871780 — MonB5G — ICT-20-2019-2020

Deliverable D6.1
Technical Report on System Integration and Operation

Document Summary Information

Grant Agreement No	871780	**Acronym**	MonB5G
Full Title	Distributed Management of Network Slices in beyond 5G		
Start Date	01/11/2019	**Duration**	42 months
Project URL	https://www.monb5g.eu/		
Deliverable	D6.1 – Technical Report on System Integration and Operation		
Work Package	WP6		
Contractual due date	M39	**Actual submission date**	M41
Nature	Technical Report	**Dissemination Level**	Public
Lead Beneficiary	IQU		
Responsible Author	Luis A. Garrido (IQU)		
Contributions from	Luis A. Garrido (IQU), Kostas Ramantas (IQU), Sergio Barrachina (CTTC), Farhad Rezazadeh (CTTC), Luis Blanco (CTTC), Engin Zeydan (CTTC), Luca Vettori (CTTC), Sarang Kahvazadeh (CTTC), Lanfranco Zanzi (NEC), Francesco Devoti (NEC), Rafał Tępiński (ORA-PL), Robert Kołakowski (ORA-PL), Georgia Pantelide (eBOS), Sihem Cherrared (ORA-FR), Fabrice Guillemin (ORA-FR), Adlen Ksentini (Eurecom), Sabra Ben Saad (Eurecom), Karim Boutiba (Eurecom), Mohamed Mekki (Eurecom), Abderahmane Tlili (Eurecom), Sofiane Messaoudi (Eurecom), Gussan Mufti (eBOS), Vasiliki Vlahodimitropoulou (OTE), Ashima Chawla (LMI), Anne Marie Cristina Bosneag (LMI).		

Revision history

Version	Issue Date	% Complete	Changes	Contributor(s)
0	15/11/2022	0	ToC	Luis A. Garrido
1	8/3/2023	100	Final format	Luis A. Garrido, Engin Zeydan

Disclaimer

Copyright message

TABLE OF CONTENTS

List of Figures

List of Tables

List of Acronyms

Acronym	*Description*
3GPP	*Third Generation Partnership Project*
5GC	*5G Core*
AD	*Anomaly Detection*
AE	Analytic Engine
AI	Artificial Intelligence
CCI	Cloud Continuum Infrastructure
CLA	Closed-loop Automation
CN	Core Network
CNF	Cloud Native function
CQI	Channel Quality Indicator
CSMF	Communication Service Management Function
DB	DataBase
DDOS	Distributed Denial-of-Service
DE	Decision Engine
DMO	Domain Manager and Orchestrator
DNN	Deep Neural Network
DRL	Deep Reinforcement Learning
DSM	Domain Slice Manager
ECATP	Enhanced Context-Aware Traffic Predictor
EPC	Enhanced Packet Core
EEM	Embedded Element Manager
eMBB	Enhanced Mobile Broadband
ETSI	European Telecommunications Standards Institute
ECA	Event Condition Action
ENI	Experiential Networked Intelligence
FCAPS	Fault, Configuration, Accounting, Performance, Security

G5IAD	Graph-based Interpretable Anomaly Detection
GCN	Graph Convolutional Network
HA-DRL	Heuristically Assisted DRL
IDM	Infrastructure Domain Manager
IDMO	Inter-Domain Manager and Orchestrator
IDSM	Inter-Domain Slice Manager
InP	Infrastructure Provider
ISM	In-Slice Management
ITU	International Telecommunication Union
K8	Kubernetes
KNN	k-nearest neighbours
KPI	Key Performance Indicator
LCM	Lifecycle Management
LDA	Linear Discriminant Analysis
LXC	Linux Containers
ML	Machine Learning
MANO	Management and Orchestration
MaaS	Management as a Service
MAN-F	Management Function
mMTC	Massive Machine Type Communications
MECapp	MEC applications
MEO	MEC Orchestrator
MGEN	Multi-Generator
MNO	Mobile Network Operator
MS	Monitoring System
MEC	Multi-access Edge Computing
MEP	MEC Edge Platform
NBI	North Bound Interface
NF	Network Function
NFMF	NF Management Function

