D6.2: Complete Report on Validation and Demonstration of the Integrated Platform

Deliverable

Document ID	NECOS-D6.2
Status	Final
Version	2.0
Editors(s)	Billy Pinheiro (UFPA) and Antônio Abelém (UFPA)
Due	31/10/2019
Delivered	30/10/2019

Abstract

This document presents pieces of evidence from the correct NECOS execution and summarizes the tests performed in each one of the five demonstrations using the NECOS prototypes in different scenarios.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF CONTRIBUTORS

Contributor	Institution
Stuart Clayman	University College London (UCL)
Francesco Tusa	University College London (UCL)
Alex Galis	University College London (UCL)
Christian Esteve Rothenberg	University of Campinas (UNICAMP)
David Fernandes Cruz Moura	University of Campinas (UNICAMP)
Celso Cesila	University of Campinas (UNICAMP)
Asma Swapna	University of Campinas (UNICAMP)
Tariqul Islam Sajib	University of Campinas (UNICAMP)
Antonio Jorge Gomes Abelém	Federal University of Pará (UFPA)
Billy Anderson Pinheiro	Federal University of Pará (UFPA)
Jeffson Celeiro Sousa	Federal University of Pará (UFPA)
Joan Serrat	Universitat Politècnica de Catalunya (UPC)
Javier Baliosian	Universitat Politècnica de Catalunya (UPC)
Ilias Sakellariou	University of Macedonia (UOM)
Polychronis Valsamas	University of Macedonia (UOM)
Sotiris Skaperas	University of Macedonia (UOM)
Antonis Tsioukas	University of Macedonia (UOM)
Sarantis Kalafatidis	University of Macedonia (UOM)
Tryfon Theodorou	University of Macedonia (UOM)
Luis M. Contreras	Telefónica Investigación y Desarrollo (TID)
Augusto Neto	Federal University of Rio Grande do Norte (UFRN)
Silvio Sampaio	Federal University of Rio Grande do Norte (UFRN)
Marcilio Lemos	Federal University of Rio Grande do Norte (UFRN)
Rafael Augusto Scaraficci	CPqD Telecom Research and Development Center
Sand Luz Correa	Federal University of Goias (UFG)
Leandro Alexandre Freitas	Federal University of Goias (UFG)
Paulo Ditarso Maciel Jr.	Federal University of São Carlos (UFSCar)
Fábio Luciano Verdi	Federal University of São Carlos (UFSCar)
André Beltrami	Federal University of São Carlos (UFSCar)
Rafael Pasquini	Federal University of Uberlândia (UFU)
Raquel Fialho Q. Lafetá	Federal University of Uberlândia (UFU)

REVIEWERS

Reviewer	Institution
Francesco Tusa	University College London (UCL)
Javier Rubio Loyola	Universitat Politècnica de Catalunya (UPC)
Christian Esteve Rothenberg	University of Campinas (UNICAMP)
Alex Galis	University College London (UCL)

Achronyms

Acronym	Description
API	Application Programming Interface
CBR	Constant Bit Rate
CPU	Central Processing Unit
CDN	Content Delivery Network
CPE	Consumer-Premise Equipment
DASH	Dash Price Chart
DC	Data Center
DNS	Domain Name System
EPA	Extended Platform Awareness
FR	Functional Requirements
FED4FIRE	Future Internet Research and Experimentation
IMA	Infrastructure & Monitoring Abstraction
IoT	Internet of Things
IP	Internet Protocol
ISP	Internet Service Provider
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
LSDC	Lightweight Software Defined Cloud
LAN	Local Area Network
MQTT	Message Queuing Telemetry Transport
M2M	Machine-To-Machine
NECOS	Novel Enablers for Cloud Slicing
NFR	Non-Functional Requirements
OTT	Over The Top or Content Providers
OVS	Open vSwitch

PDT	Partially Defined Template
PoCs	Proofs of Concept
QFD	Quality Function Deployment
QoS	Quality of Service
RAN	Radio Access Network
REST	Representational State Transfer
RSPEC	Resource Specifications
SDN	Software Defined Networking
SLA	Service Level Agreement
SP	Slice Provider
SRA	Slice Resource Alternatives
SRO	Slice Resource Orchestrator
SWI-Prolog	Constraint Logic Programming System
TCP	Transmission Control Protocol
TRL	Technology Readiness Level
UDP	User Datagram Protocol
VDU	Virtualisation Deployment Unit
VXLAN	Virtual Extensible LAN
VIM	Virtual Infrastructure Manager
VLSP	Very Lightweight Network & Service Platform
VM	Virtual Machine
VNF	Virtual Network Function
VoD	Video-on-Demand
VPN	Virtual Private Network
WIM	Wide-area network Infrastructure Manager
WISE	Wireless Slicing sErvices
WLAN	Wireless Local Area Network
YAML	YAML Ain't Markup Language

Executive Summary

This document is a report of the work carried out in the context of WP6, with particular focus on task 6.2 (System & Platform Validation & Demonstration). Firstly, this deliverable presents the deployment, test and validation activities in scope of the Novel Enablers for Cloud Slicing (NECOS) platform. Secondly, the deliverable describes how the different testbed islands were interconnected in order to validate and demonstrate the multi-domain federation model. In order to endorse the NECOS validation plan, WP6 activities executed within the second year of the project were focused on (1) setting up the network to interconnect the Lightweight Software Defined Cloud (LSDC) islands; (2) deploying the network and computing resource description and discovery of LSDC over the federated islands; and (3) performing the use case demonstrations.

Five demonstrations are presented and evaluated: MUlti-Slice/Tenant/Service (MUSTS), Marketplace (MARK), Experiments with Large-scale Lightweight Service Slices (ELSA), Machine-Learning based Orchestration of slices (MLO), and Wireless Slicing Services (WISE). Altogether, this document presents the evidence of the suitability of the NECOS proposition by means of running platforms and experimental evaluations.

Scope

This document is the final result of task T6.2: Complete report on validation and demonstration of the Integrated Platform NECOS project's WP6.

In the context of this document, the NECOS software solution is referenced as Lightweight Slice Defined Cloud (LSDC) and as NECOS platform interchangeably.

The term prototype is used as a physical and logical model used to evaluate and test both the slicing concept and workflows in network clouds.

The term demonstration is used as the set of actions and operations put together with the primary purpose of showcasing feasibility, performance and method of slicing in networked clouds.

1. Introduction

The main objective of this deliverable is to describe the NECOS validation process and presents evidence of the correct implementation and execution of NECOS workflows. In D6.1, we described the Proofs-of-Concept (PoCs) in order to explore some of the systems related to the NECOS architecture. The results of the analysis of these PoCs were used to develop both a revised version of the architecture and of the Application Programming Interface (API). Figure 1 provides an overview of the relationship between D6.2 and other project deliverables.

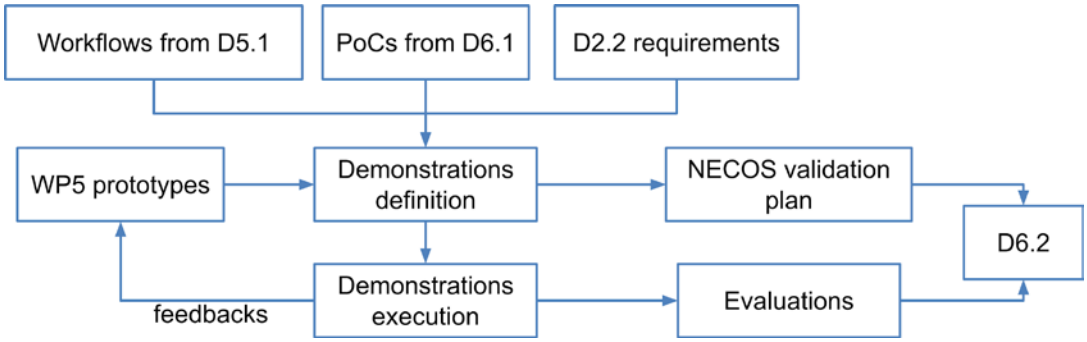

Figure 1. Relationship between D6.2 and other project deliverables

Technology-readiness-level (TRL) [H2020 2017] provides a standardized, systematic, and shared view of how to manage innovation. The NECOS experimental results presented in this document were performed using technologies validated in the lab (i.e., in reduced scale prototype developed and integrated with complementary subsystems at laboratory), defining the current NECOS technology-readiness-level as 4. Furthermore, some characteristics are validated through numerical analysis and measurable Key Performance Indicators (KPI), reinforcing the NECOS classification as TRL 4.

1.1. Structure of this document

This document is structured in five sections. Section 1 is this Introduction. Section 2 presents the NECOS platform validation plan and discusses the inputs from others deliverables. Section 3 presents the infrastructure hosting the prototypes used in the NECOS experimental evaluation. Section 4 is devoted to describing the demonstrations performed in the experimental environment. Finally, Section 5 summarizes our conclusions and outlook.

1.2. Contribution of this deliverable to the project and relationship with other deliverables

WP6 targets directly Objective 4 of the NECOS project, which is stated as to demonstrate the full impact of the NECOS solutions by means of the use case implementations. In other words, this WP evaluates and shows how the NECOS LSDC platform is able to tackle the challenges presented in specific cloud network slicing use-cases/scenarios.

To achieve the above general project objective, this WP is related to all the others technical WPs. In fact, WP6 took inputs from WP2, where the two project use cases were refined into more elaborated scenarios. All these scenarios require physical and virtual infrastructures interconnected by means of physical and virtual networking resources. This means that WP6 acts as an enabler of the appropriate

resources as dictated by the project use cases and scenarios. In addition, WP6 is also related to WP3, WP4 and WP5 together getting inputs and providing outputs to each one of them. In fact, WP4 and WP5 brought the design of the mechanisms and artefacts, all together within the architecture framework of WP3. The functional and non-functional capabilities of such designs had to be reflected in the evaluation tests and showcases to be implemented in WP6. In other words, WP6 designed its set of evaluation tests (i.e., what is called demonstrations in WP6) oriented to specific capabilities and KPIs, from what was designed in WP4 and WP5. Finally, from the results obtained in those tests, feedback was provided to the same WPs to refine architectural aspects and specific characteristics of NECOS LSDC and APIs.

2. NECOS Platform Validation Plan

This section presents the NECOS platform validation plan, which is depicted in Figure 2. These tests are driven by the prioritization of D2.2 requirements and architectural concepts in order to fulfil the acceptance plan defined in subsection 2.3.

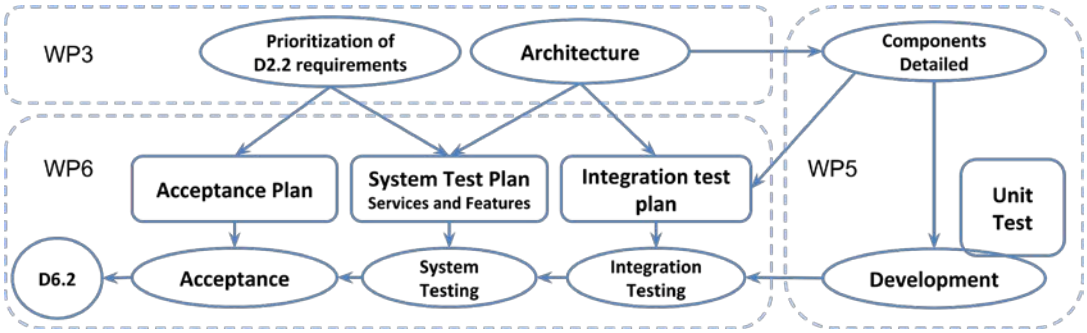

Figure 2. Overall NECOS platform validation plan.

Testing a system involves executing the software system with test cases that are derived from the specification of the real data to be processed. The tests aim to demonstrate to both developers and customers that the produced software meets the expected requirements. As shown in Figure 2, the tests were divided into 4 stages following well-known methodologies from the literature [Sommerville 2011].

Unit test: includes all testing activities carried out by the team developing the system. Unit testing, where individual program units or object classes are tested, should focus on testing the functionality of objects or methods. Component testing, where several individual units are integrated to create composite components, should focus on testing component interfaces.

Integration tests: the objectives of these testing activities are to detect faults due to interface errors or invalid assumptions about interfaces, including: interface misuse refers to a situation where a component calls another one and produces an error while using the related interface e.g., the parameters are provided in the wrong order; interface misunderstanding is raised when a calling component embeds assumptions about the behaviour of the called component which are incorrect; timing errors happen when the called and the calling components operate at different speeds, and out-of-date information is accessed.

System testing: the system testing of an application is done on the whole software system in order to check the overall compliance of the product with the functional requirements. It is performed when some or all of the components in a system are integrated and it can, thus, be tested as a whole. In this deliverable, the results of both the functional and non-functional testing will be presented. The behaviour of the system is tested to check if it meets the specified requirements using the real data.

Acceptance verification: all features (Table 1) desired in a system should be covered by the system tests for the acceptance to be properly completed.

In order to validate and demonstrate the NECOS overall approach, we have defined and setup five interrelated demonstrations: MUlti-Slice/Tenant/Service (MUSTS), Marketplace (MARK), Experiments with Large-scale Lightweight Service Slices (ELSA), Machine-learning based orchestration of slices

(MLO), and Wireless Slicing Services (WISE). Note that the integration test is presented only for the demonstration MUSTS, since it involves several components developed by different teams.

2.1. Prioritization of Requirements

The NECOS architecture was designed considering approximately 100 different requirements, whose prioritization process is presented in Figure 3. In deliverable 3.1 [D3.1] the prioritization of scenario's requirements identified in deliverable 2.1 [D2.1] was assessed making use of the Quality Function Deployment (QFD) method. Based on the requirements prioritization, a set of features was identified. Next, the deliverable 2.2 [D.2.2] presented a new set of requirements focusing on the NECOS platform but using the scenario's requirements from D2.1 as baseline. Thus, in order to reuse the QFD analysis, it was necessary to provide a link between the requirements from D2.1 into D2.2, and then, generate a prioritization from D2.2 requirements that was presented in D3.2.

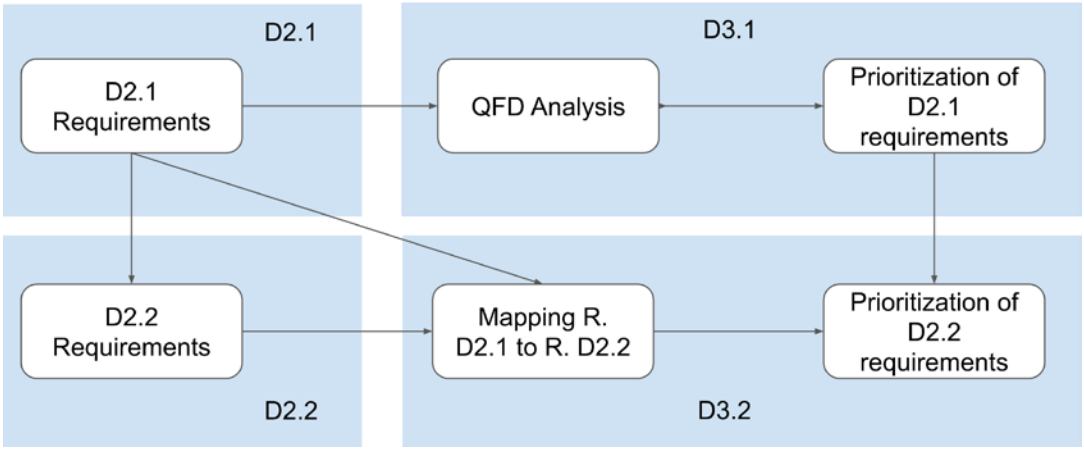

Figure 3. D2.2 requirements prioritization process.

Table 1 presents revised prioritization of D2.2 requirements. This is the set of requirements and features that will drive the all validation process.

Table 1. Prioritization of requirements from D2.2.

Feature	D2.2 requirements
Slice Provisioning	NFR-4.1, NFR-4.3, FR-1.1, FR-1.2, FR-1.3, FR-2.8, FR-2.10, and FR-9.1
Isolation	NFR-2.X, NFR-4.1, NFR-4.3, FR-1.1, FR-1.2, FR-1.3, FR-2.3, FR-2.6, FR-2.8, FR-2.10, FR-5.2, and FR-9.1
Management	NFR-4.1, NFR-4.2, NFR-6.X, NFR-7.X, FR-1.2, FR-1.3, FR-2.1, FR-2.4, FR-5.3, and FR-6.1
Elasticity	NFR-4.X,, NFR-5.3, NFR-5.4, NFR-12.3, FR-2.11; FR-2.12, FR-4.1, and FR-4.4
Scalability	NFR-4.X, FR-1.2, FR-2.3, FR-2.6, FR-2.7, FR-2.8, and FR-2.9
Monitoring	NFR-12.4, FR-1.1, FR-2.3, FR-2.6, FR-2.7, FR-2.8, FR-2.9, FR-2.10, FR-3.4, FR-5.1, FR-5.2, FR-7.2, FR-7.3, FR-8.1, FR-8.2, FR-9.5, and FR-10.2

EUB-01-2017

VIM-independence	NFR-5.3 and FR-4.3
Bare-metal slice	NFR-6.X

The 8 main requirements, which are underpinning the validation process, are as follows:

- Slice Provisioning: is the functional architecture feature related to supporting rapid resource/service provisioning using the Marketplace to discover the resources that will be deployed by the DC/WAN Slice controllers;
- Isolation: is the factor that distinguishes slicing from other cloud-based solutions. Since the slices are isolated from each other in all network, computing, and storage planes, the user experience of the slice will be the same as if it was a physically separate infrastructure;
- Management: the tasks related to this feature are executed by the LSDC Slice Provider. It includes the Slice Resource Orchestrator (SRO), which combines the slice parts that make up a slice into a single aggregated entity. It is responsible for the orchestration of several elements that are utilised for the creation of the end-to-end slices (either Virtual Machines (VMs) and virtual links or Physical Machines and tunnels in the resource domains), as well as for the actual deployment of the service elements on the above slices, on the basis of the embedding decisions performed by the Tenant via the Service Orchestrator;
- Elasticity: when a slice has to be augmented with resources regardless of their location (or other requirements that cannot be fulfilled with the already allocated slice parts), the SRO will contact the Slice Builder to delegate the process of looking for additional resources that can be attached to the existing slice as new slice parts;
- Scalability: is related to the system ability to increase workload size within the existing infrastructure (hardware, software, etc.) without impacting performance of the running services. We can think in scalability in two different dimensions: scalability of a particular provisioned slice and scalability of the number and size of the slices provided.
- Monitoring: The Infrastructure and Monitoring Abstraction (IMA) component, allows the Slice Provider to interact with various remote Virtual Infrastructure Manager (VIMs) and Wide-area network Infrastructure Manager (WIMs) using plug-in adaptors with the relevant API interactions. Besides that, it allows the SRO to interact with the remote clouds and the monitoring subsystems therein in a generic way, in order to provision the actual tenant services and to monitor the remote resources running those services;
- VIM-independence: The tenant has direct control of the VIM, including the decision of when and where to deploy the VIM;
- Bare-metal slice: It is a slice created using physical resources instead of virtual resources.

2.2. NECOS key Performance Indicators

The NECOS KPIs were formerly defined in D2.1. Based on the prioritization of requirements, we have then identified a subset of KPIs to be used in the validation process. This subset is presented in Table 2 together with the information about the associated features, the proper way to measure them and relevant test implementation aspects.

Table 2. Prioritized KPIs associated to NECOS features.

Features	KPI	How to measure
Slice	KPI4 - Average slice	Collect the time measured from the tenant

Provisioning	provisioning time (in seconds)	during the Create slice workflow.
Isolation	KPI15 - Slice isolation index	Collect Central Processing Unit (CPU) and / or RAM utilisation data from two slices while stressing the resources of a given slice without affecting the resources of the other slice.
Management	KPI3 - Average service provisioning time (in seconds)	Collect the overall time for deploying the service measured from the perspective of the Tenant.
Elasticity	KPI1 - Average elasticity response time (in seconds)	Measure the time between the instant when a trigger for the elasticity is generated and the instant when the elasticity operations have been accomplished.
Scalability	KPI4 - Average slice provisioning time (in seconds) during the Resource Discovery Create slice workflow	Measure resource discovery characteristics for slices of different sizes and with a varying number of providers. The former will include the number of messages exchanged, alternative offers on Data Center (DC) and Net slice parts, the number of alternative slice instantiations generated by the offers, and time to complete the discovery.
Monitoring	KPI 7 - Monitoring-data availability	Show the amount of data from IMA collected in each slice over the slice life-time.
VIM-independence	KPI4 - Average slice provisioning time (in seconds)	Collect the time spent to deploy different VIMs in the same Slice.
Bare-metal slice	KPI9 – Physical Server Utilization	Collection of the measurements related to the allocations of different (physical) resources that form the slice parts. Demonstrating that overloading one end-to-end Slice does not affect the other Slices in the same Resource Provider, as they use a disjointed set of physical resources.

