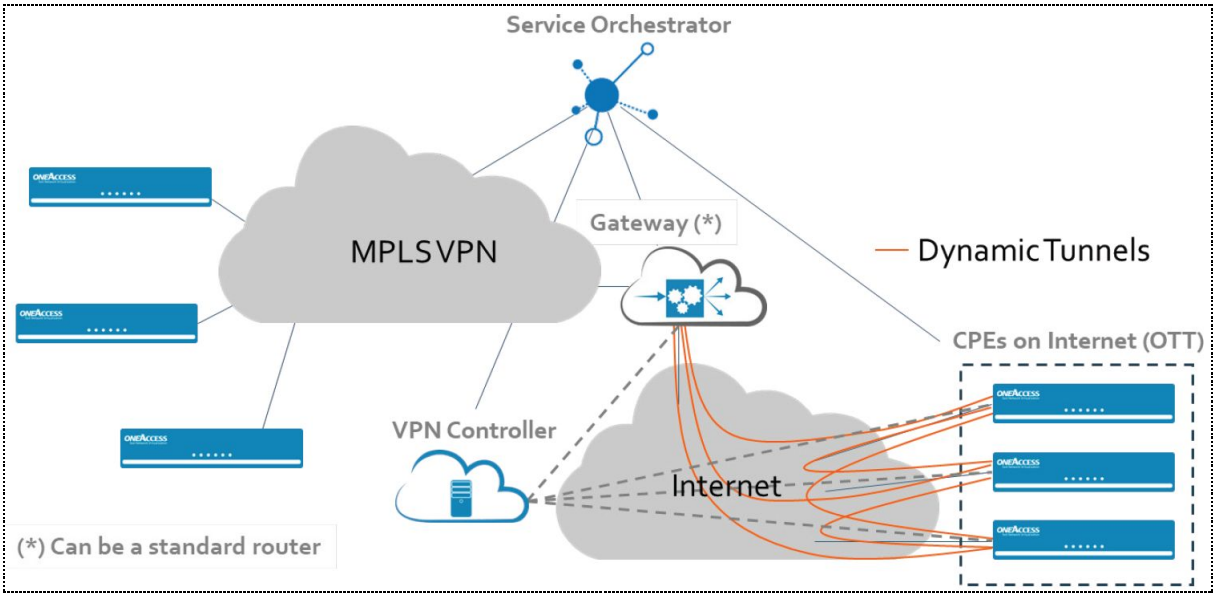

Figure 3.2.12 - Ekinops' SDWAN TestBed Infrastructure

The aforementioned Key Server is a μ-controller functionality of the SDWAN Controller that we want to export as Unikernel. It is responsible for group member authentication and handles the distribution and renewal of encryption keys.

While for the vCPE use case, the deployment platform that will be used is a laboratory-based platform. It is composed of two connected physical CPEs. The first CPE hosts two VMs, the first one is a Debian distribution VM which represents the OneOS6 control plane, while the second VM is a unikernel for the data plane. The second machine is a TRex[6] . TRex is an open source realistic traffic generator, low cost, stateless and stateful generator of traffic fuelled by DPDK. TRex is used to measure the maximum sustainable throughput of a Device Under Test (DUT) under certain conditions. The DUT here is the OneOS6.

3.2.3.3 UNIKERNEL Requirements

Regarding the requirements that Unikernels should satisfy for both the SDWAN and vCPE use cases, we have classified them into two categories; the functional requirements and the non-functional requirements. The functional requirements represent all the utilities that are needed by our use cases in order to have the same behaviour when the Key Server/vCPE is deployed as a container, a traditional VM or as bare metal. These utilities include the libraries whether low level or application level ones including the following:

- Openssl
- Freeradius-client
- Libc
- Libpthread

- Sqlclient
- Lwip
- DPDK
- mariadb

For the non-functional requirements, we are expecting that unikernels will bring better performance than in traditional ways wherein the key server/vCPE is deployed as container, VM or even in bare metal. Therefore, Unikernels should satisfy a list of requirements that are described hereafter.

