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

D5.2 Initial Deployment

Due date of deliverable: August 31, 2020

Actual submission date: August 31, 2020

Start date of project	January 1, 2020
Duration	36 months
Lead contractor for this deliverable	Consorci de Serveis Universitaris de Catalunya (CSUC)
Version	1.0
Confidentiality status	”Public”

Abstract

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. The key to this is to compile an application into very light-weight 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 initial deployment status for each of the use-cases, describing the original plan, the current status of the deployment, the issues encountered and the features used during months 12-18. Also the document details the progress done in regards Unikernels security on lightweight virtualization.

Target Audience

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

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 workpackage	WP5 - Unikernels in Practice
Editor	Consorci de Serveis Universitaris de Catalunya (CSUC)
Project Co-ordinator	Emil Slusanschi, UPB
Technical Manager	Felipe Huici, NEC
Copyright notice	© 2020 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 Deliverable 5.2 Initial Deployment - Intermediate contains a detailed report of Initial Deployment of real world UNICORE use-cases identified in deliverable D2.3. This report gives for each of six use-cases belonging to the four application domains a short summary of the expected plan at the beginning and focus on the current deployment status by describing the state of each of the use-case and identifying the different difficulties and lack of features found during the implementations in order to fed back the Core WPs. Also it is important to remark the WP3 and WP4 developed features and tools used in the deployment workflow. Also it focus on the enhancements performed to improve the security in lightweight virtualization environments by using the address space isolation techniques that will be integrated in the future.

List of Authors

Author	Ioan Constantin (ORO), Marius Iordache (ORO), Cristian Patachia (ORO) Franck Messaoudi (OA), Thierry Masson (OA) , Stephen Parker (XLRN), Esteban Martinez (CSUC), Xavier Peralta (CSUC), Gabriele Scivoletto (NXW), Gino Carrozzo (NXW), Cristina Basescu (EPFL), Gaylor Bosson (EPFL), Felipe Huici (NEC), Radu Stoenescu (CNW), Razvan Deaconescu (UPB), Emil Slusanschi (UPB), Mike Rapoport (IBM)
Participants	ORO, CSUC, OA, NXW, EPFL, NEC, UPB, CNW, IBM
Work-package	WP5 - Unikernels in Practice
Security	PUBLIC
Nature	R
Version	1.0
Total number of pages	31

Contents

List of Figures

List of Tables

Acronyms

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	Berkley 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
CVE	Common Vulnerabilities and Exposures
DALI	Digital Addressable Lighting Interface
DEDIS	Decentralized and Distributed Systems
DHCP	Dynamic Host Configuration Protocol
DMA	Direct Memory Access
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 Berkley Packet Filter
ELF	Executable and Linkable Format
EPC	Evolved Packet Core
EPFL	cole Polytechnique Fdrale 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, Centilation, 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 Intercace 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
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
pCPE	virtual CPE
vCPE	virtual CPU
VM	Virtual Machine
VMM	Virtual Machine Monitor
VoD	Video on Demand
VoIP	Voice Over Internet Protocol
VNF	Virtual Network Function
vRAN	virtualised Radio Access Network
XDP	eXpressive Data Path

1 Introduction

This document is the deliverable D5.2 of the UNICORE project. Its purpose is to detail the results of the initial deployment of the core project tools and use-cases by explaining a summary of the original plan, the current deployment state, what problems have been found and what functionalities have been used or are missing. This input will be fed back to the core WP in order to further refine the UNICORE tools to meet the demands of the projects use-cases This report is structured in three chapters, which are:

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

Chapter 2 provides a detail of the initial deployment for the six use-cases by providing the expected plan, the current state of the deployment, the different issues found in the implementation and the features developed in the core WPs used.

Chapter 3 explains how to enchance the security when running unikernels in containers

Chapter 4 summarizes what has been achieved in this document, identifying shortcomings in order to improve and solve the problems face in the future.

2 Use-cases' Initial Deployment

2.1 Serverless Computing

2.1.1 Original Plan

As stated in the deliverables 2.1 and 5.1 CSUC use-case is focused on the process to convert the images before uploading them to the repositories. In order to leverage the Unikernel technology and Unikraft tools CSUC designed a new methodology based on serverless computing by launching a function per image conversion and storing it on an S3 bucket.

The issues identified to build image conversion serverless solution in a Unikernel environment are:

- **Build an image conversion Unikernel using the Kraft Tool**

  This step requires to port the libraries needed by Pillow and S3FS and integrate them on the Kraft Tool in order to build a Unikernel image using Kraft.

- **Performance and Testing**

  After the creation of the Unikernel image, it will be validated to see if it meets the requirements, the performance and the consolidation.

- **Automation and orchestration**

  The final solution has to be capable of run in an environment which can execute a function automatically when a new file is in the input bucket.

2.1.2 Deployment Status

During the months 12-18 we focused on the pillow library dependencies and by now it is possible to build a unikernel image with Kraft Tool which can process the formats like jpeg, png and tiff. It is also capable of using the CMS for color management. The figure 2.1 shows a unikernel execution creating a thumbnail of an image.

A new application that employs a kraft tool of unicore has been generated for jpeg image resize and the next steps will be to support the rest of pillow functionalities and to support the S3 protocol by using the boto3 library in order to read and store the images.

After testing the different functionalities needed the next step is to develop the tool capable of run like a function as a service and integrate it in an orchestrator which permits to execute the function automatically.

At the end we are in the first phase of deployment process testing unikernels builds with every library ported.