The KPIs presented are described next:

- KPI 1 - Average elasticity response time (in seconds): it is the time required to perform the elasticity action. It starts with the first elasticity request performed by the tenant until the operation is completed when the slice is reconfigured;
- KPI 3 - Average service provisioning time (in seconds): it is the time required to perform the service provision. It includes the time interval since the service request is performed by the tenant (after the provision) until the service is completely instantiated on the slice;
- KPI 4 - Average slice provisioning time (in seconds): it is the time required to perform the slice provision. It starts with the request performed by the tenant until the slice is completely allocated and instantiated together with the VIMs.
- KPI 7 - Monitoring-data availability: Provision of monitored data to the Slice Provider operator and the tenant;

- KPI 15 - Slice isolation index: Slice isolation to highlight data segregation over a multi-domain environment.
- KPI 9 - Physical Server Utilization: Quantifies the resource-efficiency of the NECOS Slice as a Service capability measuring the physical utilization of CPU and / or memory.

2.3. Acceptance Plan

The subset of non-functional requirements presented in Table 1 are supported through the KPIs identified in Table 2. Since the Slice Creation and Slice Elasticity workflows make reference to most of the presented functional requirements (see Figure 4), we decided to exercise them in order to validate the functional requirements.

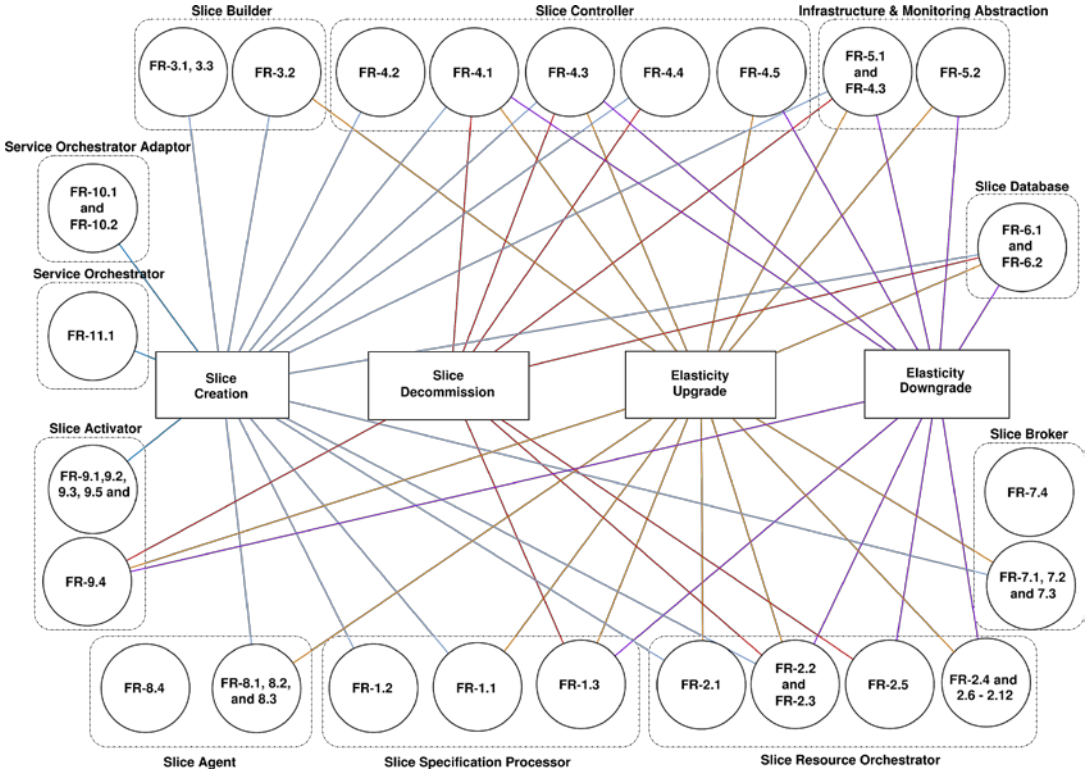

Figure 4. Functional requirements used in the workflows.

For system testing, each of the functional and non-functional requirements listed in Table 2 will be validated separately. The functional requirements validation will be performed with the focus of verifying that they have been implemented correctly and completely. If errors are found, these will be reported, and corrections or changes will be implemented to meet the requirements. For non-functional requirements, Table 2 presents how to evaluate these requirements, how to measure and implement the test. The acceptance criteria for system tests are: i) all the priority bugs are fixed and verified; ii) all the test cases have passed.

System testing is performed end-to-end by developers and testers to check if the software meets the specified requirements. While testing how the system is behaving as a whole, also its functionality and performance are checked by using demo data (including possible dummy inputs) instead of the production one. Through these tests the software is checked for complete specification including

hardware and software, memory, and number of users. Defects found during the system testing will then be fixed based on priorities.

For acceptance verification, the system tests results will be verified to approve the features coverage. All features must provide evidence that: i) there are no critical defects left open; ii) all functional and non-functional requirements attached to the features are correct and complete; and iii) the solution process is working fine, receiving entries correctly, processing as specified and returning correct outputs.

3. Integrated NECOS Platform Testing Environments

The NECOS integrated test environment is a distributed resource infrastructure that was used to host the NECOS prototypes in different locations in order to show that the devised distributed architecture is suitable for deployment in realistic scenarios.

3.1. Prototypes

The NECOS platform components used in the demonstrations are either developed from scratch or based on existing software components. Implementations from scratch usually imply fewer dependencies and requirements from external software and libraries, while, in general, an implementation based on re-using solutions contribute to an increased number of requirements. The prototypes that were fully presented in deliverable 5.2 [D5.2] are briefly summarised in this deliverable.

3.1.1. The Slice Builder

Description

The Slice Builder is responsible for building a full end-to-end multi-domain slice from the relevant constituent slice parts. When the Partially Defined Template (PDT) message has been specified, the Slice Specification Processor sends such a message to the Slice Builder invoking the initiate_slice_creation method.

Functional requirements implemented

- FR-3.1 - Analysis of specific policies and rules for slice requests
- FR-3.2 - Request resources for slice parts
- FR-3.3 - Provide different mechanisms for defining the final slice specification
- FR-3.4 - Creation of contracts for resources reservation among resource providers, NECOS platform and tenants
- FR-3.5 - Perform reservation and activation of slice parts
- FR-3.6 - Provide slice and slice parts information

3.1.2. Slice Spec Processor

Description

The Slice Specification Processor component handles the requests for slice creation coming from the Slice Activator (in the tenant's domain). In our prototype implementation, the Slice Specification Processor is implemented as a RESTful service using Python.

Functional requirements implemented

- FR-1.3 - Provide slice management interface (SSP)
- FR-2.1 - Management of connections between slice parts to have an end-to-end slice
- FR-2.2 - Interact with Slice Database for querying purposes and to provide slice updates
- FR-2.3 - Continuously update IMA regarding VIM, WIMs and Monitoring pointers
- FR-2.4 - Provide operational functions for slice management
- FR-2.5 - Request slice parts removal
- FR-2.6 - Process slice monitoring information provided by the IMA
- FR-2.7 - Evaluate the need for upgrading or downgrading the resources within a slice
- FR-2.8 - Evaluate requests for upgrading or downgrading the resources within a slice
- FR-2.9 - Selection of resources that will be allocated or freed in the elasticity process

- FR-2.10 - Request the addition or removal of resources within slice parts
- FR-2.11 - Provide an elasticity process that prioritizes vertical elasticity over a horizontal approach
- FR-2.12 Implement horizontal elasticity by requesting the creation or removal of slice parts
- FR-3.1 - Analysis of specific policies and rules for slice requests
- FR-3.2 - Request resources for slice parts
- FR-3.5 - Perform reservation and activation of slice parts

3.1.3. DC and WAN Slice Controllers

Description

The DC Slice Controller is the component of the NECOS Architecture in charge of creating DC slices within the data centre. It is responsible for allocating the required compute and storage resources for a given slice part, and returning a handle to a VIM running on it, for each data centre.

The WAN Slice Controller component resides inside each Network Provider and that dynamically creates a network slice, as a part of a full cloud network slice. A network slice is a set of virtual links that connects two DC slices. In order to create a network slice, the WAN Slice Controller manages all of the network resources in the network provider domain that are allocated to participate in slicing and keeps track of which network resources have already been allocated to which slice.

Functional requirements implemented

- FR-4.1 - Support allocation, removal and modification of resources for slice parts
- FR-4.2 - Accept or reject contracts for resource reservation
- FR-4.3 - Instantiation and removal of VIMs and WIMs for slice parts
- FR-4.4 - Support requests for connecting and disconnecting slice parts together
- FR-4.5 - Process requests for upgrading and downgrading resources within slice parts

3.1.4. WAN Slice Controller

Description

An (additional) implementation of the WAN Slice Controller has been developed in support of the Wireless Slicing sErvices (WISE) demonstration (see Section 4). The prototype leverages virtualization capabilities from both OpenWrt and Open vSwitch to create WiFi Local Area Network (LAN) slices on top of an off-the-shelf WiFi access point.

Functional requirements implemented

- FR-4.1 - Support allocation, removal and modification of resources for slice parts
- FR-4.2 - Accept or reject contracts for resource reservation
- FR-4.4 - Support requests for connecting and disconnecting slice parts together
- FR-4.5 - Process requests for upgrading and downgrading resources within slice parts

3.1.5. Slice Resource Orchestrator (SRO)

Description

It is responsible for combining the slice parts that make up a slice into a single aggregated Slice. It is also responsible to invoke the proper IMA resource adapters in order to instantiate the required service elements on the different slice parts (of the end-to-end Slice). This is done according to the particular service embedding strategy that was requested by the Tenant via their Service Orchestrator.

Functional requirements implemented

The current SRO implementation took the functional requirements in Deliverable 2.2 as input and covered several of them. Currently, the SRO implements the following functional requirements.

- FR-2.1 - Management of connections between slice parts to have an end-to-end slice
- FR-2.2 - Interact with Slice Database for querying purposes and to provide slice updates
- FR-2.3 - Continuously update IMA regarding VIM, WIMs and Monitoring pointers
- FR-2.4 - Provide operational functions for slice management
- FR-2.6 - Process slice monitoring information provided by the IMA
- FR-2.7 - Evaluate the need for upgrading or downgrading resources within a slice
- FR-2.10 - Request the addition or removal of resources within slice parts
- FR-2.11 - Provide an elasticity process that prioritizes vertical elasticity over a horizontal approach
- FR-2.12 - Horizontal elasticity by requesting the creation or removal of slice parts

3.1.6. Slice Database

Description

The Slice Database is a module that interacts with the Slice Resource Orchestrator in the Slicing Orchestrator. The Slices Database keeps information about the topology and the resources of the slices for their whole lifetime. The Slices Database stores the data of the deployed slices such as location, pointers and identifiers of its parts.

Functional requirements implemented

The Slice Database implements the following functional requirements.

- FR-6.1 - Provide an interface for slice management
- FR-6.2 - Provide information about slices, slice parts and services

3.1.7. Infrastructure & Monitoring Abstraction (IMA)

Description

IMA is responsible for ensuring the monitoring process of the end-to-end Slices created in the context of a NECOS Slice Provider. Each end-to-end Slice will include different slice elements, i.e., the resource substrate represented by the slice parts, as well as the virtual resources associated with the service elements running on those slice parts. Since IMA provides an adaptation layer between the Slice Resource Orchestrator and the VIM / WIM running in each slice part, it will also be responsible for the management of the services in terms of deploying, re-deploying and deleting the service components running inside each of the slice parts.

Functional requirements implemented

- FR-5.2 - Provide slice monitoring metrics
- FR-5.3 - Provide management operations for deployment of virtual functions
- FR-6.2 - Provide information about slices, slice parts and services (Slice Database)
- FR-9.3 - Start service deployment
- FR-9.4 - Support management functions for running slices
- FR-10.1 - Process service deployment requests
- FR-10.2 - Provide service access and monitoring interfaces to the tenant

- FR-11.1 - Provide service deployment within a slice

3.1.8. Slice Broker and Slice Agent with RabbitMQ

Description

The Slice Broker and Slice Agents components are part of the NECOS Marketplace. The Slice Broker is the component that receives a PDT Message request, decomposes it to its constituent slice parts and addresses each slice part request to different Slice agents. The SRA message returned at the end of this process, contains all the alternative offerings from providers for each part. This prototype was implemented in Python.

The Slice Agents are responsible for answering slice parts requests by matching the latter (i.e., ensuring "coverage", see D5.2) to provider resources and reporting back with an offer that minimizes the slice part cost. The prototype implementation includes components in Python and in SWI-Prolog (Constraint Logic Programming System).

The infrastructure that allows message exchange between the Slice Broker and the Slice Agents relies on the RabbitMQ[1] open source messaging broker that also handles the Slice Agent Registration process.

Functional requirements implemented

- FR-7.1 - Support a search mechanism for requesting resources from infrastructure providers
- FR-7.2 - Capability of identify potential network resource providers based on DC resource offers
- FR-7.3 - Provide resource offers in the form of alternative resources
- FR-7.4 - Accept registration of Slice Agents
- FR-8.1 - Process resource requests and check local availability
- FR-8.2 - Provide resource options as answers for resource requests
- FR-8.3 - Constantly check the availability of resources in the local domain
- FR-8.4 - Registration with the Slice Broker

3.1.9. Slice Broker with HUG

Description

Provides a rendezvous point where the Resources Providers can register and accept the requests from the Slice Provider that are routed by the Broker. The prototype is implemented as a RESTful service using Python and HUG[2].

Functional requirements implemented

- FR-7.1 - Support a search mechanism for requesting resources from infrastructure providers
- FR-7.2 - Capability of identify potential network resource providers based on DC resource offers
- FR-7.3 - Provide resource offers in the form of alternative resources
- FR-7.4 - Accept registration of Slice Agents

[1] https://www.rabbitmq.com/
[2] https://www.hug.rest/

3.1.10. Slice Agents with HUG

Description

Allows the Resource Providers to expose information about the available WAN and DC resources. After the Agents registration in the Broker, requests originated from the Slice Provider can be processed. This prototype is implemented as a RESTful service using Python and HUG.

Functional requirements implemented

- FR-8.1 - Process resource requests and check local availability
- FR-8.2 - Provide resource options as answers for resource requests
- FR-8.3 - Constantly check the availability of resources in the local domain
- FR-8.4 - Registration with the Slice Broker

3.2. The NECOS roles

This section describes the types of NECOS roles namely, Slice Provider, Resource Provider, and Resource marketplace. These roles have been described in the deliverable "Consolidated definition of use cases, business models and requirements analysis" [D2.2]. Also, the needed NECOS software components for each NECOS role are listed. Also, the NECOS software components needed for each NECOS role are listed.

3.2.1. Slice Provider

The Slice Provider is composed by following NECOS software:

- The Slice Builder and Slice Spec Processor;
- SRO;
- Slice DataBase;
- IMA.

3.2.2. Resource Provider

The Resource Provider is composed by the following NECOS software:

- DC/WAN Slice Controllers;
- Slice Agents.

3.2.3. Resource Marketplace

The Resource Marketplace is composed by the following NECOS software:

- Slice Broker.

3.3. Infrastructure for validation

The infrastructure for validation is a combination of different Islands (i.e., a set of resources made available from a single administrative domain) together with the network interconnecting them. Connectivity between Islands could be through either private or public infrastructures like Software Defined Networking (SDN) infrastructures, Virtualized infrastructures or standard Internet Service

Provider (ISP) connectivity. Due to security constraints, private networks through public Internet are set up between the different islands for our validation purposes.

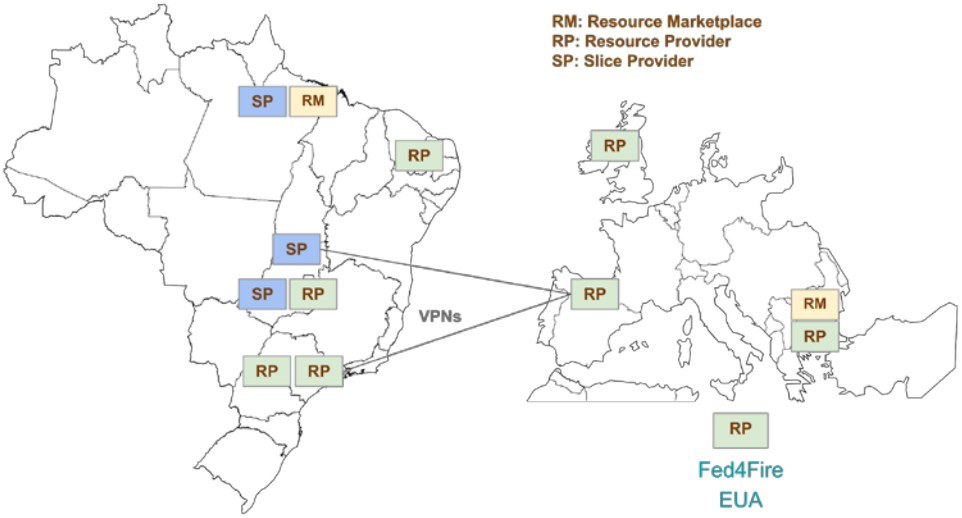

Figure 5. NECOS Integrated test environment.