- **Scalability:** Deploying the key server as a unikernel should provide high scalability in terms of number of edge devices that can be managed by one key server (approximately, 1000 devices), but also in terms of number of key servers that can be instantiated on the same physical machine which should be high. In the same way, with vCPE as a unikernel, we should be able to route thousands of packets per second, but also be able to instantiate a large number of vCPE unikernels on the same physical machine.
- **On-demand, transparent deployment:** The generated unikernels (for both key server and vCPE) should behave the same way as standard VMs. Indeed, these unikernels deployment should follow the European Telecommunication Standards Institute (ETSI) Network Function Virtualization (NFV) principles, this is to reuse the NFV Management and Orchestration (MANO) framework. Hence the unikernels should be deployed using orchestrators (such as OSM, Cloudify, and ONAP), over a virtualized platform using a Virtual Infrastructure Manager (e.g., Openstack, OpenNebula, KVM, and Kubernetes).
- **Low SER:** As the Unikernel follows the NFV principals, both the key server unikernel and vCPE unikernel Service Creation Time (SER) should be very short (in term of seconds). This time includes the deployment of the unikernels and their boot. This deployment should also support the scale-in and the scale-out with various flavours in a very short time.
- **Low resource consumption:** Using unikernels rather than VMs should bring important gain in terms of resource consumption (CPU, RAM, and Energy). Indeed, as unikernels are lightweight VMs, we are expecting a very low consumption in comparison with VMs. This should bring gain with a factor of at least x10.

- **High performance:** Using unikernels should not affect the performances of both key server and vCPE. Indeed, because the unikernels are lightweight VMs with less libraries and utilities than standard Linux VM, this should not impact the performance of the key server/vCPE. On the contrary, it should bring better performance in regards to the number of unikernels that can be instantiated on the same physical machine.

3.2.3.4 Orchestration and Management Integration Requirements

In what concerns the orchestration and management of these use cases, Ekinops, at the time being, has not a constrained platform with a specific orchestrator or infrastructure manager for these use cases. However, this may change in the future by creating a trial platform, which should fellow the ETSI MANO framework.

3.2.3.5 Description of Business Case

Introducing unikernel addresses the need for more efficiency and performance which are becoming strong market requirements since SDWAN components are virtualized and their resources requirements are not negligible.

SDWAN solutions leverage recent developments in networking technology to provide enterprise customers with a more flexible approach for building and operating WAN networks. IT departments of enterprises are increasingly challenged to provide more agile network services supporting the enterprise in their digital transformation journey. Traditional Communication Service Providers (CSPs) services do not offer the flexibility, optimization and time to market required to meet these challenges. Using a combination of VPN technologies, Hybrid WAN, Application Awareness and Centralized Orchestration, SDWAN solutions provide enterprises the means to become more independent from existing CSPs and build their own private overlay network on top of existing Multi-Protocol Label Switching (MPLS) and Internet services in a more flexible and optimized way. The Centralized Orchestration of the SDWAN network also puts the enterprise back in control over large parts of its network and on-the-fly re-allocation of network resources provides enterprises the needed flexibility to cope with the dynamic nature of their businesses.

Thanks to its lower resource requirements/cost, unikernel SDWAN and vCPE propositions will be more appealing for small and medium sized companies interested to operate their WAN in a more agile and optimized way.

3.3 Home Automation and IoT

3.3.1 Description of the Use-Case

The Home Automation use case is based on Symphony, the Smart Home and Smart Building Management platform by Nextworks. It integrates home/building control functionalities, devices and heterogeneous sensing and actuation subsystems, allowing, among others, the creation of scenarios for the control of lighting, doors, climate, etc. As previously described in deliverable D2.1 nad D2.3, Symphony can communicate with a large number of commercial automation controls, and it integrates different field protocols under a coordinated, unified management middleware .
The migration of some functionalities to unikernels might be an interesting evolutionary step for the Symphony platform in the direction of optimization of the split of its business logics into unitary services and lightweight image footprints. These expected benefits can ease more dynamic and faster upgrades, as well as dynamic configuration of services.