2.1.3 Implementation Issues

With the creation of the Kraft Tool some parameters for unikernel generation were not valid anymore so it was needed to use the Kraft Tool to run them. Because of this, the external libraries have to be in a remote Unikraft repository in order to use them in the Unikernel build time. In order to test the different new libraries

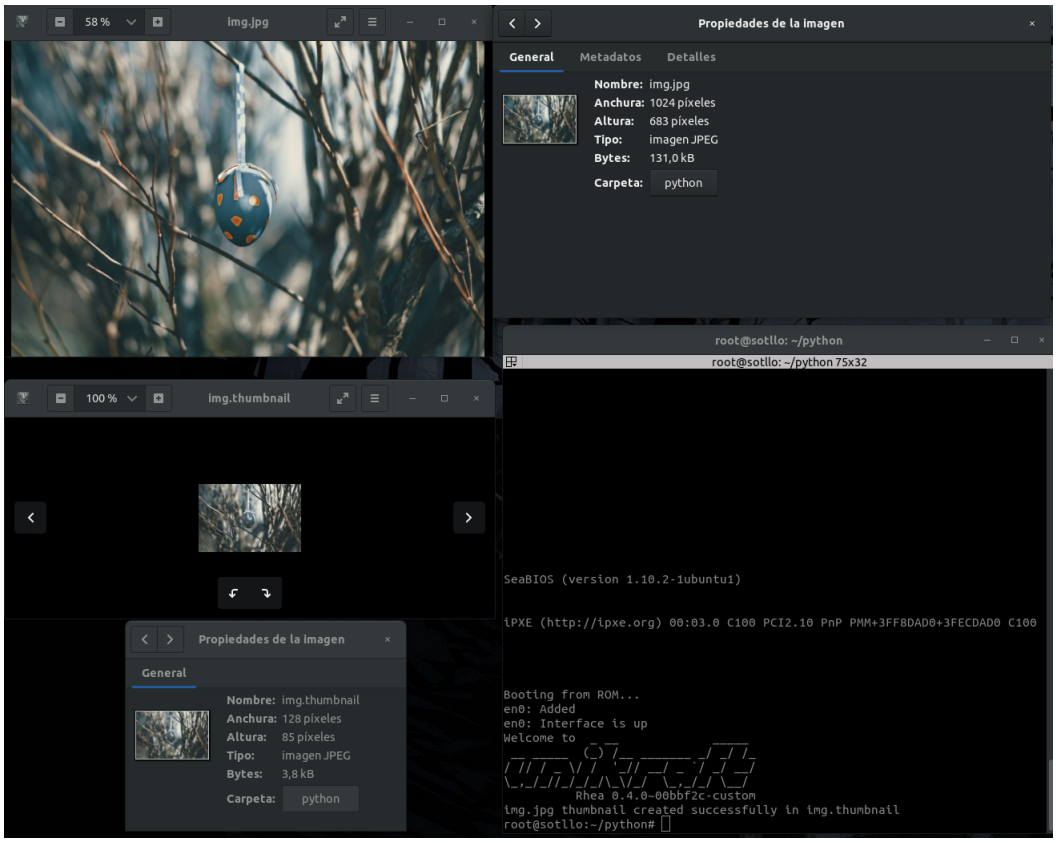

Figure 2.1: CSUC use-case execution

ported and because of them were not available on the remote repository these had to be added manually to Unikraft Core.

Other challenges have been found fixing the errors encountered during the library compilation for the use-case due to some lack of information that has been solved quickly. Also the generation of the code patches in the downloaded libraries to work properly in a unikernel environment has been quite tricky.

2.1.4 Core WP Features Involved

As we are still in the first phase of the use-case CSUC has only used partial core WP Features like some libraries which are dependencies of Pillow and list in the following table 2.1

TOOL/LIBRARY	VERSION
unikraft	0.4
lib-lwip	0.4
lib-Python3	0.4
lib-pthread-embedded	0.4
lib-newlib	0.4

Table 2.1: list of Unikraft libraries used by Pillow

2.2 Network Function Virtualization

2.2.1 Broadband Network Gateway for wired Internet Access

2.2.1.1 Original Plan

Oranges Initial deployment plan for the Unikernel BNG has two distinct phases - a move, test and validation cycle from the currently-deployed Physical BNG followed by a deployment, test and validation of a virtualized BNG infrastructure based on Unikernels for the BNG services and components. As per the description in D5.1 - Deployment Plan, Requirements and Business Cases, Oranges plan consist of successfully moving from a Physical BNG to a virtualized BNG with decomposed micro-services, allowing a fine-grain control of distribution and usage scenarios. Finally, all deployments - Physical BNG, monolithic virtual BNG and Unikernel, decomposed micro-services BNG will be tested against the KPIs set forth in D2.1 and D5.1 and the results will be compared to show differences of moving horizontally across deployment type, validating the improvements in using Unikernels.

2.2.1.2 Deployment Status

The second important phase of our transitioning process from the hardware Broadband Network Gateway (BNG) to a completely virtualized, disaggregate and on-demand BNG service is commissioning and testing a Monolithic BNG instance on a virtualized environment. A Virtualized Monolithic BNG is essentially the same BNG instance from the OS layer upwards - it maintains its OS structure, functions and applications; the major change is the abstractization of the underlying Physical Layer with a virtualized layer on top of a Type 1 or Type 2 Virtualization Hypervisor.