3.3.1. Islands

The NECOS integrated test environment is composed, as presented in Figure 5, by the following islands:

- Slice Provider at UFG: 1 Dell EMC PowerEdge R740 server, equipped with two Intel Xeon Silver 4114 processor, 128 GB (8x 16GB RDIMM, 2666MT/s, Dual Rank) of RAM, and 12 TB of HD;
- Slice Provider at UFU: 1 DELL PowerEdge R740 of 64GB-RAM and two Intel Xeon Silver 4114 2.2G 10C/20T, each server;
- Resource Marketplace at UOM using Slice Broker with RabbitMQ: One Dell PowerEdge R630 (8 CPU cores @2.1Ghz, 16-48GB RAM) server hosting (XEN Server virtualization environment) hosting VMs for the Broker and the RabbitMQ messaging platform;
- Resource Marketplace at UFPA using Slice Broker with HUG: The Slice Broker is running in a VM with 2 CPUs and 4 GB of memory hosted in a Server Dell EMC PowerEdge R740, Intel Xeon Silver 4114 2.2G, 10C/20T, 9.6GT/s 2UPI, 14M Cache, Turbo, HT (85W) 64GB DDR4-2400;
- Resource Provider at UFU: The resource provider located at UFU consists of a rack with three DELL PowerEdge R740 of 64GB-RAM and two Intel Xeon Silver 4114 2.2G 10C/20T, each server. The servers compose an OpenStack cluster in which virtual machines are used to deploy slice parts by adopting Kubernetes as VIM;
- Resource Provider at UNICAMP: The resource provider located at UNICAMP consists of a Dell PowerEdge R740, with OS: Linux Ubuntu 18.04 LTS, Kernel: 4.15.0-51-generic x86_64, two Intel Xeon Silver 4114 CPU @ 2.20GHzD-1518 processors, RAM 64GB DDR4 and HD 2TB;
- Resource Provider at UFSCar: The resource provider located at UFSCar Sorocaba consists of a Supermicro model X10SDV-TP8F, with OS: Linux Ubuntu 18.04 LTS, Kernel: 4.15.0-58-generic x86_64, CPU Quad core Intel Xeon D-1518 2.2GHz (-MT-MCP-) with 8 threads, RAM 64GB DDR4 and HD 2TB;

- Resource Provider at UOM: This resource provider (acting as an edge cloud server) is located at the University of Macedonia and has OS: Linux Ubuntu 18.04 LTS, kernel: 4.15.0-55-generic x86_64, CPU Intel(R) Core(TM) i7-3770K CPU @ 3.50GHz, 16GB DDR3 RAM, and HD 1TB;
- Resource Provider at UFRN: UFRN's REGINA-Lab testbed is built on top of an OpenStack-empowered cluster enclosing three Dell PowerEdge R740 servers interconnected by OpenFlow-enabled wire-meshed switches with 1 Gbps interfaces. The compute infrastructure is composed by three servers each featuring 40 vCPUs and RAM of 64 GB. The SDN infrastructure entails 6 Mikrotik RB951G-2HND (CPU of 600 Mhz and RAM of 128 MB) switches, along with two TP-Link 802.11b/g/n TL-WR1043ND v3 Access Points to provide broadband WiFi-sharing connectivity;
- Resource Provider at UCL: The resource provider located at UCL is based on: 1x Blade server M630, relying on 2 x Intel(R) Xeon(R) CPU E5-2680 2.70GHz and 160Gb of RAM; 1x Dell PowerEdge R730 with Intel Xeon E5-2680 v3 2.5GHz, 12 cores, 192GB RAM, 1.2TB HDD; 4x servers with Quad-Core AMD Opteron(R) Processor 2347 HE, 32 GB RAM, 700GB HDD;
- Resource Provider at 5TONIC: the resource provider enabled by TID at 5TONIC premises is based on 1x PowerEdge R720 with 2x Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz, 6 cores per socket (for a total of 12 cores or 24 vCPU), 128GB of RAM, and 2TB as HDD;
- FED4FIRE Experimental Testbeds for the Marketplace Resource Discovery Demonstration: Information regarding resource availability is going to be obtained by the FED4FIRE testbeds, that will be considered as resource providers in the marketplace demo. Hosts in some of the testbeds listed below are going to host the Slice Agents of the Marketplace Resource Discovery. We list below information for the testbeds used and examples of configurations of nodes (hosts/servers) in clusters, where the latter are groupings of identical nodes. For conciseness, we omit full node specs of all clusters in a testbed.
    - Virtual Wall 1: 3 clusters, 206 nodes (servers), e.g. cluster:
        - 100 x pcgen2 nodes: cpu 2x Quad core Intel E5520 (2.2GHz) CPU, ram 12GB, hdd 1x 160GB harddisk, lan 2-4 gigabit nics per node.
    - Virtual Wall 2: 5 clusters, 162 nodes (servers), e.g. cluster:
        - 100x pcgen3 nodes: cpu 2x Hexacore Intel E5645 (2.4GHz) CPU, ram 24GB, hdd 1x 250GB harddisk, lan 1-5 gigabit nics per node.
    - Cloudlab Utah: 1 cluster, 315 nodes (servers), e.g. cluster:
        - 315 x m400 nodes: cpu: Applied Micro X-Gene system-on-chip, Eight 64-bit ARMv8 (Atlas/A57) cores at 2.4 GHz, ram 64 GB, hdd 120 GB of flash (SATA3).
    - Cloudlab Wisconsin: 2 clusters, 100 nodes (servers), e.g. cluster:
        - 90 x Cisco UCS SFF 220 M4 nodes: cpu: 2x Intel E5-2630 v3 85W 8C at 2.40 GHz for a total of 16 cores, ram 128 GB, hdd 375 TB.
    - Grid5000: 33 clusters, 1064 nodes (servers), e.g. clusters:
        - 32 x Grenoble dahu nodes, cpu: 2 x Intel Xeon Gold 6130 16 cores/CPU, ram 192 GB, hdd 240 GB SSD + 480 GB SSD + 4.0 TB HDD, lan 10 Gbps.
        - 64 x Nancy grvingt nodes, cpu: 2 x Intel Xeon Gold 6130 16 cores/CPU, ram 192 GB, hdd 1.0 TB HDD, lan 10 Gbps

3.3.2. Private Interconnection between Europe and Brazil

This section describes how the interconnection between the islands was achieved and the technologies used in the NECOS demonstrations.

Islands connectivity

All the islands running the Slice Provider components are capable of providing network connectivity to the Tenant, in a way that instantiated DC's resources at an island can have network connectivity to another DC resource instantiated on a remote location. This will include the case of different slice parts that have been allocated in several NECOS islands but belong to a common end-to-end Slice. On the control level, all NECOS components use standard network connections to communicate, making use of the Internet and VPN.

Software

The connectivity at the slice parts level is provided by Virtual Extensible LAN (VXLAN) tunnels using Open vSwitch (OVS)[3]. An OVS bridge is created on demand for each instantiated slice part and is then connected to another OVS bridge instantiated on the remote edge where the connectivity must take place.

We have implemented a solution that takes care of the setup of the multiple tunnels as part of our WAN Slice Controller component, as described in subsection 3.1.1. In practice, the solution is based on multiple agents located at each DC resource provider and one central element that communicates with the agents while also managing a database containing information regarding all created connections and their respective attributes. The resulting connectivity is a L2 network overlay over the existing L3 network layer, that in most cases relies on the Internet.

Issues

By default, the VXLAN tunnels do not implement encryption, so security concerns might be an issue to be considered. This can be overcome by setting Virtual Private Network (VPN) tunnels under the VXLAN tunnels, but at the moment it is not part of our solution. As one of the requirements of our network connectivity solution, the Slice Provider edges to be connected need to be able to communicate to each other over the network already, since this connectivity will be the base to the overlay network to be instantiated.

Most of our islands have a public IP address reachable over the Internet. However, in the case of the Telefonica island, the only way of accessing the 5TONIC server running the DC component was through a VPN connection. Therefore, the VXLAN tunnels for instances located on this edge were configured using VPN tunnels. Although this approach allowed solving the above issue, it required an additional layer of complexity which also impacts the performance of the connection.

[3] http://docs.openvswitch.org/en/latest/faq/vxlan/

4. Demonstrations and NECOS Validation Results

In the context of the NECOS project, five demonstrations have been worked out. The main one involves multiple tenants who request services provided via the NECOS platform running in an integrated test environment. The remaining demonstrations focus on more narrowly defined and specific aspects like scalability, intelligent mechanisms, and so on. Each demonstration has an objective with a series of inputs provided by the tenant, and some expected outputs. The tenant is responsible for the service running in the slice, playing the role of a customer requesting the slice in order to accommodate the deployment of a service.

4.1. MUlti-Slice/Tenant/Service (MUSTS)

MUSTS is the main NECOS demonstration that creates 2 slices using one slice provider hosted by UFG, the marketplace hosted by UFPA, and four resource providers (UFSCar, 5TONIC, UoM and UNICAMP), as presented in Figure 6. Slices requested by two different tenants are shown through their complete life cycles. Each tenant has specific resource and service constraints that must be supplied and guaranteed by NECOS. More specifically, one tenant runs a Touristic service while the other runs an Internet of Things (IoT) service, which are derived, respectively, from the Network Slicing for Touristic Content Distribution and Network Slicing for Metropolitan Integrated Monitoring scenarios [D2.2].

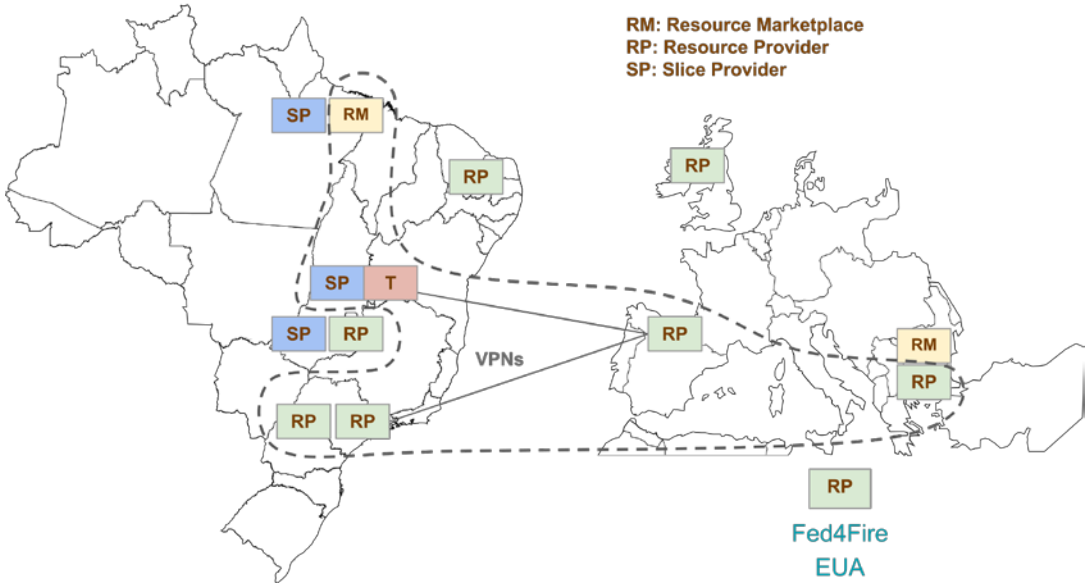

Figure 6. Instantiation of the MUSTS demo on the distributed experimental infrastructure.

4.1.1. Objectives

This demonstration aims at exercising the following key features of NECOS: slice creation, slice decommission, slice monitoring, service deployment, service update, VIM heterogeneity, and elasticity upgrade (both vertical and horizontal). The features slice creation, slice decommission, slice monitoring and service deployment are features exercised in both slices, as they are essential for every slice. However, the feature VIM heterogeneity is exercised only in the Touristic service slice to showcase the capability of NECOS to support different VIM technologies, in our case, Docker and XEN. The elasticity feature is exercised in the IoT service slice, in which we demonstrate the NECOS capability of vertical and horizontal elasticity upgrades.

Touristic Service for End-to-End Slice

The touristic Content Delivery Network (CDN) service is a cloud network slice use that delivers touristic content to users based on their geographic location. The idea is that tourists visiting specific locations (e.g., cities, archaeological sites, museums, churches, or towers) are more likely to request content related to a geographical site.

Assumptions of the touristic CDN scenario include: i) a central Web server hosting all content (videos/web pages), and ii) three edge cloud nodes hosting a single video and a web site related to their geographic location. For instance, the core cloud node could be Brazil-Unicamp, an edge cloud node with Greek Touristic content could be Greece-UOM, an edge cloud node with Spanish content could be 5G-Tonic and another edge cloud with Brazilian content could be Brazil-UFSCar. Touristic content requests (video or web content) from a visitor sightseeing Spain are directed either to the local edge cloud server, in case that the requests are related to Spain (e.g., visiting hours for the Royal Palace of Madrid), or to the core server if his requests are irrelevant to his position (e.g., a youtube video for healthy eating).

Our implementation approach regarding the aforementioned service aims at highlighting how the CDN services benefit by using the NECOS Platform. More importantly, we demonstrate that CDN technology brings services close to the end-users achieving efficiency not only in terms of performance (e.g., connection time, download time) but also in respect to the processing resources required. For example, in the touristic CDN service, applications running at the network edge are lightweight, and the edge infrastructure computing resources are much less powerful than the ones at the core. What is more, the NECOS Platform facilitates the deployment of such services.

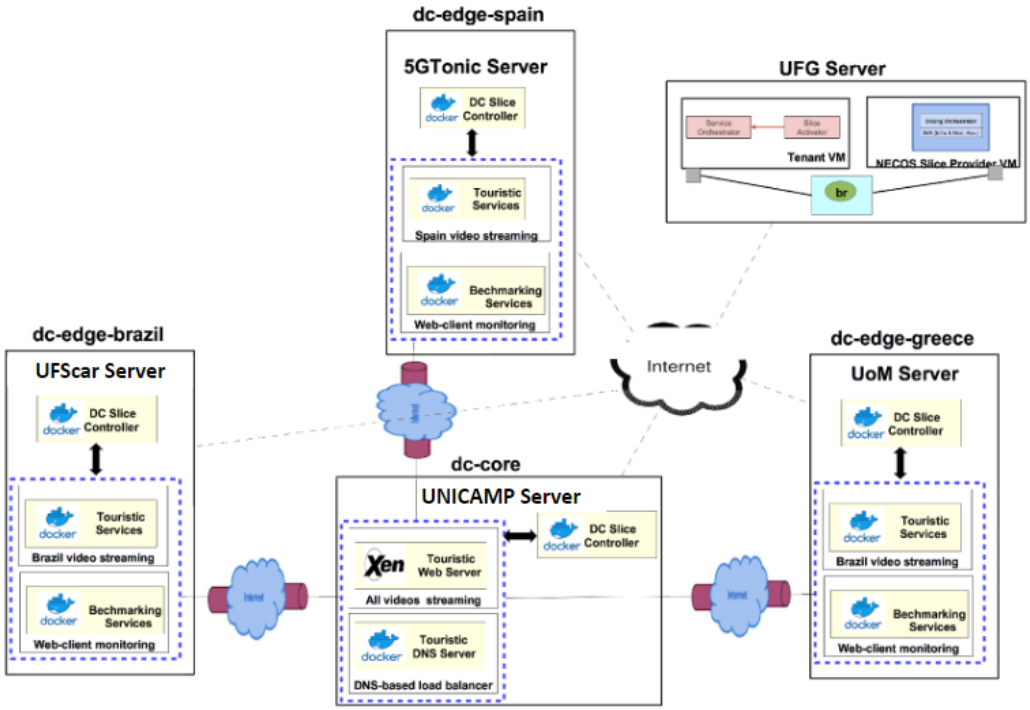

Figure 7. CDN Slice Overview.

Figure 7 shows the connections among the components in the touristic CDN service. Boxes in blue represent the slice-parts holding the CDN service components. Each slice part in the touristic CDN deployment is composed of a single VM. Such figure also shows the different VIM technologies being used in a geographically distributed slice: the dc-core is using XEN as the VIM and the dc-edges are

using Docker. The dc-core slice part accommodates the content services and the Domain Name System (DNS) load balancer is responsible to direct the requests to the appropriate DC slice part (core or edge). As previously mentioned, the requests are shared among the DC slice-parts according to the client's geographic location. The dc-edges host the content services and benchmarking tools (e.g., load testing tools), which are used to test the performance of touristic service.

Experimental setup

Core DC - the applications that are running in the core are the following: the DNS load balancer, an Apache web server which provides all the web pages, VLC video streaming servers, and the Grafana and Influx-db monitoring tools in the touristic Web Server. The inputs in the monitoring tools are the results from the benchmarking tools, which are running on edge cloud nodes.

Edge DC - the services at the edge dc slice parts have been containerized (e.g., Docker). For the web services, we use flask and for the video streaming services we use VLC. The benchmarking load testing tool used is jmeter.

DNS Load balancer - it has a major role in the touristic CDN deployment. The DNS server uses a JavaScript Object Notation (JSON) configuration file (shown in Figure 8) which has the current information about domain names and slice parts' IPs. In this section, we present only the part of the JSON file which is used for the touristic CDN scenario for an edge cloud slice (e.g., dc-edge-brazil).

```
"dnsEntries" : [
   {"domain": "core.swn.uom.gr",
    "config" : { "algorithm" : 1},
    "records": {
       "core" : ["195.251.209.201"]  }      },
   {"domain": "brazil.swn.uom.gr",
    "geoLocation" : {  "brazil" : ["195.251.209.0/24"] },   <- If the request is from Brazil
    "records": {       "brazil" : ["195.251.209.218"],      <- send to Brazil's edge
                       "core" : ["195.251.209.201"] }       <- else send to core dc
   } ]
```

Figure 8. JSON file used for touristic CDN scenario.

The DNS can be reconfigured while it is running so we can easily apply any changes that happened after the deployment (for example the DC's IPs). So particularly, the DNS should know the slice parts' IPs (the IPs are defined in the JSON configuration file), while the core's IP should be added in the edge slice parts' *resolv.conf* file.

4.1.2. IoT Service

The IoT Demonstration aims to show that NECOS is suitable and also a facilitator in supporting the deployment, management and operation of real IoT solutions; being able to provisioning and monitoring resources, deploying services and scaling slice resources according to load changes. As mentioned above, beyond demonstrating the slice creation, slice decommission, slice monitoring and service deployment, this slice specifically showcases the service update and elasticity upgrade, both vertical and horizontal.

The chosen IoT scenario consists of a real-time cargo monitoring and tracking IoT solution, where a monitoring device with wireless communication and multiple sensors (temperature, humidity, light,

and gps) is attached to a cargo container and rides with it all along its journey. This device periodically "wakes-up" and transmits precise monitored data to a centralized system, which allows customers to have visibility of their goods in movement and being notified when something happens with their cargo (door openings, extreme temperature shifts, etc.). This scenario is illustrated by Figure 9.

Figure 9. Real-time cargo monitoring and tracking.

The IoT demonstration was built using Dojot, an open source IoT Platform whose development is led by CPqD, and a load testing tool for Message Queuing Telemetry Transport (MQTT) IoT devices, which was customized for this demonstration. The software components were deployed, initially, in two slice parts as depicted in Figure 10.

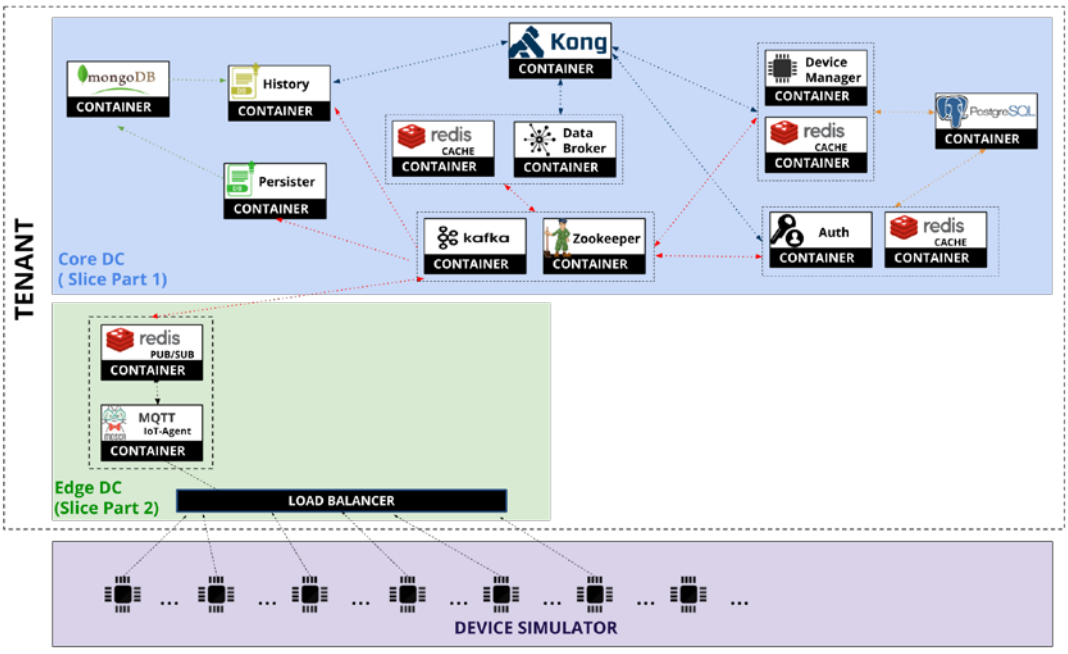

Figure 10. IoT Demonstration setup based on Dojot micro services.

The Core DC (slice part 1) runs the cloud Dojot micro services, which are responsible for: managing the IoT devices life cycle, storing telemetry data, and providing Representational State Transfer (REST) and socket.io interfaces to retrieve, respectively, historical and real time data about the devices. In this

Demonstration the slice part 1 is a set of three VMs managed by the Kubernetes VIM (one VM hosting the master node and the other two representing the worker nodes). The Dojot micro services are deployed in the worker nodes.

The Edge DC (slice part 2) runs the edge Dojot micro service, called MQTT IoT-Agent, which is responsible for establishing connections with the IoT devices and transform the data to be transmitted to the cloud micro services. In this Demonstration the slice part 2 is a set of two VMs managed by the Kubernetes VIM (one VM hosting the master node and the other representing the worker node). The edge Dojot micro service is deployed in the worker node.

The device simulator runs outside the NECOS infrastructure and simulates connected IoT devices publishing telemetry data according to some input settings.

Figure 11 shows the IoT Slice overview. The Core DC is at Unicamp, Campinas, and the Edge DC is at 5TONIC, in Madrid. Note that for horizontal elasticity upgrade, a new slice part is created at UFSCar, Sorocaba.