As described in deliverables D2.1 and D2.3 and given the complexity of the Symphony IoT platform which spans from the dometic field bus controls to Human Machine Interaction, the functions which can be more reasonable to port to unikernel include

- somedomotic or automation protocol gateways (e.g. MQTT, Zigbee, Z-Wave, Bluetooth LE);
- functions for data storage;
- specific network functions (e.g. local routers for NAT/Firewall);
- specific media service gateways and/or voice/video communication handlers which are targeted to be hosted on constrained devices (e.g. Raspberry Pi, micro-PC, etc.).

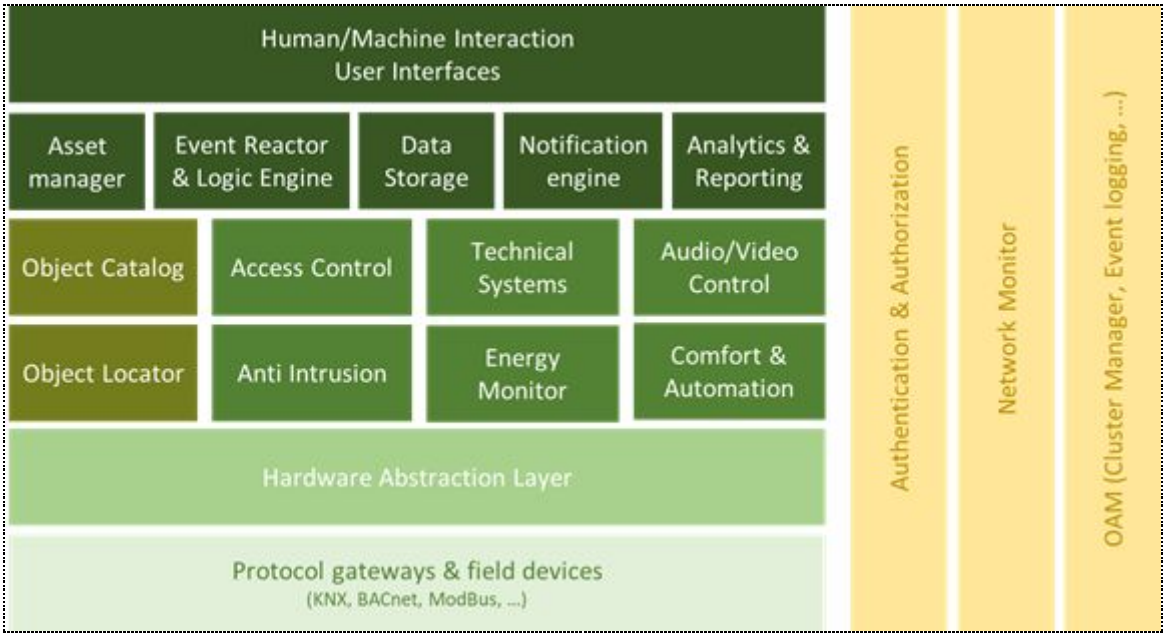

Figure 3.3.1 - Symphony Building Blocks

The deployment and validation strategy of Symphony functions with UNICORE will follow the three stages introduced in deliverable D2.1 and D2.3:

- ***Stage 1. Functional validation***. Core focus of this stage is the validation of the krafting aspects and the verification that the resulting unikernel implements all the functions the origin Symphony element can offer.
- ***Stage 2. Performance evaluation.*** This phase will consider performance aspect in order to evaluate if the implementation via unikernel can maintain or even exceed some specific performance indicators e.g. in terms of processing (speed, number of operations), throughputs at network interfaces, time to boot, etc.
- ***Stage 3, Automation and Upgrades***. This final stage of experimentation will evaluate the feasibility and potential benefits of an automatic deployment of the Symphony Building Management System (or parts of it) through distributed controller nodes in which unikernels are generated at run-time, taking into consideration characteristics, constraints and location of the available hardware nodes.