NFVI	Network Function Virtualization Infrastructure
NFVO	Network Function Virtualization Orchestrator
NSD	Network Service Descriptor
NSM	Network Slice Manager
NSO	Network Service Orchestrator
NSP	Network Service Provider
NSPR	Network Slice Placement Request
NSI	Network Slice Instance
NSM	Network Slice Manager
NSMF	Network Slice Management Function
NSSMF	Network Slice Subnetwork Management Function
NST	Network Slice Template
NSSI	Network sub-Slice Instance
NGMN	Next Generation Mobile Networks
NFVI	NFV Infrastructure
OAI	Open Air Interface
ONAP	Open Network Automation Platform
OSM	Open-Source MANO
OSS	Operation System Support
PaaS	Platform as a Service
PCA	Principal Component Analysis
PoC	Proof of Concept
PRB	Physical Resource Block
PSN	Physical Substrate Network
PV	Persistent Volumes
PVC	Persistent Volume Claims
QoE	Quality of Experience
QoS	Quality of Service
RAN	Radio Access Network
RBAC	Role-Based Access Control

RNN	Recurrent Neural Networks
SDN	Software Defined Networks
SFC	Service Function Chain
SON	Self-Organizing Network
SLA	Service Level Agreement
SFL	Slice Functional Layer
SML	Slice Management Layer
SM	Slice Manager
SO	Slice Orchestrator
TASAC	Time Aware Slice Admission Control
TD	Technology Domain
UE	User Equipment
uRLLC	Ultra-Reliable Low-Latency Communication
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VNF	Virtual network Function
VNFM	Virtual Network Function Manager
VoD	Video-on-Demand
XAI	interpretable Artificial Intelligence
ZSM	Zero-touch network and Service Management

1 Executive summary

The scope of this deliverable focuses on validating the inter-operability of the Monitoring Systems (MS), Analytics Engines (AE), Decision Engines (DE), Actuators (ACTs) and the fulfilment of the control-loop capabilities envisioned by the MonB5G Architecture specified in Work Package 2 (WP2). This deliverable is not only different in that it greatly extends the work presented in D4.3, but it in fact joins up the work presented in Work Packages 3, 4, and 5 (WP3, WP4 and WP5, respectively) with the objective of materializing the mission that the MonB5G Project had set for itself in the first place. This mission can be summarized in the following way: providing a solution for decentralized control in Beyond-5G communication networks.

Whereas WP3 focused on the design and integration of the MS and AEs, WP4 focused extensively on the design choices for the implementation of the DE, and WP5 focused on the security and energy efficiency aspects of the MonB5G components, this current deliverable (D6.1) will focus on the first effort of putting all of these components together. This means it will focus on the functional integration of the MS/AE/DE/ACTs for different monitored systems, geared towards providing autonomic management (control-loop automation) with the features envisaged in the early stages of the MonB5G Project.

In order to provide a convincing, systematic and clear demonstration of this integration effort, this deliverable will also present, from a very high-level architectural perspective, the experimental platforms in which the MonB5G components have been integrated and functionally tested. This is necessary information to provide context to the reader about the testing procedure and the properties of the environments in which MonB5G has been demonstrated to thrive and report the benefits it envisioned. In order to enhance the substance of the experimental platform descriptions, some details of the tools to implement the platforms will also be included, in order to contextualize further the scope of the functional validation.

In the journey towards reaching this stage of development for MonB5G, the partners have provided multiple contributions to achieve the MonB5G objectives and management features. These contributions have been presented in the previous deliverables, but a sub-set of them have been committed for functional validation and posterior performance evaluation (for the upcoming D6.2). In this deliverable, we revisit these enablers in a wider scope, a scope that this time includes the interoperability of components of enablers, how they map to the MS/AE/DE/ACTs components at the core of MonB5G specifications, how they map to the static components of the MonB5G Architecture (IDMOs (Inter-Domain Manager and Orchestrators), DMOs (Domain Manager and Orchestrators), MonB5G Portal).