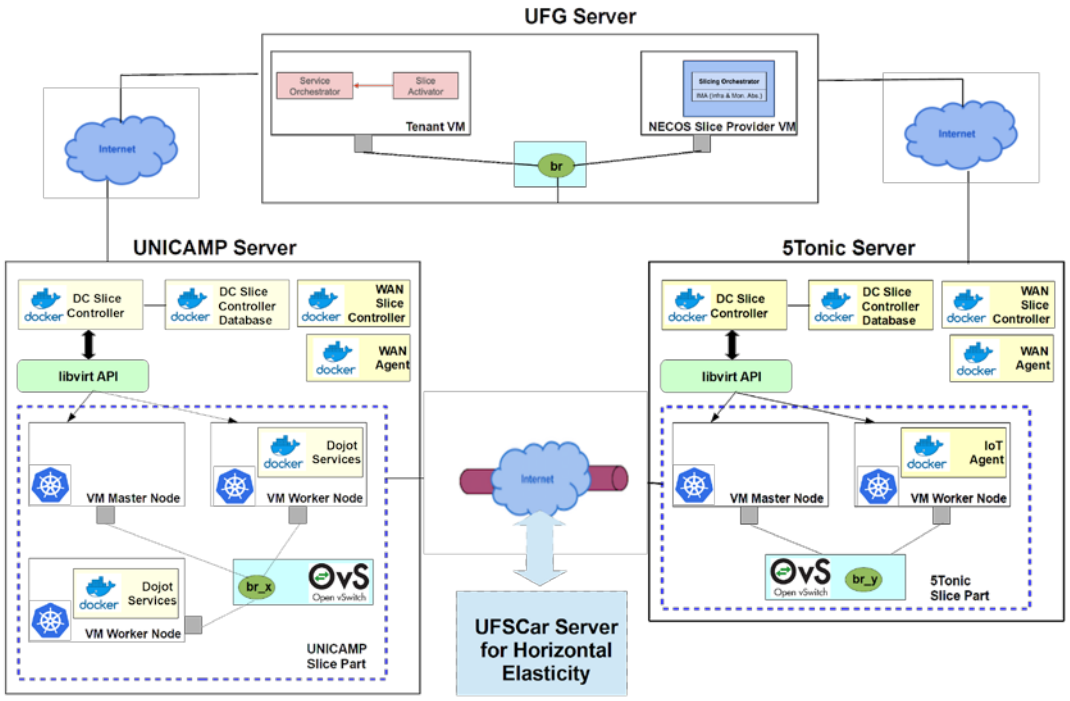

Figure 11. IoT Slice overview.

The major flow of the telemetry data inside the platform is described in Figure 12 with each step identified by a red circle with a number inside. A detailed description of each step is given below:

1) The simulated device opens an MQTT connection through a TCP Load Balancer, publishes some telemetry data, and closes the connection;
2) The TCP Load Balancer redirects the connection/telemetry data to an instance of the MQTT IoT-Agent;
3) The MQTT IoT-Agent authorizes the device connection and sends the telemetry data to the REDIS DB, responsible for implementing the PUB/SUB of the MQTT Broker, and also sends it

EUB-01-2017

with some extra metadata to the Apache Kafka, responsible for redistributing it to the other dojot micro services;

4) The Persister micro service consumes the telemetry data from Kafka and stores it into the MongoDB;
5) The DataBroker consumes the telemetry data and provides it through socket.io to the registered clients.

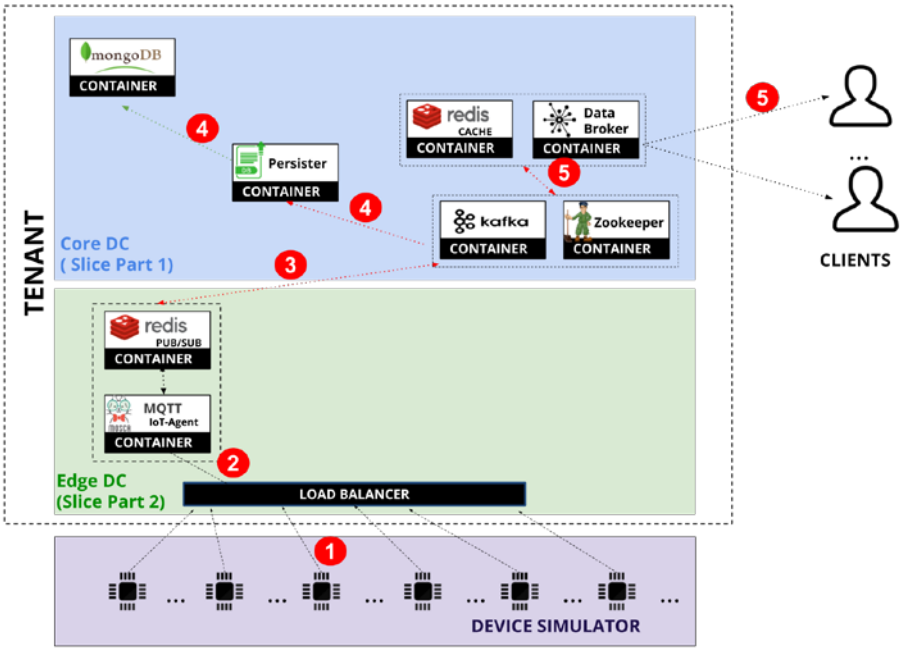

Figure 12. Dojot data flow.

An increasing number of cargo containers was simulated, publishing telemetry data (temperature, humidity, lightness, gps) with a given periodicity. When the machine hosting the IoT Agent in the edge was overloaded due to an increase in the number of requests to the IoT Agent, elasticity took place to avoid a degradation of the quality of the service (connections rejections, messages losses and long response time). First of all, NECOS would try to do a vertical elasticity upgrade, which means to add a new machine to host another IoT Agent in the same slice part. Specifically, in our case NECOS will make vertical elasticity in the Edge DC (slice part 2). Then, after making vertical elasticity, the service is redeployed so that the new machine receives the new IoT Agent and the requests coming from the sensors are now balanced between both IoT Agents.

A second experiment that was done is related to horizontal elasticity. In this case, NECOS tries to make vertical elasticity in the Edge, but there are no more resources in that slice part. Then, SRO triggers the horizontal elasticity upgrade which means to add a new slice part to the existent slice. Specifically, for this experiment, a new slice part at UFSCar is created. After the SRO receives the return about the creation of the new slice part, it triggers the service update (redeployment) so that the new slice part is used by the Dojot service. After the service redeployment, requests coming from the sensors are balanced between the IoT Agent at 5TONIC (the overloaded slice part) and UFSCar.

EUB-01-2017

4.1.3. Workflow

Since this demonstration is the one which exercises the majority of the NECOS features and follows the working flows defined in D5.1 and D5.2, the working flow presented in Figure 13 is depicted in a higher level of detail.

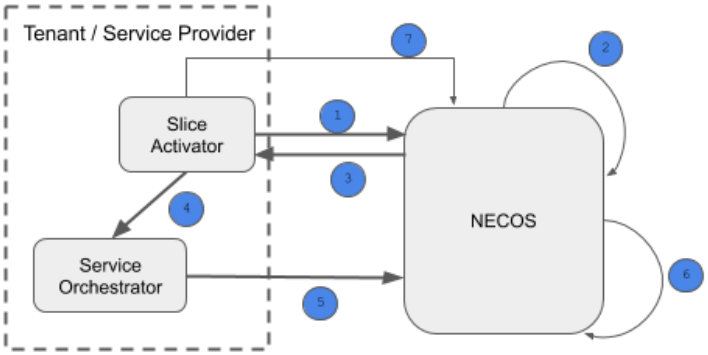

Figure 13. MUSTS demonstration workflow.

- Step 1: The tenant calls the NECOS system to create a slice. In this call, the tenant must inform the specification of the slice so that NECOS will be capable of building it;
- Step 2: This step, abstracted as a single step in this figure, follows the complete slice creation working flow presented in D5.2. In this step, actions such as looking for candidate slice parts in the Marketplace and start the monitoring of the slice are performed;
- Step 3: The NECOS system returns to Tenant all the details about the slice that was created, including information about every slice part, physical resources, location, etc.;
- Step 4: The Tenant calls its Service Orchestrator so that it can embed into the slice the service to be run. The Service Orchestrator can be a very smart engine or a very simple solution (even performed by hand by an operator) to decide where to put every service inside the slice;
- Step 5: Service Orchestrator triggers the deployment of the service. NECOS is responsible for parsing the YAML file received and dispatch specific commands in each slice part to deploy the correspondent service inside the slice;
- Step 6: This step represents the elasticity. Specifically, for this Demonstration, it refers to the vertical and horizontal elasticity upgrade, which means adding a new resource inside a slice part or adding a new slice part, respectively. This is done when a defined threshold is reached. For this Demonstration, the SRO is observing the CPU load of the machine where the IoT Agent is running. When the load reaches 80%, the elasticity is triggered. Again, as with step 2, the elasticity upgrade (both vertical and horizontal) abstracted here in a single step, is fully described in D5.1;
- Step 7: This final step represents the slice decommission call done by the Tenant to delete the slice as well as the service running inside it.

4.1.4. Results

This demonstration validates 6 of the prioritized NECOS features. Each one of them is presented together with the respective results from the experiments performed following the steps defined in the last subsection.

In order to create a slice, the steps 1 to 3 must be performed. Once this creation is started, the time is collected in order to provide the average slice provision time in seconds (KPI 4). The provision time includes the reservation time and the instantiation time, as presented in Figure 14. The instantiation time is bigger because it incorporates both times to deploy the VIM on-demand and configure all the resources to be part of the slice.

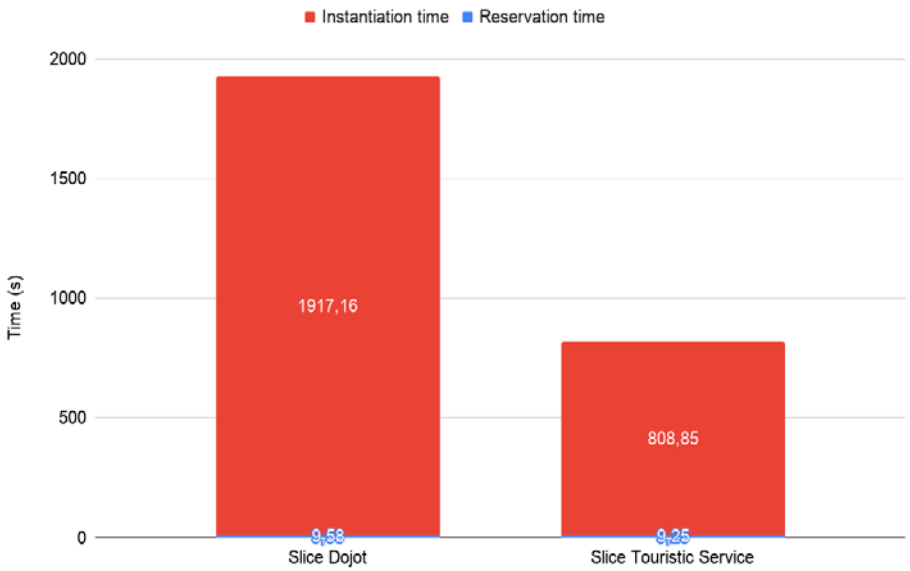

Figure 14. Average slice provisioning time (KPI 4).

Once all the slice parts are allocated, steps 4 and 5 are executed and the service is deployed as part of the slice management tasks. Figure 15 presents the overall time for deploying the service from the tenant.

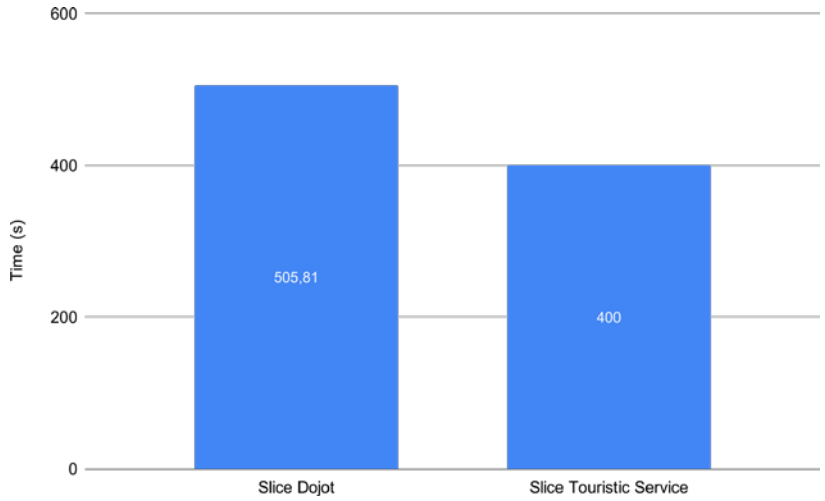

Figure 15. Average service provisioning time (KPI 3).

After steps 4 and 5 are performed the service is running in the slice, so it is possible to overload one slice in order to see if the isolation is working properly. Figure 16 shows the last 100 seconds before SRO triggered elasticity. As can be seen, after the second 50, the service running on slice 2 (Dojot)

EUB-01-2017

started to use a high percentage of CPU without affecting the CPU usage of slice 1 (Touristic), showing that the isolation works among slices.

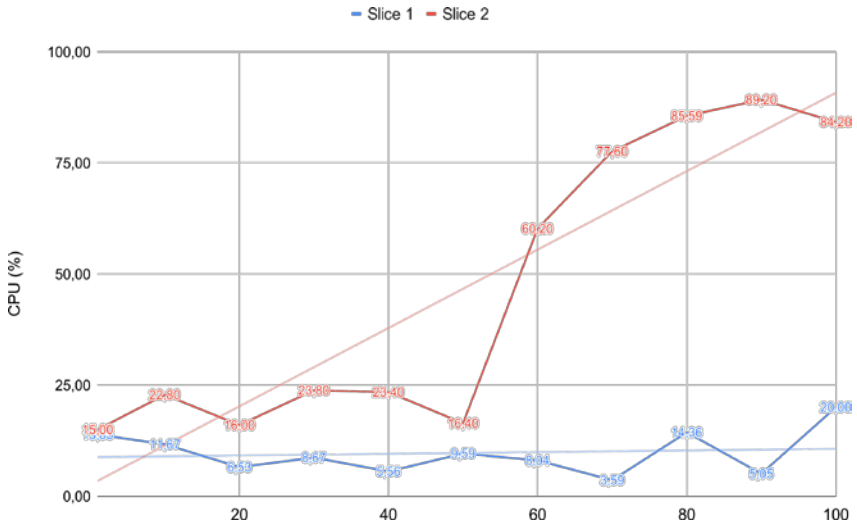

Figure 16. CPU Isolation.

The step 6 perform the elasticity. Figure 17 presents the results for the horizontal elasticity upgrade, which means adding a new slice part to the slice. The time reported in this graph includes the call to the slice builder to find the best option to place the new slice part and deploy it, two calls to IMA for updating monitoring and management and two more calls to IMA to redeploy and monitor the service.

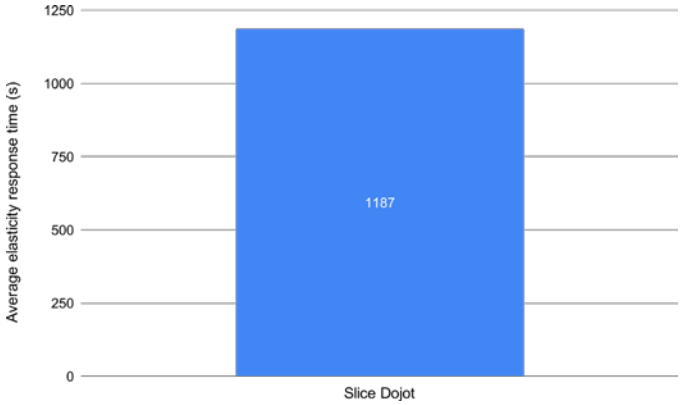

Figure 17. Average elasticity response time (KPI1).

During the slice execution, the IMA is performing the monitoring, thus, Figure 18 shows the number of metrics that are collected from each Slice every 1 minute. In this case, each Slice has different amounts of resources (Virtual Machines) to be monitored, so the number of metrics collected is different between them. It can be noted that the metrics grow linearly, and the distance between the number of metrics per Slice increases as well, although this difference may vary as the features for each Slice change, this is a characteristic of the IMA database.

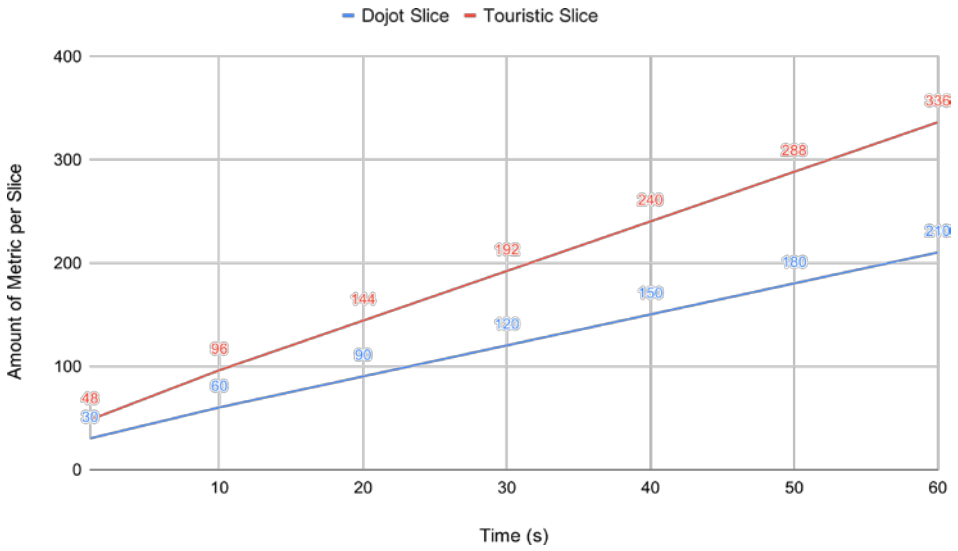

Figure 18. Monitoring-data availability (KPI 7).

In order to showcase the VIM on-demand concept, Figure 19 presents the average slice provisioning time, focusing on the VIMs deployment time. As highlighted in the figure, the Dojot slice uses only one type of VIM while the Touristic slice uses 2 VIM types. The difference in the VIMs deployment time for the slices is due to the different number of VMs used by each slice, i.e., the Dojot slice instantiates more VMs per slice-part than the Tourist slice. The VIM deployment time is also influenced by the complexity of the VIM.

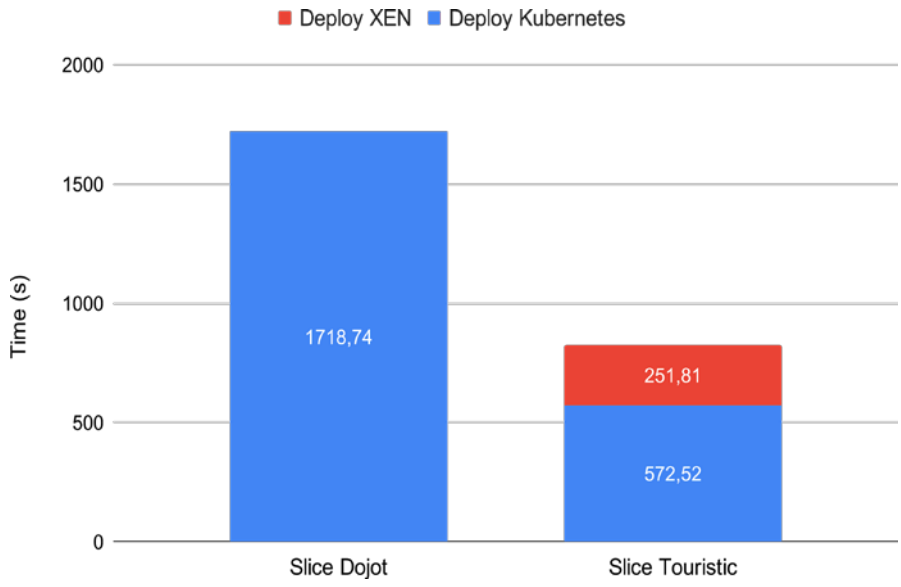

Figure 19. Average slice provisioning time (only VIMs).