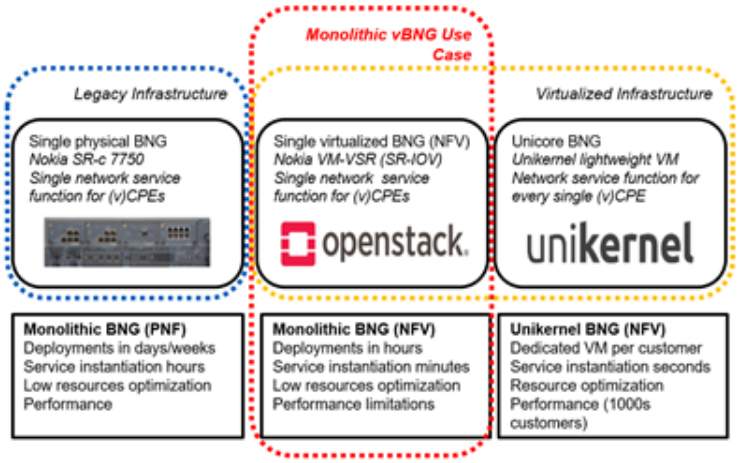

Figure 2.2: ORO Monolithic vBNG Scenario

As specified in the figure 2.2, we are at the second step of our objective, marked by a virtualization of the underlying physical layer of the Monolithic BNG setup. We have deployed our Monolithic Virtual BNG instance on top of our existing virtualized framework, hosted on hardware specifically allocated for telecom research projects, within one of Orange Romanias lab environments. The vBNG software used for deployment was the Nokia 7750 Virtualized Service Router (Nokia 7750 VSR), release 20.2.R1 and the virtualized

infrastructure used for provisioning the vBNG is OpenStack Ocata.

We provisioned the Virtualized Infrastructure in six distinct steps:

(i) Optimization of BIOS and Linux Kernel Settings

(ii) Resource Allocation of Compute Node

(iii) Nova Scheduler Parameter Implementation

(iv) Enable CPU Pinning and SR-IOV on OpenStack Controller and Compute Node

(v) Create Volume Drive using Cinder

(vi) Create Nova Flavor for Monolithic vBNG Nokia 7750 VSR

Following this second phase in implementation, Orange will move to the third and final stage, running a decomposed BNG provided by CNW atop a bare metal deployment in Oranges Test bed. This third phase is scheduled to begin by End of Month 20 of the project with an implementation time of two months. For this weve established remote access capabilities with CNW, allowing monitored remote access to Oranges Test bed components that are going to be used to host the Unikerneld services.

2.2.1.3 Implementation Issues

One of the constraints that we came across when provisioning the VM (at least on the setup we have in place, based on OpenStack Ocata), were that the VM provisioning would not finalize if we had not set up CPU Pinning and a Single root I/O virtualization (SR-IOV). The dedicated compute node from which we have allocated the resources for the vBNG is based on the Skylake SP Intel infrastructure (using 6xxx or 8xxx Intel Xeon processors).

CPU Pinning

In order to achieve optimal performance on the vBNG instance the following CPU Pinning Best Practices had to be implemented: each vCPU was pinned to a hardware CPU of the single NUMA node; the first 2 vCPUs were pinned to sibling threads on the same Physical CPU Core, the next 2 vCPUs were pinned to sibling threads on the next Physical CPU Core and so on.

SR-IOV

Single root I/O virtualization (SR-IOV) is a PCI-SIG standard that allows a single physical Ethernet port to appear as multiple separate physical devices, each device associated with its own PCIe function called a Virtual Function (VF). In a Network Function Virtualization host (in this case the Monolithic vBNG), the hypervisor can assign Virtual Functions to Virtual Machine so that they appear as vNIC interfaces to the vBNG.

SR-IOV enables almost bare-metal I/O performance because data is transferred directly using Direct Memory Access (DMA) between the NIC hardware and guest memory (with address translation provided by Intel VT-d).

The vNIC interfaces were set with VirtIO, E1000, or VMXNET3 drivers, as requested by the Nokia 7750 VSR manual. When the VSR VM uses a VirtIO, E1000, or VMXNET3 driver for one of its vNIC interfaces (ports), the abstraction provided by the hypervisor allows any type of physical NIC to be used to transport the traffic associated with the vNIC interface.

2.2.1.4 Core WP Features Involved

As we have not, as of yet, reached our phase 3 for implementation of an Unikernel BNG, Orange has not used any core WP Features. The activities that are going to involve the using of core UNICORE features are scheduled to begin by end of Month 20 and all activities will follow the initial implementation planning, KPIs and Objectives as set in D2.1 and D5.1

2.2.2 Wireless 5G vRAN NFV Clusters

2.2.2.1 Original Plan

As mentioned in D2.1 and D5.1 the telecom world is in the middle of a transformative shift away from monolithic RAN solutions running on proprietary hardware towards disaggregated and virtualized open solutions running on COTS hardware. Many attempts at virtualizing RAN deployments consist of virtualizing a monolithic software application that was originally running on the BBU and getting it cross-compiled and running as a monolithic entity on a server in a data centre. These deployments typically consist of a single application executing in a Virtual Machine or perhaps even a Docker container. This addresses the virtualisation aspect of the problem but it does not address the open aspect i.e. the deployed system will still likely contain proprietary mechanisms and will not interact with systems supplied by alternate vendors. Equally, such monolithic deployments do not scale well as typically different aspects of a RAN system will scale orthogonally. Accelleran is committed to developing 4G and 5G RAN solutions based on general-purpose, vendor-neutral hardware with open north and south bound software interfaces.
Accelleran has identified several key issues that need to be addressed to tackle the problem of deploying disaggregated microservices using Unikernels:

- Portable software code that is platform agnostic.
- Extracting and isolating key software modules that scale orthogonally.
- An intra-module communication mechanism that is neutral to the underlying hardware architecture.
- A distributed data store that allows disaggregated components to share a common configuration, state and data set.