The key achievements covered by this deliverable are:

- Demonstration of the integration of the MS/AE/DE/ACTs for control-loop realization
- Fulfilments of Architectural Vision of MonB5G
- Functional validation of partners' enablers and their compliance with MonB5G specifications.

2 Introduction

2.1 Objectives of MonB5G Integration Process

This deliverable presents the results accomplished in the functional integration and validation of the MonB5G components within the scope of the project. This validation and implementation consist of integrating the MS/AE/DE/ACT components to materialize autonomic control-loops within the framework of Beyond-5G communication networks. This deliverable will shed further light on the implementation and testing of the AE components, the implementation of the MS, as well as the integration of these components with the MS.

This is a necessary objective to fulfil in order to prepare the contributions for further validation steps within the framework of the MonB5G Project. After the functional validation presented and achieved here, it will be necessary to demonstrate the ability of the integrated MonB5G components to fulfil the KPI objectives presented in WP2, as part of the MonB5G Architectural vision for distributed management. The achievement of this KPIs is critically dependent on the inter-operability and integration of the MonB5G components, presented in this deliverable.

2.2 Deliverable overview and Structure

The outline of this deliverable is as follows. Section 3 will provide a recap of the overall MonB5G Architecture and its relation to the MonB5G core components, namely the MS, the AEs, the DEs and ACTs. This section will briefly explain the different two Proof-of-Concepts (PoCs) that were envisaged for MonB5G which establish the framework for functional and posterior performance validation. This information is given here in order to provide enough context for the prototype reference architecture of the experimental platforms used for functional validation in this deliverable. This is necessary because the different Proof-of-Concepts require reference architectures with different properties and features.

Section 4 gives a summary of some of common network and cloud related faults available in the telco environment. This helps in developing more tailored cloud and network related faults to show the benefits of the enablers in PoCs. Section 5 provides demonstrations of the functional validation of Enablers for Proof-of-Concept 1. In this section, a sub-set of the contributions by partners presented in previous work packages are now revisited with special focus on component integration and autonomic control implementation. In this same vein, Section 6 continues by providing demonstration for the functional validation of contributions of Proof-of-Concept 2, which are enablers oriented towards security and energy efficiency. Section 7 wraps up this document by providing a general summary of what the deliverable presents, and the subsequent steps for the fulfilment of the MonB5G objectives.

3 Architecture of Experimental Platforms for the PoCs

3.1 Architecture Mapping

The design philosophy of MonB5G, as it was defined in WP2, is to provide hierarchical, feedback-loop-based control for faults, configuration, accounting, performance, and security (FCAPS) management. Moreover, its design also was oriented towards slice orchestration, featuring different control loops with different control scopes, goals, and timescales. This control strategy with all its features is expected to work at the Global OSS/BSS level, the Technological Domain level, Slice Level, and the Node (Virtual Network Function/Physical Network/Cloud Native Network Function) level.