EUB-01-2017

4.2. Marketplace (MARK)

This demonstration requests multiple slice allocations using one marketplace hosted by UOM and the Resources Providers from the FED4FIRE as presented in Figure 20.

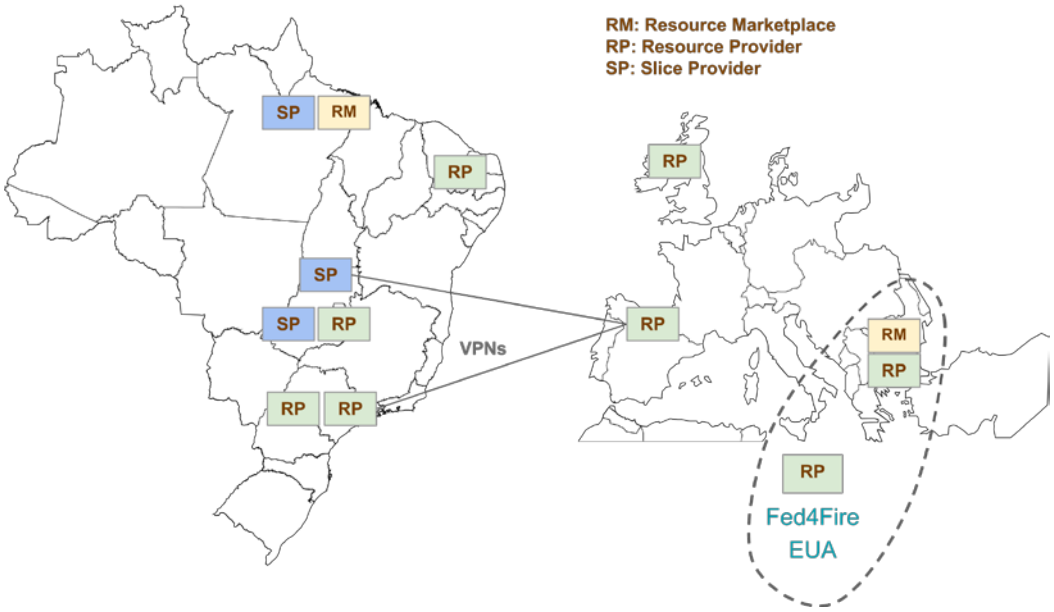

Figure 20. Instantiation of MARK on the experimental infrastructure.

4.2.1. Objectives

The main objective of this demo is to demonstrate that the marketplace concept introduced in NECOS as a dynamic resource discovery mechanism can cope with slices of significant size and multiple geographically distributed resource providers. We assume multiple slice requests and demonstrate the behavior of the marketplace components, the involved workflows and the relevant resource discovery performance.

In order to obtain real world data, namely the status of resources, we use 6 FED4FIRE testbeds (http://www.fed4fire.eu/testbeds). We developed a (Python) Translator component that is responsible to directly communicate with the corresponding test-bed control interface (e.g., jFed CLI), and to translate the response message into a uniform format. In practice, we maintain a local representation of the resources in JSON format from the following open-access test-beds: Virtual Wall 1, Virtual Wall 2, Cloudlab Utah, Grid5000, Cloudlab Wisconsin, and w-iLab2. This treatment of the resources' features is critical because FED4FIRE represents their resources through Resource Specifications (called RSPECs), but not in a uniform manner throughout the test-beds, e.g., the resources may have incomplete details or present different attributes.

FED4FIRE testbeds are organised in clusters of nodes, where nodes in a cluster are machines of identical configuration. Each node in a cluster is associated with a cost and in all experiments, we assume that one Virtual Deployment Unit (VDU) service is allocated to a single cluster, thus that cluster should “cover” the VDU Extended Platform Awareness (EPA) attributes.

We place the Slice Agents in Fed4Fire testbeds. To facilitate the experimentation process, we reserved three hosts in a subset of the aforementioned testbeds, and allocated our agents on these three hosts. The allocation is the following:

- Virtual Wall 1 (Europe): 4 DC Agents responsible for resource discovery of all the European Testbeds and one WAN Agent;
- Virtual Wall 2 (Europe): 6 DC Agents with “semi-artificial” resource data, and one WAN Agent;
- Cloudlab Utah (USA): 2 DC Agents responsible for resource discovery of all the USA Testbeds and one WAN Agent.

Each DC Agent is responsible for a testbed, communicating with the latter through the translator. Each Slice Agent has a different cost for the hosts it offers. Since FED4FIRE is an open platform, this cost was generated using random values. Since the time required by the Slice Agents to report back testbeds availability information is far greater than the actual time required to deploy them, we consider that the allocation of the Slice Agents in different testbeds would not significantly change our results. In order to further investigate further the proposed approach, we have generated “semi-artificial” data regarding resource providers, that are variations in terms of resource availability and host characteristics based on the real data obtained by the FED4FIRE testbeds. We also included 3 WAN-Provider Agents that offer connectivity between DC slice parts; in our tested we considered a fully connected graph between our slices, i.e. all WAN Agents are able to connect to all the testbeds albeit at a different cost.

The Slice Brokers (that use the RabbitMQ service) are hosted on a server on the UOM premises. We have generated multiple slice request cases, by varying the number of slice parts, services hosted in each slice part and their tenant requested geographic constraints. In all cases, we investigated the number of messages exchanged, the number of slice parts alternatives and the total number of alternative slice instantiations and their total cost, from which we derived the slice instantiation of the minimum cost.

Finally, since the testbed deployment is time consuming and in order to further investigate the scalability of the proposed approach, we report also on single host experiments, that would allow us to conduct (more easily) an experimental evaluation. This single host experiments are reported in the corresponding section below. It should be stressed that the implementation tested in the Fed4Fire Demo and Single Host experiments is identical.

4.2.2. Workflow

The Slice Broker and Slice Agents prototypes implement fully the NECOS Marketplace functionality. In particular, the current implementation is based on the workflow depicted in Figure 21 (Deliverable D5.1).

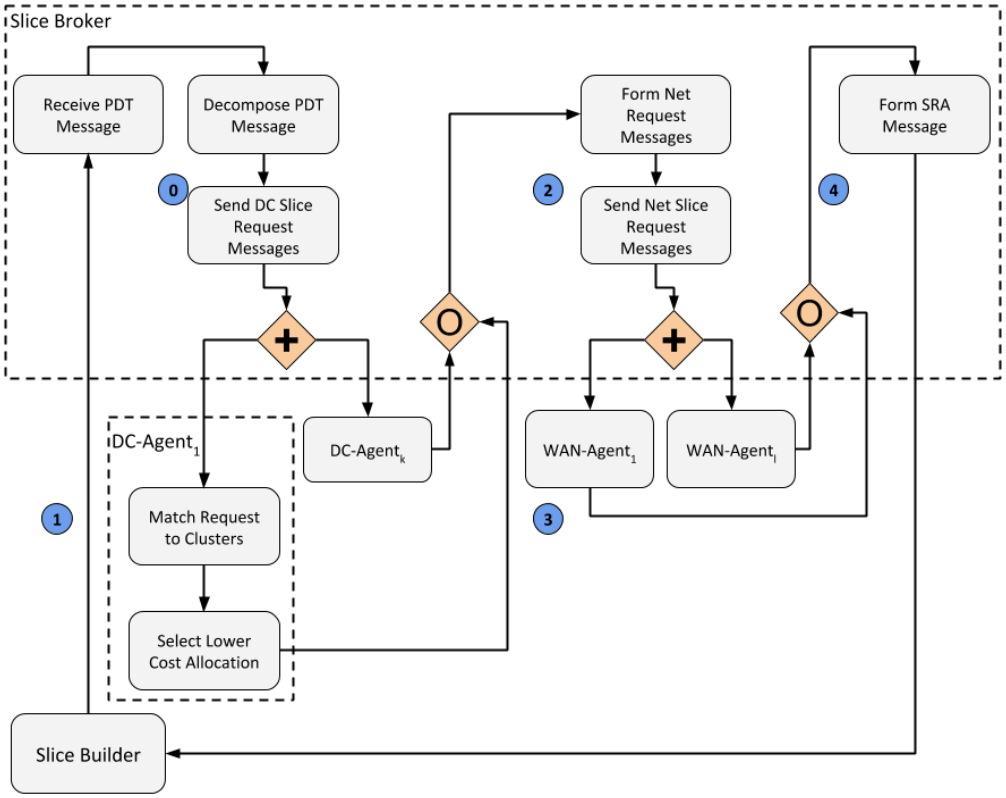

Figure 21. Marketplace Resource Discovery Workflow.

- **Step 1:** The Broker receives the PDT Message from the Slice Builder and extracts from it the different DC slice parts. Each of such DC slice parts is enriched with the necessary information regarding its host constraints, obtained by the corresponding service specification that is included in the PDT Message (please see deliverables D4.2 and D5.2) and is forwarded to the agents via the RabbitMQ messaging platform, annotated with the corresponding geographic constraints.
- **Step 2:** A Slice agent that receives a slice part request message, computes the clusters that can host DC part requested. Since multiple clusters can host a VDU in a slice part (please see D5.2) the agent computes its minimum cost answer, based on any allocation constraints regarding host availability. Finally, this answer is communicated back to the Broker.
- **Step 3:** After having collected all available DC-Agent answers for all the DC slice parts, the Broker forms requests addressed to Net-Agents. In order to do so, the Broker extracts connectivity information from the slice graph described in the PDT message. Thus, for each net-(slice) part it forms a request annotated with the network details (IP) of the providers at the ends of each connection. Each such message is sent to all the WAN-Agents.
- **Step 4:** WAN-Agents report on the availability and the cost of providing the specific network connection.
- **Step 5:** After receiving all requests, the Slice Broker combines them into a single SRA message and sends it to the Builder.

4.2.3. Results

We demonstrated geographically distributed resource discovery for a slice request, emphasizing on:

- **Scalability:** the Marketplace concept can scale to a larger number of providers and in the case of the Fed4Fire demo we included results that concern 6 real world testbeds and 6 "semi-artificial" resource providers and in the case of the SingleHost demo we experimented with 20 DC providers. We also demonstrated scalability in terms of the requested slice, i.e., we reported on resource requests with a large number of slices/resources;
- **Heterogeneity:** Resource Discovery can cope with a diverse range of server specifications, as the latter are defined by the different testbeds;
- **Cost efficiency:** Selection of the minimum cost slice among alternatives, since by having information regarding all possible slice instantiation annotated with cost allows the builder to apply any technique, either complete or heuristic in order to decide on the final slice instantiation;
- **Performance:** We reported on the total number of messages exchanged, the wall elapsed time since the builder sent the original request, until the final SRA message was received.

Single Host Experiments

In order to test and validate the Marketplace implementation, we ran a set of experiments on a single host, i.e., all Marketplace components were running on the same Linux machine. On this experimental setting, we generated 20 DC providers, with "semi-artificial" data regarding host characteristics and availability and 3 WAN Providers.

Our experimentation involved requests generated in a random manner, as that is reported in the corresponding section of D5.2. We have generated multiple requests organised in 5 classes. DC slice parts were distributed between Europe and America. The 5 classes of requests were characterized by a triplet (DC, NET, VDUS), i.e.,the number of DC and Net slice parts and the total number of VDUs allocated in the slice respectively. Thus, in the following the notation (2,1,4) stands for a slice request with 2 DC parts, 1 Net part and a total of 4 VDUs allocated in the slice.

For each of those requests that were executed, we recorded the wall time elapsed since the builder sent the original request till it received the SRA answer message from the Broker, as well as the number of alternative DC/NET slice parts received. For each SRA received message, using brute force search we computed the number of alternative slice instantiations generated by the SRA message, as in D5.2 as well as the minimum cost slice. However, when the slice request involves a larger number of slice parts, and since we assumed a fully connected network infrastructure, this number can increase significantly. Thus, we stopped this computation when there were fifty thousand (50000) alternative slice instantiations generated and reported on those. Results showing average values from 5 different requests in each class and are summarised in Table 3.

Table 3. Resource Discovery Marketplace on a Single Host.

Slice Request	Avg Wall Time (sec)	Alternative DC Parts	Alternative Net Parts	Slice Instantiations	Avg Cost
(2,1,4)	11.31	11	27	27	3.12
(3,2,6)	16.81	14.2	39.6	320.4	6.68
(4,5,8)	91.40	18.2	256.2	26924.4	17.68
(6,7,24)	605.867	55.4	1755	50000	78.08

(8,9,32)	763.10	73	2205	50000	103.90

In terms of scalability regarding the number of providers, the implementation managed to successfully accommodate 20 DC providers with no reported problem. It should be noted that this is not the maximum number of providers that the marketplace can host, but an indication demonstrating the capabilities of the implementation. The implementation is also able to handle slices, ranging from 2 DC- and 1 Net- parts to 8 DC- and 9 Net- parts and from 4 to 32 VDUs. Regarding the alternative slice instantiations that the builder can generate from the received SRA message, these range from a low value 27 to more than 50000 alternatives.

Regarding heterogeneity, the resource discovery process managed to accommodate providers with a varying number of resources as well as slice requests with a varying range of host specifications, since both were generated using random values. For instance, regarding slice requests, EPA attributes for a service that concern storage range from 2 to 30 GB, RAM from 4 to 16GB, and the number of hosts in each slice part from 1 to 10 hosts.

In terms of cost efficiency, given that all alternatives in the SRA message carry a cost, i.e., all alternative DC and Net slice parts, have an associated cost, the builder is free to select the lowest cost slice. In the current implementation, we select the latter by brute force techniques, which obviously cannot be used when the number of alternatives grows significantly. However, given that the complete information exists in the SRA message, any technique can be used to compute the desired slice instantiation.

Regarding performance, the wall time since the request was sent to the Broker till the SRA message was received, grows with the size of the slice, i.e., DC and Net slice parts. This is expected, since in the current implementation the Broker decomposes its slice request into its parts and follows a query-answer cycle for each. This poses a time penalty, especially in the case of Net-parts, since the number of these queries depends on the number of alternatives for each DC slice part. For instance, in the set (8, 9, 32) in Table 3, the average number of alternative Net parts (each received by a message to the broker from a WAN agent) is 2205, indicating 735 cycles of queries. Efficiently handling this problem can be an interesting research direction.

Fed4Fire Testbeds Results

Similar experiments were executed in the FED4FIRE testbeds Virtual Wall 1 & 2 and the Cloudlab Utah, as described in the corresponding section. This set of experiments was conducted in order to (a) verify that the application can indeed be executed in a distributed environment and (b) discover any problems that might occur when the DC agents query the real testbeds for up-to-date information on available resources.

As stated above there were a total of 12 DC agents, i.e., 6 accessing real testbed resource data and 6 “semi-artificial” data and we tested the discovery process in similar queries as in the single host experiment. Results are summarised in Table 4.

Table 4. Resource Discovery Marketplace on the FED4Fire testbeds.

Slice Request	Avg Wall Time (sec)	Alternative DC Parts	Alternative Net parts	Slice Instantiations	Avg Cost
(2,1,4)	24.81	8.6	55	55	5.04

(3,2,6)	53.93	13.6	125	828.2	10.49
(4,5,8)	70.57	15	162.6	3506.6	13.94
(6,7,24)	161.01	26.3	381	50000	49.92
(8,9,32)	289.00	36.3	518	50000	63.41

Regarding point (a), since the RabbitMQ message passing platform was used as the marketplace implementation, no problems were raised when running the software on a distributed set of machines. Regarding point (b), we can confirm that the python translator component to the Fed4Fire testbeds operated without any problems and that the execution behaviour of the marketplace was similar to that of the single host experiments, i.e., the execution time increases as the number of alternative slice parts increases since there are more query cycles. Overall, we can safely conclude that the implemented functionality is not affected by distributing components in geographically remote hosts.

4.3. Experiments with Large-scale Lightweight Service Slices (ELSA)

This demonstration is composed by one slice provider hosted by UFPA, the Tenant to make the requests hosted by UFRN, and the resource providers hosted at UCL, as presented in Figure 22.

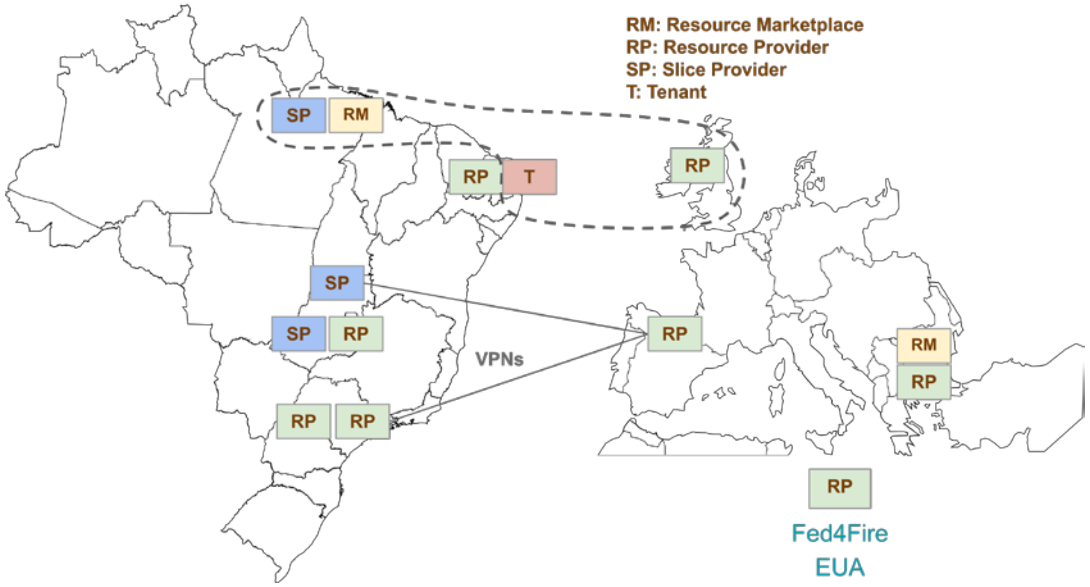

Figure 22. Instantiation of ELSA on the experimental infrastructure.

4.3.1. Objectives

The idea of this demo is to show the deployment of end-to-end Slices that will be utilised by a Tenant in order to host services consisting of a very large number of lightweight elements (i.e., Virtual Network Functions (VNFs) and vLinks) deployed at the Edge of the infrastructure. We will demonstrate how the Tenant is able to reuse their existing software components by attaching them to the allocated end-to-end slice in a completely transparent way. The description of the desired end-to-end Slice is provided as YAML input by a Slice Activator component in the Tenant domain to a software component that

implements some of the functionalities of both the NECOS Slice Builder and the Slice Resource Orchestrator.

The descriptor will include information (i.e., type of VIM, size, etc.) about the slice parts to be created, the links between the slice parts and the monitoring parameters (KPIs) to be collected from each of them. In this demonstration, the marketplace-related workflows will not be considered, and the YAML descriptor provided to the Slice Builder will contain a predefined set of entry-points of the Slice Controllers to be contacted in the Resource Providers.

The end-to-end Slice specified via the above descriptor will automatically be built on a Slice Provider hosted at UFPA. At the UCL premises, 12 of the available 14 interconnected physical servers will be hosting different instances of the DC Slice Controller, in order to emulate different NECOS Resource Providers. The created DC slice parts will be based on the on-demand instantiation of the Very Lightweight Network & Service Platform (VLSP) VIM, which will support the creation of simple lightweight service topologies across the different slice parts of an end-to-end Slice (mainly simple video streaming services).

In order to orchestrate the deployment of the above mentioned large-scale services on an end-to-end Slice, an instance of the open source (5GEx) ESCAPE[4] Orchestrator will be configured to use the resources of that end-to-end Slice as substrate for embedding the required service elements (i.e., VNFs and virtual Links). This will happen transparently as the Tenant will attach their existing service orchestrator (ESCAPE) to the newly created slices.

Finally, as soon as the end-to-end Slices will be up and running, the NECOS IMA will start collecting relevant KPIs related to it. The implementation of the DC Slice Controller deployed at the UCL Resource Provider is based on the instantiation of bare-metal slices. As such, the above collected measurements related to the KPI-9 (Physical Resource Utilization) will highlight how the execution of the large-scale services deployed on one of the slices will not affect the physical resources of the other slices in the same Resource Provider.

4.3.2. Workflow

The Experiment Controller component implementing some of the functionalities expected for the NECOS Builder and SRO architectural components will ensure large-scale system operations while the slices are created, operated and monitored.