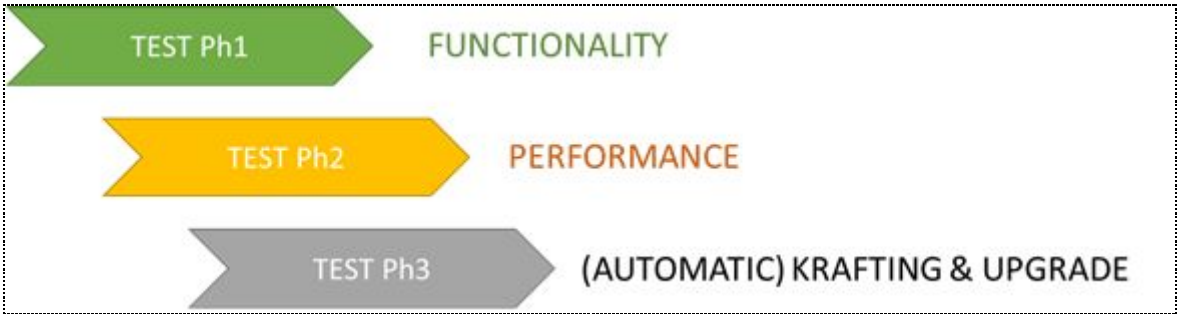

Figure 3.3.2 - Testing stages

The current plan for porting of Symphony IoT functions includes :

1. Symphony Event Reactor (ER);
2. Symphony MQTT driver (MQTT);
3. Symphony environmental sensors driver (F1);
4. Symphony lightning driver (F2);
5. Symphony energy monitor (F3);
6. Network firewall (F3);

Following the three-scenario approach described above. The process has been divided into different phases, from the selection of the component to be ported till its final integration and testing, in order to draw a time plan according to the project deadlines.

In the first stage, the pre-start, we have analysed the candidate services and all of their technical requirements. This includes taking into account all of the dependencies that Symphony comes with. It might require modules (e.g. CORBA), languages (e.g. Go), libraries that could not be supported by Unikraft at the time we write this deliverable. This implies that either adaption will be needed or even some components will have to be dropped, due to the impossibility of the migration.