Software portability can be addressed by effective use of coding standards e.g. MISRA. Accelleran has identified key 5G gNB CU RAN microservice components which scale orthogonally, for example;

(i) ngInterface - 1 per Core Network connection - potentially up to 6 in a given neutral host deployment.

(ii) gNB Controller - 1 per 5G base station with potentially hundreds of gNBs deployed in a single network.

(iii) UE Data Session Controller - 1 per active UE with potentially 1000s in a network deployment.

For intra-module communication Accelleran will use YANG for message definitions, Google Protobuf for message serialisation and NATS as a publish-subscribe microservice message exchange mechanism. Redis will be used to provide a distributed, in-memory keyvalue datastore.

2.2.2.2 Deployment Status

Due to the sheer complexity of RAN software - typical systems can often run into millions of lines of code - Accelleran is taking an iterative approach to deploying a disaggregated, virtualized, open 5G RAN. The steps involved can be summarised as follows:

(i) Identify key microservice components. This aspect is complete and as mentioned above components consist of e.g. the ngInterface, the gNB Controller and the UE DS Controller.

(ii) Define the (message-based) interfaces between these components in YANG and implement Protobuf message encoders and decoders. This stage is also complete although future development will entail the definition of new messages and their corresponding YANG definitions and Protobuf encode and decode functions.

(iii) Instantiate the microservice components identified above as Docker containers that can be orchestrated using Kubernetes. This step can be considered to be an intermediate step in that the end goal is not to use Docker containers, however, as Docker containers are a reasonably mature and well documented and understood technology, it was deemed to be a sensible interim step as it was felt that trying to deploy 5G RAN microservices directly in Unicore Unikernels would add unnecessary complexity. This step is in progress.

(iv) Convert the microservice Docker containers into Unikernels that can be orchestrated using Kubernetes. Once the 5G RAN CU is shown to work in terms of orchestration and message routing etc., the plan is to convert the microservice Docker containers into more optimal Unicore Unikernels. This step has not yet started.

2.2.2.3 Implementation Issues

The implementation issues identified and addressed to date are not specific to Unicore Unikernels but are more down to the complexity of splitting a pre-configured monolithic software block into independently deployable, scalable and addressable microservices e.g. message routing and delivery is a particularly complex problem when ”publishers” and ”subscribers” are not known in advance.

2.2.2.4 Core WP Features Involved

As Accelleran has yet to start converting the microservice Docker containers into Unicore Unikernels, no Core WP features are currently involved in our deployment but the key requirements have been identified both here

in this document and more extensively in D2.1, namely; support for standard OS primitives, networking and libc support and support for third party software, such as Redis, NATS and Protobuf.

2.2.3 SDWAN Controller Key Server and vCPE as Unikernels

2.2.3.1 Original Plan

As stated in D2.1 and D5.1, Ekinops has selected two use cases to be built as Unikernels namely; SDWAN Controller Key Server as Unikernel and the vCPE based Unikernel. In what follows, we use interchangeably the SDWAN Controller Key Server and Key Server.

The Key Server and vCPE as Unikernel use cases have been described in the previous deliverables (D2.1 and D5.1) in which, the functional and non-functional requirements, the used infrastructure, as well as the management and orchestration process were highlighted. Please refer to these documents (i.e, D2.1 and D5.1) for details concerning these two use cases. In what follows, we are more interested in the roadmap for these use cases. As the Key Server use case is more advanced in the process of Unikernel building compared with the Unikernel based vCPE use case, hence our focus in this document will be more on this first use case (i.e, Key Server as Unikernel).

The plan that has been drawn for this use case is split into three main activities:

- **First Activity: Krafting the Key Server**

  In this first activity, we have identified the functionalities for the SDWAN Controller that should be used. We restrained the choice to the Key Server functionality that is responsible for the establishment of VPN tunnels between (v)CPEs, therefore this functionality manages the encryption keys to be used when new VPN tunnels need to be instantiated between (v)CPEs. Next, the Key Server functionality with the required libraries will be KRAFTED.

- **Second Activity: Performance evaluation**

  After the Key Server based Unikernel has been built correctly. Its behaviour will be validated and compared to the Key Server functionality when it is run as Docker Container or as a standalone application. Only after that will be considered the benchmarking and performance evaluation. A test campaign will be performed with a focus on the scalability KPI as we are interested in the number of (v)CPEs that can be managed by the same Key Server.

- **Third Activity: Orchestration, On-demand Deployment, and Upgrade**

  This is the last step for this use case. The built Unikernel will be considered as a VNF that will be orchestrated using an NFV Orchestrator such as OSM or ONAP, and deployed on-demand. As a wish, we would like also to integrate this unikernel in the entire SDWAN solution.

For our second use case vCPE as Unikernel, we have identified exactly the same activities; from KRAFTING the data plane of the vCPE to the build of the Unikernel, its orchestration, and its dynamic deployment. Here also, we would like to integrate the built unikernel in the Ekinops solutions like SDWAN.

Currently, for the first use case, we are at the second activity, where the behaviour of the built Unikernel is under validation. Concerning the second use case, we are at the end of the first activity as several libraries have already been KRAFTED, however we need to KRAFT the data plane functionality of the vCPE.