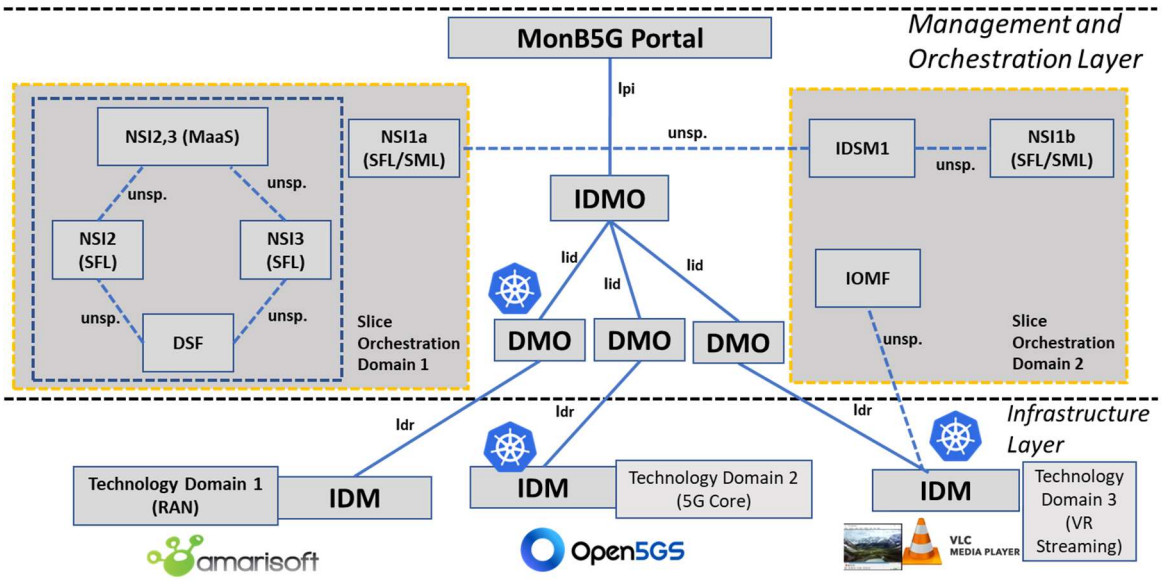

Figure 1. MonB5G architecture in implementation testbed

Figure 1 shows how MonB5G architecture is overviewed from the perspective of implementation testbed setup of PoCs that are studied in WP6. In this figure, there are various technological domains and MonB5G components are distributed and deployed on top of these multi-domain technological locations. The MonB5G architecture is composed of static and dynamically deployed components. The architecture as presented in Figure 1 includes the MonB5G Portal, the IDMO (Inter-Domain Manager Orchestrator), the DMO (Domain Manager Orchestrator) and the Infrastructure Domain Manager (IDM) and includes various interfaces between them. The dynamically deployed components, on the other hand, are associated to the autonomic management capabilities within the slice, referred to as In-Slice Management (or ISM). This is shown in Figure 2 (taken from D2.4 – Figure 8).