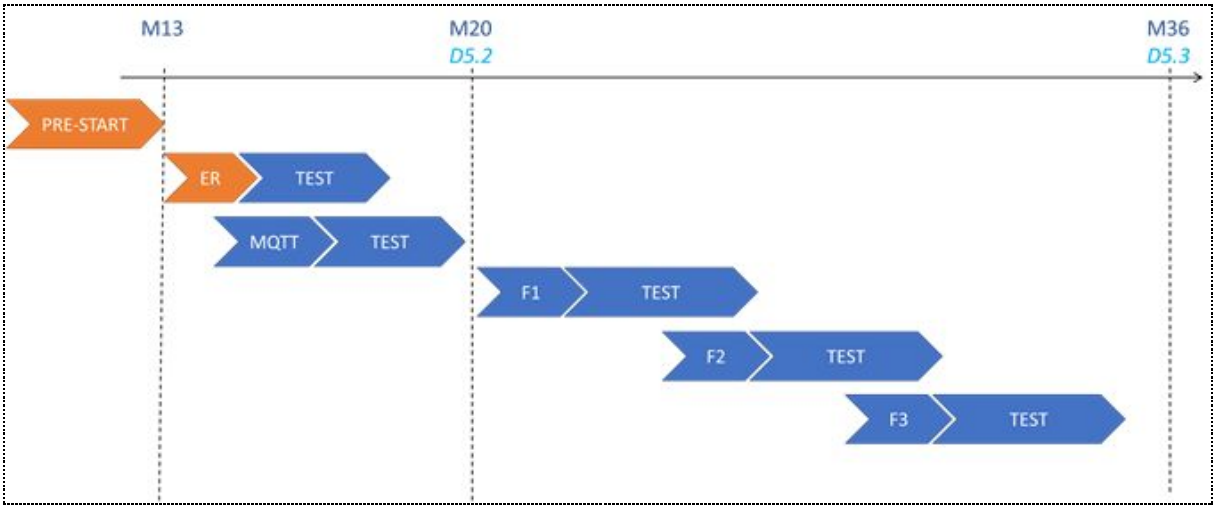

Figure 3.3.3 - Smart Home UNICORE porting time plan

For the deployment phase 2, which targets the release of D5.2 in M20, we plan to complete the work and validation of the the Event Reactor function and to also complete the MQTT driver. For the deployment phase 3, which will be completed by M36 with deliverable D5.3, we will complete the rest of the functions listed above.

3.3.2 Description of the Infrastructure

The testing infrastructure for this use case will be the Nextworks' premises in Pisa (Italy). Nextworks offices are placed in a (almost) completely automated building, equipped with many different IoT devices, all controlled by Symphony platform. The infrastructure made available to the project use case will be selected from this actual installation and it will depend on the technical requirements of the demonstrated scenarios.

The following pictures depict a map of some installations in the building:

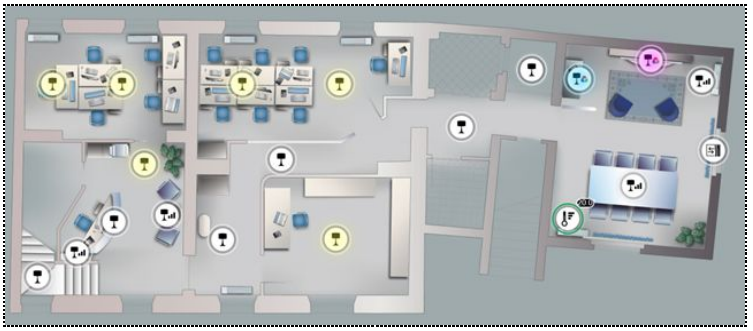

Figure 3.3.4 - Ground floor installation layout

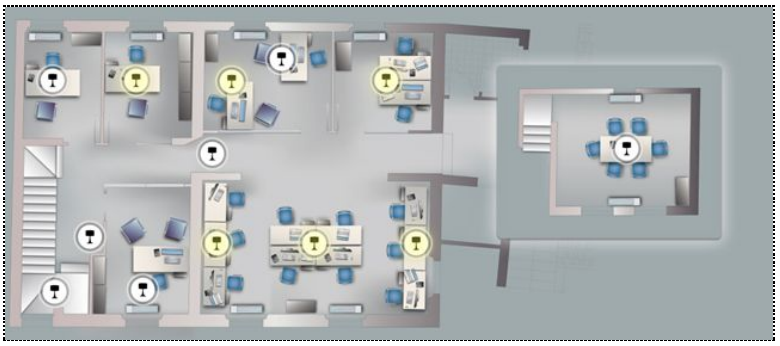

Figure 3.3.5 – First Floor Installation Layout

Figure 3.3.6 – External Area Installation Layout

The testing environment includes up to 50 types of sensors and actuators, ranging from Comfort Living sensors (e.g. Lighting switches/dimmers, curtains / blinds, thermostats, electricity meters, etc.), to Security (e.g. motion detection sensor, smart floor sensor readings, room/building accesses, etc.), Environment (e.g. indoor/outdoor temperature, pollution, etc.) and so forth..After an initial phase of selection of the IoT devices to be integrated, the related control modules will be ported into unikernels via UNIKRAFT and then deployed in the Symphony infrastructure to run integrated with the rest of the system. The targeted will run in both servers with limited computing power, such as Raspberry PIs, APUs, etc. and in virtualization infrastructures in the form of very lightweight virtual machines.