2.2.3.2 Deployment Status

It is much difficult at this stage to provide an exact status for the use case deployment as we are still in a debugging phase. However, what we are sure about at the time being is:

- All the needed libraries are KRAFTED and compiling without compilation errors.
- Several libraries unit tests are succeed.
- The application itself is KRAFTED and correctly compiled.
- The Unikernel boots correctly and starts the IPSECMD task.
- The SSL and Crypto libs are initialized.

To sum up, at this stage we are testing the key server behaviour.
Figure 2.3 depicts a very initial deployment of the first use case SDWAN Controller Key Server as Unikernel.

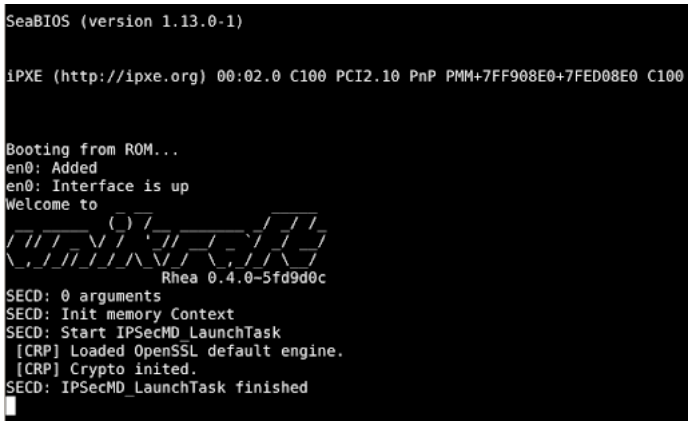

Figure 2.3: Initial deployment of the key server as Unikernel

Concerning the second use case; the Unikernel based vCPE, The first step has been done. This step consists of KRAFTING the required libraries, including DPDK, IPSEC, PROTOCOLS, and ROUTE libraries. The next step is to KRAFT the data plane of the vCPE and then build the Unikernel as well as checking its behaviour.

2.2.3.3 Implementation Issues

During the process of porting libraries to UNIKRAFT and building of the Unikernel, we faced several challenges. Some of them were solved, and some others are under investigation. Table 2.4 describes the most representative issues for the use case SDWAN Controller Key Server as Unikernel.

ISSUE NAME	ISSUE DESCRIPTION	STATUS	RESOLUTION
Syscall layer	Some of the syscall APIs were not found in the syscall_nrs2 header	SOLVED	A patch has been provided with the missing syscall APIs
NewlibC is not Linux specific lib	Our key server application is a Linux-specific implementation. Therefore, compiling this application with the NewlibC issued with several missing header errors as these files are linux-specific	SOLVED	The missing files have been imported from Musl library to NewlibC, the Makefile.uk has been updated accordingly, and the NewlibC recompiled
Start Unikernel with configuration file	The key server needs a configuration file to interact with the different vCPE members and assign keys	SOLVED	We are using the 9PFS and Devfs libraries to boot the unikernel with a configuration file
Support of Openssl version different from the already-KRAFTED one	The Openssl version that has been KRAFTED is 1.1.1c. However, the key server application needs an older version (1.0.2q).	SOLVED	The Openssl v1.0.2q was KRAFTED. We also updated the code of the key server to use recent versions of Openssl, however, there are some bugs that should be fixed.
An OPENSSL_assert failed when executing this lib	The assert (ssl_mac_secret_size[SSL_MD _MD5_IDX] >=0) failed as the result is equal to zero	SOLVED	A bug in the openssl-1.02q was solved
Musl still under development	Several functions have not been implemented	UNDER REVISION	The missing functions were implemented by the Unicore developers
Drivers for network stack	The LWIP lib does not support drivers for the supported platforms. Indeed, Netfront drivers used by Xen and Tap for linux-user space were not implemented. Only Virtio for KVM is supported	UNDER REVISION	The missing drivers were developed by the Unicore developers team
Compiling the Openssl lib in a shared mode	The key server uses Openssl in a shared mode. That is to say, that Openssl is compiled with <shared> argument	OPEN	The lib was compiled in a shared mode, however as we are facing some issues with booting the Unikernel with a configuration file, we do not know yet if this will pose problems

Figure 2.4: Table of issues faced by Ekinops and its status

The second use case Unikernel based vCPE is not completely KRAFTED, the work in progress. Therefore, at this stage we can say that the most relevant implementation issue was the incompatibility of several libraries used by the data plane that are linux-specific with the NewlibC, which is a generic one.

2.2.3.4 Core WP Features Involved

During the building process of the Unikernel, we have used the following tools and libraries represented in Table 2.2.

TOOL/LIBRARY	VERSION
unikraft	0.4
Newlib	0.4
Openssl	1.02q
lib-lwip	2.1.2
freeradius-client	1.1.6.orig
9PFS	0.4
devfs	0.4
Unity	1.0
arch_al*	r3_9pa2
orgfs*	r5_11pa0
netns*	r1_1pa14

Table 2.2: List of tools and libraries used in the Key Server Unikernel build process

For the complete version of the SDWAN controller, we have KRAFTED the following list of libraries shown in Table 2.3. Currently, we are testing the Key Server functionality. The complete version of the SDWAN Controller contains several other functions that we may not need. Indeed, as the main tools and libraries are still under development and debugging (such as, UNIKRAFT, Newlibc, musl, lwip), considering the entire functionalities will increase the complexity of the unikernel building process and therefore debugging will be very arduous. However, we believe that doing this exercise KRAFTING very different libraries (even some of these libraries may not be used for this use case) is very useful for the rest of the project. Indeed, several of these libraries already KRAFTED will be used in our second use case (i.e., Unikernel based vCPE).