[4] https://github.com/5GExchange/escape

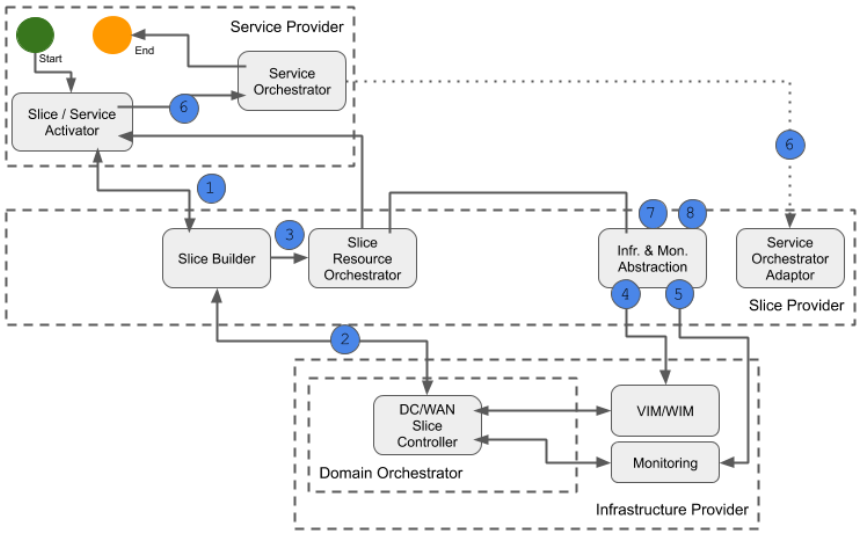

Figure 23. ELSA workflow.

More specifically the demo shows the following different steps (see Figure 23):

- **Step 1:** The YAML end-to-end Slice Specification (according to the NECOS information model) is provided by the Slice / Service Activator component in the Tenant domain to the software module implementing the required Slice Builder functions, in order to start the instantiation of a new end-to-end slice;
- **Step 2:** The above Slice Builder module interacts with different DC Slice Controllers instances that are already deployed in the testbed (via the Slice Instantiation Interface), in order to request the creation of different DC slice parts, each based on an on-demand instance of the VLSP VIM;
- **Step 3:** The information about the different allocated DC slice parts is returned back from the Slice Builder to the component implementing the functions of the Slice Resource Orchestrator. The latter will take care of interconnecting the allocated slice parts via creating an emulated tunnelling that will be based on the interaction of custom instantiated VLSP edge routers;
- **Step 4:** Resource Adapters attached to the allocated VIM endpoints are dynamically created and the handlers to the adapters are provided back to the Slice Resource Orchestrator;
- **Step 5:** Monitoring Adapters are requested to the IMA according to the allocated type of VIM and Monitoring Subsystem that were deployed in each slice part in order to gather monitoring data in a uniform way;
- **Step 6:** An instance of the open source ESCAPE service orchestrator is attached to the newly deployed end-to-end Slice and the Service Activator will receive a handle to the northbound interface of that Service Orchestrator instance. This will be utilised to request the 'embedding' of a large service request (in terms of number of involved service virtual elements) on the previously created end-to-end slice, which will act as the resource substrate. The orchestrator will also act as Service Orchestrator Adapter as service requests will be translated in a format supported by the Slice Provider;
- **Step 7:** IMA collects and aggregates data (via an augmented Data Aggregator) from the different slice parts in order to generate KPIs related to the end-to-end Slice;

- **Step 8:** measurements related to the KPI-9 collected from IMA in the previous step that are related to (at least) two end-to-end Slice in the UCL Resource Provider will be considered. The bare-metal slices built by the DC Slice Controllers will guarantee resources isolation at the physical layer. As such, deploying a large-scale service instance on one of the slices, will not affect the utilization of the physical resources of the other ones.

4.3.3. Results

We demonstrated how NECOS is able to provide a Tenant with the abstract concept of end-to-end Slice, i.e., a fully manageable bundle of resources that can be requested, allocated and then transparently used by a Tenant. The latter was eventually able to attach a newly created slice to their existing systems (such as Service Orchestrators, etc.) in order to perform the deployment of specific services on the end-to-end Slice resource substrate (this is related to the KPI Slice Provisioning). The scalable embedding of lightweight virtual service functions could be performed by a Tenant on the end-to-end Slices in the above described emulation environment, where the scale of the instantiated service elements ranges from hundreds to ~2K VLSP virtual router elements (this is related to the KPI Management). The time related to the service instantiation in this particular demonstration (i.e., deployment of lightweight VLSP services embedded via the ESCAPE service orchestrator) is reported in Figure 24.

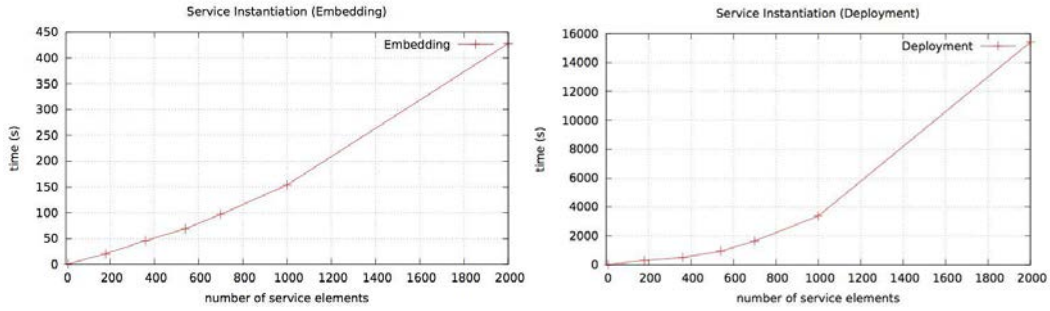

Figure 24. Service instantiation (embedding and deployment).

The NECOS architecture and system allowed using heterogeneous types of resources coming from different segments of the infrastructure (e.g., from resources constrained edge domains) that were bundled as a single object and used for the deployment (embedding) of the desired service components. End-to-end Slices could be provisioned via interconnecting slice parts coming from multiple Resource Providers (edge domains), which were emulated by partitioning the Data Center testbed available at UCL. Each slice part hosted a separate instance of the lightweight VLSP VIM (this is related to the KPIs VIM-independence and Slice Provisioning). Figure 26 highlights the time required for the allocation of end-to-end Slices in this particular demonstration. The graph shows the number of physical resources involved in the creation of different Slices. Please note that sizes on the x-axis from 1 to 3 refer to an increasing number of slice parts (i.e., 1 to 3). Values greater than 3 are all based on 3 involved slice parts. The parallelism in the required operations allowed the creation of the different parts and abstractions in a way that was not related to the number of involved physical servers (however this is related to the particular implementation of the DC Slice controller and the VLSP VIM). On the other hand, due to the specific way the interconnection between the slice parts was performed, it required the propagation of the information about the edge points of a slice parts to the adjacent ones. As this had to be performed sequentially, the required time for its execution grew linearly with the number of slice parts as reported in Figure 25.

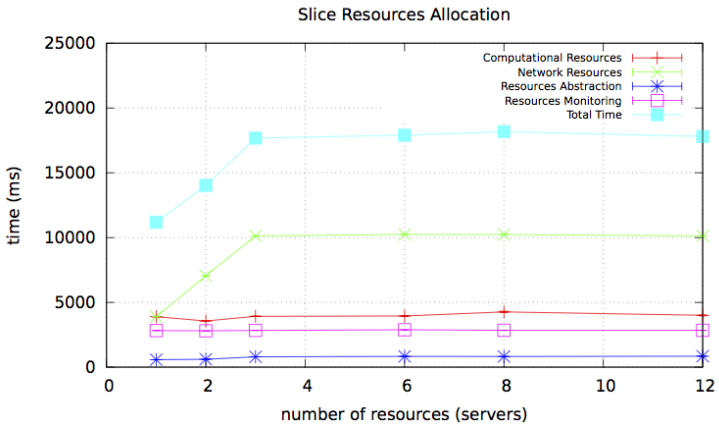

Figure 25. Creation of end-to-end Slices.

This demonstration also showed the process of gathering monitoring data for (i) the service elements, (ii) the hosts, (iii) the slice parts, and (iv) the whole slice via IMA (this is related to the KPI Monitoring). The measurements collected by the IMA Aggregators from at least two different end-to-end slices in the same Resource Provider, showed that the physical resources utilisation (i.e., the average percentage of CPU utilisation) measured on the physical resources of an end-to-end slice running a service grows according to the load, whereas the same type of KPI in the other slices (where no services are running at the same time) does not change. This is shown in Figure 26 and demonstrates the better level of isolation introduced by our implemented slicing approach, especially when the slices are created directly on the bare-metal. The average CPU utilization (percentage) of the slice running the service (slice1 in green) was affected by the load introduced by the service execution, whereas the other slice in the same Resource Provider (slice2 in yellow) was not impacted by that as it was using a separate set of physical resources.

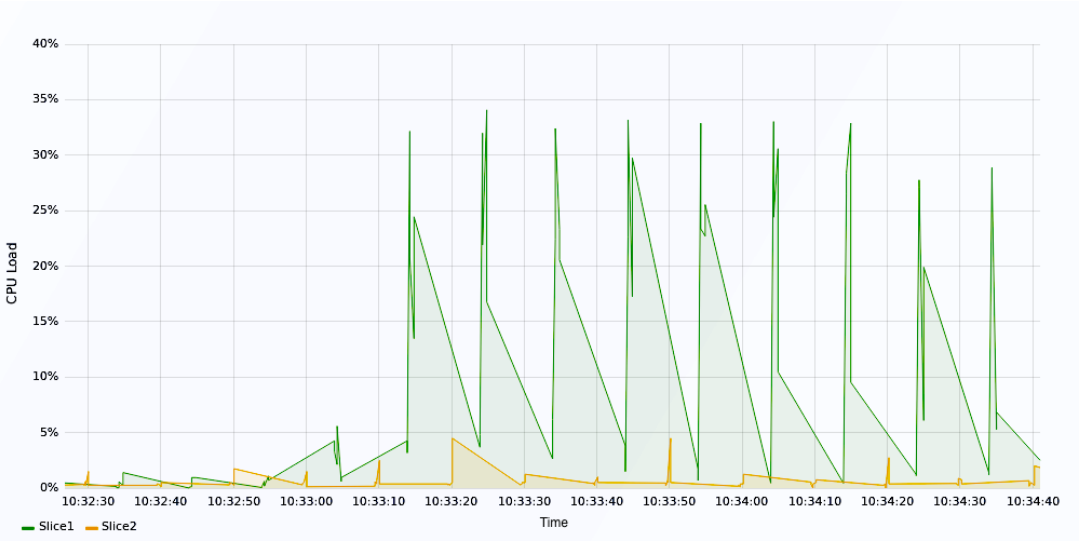

Figure 26. CPU Utilization on two different bare-metal Slices.

4.4. Machine-learning based orchestration of slices (MLO)

In this demonstration, the Tenant hosted at CPqD requests multiple slice allocations to the Slice Provider hosted by UFU that uses the Resources Providers hosted in the same institution as presented in Figure 27.

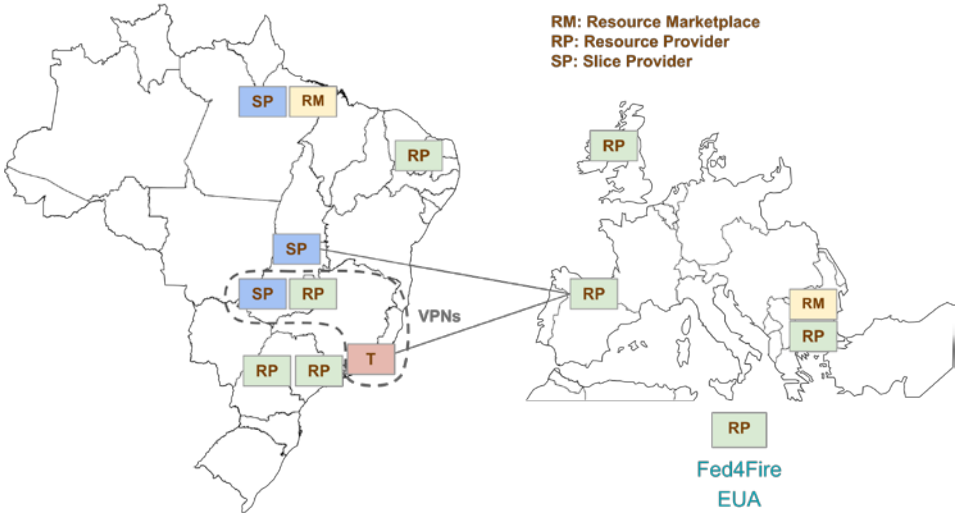

Figure 27. Instantiation of MLO on the experimental infrastructure.

4.4.1. Objectives

The objective of this demo is to show how machine learning algorithms can be useful for the orchestration of slices. Two modules of NECOS Architecture, IMA and SRO, were specifically extended to enable intelligent monitoring and intelligent elasticity orchestration, respectively.

Given the multitude of elements composing slices' infrastructure and the presence of multiple domains in which slice parts are spread to create an end-to-end slice, it is natural to expect considerable overhead to monitor and to move such data from the IMA towards the SRO. Considering such characteristics that can pose a scalability problem, this demo provides intelligence to IMA, enabling it to perform automatic selection of features to be monitored. Such selection is performed on a per-slice basis, requiring that the SRO provides a target KPI to IMA, so it can select a set of features that better describes the behavior of such KPI. We refer to the entire set of infrastructure metrics as full feature set, i.e., the set of metrics regarding the entire infrastructure composing the slice. We refer to the result of the selection mechanism based on machine learning as selected feature set, i.e., a set of K essential metrics selected among all infrastructure metrics composing the slices.

The IMA, in this demo, performs the collection of all infrastructure metrics (full feature set) in longer intervals (less frequently) when compared to the frequency in which the selected feature set is collected. It is expected that along the lifetime of a given slice, the target KPI can evolve and, as a consequence, the selected feature set can probably present a different composition. This effect is justified, for example, by sazonalities related to the usage of the slice. By collecting the full feature set in less frequent intervals, the IMA is capable of automatically updating the selected feature set.

IMA, after the selection of the features, provides the monitoring data to the SRO, according to the time interval specified by IMA. As a result, the intelligent version of the IMA presented in this demo reduces the volume of data being pushed towards the SRO (the selected feature set), not only improving monitoring scalability, but also removing noisy data from the monitoring information. Such noise

removal has the potential to improve accuracy in the operations performed by the SRO during slices orchestration.

The intelligent SRO implemented in this demo consumes the selected feature set provided by the intelligent IMA with the goal of supporting elasticity decisions. To do it, the intelligent SRO performs four steps: 1) KPI Estimation; 2) SLA Prediction; 3) Slice Resources Optimization; and 4) Enforcement of Slice Modifications.

The SLA for a given slice, in this demo, is associated to a KPI. Such KPI can be related to the slice infrastructure, considering metrics related to, for example, CPU, memory, network traffic, and others. The most interesting aspect of this demo is that, differently from the other SRO implementation, such KPI does not have to be generic and can also be directly related to a Service KPI. As such in the demo, we actually used Read and Write Response Times of a Cassandra-DHT running as the Service deployed in a given slice. The SRO estimates the current state of such Service KPIs as a way of verifying whether the slice SLA is being violated or not. An example SLA that might be adopted during the demo is to target Read Response Times below 50 milliseconds for verifying the Slice SLA conformance. Besides continuous values, the same solution presented in this demo can be applied for services with discrete classes, like videos in high or low resolution.

The first step taken by the SRO, named as KPI Estimation, consumes the monitoring data provided by the intelligent IMA, feeding it into a supervised machine learning model (regressor/classifier), which is trained to estimate the current state of the target Service KPI. Basically, it is done in order to associate the slice's infrastructure measurements fluctuations with the chosen Service KPI.

By having the history of estimations, the intelligent SRO is capable of performing the second step, named SLA Prediction, which foresees what the state of the SLA will be at a given point in the future. This enables the SRO to proactively tune the slice, preparing it for the condition seen in the future.

The third step, perhaps the most complex out of the four steps listed, has the duty of designing the new slice infrastructure arrangement, capable of handling the condition foreseen by the second step taken by our intelligent SRO. This third step, by itself, opens several research possibilities, including root cause analysis, resource optimization, and others of even higher complexity.

In this demo, we consider the existence of a set of slice flavours. A tenant, when requesting a slice to NECOS – similarly to what is done nowadays when requesting a virtual machine at Amazon – informs the flavour which will be applied to its own slice and, additionally, a set of flavours allowed to accommodate possible SLA fluctuations. The demo detects in step number two whether the SLA is going to be violated or if it is going to be under conformance with a lighter KPI condition, and step number three optimizes the slice flavour, among a set of flavours, which is capable of keeping the SLA under conformance and, at the same time, with moderate resources consumption, i.e., the service will not face SLA violation due to lack of resources, but it will not be running with a set of resources that is not really necessary, given the current service condition estimated by the SRO, as well.

The fourth step does not restrict itself to adjustments in terms of slice infrastructure, already specified in NECOS Architecture, it also encompasses intelligent adaptations. Besides the communication among SRO and DC/WAN Slice Controllers, the intelligent SRO: 1) requests to IMA the update in the composition of the feature set being monitored, ideal to the new slice flavour; 2) updates the trained model being used in step number one, which estimates the Service KPI; and 3) updates the predictive

models used in step number two. In short, step number four performs an overall, infrastructure and intelligence, adjustment in this demo.

4.4.2. Workflow

The demo departs from an established end-to-end slice with a certain service running inside it. As a first step in the demo, the SRO provides IMA with the KPI that it wants to be estimated. As mentioned before, the demo showcases a Cassandra-DHT type of service. It also has a second application, which is a Video-on-demand Service based on Dash Price Chart (DASH).

As seen in Figure 28, upon receiving the KPI from SRO (step 1), the intelligent IMA recovers (step 2) from the infrastructure providers the full feature set (i.e., the measurements related to the infrastructure) already monitored by the VIM/WIM. Figure 28 suggests the usage of local databases at infrastructure providers to store monitoring data, but depending on the monitoring technology within infrastructure providers, IMA might assume the responsibility of storing it.

After step 3, with all monitoring data at hand (full feature set), the IMA performs feature selection using machine learning. It selects the features to monitor using the Service KPI as target metric, i.e., it defines the composition of the selected feature set with K essential metrics that better represent the given Service KPI and executes some tasks in parallel as a consequence of step 4. The set of K features and the respective monitoring history is returned to SRO (step 5), so it can train its KPI Estimation module, at the same time that IMA adjusts the monitoring tools deployed within infrastructure providers. As mentioned before, the intelligent IMA deploys two monitoring instances, one responsible for collecting the full feature set (step 6) at longer time intervals (mainly used to support feature selection) and a second instance responsible for delivering live monitoring data corresponding to the selected feature set (step 7). The definition of both collecting intervals can be explicitly specified by the tenant or by analysing learning curves.

Figure 28 also suggests a direct communication from VIM/WIM Monitoring module towards the SRO to deliver live monitoring data (step 8). This demo implementation uses a Pub/Sub system, based on Apache Kafka, which removes several push cycles from the overall monitoring system. Basically, we assume that required adaptations in the monitored data is performed locally at the infrastructure providers, for example, by deploying the functionalities of IMA in a distributed manner among the slice parts composing slices. Such design contributes to the deployment of real time orchestration of slices.

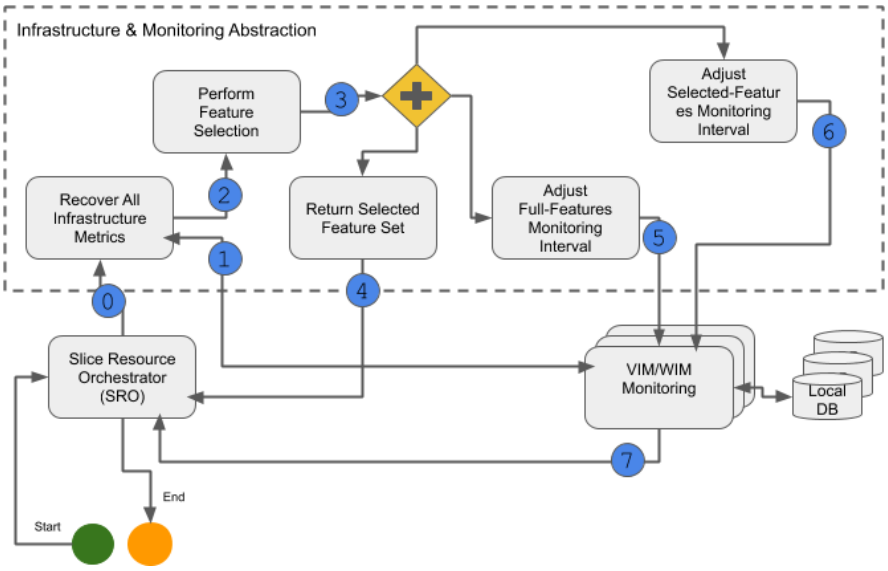

Figure 28. Intelligent IMA workflow for feature selection.