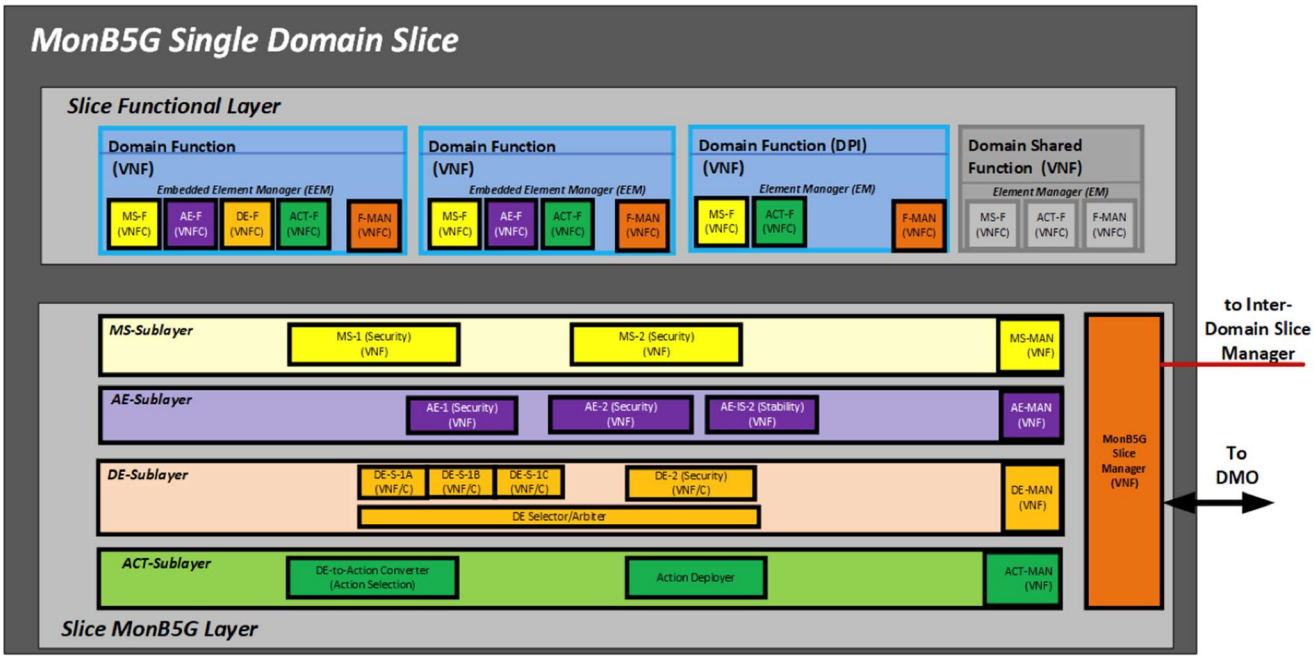

Figure 2. MonB5G Single Domain Slice

Whether we refer to the static or dynamically deployed components, or whether they are deployed at any technological domain or node, all the components of the MonB5G architecture consist of the following components:

1. **MS Sublayer:** responsible for collecting, aggregation and processing of the monitored data
2. **AE Sublayer:** responsible of performing different types of data analytics on the data stored from the system being monitored
3. **DE Sublayer:** responsible of generating orchestration decisions of different types
4. **ACT Sublayer:** responsible for the execution of the DE orchestration directives by calling onto more primitive actions related to the monitored system.

These components of the MonB5G architecture are explained in Figure 2. In this figure, the Functional Layer refers to the layer corresponding to the components that actually provide the service while the MonB5G layer holds the aforementioned MS, AE, DE and ACT sublayers. Within the context of the slice, the SFL consists of (among other things) the Service Function Chains (SFCs) that the tenants deploy as part of the services their users consume. Tenants design their services and request the resources they need from the Infrastructure Provider (InP) at the different technological domains of the infrastructure. The tenants will require slice-level orchestration and management in order to orchestrate their services satisfactorily and ensure that their users experience the desired QoS. For this reason, the MonB5G design supports the ability for in-slice management.

At the same time, the InP needs to have orchestration and management capabilities to manage the slices themselves, and this needs to be done at each TD in which the slice (or parts of it) are deployed. The MonB5G

architecture provides support for orchestration and management at each TD and provides also the capabilities to do so across multiple TDs. As can be seen from Figure 2, in the design philosophy of MonB5G, the DMO (a static component of the MonB5G architecture) would look like as shown in Figure 3. In this figure, the SFL consists of the components associated to support 5G-related functionalities at the specific Domain in which a DMO is deployed, and it also includes its own OSS/BSS Layer that communicates with the IDMO. As we shall we see, a lot of the enablers that have been tested so far fit well into a DMO with the corresponding sub-layers.