LIBRARY	VERSION	LIBRARY	VERSION
pyang	r1_2pa0	cplib_evdelegate	r1_1pa12
arch_al	r3_9pa2	cplib_zthreadutil	r2_2pa13
orgfs	r5_11pa0	libevent	r1_1pa18
protocols	r4_31pa0	cp_evloop	r2_2pa14
ipsec_dpapi	r10_12pa0	cpdm_ciroot	r5_5pa3
memory_al_api	r1_1pa8	cpdm_ifmmgr	:r42_49pa2
netnslib	r1_1pa14	cpdm_crypto	:r1_10pa9
cplib_log	r1_1pa18	osal_api	r1_3pa0
cplib_hash	r1_1pa14	soc_core	r7_7pa0
usdpaa	r3_4pa3	multicore_lib	r10_14pa0
soc_info	r3_6pa2	mem_pool	:r1_3pa17
memory_al	r1_1pa26	oamcapi_arch	r5_15pa0
mem_lib	r3_6pa17	ipsec_msg	r4_4pa9
osal	r1_4pa1	tp_openssl	r2_2pa13
sys_utils	r15_16pa1	cptplib_dsutil	r1_1pa12
util_lib	r19_22pa3	oasystemd	r8_8pa18
oamcapi	r2_3pa7	cplib_confdlib	r1_13pa1
pkt	r34_37pa0	trc_pki	r2_2pa1
cplib_ifmg	r3_22pa2	trc_user_routing	r1_1pa6
cplib_zinclude	r1_1pa15	trc_event_data	r1_6pa0
cptplib_ztlv	r1_1pa10	cplib_aclmgr	r8_10pa0
cptplib_zvector	r1_1pa14	cplib_trapd	r1_2pa1
liburcu	r1_1pa14	cplib_routeinterface	r1_4pa7
lttng_ust	r1_3pa7	encaps_msg	r9_9pa1
trc_ipsec	r1_1pa6	encaps_oper_lib	r26_33pa0
flib	r1_1pa5	cp_devcon	r1_6pa13
cpdm_prefixlist	r1_1pa26	cplib_dnshelper	r1_1pa62
cpdm_route	r1_2pa4	addrpools	r1_1pa6
cptplib_zutils	r11_14pa3	cplib_messageutil	r1_1pa26
pki	r2_3pa8	cplib_vrfmgr	r1_8pa7
cptp_pammodule	r2_3pa4	trctool	r1_1pa12
dma_userspace_driver	r1_2pa1	cplib_aaa	r3_4pa0

Table 2.3: List of Ekinops krafted libraries

2.3 Home Automation and IoT

2.3.1 Original Plan

The Home Automation and IoT use case is based on the integration of some unikernel-based functions in the Nextworks Smart Home and Smart Building Management platform called Symphony. As initially described in D2.1 and D5.1, some of the many Symphony platform functionalities can be considered for migration to unikernel, in order to implement new scenarios with tiny virtual functions instead of standard VMs. The strategy initially adopted for the deployment and validation in this use case has been described in D5.1 and based on the following three stages/phases:

- **Stage 1. Functional Validation:** functional validation of the krafted function w.r.t. the original one.
- **Stage 2. Performance Evaluation:** benchmarking operations in terms of speed, number of operations, throughput etc.
- **Stage 3. Automation and Upgrade:** evaluation of the feasibility and potential benefits of automatic deployments based on run-time unikernel generation.

Due to the complexity of the Symphony code base in terms of modules dedicated to implement various IoT control functions, their required libraries to access system resources and the links to third-party application, and due to the ongoing development in UNIKRAFT of many of such system primitives and libraries, the selection of the first Symphony modules to be ported to unikernels focused on those elements which showed the simpler dependency graph and required libraries just supported by UNIKRAFT.
As such, the original plan defined in D5.1 was structured to experiment:

- By this deliverable D5.2 (Aug 2020)
  (i) Symphony Event Reactor module.
  (ii) Symphony MQTT driver module.
- By the end of the project, i.e. for D5.3 (Dec 2021)
  (i) Symphony environmental sensor driver.
  (ii) Symphony lightning driver.
  (iii) Symphony energy monitor module.
  (iv) Network Firewall.

Therefore, in scope of activities reported in this deliverable, the krafting and functional validation (stage 1) of Symphony Event Reactor and the MQTT driver was done.

2.3.2 Deployment Status

The deployment of Symphony Event Reactor and the MQTT driver function in unikernels, was carried out in three pipelines activities:

(i) Krafting via UNIKRAFT of the selected module and instantiation in Symphony testbed at Nextworks.

(ii) Definition of functional tests to validate the krafted Symphony function.

(iii) Execution of the validation tests.

The krafting of the two selected functions required at first software adaptation, in order to remove from the original codebase all the unused objects and library functions which enlarged the dependency graph and which were not supported by UNIKRAFT. Then, once the module was ready to be krafted/compiled, a phase of code fixing/adjustment and compiler tuning was started in order to obtain a running executable. During this process, several issues have been encountered, which required debugging of UNIKRAFT or compiler configuration directives which were executed together with the UNIKRAFT developers.

For the Symphony MQTT Driver, the krafting process was successful and the function has been correctly run in unikernels and functional tests executed. Figure 2.5 shows a simple application scenario in which MQTT clients and broker in unikernel versions have been deployed.