3.3.3 UNIKERNEL Requirements

The initial selection of Symphony functions to be possibly “unikernelized” has allowed to identify and confirm the requirements towards unikernels initially identified in deliverable D2.1 and D2.3.

In fact, during the first stage (Phase-0 as per Section 2) we have analysed the candidate services and all of their technical requirements. This includes taking into account all of the dependencies that Symphony comes with. It might require modules (e.g. CORBA), languages (e.g. Go), libraries that could not be supported by Unikraft at the time we write this deliverable. This implies that either adaption will be needed or even some components will have to be dropped, due to the impossibility of the migration. The first component which has been evaluated for porting to unikernel is the Symphony Event Reactor (see description in deliverable D2.1). It triggers actions and alarms (e.g. act on notification, act on explicit sensors’ values, etc.), logs and manages alarms lifecycle, through two submodules:

- Event Manager – written in Python
- Alarm Manager – written in C++

It also uses Blockly [7], an open source client-side JavaScript library for creating block-based visual programming languages (VPLs) and editors, as a GUI for connecting events and reactions to events.

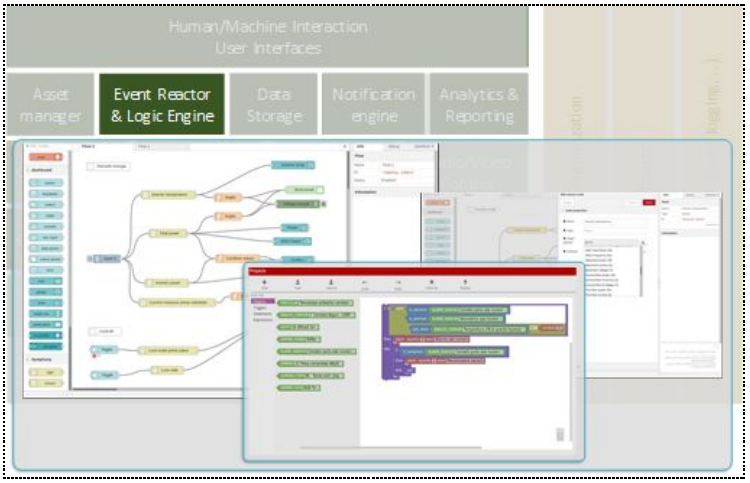

Figure 3.3.7 - Human/Machine Interaction User Interfaces

Three reasons have led us towards Event Reactor (and in particular to Event Manager) as the first module to work with unikernels:

- It is implemented in Python
- It has REST API in addition to CORBA interfaces
- It has reduced number of dependencies/interfaces with other modules

Nevertheless, not all the dependencies are satisfied:

- *Tornado* and RabbitMQ (*pika* library) are supported
- *Python2.7* is not supported, so the component had been ported to *Python3*
- *OmniORB* (CORBA python library) was not supported and it had removed
- Additional 2nd-level dependencies are ongoing to be investigated

Concerning more specific use-case related functionalities, we expect the UNICORE toolkit and krafted unikernels to:

- implement the same functionality across the current interfaces (at least string-based TCP protocol interfaces);
- support the same number of messages (e.g. AMQP, MQTT) with respect to standard containers or VM solutions;
- support the same number of network flows in network functions (e.g. NAT translations, firewall rules, etc.) and packets processed per second in similar conditions of assigned resources;
- support automated packaging of Unikernel functions with variable configuration profiles, to allow automatic on-demand spawning of functions for service scaling or event-based processing;
- lower resources consumption for the Symphony middleware to allow it to fit into small-scale computing elements (e.g. domestic NAS);
- lower time for delivering software upgrades in field installations to reduced footprint images (unikernel-based) and automated build procedures.