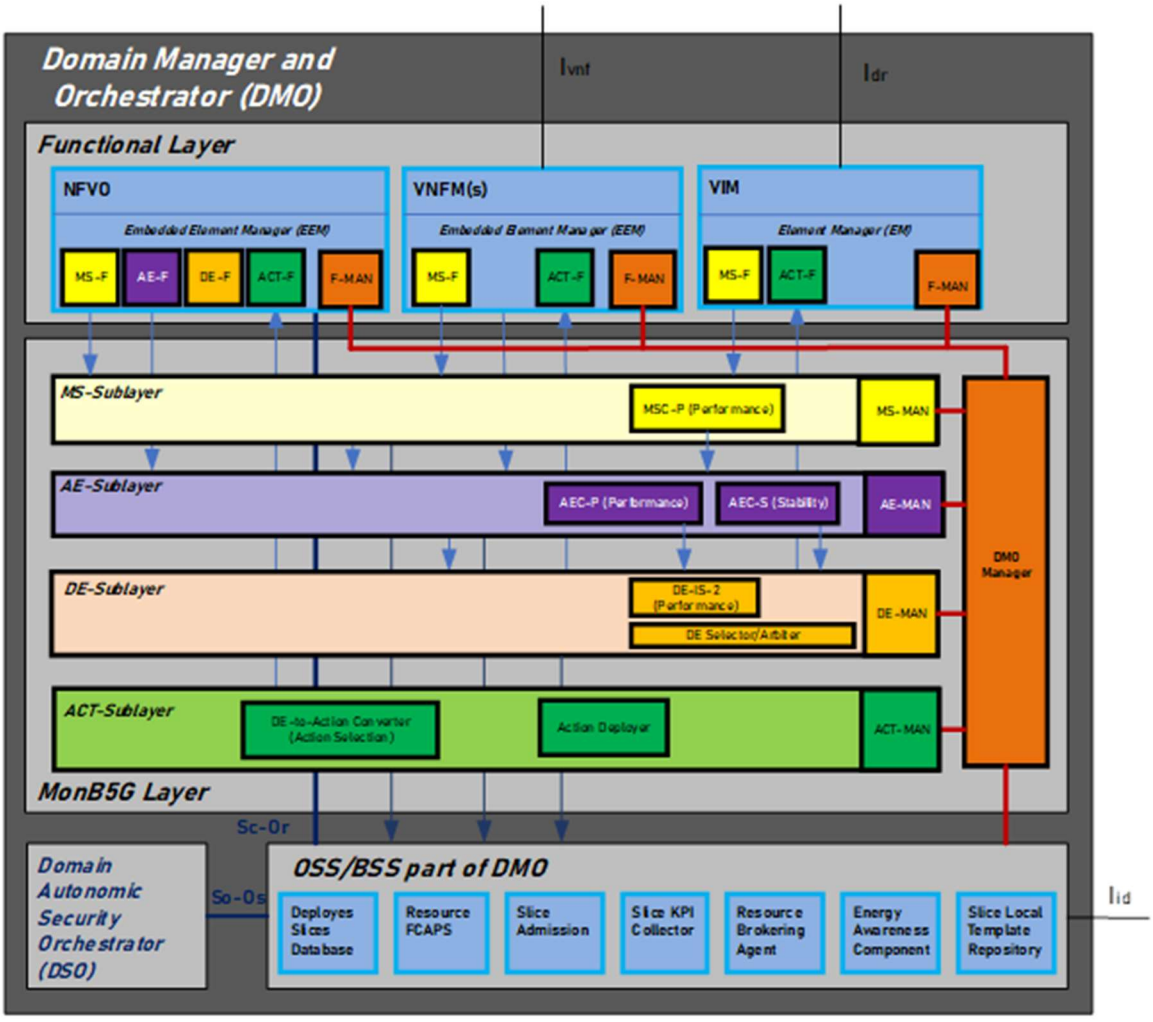

Figure 3. Architectural Diagram of the DMO as per MonB5G.

Starting from Section 5, we will detail how the different contributions developed by each partner satisfies the architectural requirements specified in WP2 and reproduced in this section for convenience. The scope of these sections in this document is to provide a detailed explanation, description and results (when

possible) of the functional integration of the MS, AE and DE in their respect sub-layer for the validation of the MonB5G Architectural template.

Within this scope, it is also necessary to provide a very high-level description of the testing platforms in which the contributions were functionally tested, and it is necessary to explain the relationship of this testing platforms to the experimental scenarios of the MonB5G project in order to validate these contributions. These high-level descriptions given for each experimental scenarios are given in Sections 3.2.1 and 3.2.2 for the first and second Proof-of-Concepts, respectively.