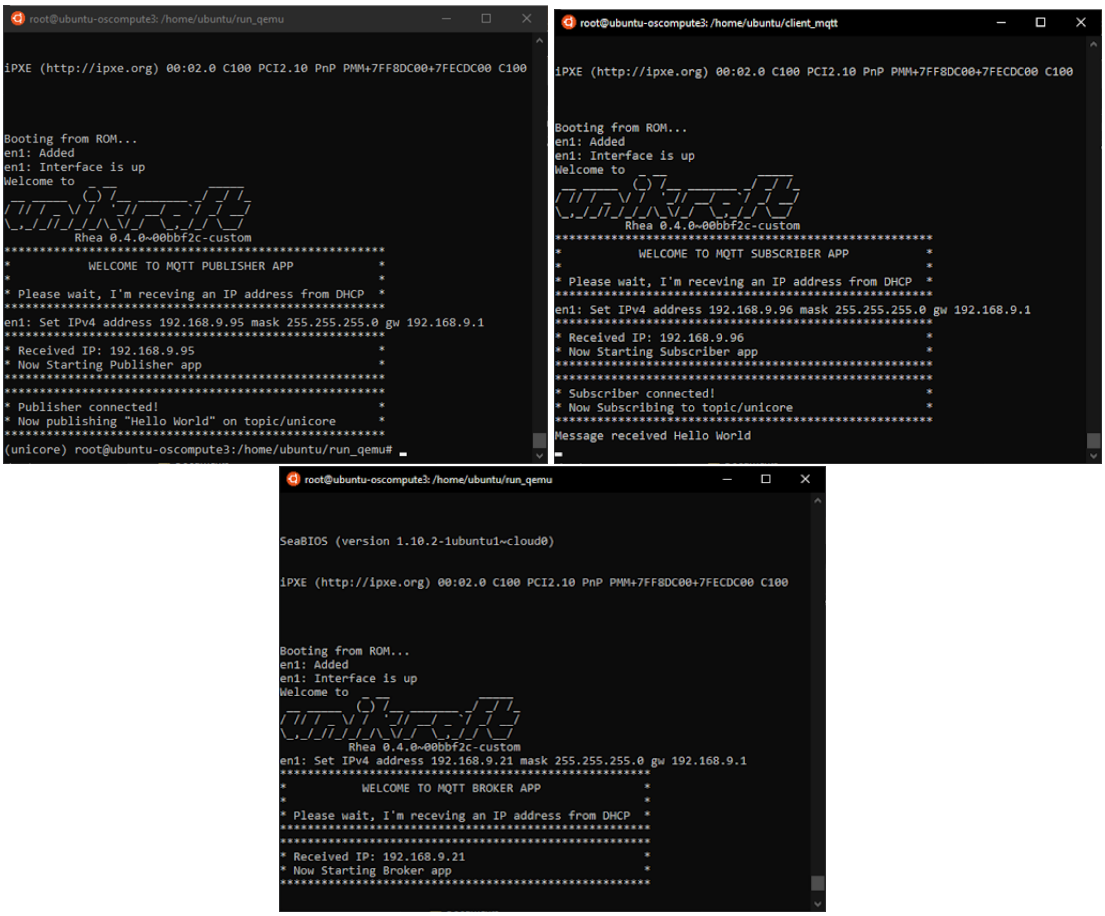

Figure 2.5: Symphony Unikernel Screenshots

For the Symphony Event Reactor, krafting was successful (i.e. executable compiled) but functional validation could not occur due to some remaining issues in UNIKRAFT, as described in the next section.

2.3.3 Implementation Issues

Next table shows the main issues that have been found during the implementation of the aforementioned Symphony functions in unikernel. Some of them have been fixed while some other remain still under investigation by the UNIKRAFT developers.

ISSUE NAME	ISSUE DESCRIPTION	STATUS	RESOLUTION
Pyhton Tornado web server with SSL support not working	*Pyhton Tornado web server with SSL support not working*	*SOLVED*	Patch released by the developers
Support of Openssl Library versions different from the latest	*Compile process stuck due to missing hardcoded version of OpenSSL library in openssl.org repo*	*SOLVED*	Multiple releases available on the server
Slow IP Address assignment (DHCP)	*The DHCP Server takes about 10 minutes to assign the IP to the VM*	*SOLVED*	Timer Frequency increased on unikernel configuration
SQLAlchemy not compiled	Compilation errors when we try to compile an application with Python3 and sqlite (missing symbol PyInit_sqlite3)	OPEN	
Unexpected Clock Behaviour	The clock of all the unikraft applications seems to be very slow (about 200 times slower then the expected execution time)	OPEN	
Incomplete socket support	Some socket functions in Python (e.g socketpair) have not been implemented	OPEN	
Nondeterministic complie error output	Repeated unsuccessful compile run using the same configuration and source code base produce different compilation errors	OPEN	

Figure 2.6: Table of issues faced by Nextworks and its status

2.3.4 Core WP Features Involved

For the development, the library and the tool version used are the following:

TOOL/LIBRARY	VERSION
unikraft	0.4
lib-pthread-embedded	0.4
lib-openssl	0.4
lib-lwip	0.4
lib-zlib	0.4
lib-uuid	0.4
lib-newlib	0.4
lib-python3	0.4
paho-mqtt	1.5.0
hbmqtt	0.9.6

Table 2.4: List of libraries used by Nextworks symphony

2.4 Smart Contracts

2.4.1 Original Plan

A distributed ledger system consists of several participants running the same application to work together on choosing the next state of the system. It is built around multiple components responsible for different steps. Those steps can basically be grouped in two phases:

- Ordering phase
- Validation phase

With regard to the UNICORE project, the ordering phase is out of context but for the sake of understanding the big picture, it is good to know that it is responsible for gathering the transactions from the clients. After a certain point in time decided by the ordering service, the transactions are ordered and batched together to be sent to the validation service.