Figure 29 updates the elasticity workflow presented in Deliverable D5.1 of NECOS, presenting the machine learning extensions as green steps. This figure depicts the four steps SRO performs in this demo to support elasticity. This figure also accommodates the proposed communication directly from the VIM/WIM monitoring systems towards the KPI Estimation module of the SRO, as mentioned above.

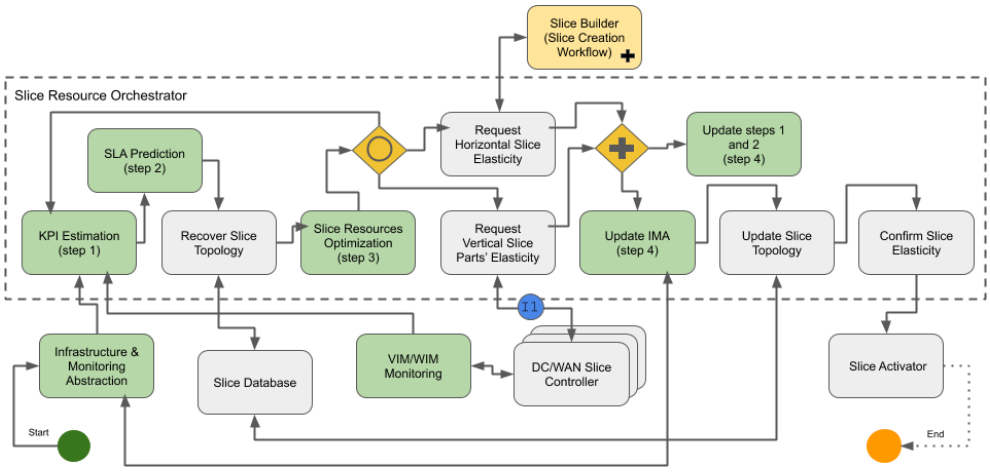

Figure 29. Intelligent SRO workflow for elasticity.

It is important to highlight the possible outcomes of step 3, the decision can indicate the need for vertical and/or horizontal elasticity, both including upgrade and/or downgrade of infrastructure resources. But, another important outcome of step 3, is to keep the slice in its current form, i.e., returning the loop to KPI Estimation (step 1) of the intelligent SRO. This latter case represents the scenario in which the optimization of slice flavours indicates that current slice arrangement is the best among the available options, for example.

4.4.3. Results

The key aspect of this demo related to intelligent monitoring and orchestration to provide elasticity. In this context, the demo provided an environment that is capable of evaluating total elasticity execution time, from the moment when elasticity was defined as necessary, up to the moment when it is completely executed. The demo also allowed to measure the effectiveness of honouring the SLA

EUB-01-2017

agreed between the Tenant and NECOS. Finally, given the machine learning nature of the demo, it was also possible to evaluate the accuracy of the proposed solution. For example, measuring how often the proposed mechanism mistakenly changes the infrastructure of slices. Figure 30 is useful to evaluate how close to the Observed Response Time is the Estimated Response Time of our neural network. It is also important to mention that the estimated values are a forecast of thirty seconds in the future. The dynamicity of the response times is due to a load generator that controls the instantiation of end users consuming the service offered by the DHT system. As seen from Figure 30, the set of estimated response times closely approximates the set of observed response times, bringing accuracy to the elasticity forecasts.

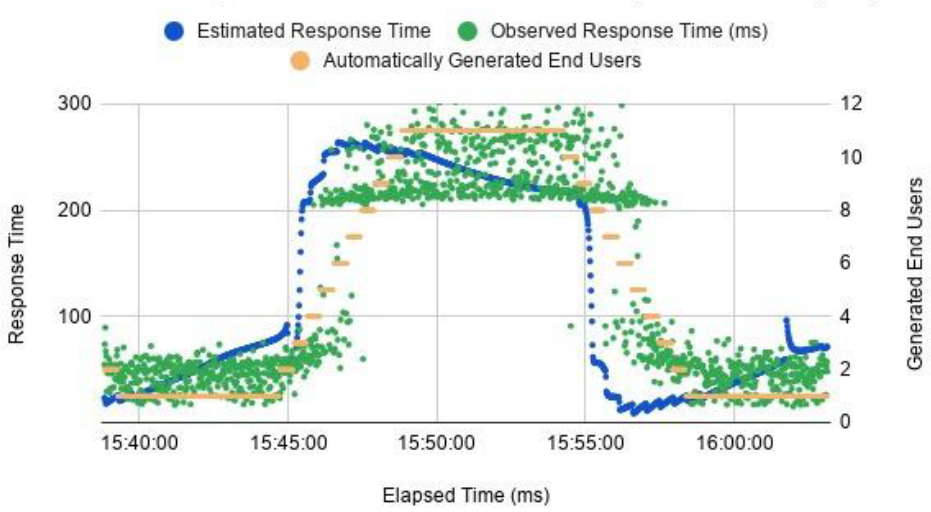

Figure 30. Estimated Response Time versus Observed Response Time as a function of the number of end users consuming the DHT service.

Figure 31 illustrates a vertical elasticity upgrade triggered by our machine-learning-based orchestrator. The SLA is defined as response times below 180 ms, so whenever the orchestrator forecasts that such an SLA will be violated in the next 30 seconds, the elasticity process is triggered to proactively avoid the violations. As can be seen in Figure 31, a few seconds after 18:18:00 the orchestrator foresees an SLA violation (blue star) and triggers elasticity. Since the elasticity is proactively triggered and the slice is modified to the slice flavour that fixes the foreseen violations, the process takes just a few seconds to be concluded. It is also possible given the fact it is a vertical elasticity and no new slice parts are included in the end-to-end slice and no new VIM/WIM needs to be instantiated, a case in which the process would last longer. After the vertical elasticity, it is possible to check that the observed response times are maintained under the specified SLA, even with the number of end users growing, i.e., the pressure over the service is increasing.

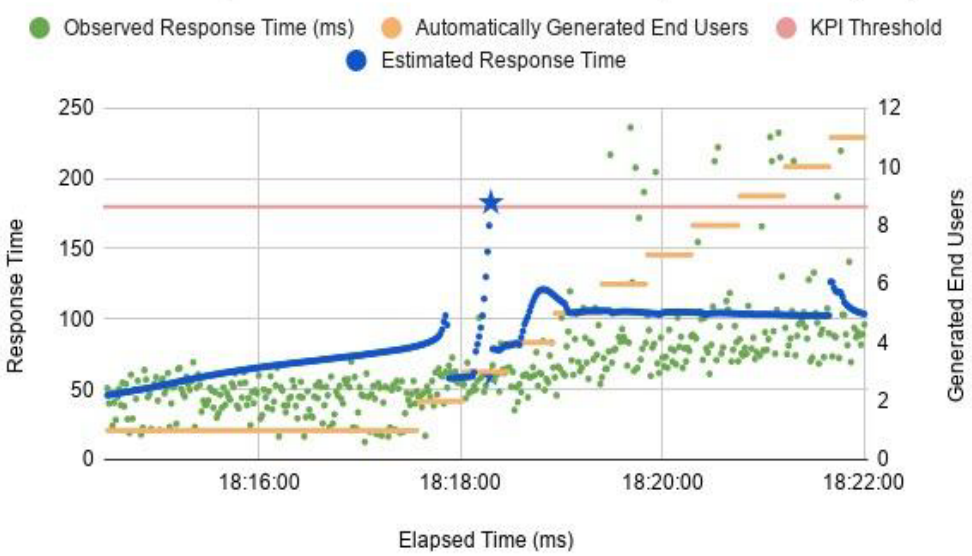

Figure 31. Estimated Response Time versus Observed Response Time as a function of the number of end users consuming the DHT service.

Other aspects closely related to monitoring that this demo allows to evaluate include the time required to perform feature selection, per slice and per Service KPI. It also opens the opportunity to investigate the impact in terms of accuracy of feature selection and SRO estimations, versus the time interval in which the entire (full) infrastructure metrics are monitored. Figure 32 details such trade-offs, illustrating how the frequency interval for the collection of the entire set of infrastructure metrics can influence in the quality of the estimations by the orchestrator, and also the number K of metrics selected by the IMA module. From Figure 32 it is possible to check that at around 15 metrics (for the scenario being considered in the DHT service) the accuracy of the machine learning model stabilizes. As a consequence of this evaluation, in the demo we consider K=15. It is also possible to check that the longer the collection interval, the worse is the accuracy of the machine learning model and this is easily explained by the fact that it "reduces the resolution of the picture" the monitoring system takes from the infrastructure composing the slice. Such effect can be controlled by the fact that online learning can be used to update the machine learning model along the lifetime of the slice, using the live monitoring data to not only forecast thirty seconds in the future, but also improving the trained model.

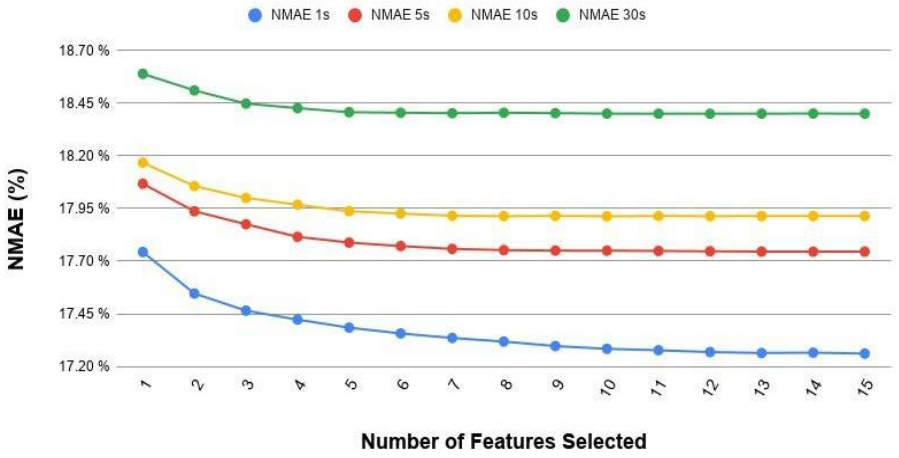

Figure 32. Accuracy of the machine-learning model (measured as NMAE) as a function of the frequency interval for the collection of the full feature set and the number K of features selected.

As a consequence of the feature selection performed by the intelligent IMA module, this demo is also capable of showing the improvement regarding the movement of monitoring data from the Infrastructure (i.e., from the VIM/WIM Monitoring via IMA) towards the SRO, i.e., it is possible to evaluate the gains in terms of overhead reduction that intelligent feature selection assigns to NECOS. Figure 33 presents the volume of data transferred from the IMA towards the SRO during the period of one hour, for both the full feature set approach and the selected feature set approach. As seen in Figure 33, the volume of data transferred for K=15 increases much slower than the amount of data transferred for the full feature set, such behaviour assigns better scalability for NECOS since it monitors in a per slice basis.

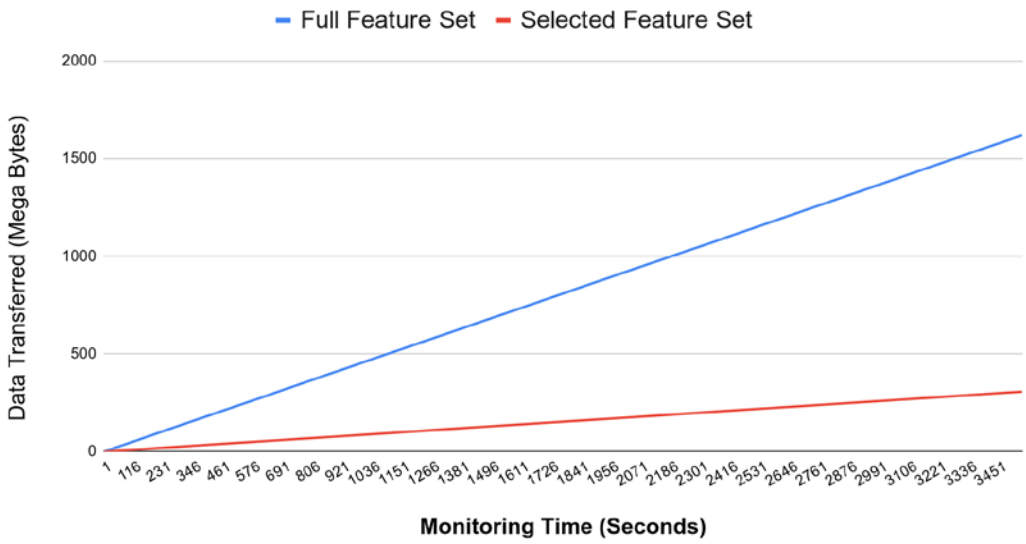

Figure 33. Comparison of the volume of data transferred by the monitoring module (IMA) towards the orchestrator (SRO) while monitoring the full feature set once a second versus the selected feature set once a second for K=15 in the slice with the DHT Service.

4.5. Wireless Slicing Services (WISE)

In this demonstration, multiple slice allocations are requested via one Slice Provider hosted by UFPA and the Resources Providers participating at the UFRN, as depicted in Figure 34.

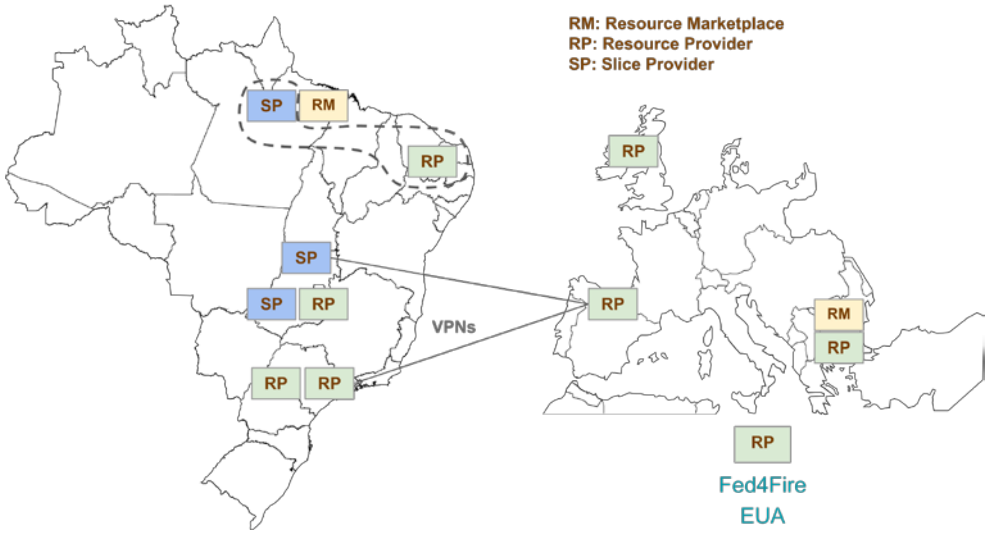

Figure 34. Instantiation of WISE on the experimental infrastructure.

1.1.1 Objectives

The Wireless Slicing sErvices (WISE) demo shows the NECOS LSDC capabilities in expanding the cloud-network slinging concept towards wireless network domains. In general terms, the WISE [Maxweel, 2019] solution outperforms the state-of-the-art in WiFi-shared access by deploying an end-to-end slice-defined approach, enabling WLAN networks tailored to serve the demands of specific scenarios and applications needs. Moreover, WISE allows carriers to be capable of both managing and controlling home-premised off-the-shelf WiFi routers at the runtime by harnessing a fully softwarized approach. The WISE solution addresses the gaps of existing WiFi sharing tools, such as the widely used FON[5] global WiFi network, which only allows traffic isolation and differentiated services at the CPE premises, as well as only permits system reconfigurations on-site (fully customer-centric).

Contrary to the FON WiFi sharing service fully deployed within WiFi router premises, WISE turns off-the-shelf WiFi routers into a simple CPE, focused on provisioning WiFi access to mobile devices through different virtual networks. The WISE WiFi sharing control services run out of the CPE, namely in virtual CPE applications running at edge node premises. Thus, all incoming traffic that belongs to different virtual WiFi networks must be subjected to respective virtual network functions before going forward. In light to achieve this, the legacy WISE solution relies on external technologies to provide all the necessary resources. The idea behind harnessing the NECOS hub of services (i.e., the NECOSization of the WISE approach), stands to rely upon the LSDC approach for orchestrating and managing all necessary resources so that the WISE solution can run as a service. Thus, NECOS foresees to provide WISE-enabled WiFi sharing systems with capabilities for end-to-end service-oriented networking services, including full isolation and auto-scaling, as well as customization and control at runtime. The goal of the WISE demo is to assess the LSDC approach performance in a lab-premised testbed deployed

[5] http://fon.com/

in the UFRN's REGINA-Lab premises, which hosts all NECOS components in the core DC. Figure 35 depicts the testbed configuration of the WISE demo.

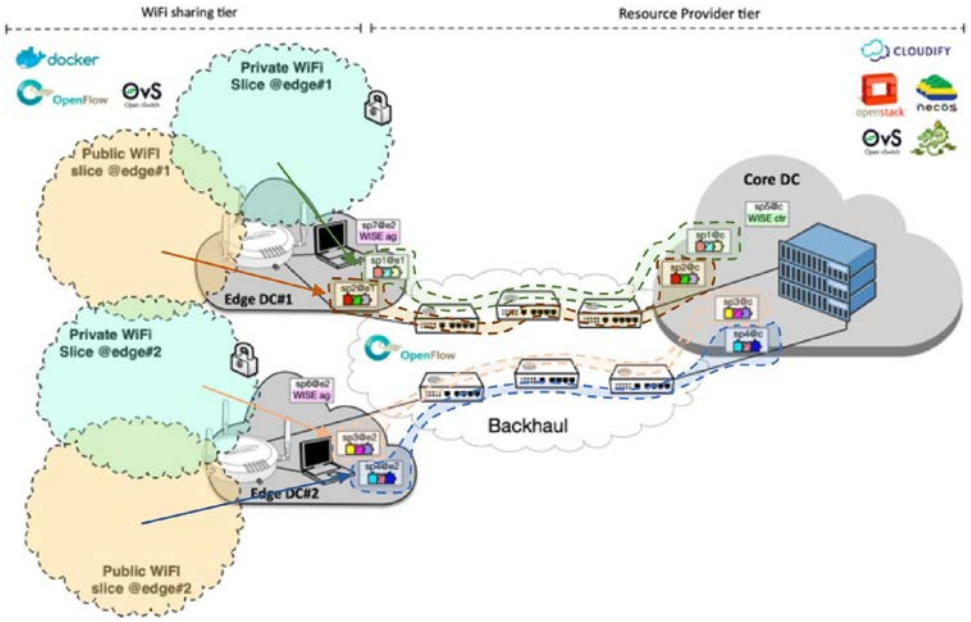

Figure 35. Testbed configuration of the WISE demo.

The WISE demo considers a scenario in which carriers exploit public WiFi coverage coming from residential hotspots to support 5G's ultra-dense networking. For instance, distinct carriers may want to purchase WISE-enabled WiFi sharing systems to complement their cellular networks with broadband wireless access featuring end-to-end isolation, customization and independent service provisioning. Such a WiFi-assisted network densification proceeds in two stages: i) by harnessing the NECOS LSDC approach to orchestrate the necessary cloud-network slice resources all the way from the core DC to the WiFi CPE. More specifically, the NECOS LSDC approach takes slicing template specifications for building (and decommissioning afterwards) a pair of containers at both edge DCs and the core DC in a per virtual WiFi granularity, each pair hosting same chaining WISE VNFs; one container in the edge DC to host the WISE Agent application; and, one container in the core DC to run the WISE Controller. In the end, the NECOS LSDC approach builds network slices connecting all the cloud slice parts and the WiFi virtual networks.