3.2 Proof-of-Concept Demonstrations for MonB5G

In this section, we present the Proof-of-Concept (PoC) experimental scenarios, in which we validate the functional integration of MonB5G. These PoCs are designed to showcase the ability of the different contributions of the partners of the MonB5G project to fulfil the architectural objectives of the proposed architecture. For the PoCs, the partners had proposed what we called “enablers”, which are a way to refer to specific implementation of an AE, DE and ACT which has been designed to provide a cloud native distributed intelligence to PoC demonstrations.

3.2.1 PROOF-OF-CONCEPT # 1

The first Proof-of-Concept consists of two experimental scenarios that differentiate each other by showcasing the MonB5G capabilities at two distinct layers: at the service level (Experimental Scenario 1) and at the network-slice level (Experimental Scenario 2). The MonB5G solutions presented in this deliverable are designed to target the objectives initially established for the project, for which a prototype experimental platform will be required. Before describing this prototype architecture and how it maps to the static and dynamic components of the MonB5G Management Architecture, we will first generally revise the objectives of the experimental scenarios for this POC, which will provide more context to understand the design of the experimental architecture for testing.

The first Experimental Scenario (Zero-Touch Multi-Domain Service Management with end-to-end SLAs) will demonstrate Automated Zero-Touch service management and multiple redundancy mechanisms (to ensure high-availability) in complex multi-domain services. Continuous monitoring and closed-loop autonomic control mechanisms will ensure self-healing, self-configuring and self-scaling of **services**, to address faults and performance issues in any of the service technological domains. The main of this experimental scenario is to demonstrate:

- Continuous monitoring of the **provided services** across all technological domains by the respective monitoring engines
- Capacity to carry out **localized decisions** by DEs placed at location with the monitored systems to enforce SLAs.
- Capacity to **aggregate decision-making policies** at central points of control i.e., central DEs, to ensure SLA compliance.
- Ability of DEs to enforce energy optimization policies, while simultaneously SLA constraints.

The objective of the second Experimental Scenario (Elastic End-To-End Slice Management) is to showcase how the MonB5G architecture addresses local performance issues in multiple technological domains as well

as changes to traffic patterns in various timescales. As the number of Network Slice Instances (NSIs) increases, the scale and complexity of lifecycle management and slice reconfiguration makes automation a necessity. Each NSI consists of multiple NSSIs, generally one per technological domain (i.e., 5G Core, RAN and transport network), while each technological domain in MonB5G has its own data-driven MS, AE and DE components. The contributions presented in this deliverable will demonstrate the capacity of the MonB5G architecture components to:

- Achieve continuous monitoring of **each NSSI** by the respective Monitoring Engine at appropriate time-scales
- Achieve pro-active decision-making related to **network slice** management through DE deployments at each domain
- Achieve distributed data analysis from multiple sources of data ingestion i.e., **multiple network slices** for traffic forecasting, in order to enhance the pro-active decision-making capabilities of the DE

By understanding the objectives of each experimental scenarios, then it is possible to come up with a specification of prototype experimental architecture on which to test the MonB5G solutions. As previously mentioned in the beginning of this section, the focus of experimental scenarios 1 and 2 are at the service-level and at the network slices, respectively. Thus, a prototype architecture needs to be able to support 5G services, which implies the inclusion of the right hardware components for 5G, and support for Network Function Virtualization (NFV) and Software Defined Networks (SDN) to support network slicing. The components of the prototype architecture (explained in Section 3.2.1.1) for testing are such that satisfy these requirements.

3.2.1.1 TESTBED FOR POC-1

For this POC, the MonB5G project will use a testbed experimental platform that includes an experimental 5G Infrastructure deployment with a corresponding network environment. This platform possesses capabilities for virtualization that enable it to work as an independent and self-contained ecosystem for Network Functions Virtualization (NFV). The prototype architecture for this testbed is shown in Figure 4. The resources available in a Testbed platform will generally include:

- 5G Core: Amarisoft Callbox 5GC, Free5GC, OAI CN, and Open5GS
- 5G RAN: Amarisoft Callbox gNBs and UERANSIM gNB
- 5G UEs: Amarisoft Simbox, Smartphones, and UERANSIM UEs.