3.3.4 Orchestration and Management Integration Requirements

One of the intentions is to deploy unikernel-based services in containers or light VMs to be orchestrated and managed via ProxMox (https://www.proxmox.com) or Kubernetes (https://kubernetes.io). The target reference processor architecture for all the selected functions is x86.

This represents the main requirement towards orchestration from this use case.

3.3.5 Description of Business Case

As described in deliverable D2.3, the idea behind this use case is to create a "light" version of the Symphony platform, that could be offered to customers with constrained hardware to support a subset of basic automation functionalities.
As previously described, Symphony integrates a large number of protocols and technologies, providing different services for domotics and entertainment on the top of these. The process of migrating to Unikernels will foster the decomposition of the platform in a set of (partially) independent software components. These will be then re-assembled together basing on the user needs, in order to provide quick and lightweight solutions, in all the cases where the full set of platform features is not strictly required.
This new flexible and modular Symphony version can address all the small customers that might require the implementation of specific functions tailored on their scenarios.

3.4 Smart Contracts

Smart contracts are snippets of code that execute in a distributed environment with potentially malicious parties. Every deployment node in the distributed environment (or just some nodes, in environments that use sharding) runs a given smart contract every time some user provides input for that particular contract. There are two actors that could be malicious: either the deployment nodes that run the smart contract or the users creating the smart contract. Because the nodes can be malicious, the user cannot trust any single node to correctly run the smart contract. The nodes themselves do not trust each other either. Thus, the user and nodes only trusts a piece of information (input, output, smart contract code)

when a majority of nodes agree with that particular information – we say the nodes reach consensus on that particular information. Because users can be malicious, we need to protect the nodes from users who provide malicious code or different code versions to different nodes for the same smart contract.

Nodes store in the blockchain information that a majority reached consensus on. A blockchain is an append-only data structure containing an ordered collection of blocks. This information stored is, for example, the smart contracts themselves that the users provide, the inputs that the users provide and the outputs obtained after running the smart contract on some inputs. Typically the deployment nodes store the blockchain, but it could be any other set of nodes.

3.4.1 Description of the Use-Case

The DEDIS lab at EPFL has developed through the years a blockchain framework[8] that implements efficient deterministic consensus protocols[9], such as ByzCoin[10] and Omniledger[11], and multiple tools that make use of the blockchain, such as e-voting or a private storage. One of these tools is the support for smart contracts execution.

Every deployment node in the distributed environment, the so-called *conode* in the DEDIS framework named Cothority (Collective Authority), runs a given smart contract every time some user provides input for that particular contract. In our framework There are five steps in running the smart contract: (1) Obtaining the smart contract; (2) Obtaining the input for an execution of the smart contract (also called transaction); (3) Running the smart contract on that input; (4) Agreeing on the result; (5) Persisting the result of the execution, which can then be retrieved by the user.

Currently the smart contracts written by the lab are bundled within the conode binary, which allows a very efficient and controlled execution. This is important as many executions of a smart contract with the same inputs must produce the same output, otherwise the consensus between participants cannot be reached and, thus, the transactions are not successfully executed.

In order to better open the framework to industrial partners, DEDIS aims to extend the support of smart contracts with the possibility of executing arbitrary code written in generic languages. This last point is important as we want developers to be able to use existing

libraries. By using Unikernels, the framework can provide support for generic smart contracts like Ethereum does, but with the difference that the developer is not limited by the features of the language and can efficiently and conveniently write code to be executed on the blockchain by reusing existing libraries. For example, zero knowledge proofs are not possible in Ethereum because of language and gas limitations, but would be possible in our framework.

3.4.2 Description of the Infrastructure

The Cothority framework is meant to be deployed on multiple servers that will participate in the network. One important aspect of this network is the heterogeneity of the architectures of the different conodes. Because there is no central party controlling the conodes, each conode independently decides the operating system running on the conode, e.g., a standard x86_64 based operating system or a Raspberry PI using an ARM64 architecture. Our simulation will therefore run on a Kubernetes cluster built from a mix of different architectures.