The WISE Controller relies on particular VIM and WIM that NECOS provides to the carrier in order to enable the dynamical control and management of all the computing, storage, and network resources within the entire cloud-network slicing topology; ii) by applying end-to-end cloud network slice definitions on top of the WISE-enabled Wi-Fi sharing technology, with the aim of offering multi tenancy and multi service support for a wide range of services.

On the basis of the Wi-Fi slicing concept, typical WiFi WLAN-sharing services implement virtualization to accommodate two virtual networks within the common Wi-Fi spectrum for shared connectivity. Similarly, NECOS maps the description of the desired service capabilities (provided as YAML file) into two end-to-end cloud-network slices: (i) a “public” one, devoted to community Internet access, and (ii) a “private” one, for particular devices attached to the Wi-Fi owner’s network. Each demanded cloud network slice comprises a set of dc and network slice parts as follows:

- A public and a private WiFi slice parts, consisting of virtual access points (vAPs) running on top of an OpenWRT-empowered off-the-shelf CPE;
- Eight (8) Edge dc slice parts consisting of different NFV service chaining instances running locally at mini dc equipment premises (a laptop), in the form of vCPE applications;
- Eight (8) WAN network slice parts consisting of the required virtual networking infrastructure (e.g., nodes, links, interfaces, etc.) for proper edge-to-core cloud slicing connectivity;
- Eight (8) core dc slice parts consisting of the required computing and storage resources to accommodate additional services such as a general-purpose software application.

Two off-the-shelf Wi-Fi router TP-LINK TL-WR1043ND v3 (CPU of 720 MHz, and RAM of 64 MB), running the OpenWRT v18.069 and the WAN Slice Controller implementation, is adopted to provision the Wi-Fi-sharing technology. A laptop DELL VOSTRO 5480 (Core I7-5500U, RAM 8GB, HDD 500GB) implements the edge DC, whilst a two clustering rack servers PowerEdge R7425 (2AMD 32-core EPYC processors, 64GB DIMM DDR4 RAM, 4 HDD 2TB, and 4 Gb Ethernet network cards) compose the core cloud. In short, the WAN Slice Controller creates vAPs that can run on a physical router to provide service-oriented WLANs for specific applications. An SDN infrastructure featuring 6 OpenFlow-enabled Mikrotik 951G-2HnD (CPU of 600 MHz, and RAM of 128 MB) meshed switch nodes provide wired connectivity between the edge and core DCs.

4.5.1. Workflow

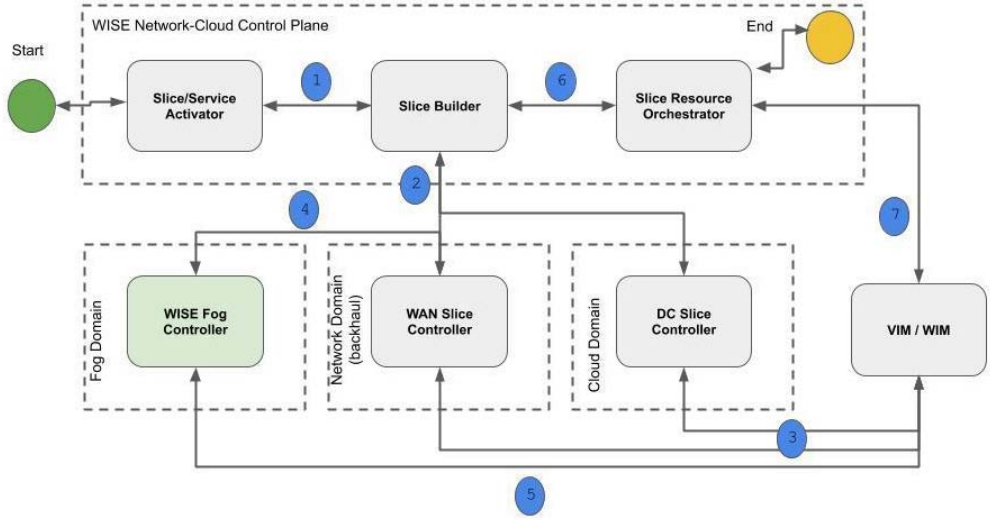

Figure 36. WISE workflow.

As shown in Figure 36, the demo has seven steps which are detailed below.

- **Step 1:** Slice Activator receive a YAML file with slice and service description (based on NECOS information model) and requests an end-to-end cloud-network slice deployment to slice builder;
- **Step 2:** Slice Builder interacts with already deployed DC/WAN controllers in order to create different slice parts (including WiFi, network and edge/core DC slice parts);
- **Step 3:** DC/WAN deploys VIM/WIM on demand in the edge DC, network and core DC domains;
- **Step 4:** WAN Controller communicates with WISE edge DC controller to deploy two new slices, one for the public and another for private purposes, upon the network domain;
- **Step 5:** WAN Controller associates the deployed WIM with the public and private slice parts;

EUB-01-2017

- **Step 6:** Slice Builder informs SRO about Slice creation process;
- **Step 7:** SRO deploy slice services in each VIM/WIM based on slice description from Service Activator. After that, SRO can manage cloud, network and WiFi resources in the edge DC, wan and core DC.

4.5.2. Results

The mains KPIs addressed in this demo are both the average slice provisioning time (KPI 4) in seconds (during the creation and decommission workflows) and the monitoring-data availability (KPI 7), in terms of the number of control-plane signalling exchanges, to estimate the networking cost impact.

In order to show the results concerning the KPIs 4, we collect the times that the LSDC approach devotes to provisioning all the cloud-network slices during the course of the testbed experiments. Figure 37 sketches the total provisioning times to create, config and decommission all cloud network slices.

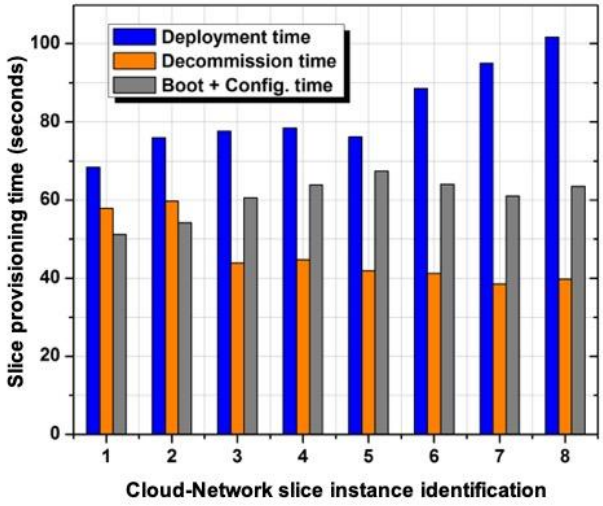

Figure 37. Average provisioning time to build and decommission 8 cloud-network slices in the testbed experiments.

The numerical analysis of the results obtained reveals that the NECOS LSDC approach spends on average 57.20 seconds devoted to VIM-centric operations (cloud network slice part), 25.26 seconds to carry out WIM-centric operations (network slice part), and 60.4 seconds to boot up the service delivery. Thus, a cloud-network slice instance creation raises a total provisioning time of 2.38 minutes on average (a peak time of 2.73 minutes) in the lab-premised testbed experiments. In regards of the decommission time, the NECOS LSDC approach takes slightly less than one minute (50.63 seconds) on average, raising a peak of 1.05 minutes in the worst case.

In order to address the KPI 7 analysis, we collect all the signaling load (in bits) entailing cloud-network slice creation and decommissioning workflows during the course of the testbed experiments. This analysis play a key role by the importance in estimating the cost that the NECOS LSDC approach impacts in the whole system, in response to carrying out the cloud-network slice creation and decommissioning workflows. The Figure 39 depicts the KPI 7 results.

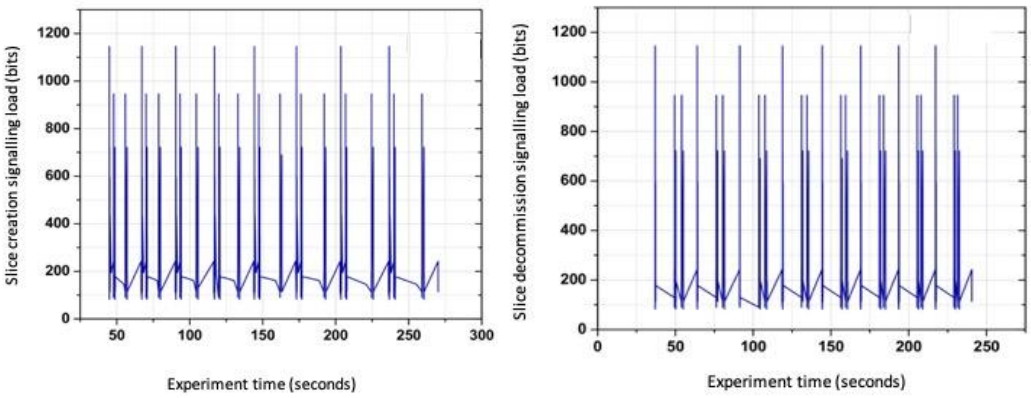

Figure 38. Total signalling load impact to create and decommission cloud-network slice instances during the course of the testbed experiments.

On observing the results of Figure 38, the testbed reveals a slight difference in the total resource enforcement signaling load impacting the experiments. The numerical analysis exhibits that the NECOS LSDC approach requires a signaling load of 37,363 bits on average to create all end-to-end cloud-network slice instances all the way from the core DC to the WiFi CPE. During the creation of the first cloud-network slice, the justification for the best resource enforcement signaling latency performance (35,640 bits) comes from the higher amount of available resources for the NECOS components to operate. Hence, the total latency coming from the resource enforcement signaling load needed to create the first cloud-network slice is of 22.24 seconds. From the first cloud-network slice on, the total resource enforcement signaling load latency exponentially increases, achieving a maximum of 33.85 seconds to create the cloud-network slice 8, since resources are progressively exhausted.

For what concerns the cloud-network slice decommissioning, the total signaling load behaves in the opposite way, since resources return to available condition progressively with the accumulative resource releasing. Thus, cloud-network slice 8 decommissioning performs better than the rest of the cloud-network slices, by exchanging a total of 17,299 bits in the whole system. On the other hand, the cloud-network slice 1 shows worst decommissioning operation performance through exchanging a total of 18,691 bits (7.45% of additional signaling load). On average, the LSDC approach spends a 25.47 seconds on average for decommissioning a cloud-network slice instance in our lab-premised testbed, from 27.03 seconds in the first cloud-network slice instance to 23.51 seconds in the very last one.

It is worth highlighting that the cloud-network slice booting time does not include WISE service deployment. Aside from that, the WISE demo adopts the same VIM and WIM technologies already running from the beginning of the tests for all of the cloud-network slice instances. Finally, WISE achieves a significant reduction in provisioning times when compared with the MUSTS demo for instance for the reason of not needing to deploy VIM and WIM instances as on-demand services.

As a future remark, we will carry out new experiments considering WISE service deployment, in addition to cloud-network slicing creation and decommission workflows, in different scenarios. Thus, we expect to get new insights by observing the incidence of more realistic provisioning times and costs.

4.6. Acceptance Verification

As pointed out in subsection 2.3, the NECOS acceptance requires that the tests for all prioritized features are complete, i.e., the solution process is receiving entries correctly, processing as specified and returning correct outputs. These results are summarized in Table 5, for each feature a specific test was conducted in one or more demonstrations and the results using the KPIs were listed as verification artefacts.

Table 5. Acceptance verification**.**

Features	KPI	Test	Verification Artefacts
Slice Provisioning	KPI 4 - Average slice provisioning time (in seconds)	Execute the Create Slice Workflow.	Figure 14 from demo MUSTS and Figure 37 from demo WISE
Isolation	KPI15 - Slice isolation index	To overload the resources of a given slice without affecting the resources of the other slice.	Figure 16 from demo MUSTS
Management	KPI3 - Average service provisioning time (in seconds)	To deploy the service from the tenant.	Figure 15 from demo MUSTS and Figure 24 from demo ELSA
Elasticity	KPI1 - Average elasticity response time (in seconds)	Measure the time between the trigger for the elasticity and the time to accomplish it.	Figure 17 from demo MUSTS and Figure 31 from demo MLO
Scalability	KPI 4 - Average slice provisioning time (in seconds) during the Resource Discovery phase.	Perform the Resource Discovery phase in Create slice workflow, considering large slices spanning over multiple providers with numerous resources.	Table 4 from demo MARK
Monitoring	KPI 7 - Monitoring-data availability	Show the amount of data from IMA collected in each slice over the slice life-time.	Figure 18 from demo MUSTS, Figure 33 from demo MLO, and Figure 38 from demo WISE
VIM-independence	KPI 4 - Average slice provisioning time (in seconds)	Collect the time spent to deploy different VIMs in the same Slice.	Figure 19 from demo MUSTS and Figure 38 from demo WISE
Bare-metal slice	KPI 9 - Physical Server Utilization	Collection of the measurements related to the different (physical) resources that form the slice parts. Demonstrating that overloading one end-to-end Slice does not affect the other Slices in the same Resource Provider, as they use a disjointed set of physical resources.	Figure 27 from demo ELSA

5. Conclusion

This document described the deployment, tests, and validation processes of the NECOS platform with the related results. The activities were organized and executed in a way that WP6 acted as an integration point for the other technical work packages, as shown in Figure 39. The scenarios and requirements from WP2 were used to create demonstrations, the distributed architecture from WP3 and the model and APIs from WP4 allowed the creation of a geographically distributed testbed to show the flexibility and power of the NECOS architecture, APIs, and model. Finally, the prototypes from WP5 made possible to test how the LSDC concept could be demonstrated in meaningful scenarios.

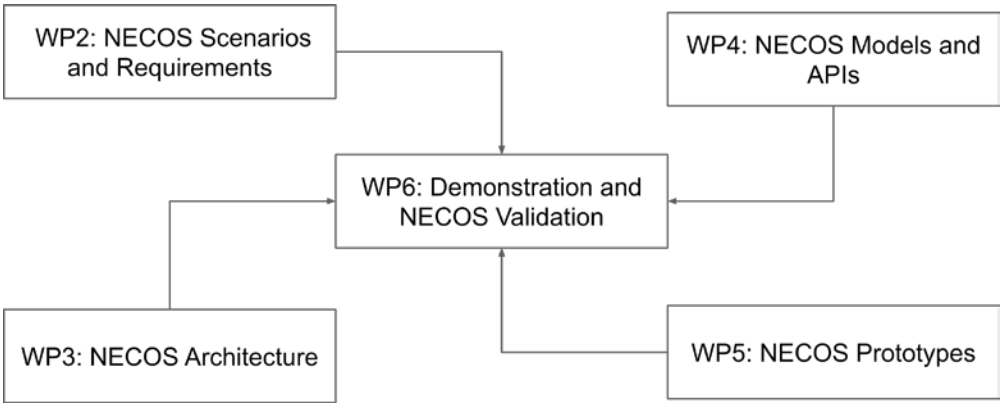

Figure 39. WP6 as an integration Work Package.

As presented in Figure 40, the initial seven (7) PoC implementations from D6.1 were used as a base for the final five (5) demonstrations: MUlti-Slice/Tenant/Service (MUSTS), Marketplace (MARK), Experiments with Large-scale Lightweight Service Slices (ELSA), Machine-learning based orchestration of slices (MLO), Wireless Slicing Services (WISE). Altogether, the different NECOS platform instances featured 2 Slice Providers, 2 Marketplaces, 7 Resource Providers, and integration with FED4FIRE testbeds uses as Resources Providers, in multi-site deployments, resulting in the first cloud network slicing embodiments between Europe and Brazil.

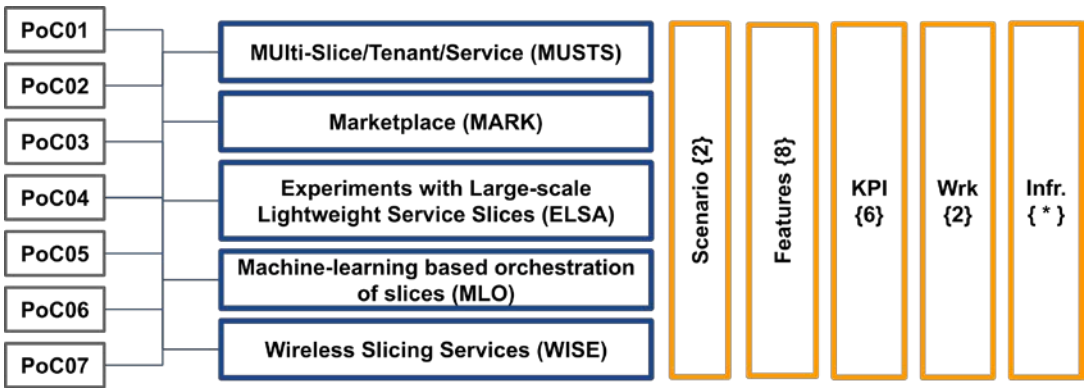

Figure 40. NECOS validation overview.

EUB-01-2017

Eight prioritized features were validated during the Create Slice, Elasticity and Decommission workflows. For each one of these features, at least one KPI was presented in the results demonstration section, for a total of 6 validated KPIs.

The system tests were performed by each demonstration, allowing us to use the related results as input for the section describing the Acceptance Verification. The latter summarized the tests and results regarding the eight NECOS prioritized features in Table 5, where all the 8 prioritized features of NECOS were validated via the execution of the NECOS workflows, altogether contributing to the overall validation of the NECOS propositions.

References

[Bradner 1997] S. Bradner, “Key words for use in RFCs to Indicate Requirement Levels”, RFC 2119, March 1997.

[H2020 2017] H2020, 2017, Technology readiness levels (TRL), General Annexes from Part 19 - Commission Decision C(2017)7124 G, https://ec.europa.eu/research/participants/data/ref/h2020/other/wp/2018-2020/annexes/h2020-wp1820-annex-g-trl_en.pdf

[Sommerville 2011] Sommerville, Ian (2011). Software engineering. Boston: Pearson. ISBN 978-0-13-705346-9.

[D2.1] Deliverable D2.1: Initial definition of use cases, 2018.

[D3.1] Deliverable D3.1: NECOS System Architecture and Platform Specification. V1, 2018.

[D4.1] Deliverable D4.1: Provisional API and Information Model Specification, 2018.

[D5.1] Deliverable D5.1: Architectural update, Monitoring and Control Polices Frameworks, 2018.

[D5.2] Deliverable D5.2: Intelligent Management and Orchestration, 2019.

[M6.1] Milestone M6.1: Report of Software Tools for Automated Infrastructure Deployment, 2018.

[Mouradian 2018] Mouradian, Carla et al. “A Comprehensive Survey on Fog Computing: State-of-the-Art and Research Challenges.” IEEE Communications Surveys & Tutorials 20 (2018): 416-464.

[Siaterlis 2013] Siaterlis, C., Garcia, A. P., & Genge, B. (2013). On the use of Emulab testbeds for scientifically rigorous experiments. IEEE Communications Surveys & Tutorials, 15(2), 929-942.

[Maxweel 2019] Maxweel Carmo, Felipe S. Dantas Silva, Augusto Venâncio Neto, Daniel Corujo, and Rui Aguiar, "Network-Cloud Slicing Definitions for Wi-Fi Sharing Systems to Enhance 5G Ultra Dense Network Capabilities", in: Wireless Communications and Mobile Computing, Vol. 2019, Article ID 8015274, 17 pages, doi: 10.1155/2019/8015274

Version History

Version	Date	Partner	Description/Comments
0.1	23/02/2019	UFPA	Draft ToC
0.2	06/03/2019	UFG	Insertion of subsection 3.3.2 Private Interconne between Europe and Brazil as content from Mil MS17
0.3	03/04/2019	All	Insertion of the content related to the initial deployment of the software used in the islands as content from milestone M18
0.4	22/06/2019	All	Insertion of Prioritized features in section 2
1.0	16/09/2019	UFPA	Draft version for Internal Revision
1.0.1	26/09/2019	UCL, UPC, UNICAMP	Revised Version 1
1.0.2	18/10/2019	All	Several contributions
1.1	22/10/2019	UFPA	Second version ready for internal review
1.2	28/10/2019	UCL, UPC, UNICAMP	Revised Version 2
2.0	29/10/2019	UFPA, UPC	Final Version