A transaction can be reduced to the client inputs that will be passed to the execution of a smart contract. It can also contain more information used by the system, for example to protect against replay attacks or to prove the identity of the client.

The validation service processes the batch of transactions created by the previous phase by going through them one by one, or according to an order that will prevent conflicts when reading or writing the new state. In order to do so, the service must execute the smart contracts, which can either be done natively if the smart contract is packed alongside the distributed ledger application, or through a virtual machine that is fed with the byte code and the different inputs.

The benefit of a native smart contract is the simplicity to write it, but it makes the system very hard to extend or update, as a new version needs to be deployed on every node so that it learns about a new contract, or an update of an existing one. Therefore, the distributed ledger should be able to accept smart contracts external to the system, that will be executed through a virtual machine. It can be for instance the Ethereum Virtual Machine (EVM), one of the numerous Web Assembly Virtual Machines (WASMVM), or a UNICORE Virtual Machine. Those virtual machines need to provide a deterministic runtime environment to reduce the risk of multiple executions of the same program to output different results.

2.4.2 Deployment Status

The DEDIS laboratory from EPFL has an historical framework that is used for most of the internal projects, which means that native smart contracts were sufficient. This project aims to open the distributed ledger system to accept smart contracts from external sources. In the scope of the WP5, a framework named Dela has been developed based on this historical framework. It allows the system to be more modular and thus more open to different kinds of integrations that will be tested for the UNICORE project.

The framework is in the early stage of development and recently got a review from some members of the laboratory which was successful by showing some weaknesses, and the current work is to address those

concerns. This will allow a very naive deployment in the upcoming months with a simple smart contract based on a unikernel. This is an important step that will allow the communication between the distributed ledger application and a deployment of a UNICORE unikernel.

After the previous points are successful, the final step is to improve the virtual machine to accept any kind of smart contracts so that any allowed developer can write its own using a compatible generic-purpose language and upload it to the system. This will allow comparisons to be done against other types of virtual machines.

2.4.3 Implementation Issues

As mentioned before, a virtual machine must provide a deterministic runtime environment that is independent from the machine it is running on. Unfortunately, our researches showed that it is impossible for a generic-purpose language to provide such an environment because of the complexity of the properties delivered by the language. Our conclusion is that the unikernel virtual machine must provide as much as possible deterministic properties but it is expected that a program can produce different outputs for the same input. For the reference, the Ethereum Virtual Machine provides a deterministic environment by forcing the language and thus the instructions available to the developers.

This issue is one of the reasons the framework needs a deep rewrite, because it can be solved by using a different algorithm to order and validate the transactions. The simple way of validating is to execute all the transactions and compare the final state, but it is instead possible to validate each execution to detect which ones are non-deterministic.

2.4.4 Core WP Features Involved

As mentioned previously, the distributed ledger system framework is developed by DEDIS to allow the validation of smart contracts running in a UNICORE virtual machine.

UPB is working on different tools that will be available to the developers to help the creation of smart contracts. The unikernel can enforce some properties to be deterministic, but it is also the intent of the developer to write a smart contract that wont produce non-deterministic outputs.

Finally, the UNICORE virtual machine is expected to support binary compatibility so that smart contracts can be executed. It also requires a communication channel with the distributed ledger application so that input and output can be transmitted.

3 Host OS Security Enhancement

Unikernels are most often run in a fully virtualized environment, but lightweight virtualization (aka containers) environments are also possible. It is important to ensure security and robustness of the host environments particularly in the light of recently discovered multiple hardware vulnerabilities.
The address space isolation in the Linux kernel techniques developed in the Unicore project can be used to enhance the security of both virtual machines and Linux containers.

- Secret memory areas - intended for applications to store secret information, e.g. private keys, or even the entire memory of the VM guest. Such areas can be completely unmapped from the kernel page tables and thus visible only to the process that owns the secret memory regions.
- Address space for Linux namespaces - Linux namespaces provide logical isolation of various operating system resources, such as mount points, system time and networking stack. The namespaces constitute the major building block of the lightweight virtualization also known as containers. For such use case, addition of a dedicated address space to network namespace would allow better privacy and security for the applications and unikernels running inside the container.

For the initial deployment we plan to integrate the memfd_secret() [1] system call that provides the userspace API for the secret memory areas in Linux. The memfd_secret() implementation is production ready and can be easily ported to the Linux kernel used in the test-bed.
As the implementation of the address space for Linux namespaces will evolve we will consider its deployment in the future versions of the test-bed.

[1] https://lwn.net/Articles/828026/

4 Conclusions

In this deliverable we presented a detailed report on the deployment status during the first mid year for each WP5 use-case. The aim of the exercise is fundamental to give a detailed view of the status of the six use-cases: serverless computing, vBNG, NFV, vRAN, Home Automation and IoT and finally smart contracts. The detailed view consisted in describing the original plan of the deployment and its current status, explaining the problems faced during this period and what solutions have been applied and listing the features developed in work packages 3 and 4.

As the document explains most of the use-cases' deployments are in the initial stage, some of them porting the needed libraries to kraft, building and testing the resulting UNIKERNELS image. Other use-cases has been working on the transition of a monolithic environment to a microservices environment. Also there is another use-case working on the development of a new framework which allows the system to be more modular and fits better in the Unicore project. The issues faced for the use-cases are most related to virtualization features needed, krafting their application or dividing the solution into a microservices environment or deterministic execution.

Finally the document explains how we are working to enhance the security when running Unikernels in a lightweight virtualization.