3.4.3 UNIKERNEL Requirements

Currently, there is no possibility to provide an environment where external developers can write their own smart contracts to the Cothority framework without using a solution similar to Ethereum which is using a specifically defined language and a virtual machine. The very first important requirement for unikernels is to support the execution of a generic piece of code in a deterministic manner so that multiple executions with the same input will produce the same output.

The system also has to process multiple transactions in a very short amount of time so that the throughput of the overall blockchain is sufficiently high. This means that unikernels will be launched beforehands and used as a pool of executors. It should be then possible to run arbitrary programs one after the other without leaking any information from the previous execution.

Finally, the conode process must be able to communicate with the pool of executors to send the program and the input, and then get the output of the execution. It should be as fast as possible as it is necessary to make a smart contract execution as small as possible to increase the global throughput.

3.4.4 Orchestration and Management Integration Requirements

Each conode participating in the network must be managed by different parties to ensure sufficient diversity and thus to prevent a group of attackers to have a high enough threshold. This means that a typical deployment of a Cothority does not assume any availability and it is necessary to provide enough solution to administrators so that they can support unikernels. UNICORE is already supporting bare-metal, KVM and Xen which is sufficient for a starter.

3.4.5 Description of Business Case

As mentioned previously, the DEDIS lab has projects in common with industrial partners. The UNICORE project can improve those contributions by improving the framework provided to the industrial partners so that they can extend the functionalities with their own smart contracts. This would improve the situation as the current workflow requires partners to prepare requirements and the DEDIS lab to write the smart contracts to fulfill the requirements.

4. Conclusions

In this deliverable we presented a detailed report on the deployment targets of UNICORE and all use-cases. The aim of Work Package 5 is to validate the practicality of UNICORE through the six use-cases: serverless computing, vBNG, NFV, vRAN, Home Automation and IoT and finally smart contracts. Towards this goal we started from information in D2.1, describing the use-cases, the infrastructure and the requirements for UNIKERNELS.

For most of the use-cases, the deployment of UNIKERNELS requires that various software components, libraries and codes be ported to UNIKERNELS and that actual logical and physical components of the underlying technology in each use-case be decomposed in UNIKERNELS. Validating the requirements for a UNIKERNEL means that the architecture for the deployment of the use-case takes into consideration the specifications and limitations of the considered UNIKERNEL and the orchestration and management frameworks to be used.

Further on, having consolidated on the business cases for each deployment, we can move forward through the timeline of WP5 towards an initial deployment followed by feeding back the findings from this activity to an ongoing optimization process that will feed forward to the final deployment and validation.

5. References

[1] ”Continuous integration vs. continuous delivery vs. continuous deployment“ - https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment

[2] “IRTA Pubpro - Open Digital Archive” - http://repositori.irta.cat/

[3] “UIC Barcelona - Open Access Archive” - http://http://repositori.uic.es/

[4] “DSpace” - https://duraspace.org/dspace/

[5] "5G; NG-RAN; Architecture description" - https://www.etsi.org/deliver/etsi_ts/138400_138499/138401/15.07.00_60/ts_138401v150700p.pdf

[6] “TRex - Realistic Traffic Generator” - https://trex-tgn.cisco.com/

[7] “Blockly-A JavaScript library for building visual programming editors” - https://developers.google.com/blockly

[8] “Scalable collective authority” - https://github.com/dedis/cothority

[9] K.Nikitin et. all - “CHAINIAC: Proactive Software-Update Transparency via Collectively Signed Skipchains and Verified Builds”, August, 2017, Vancouver, BC, Canada

[10] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford - “Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing”, August, 2016, Austin, TX, USA

[11] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, Bryan Ford - “OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding”, 2017