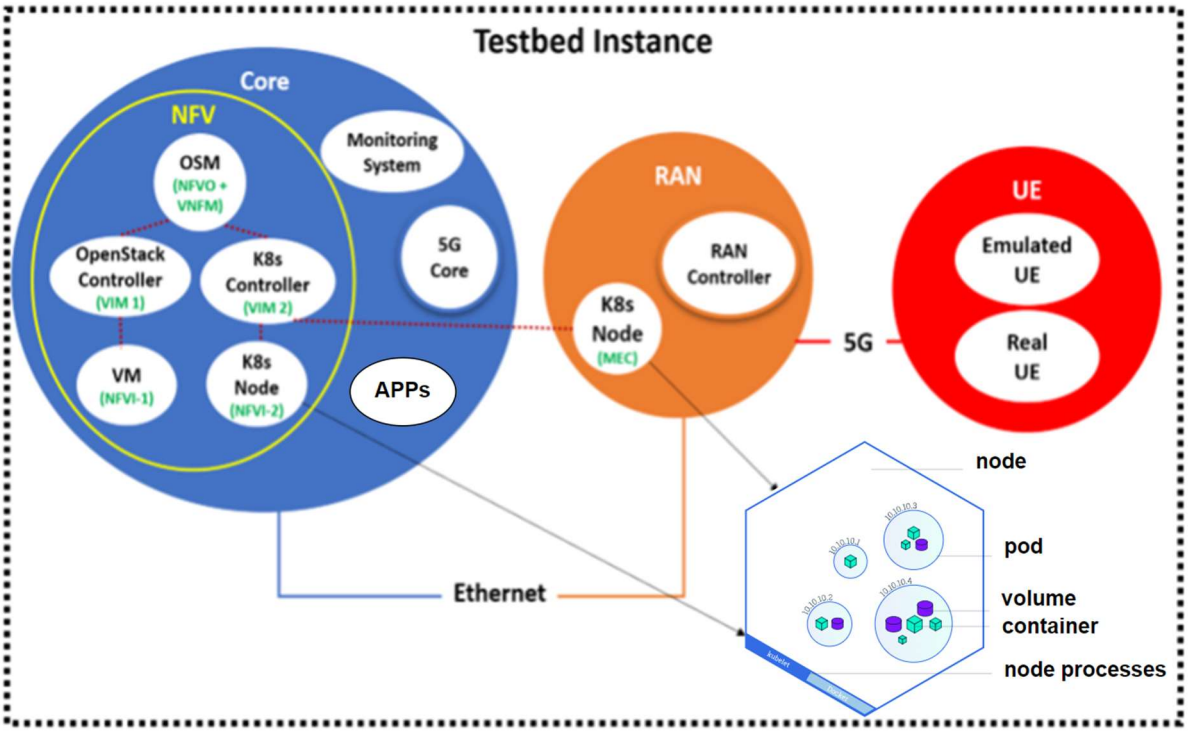

Figure 4. Typical Testbed Instance offered by CTTC.

The 5G UEs, 5G RAN, and the 5G Core are critical components to support the **5G services** for experimental scenarios. The 5G RAN and 5G Core have physical servers deployed on-site within their domains (UEs do not have physical servers added). The purpose of these physical servers are to comprise a cloud computing infrastructure (CCI) that provides the capabilities to deploy containers and Virtual Machines (VMs), both of which are computing primitives necessary to support NFV/SDN, and thus **network slicing**. The CCI in the 5G RAN domain is commonly referred as an Edge CCI. These computing primitives in turn enable the testbed to create different 5G services and to create network slices, both elements that are required to satisfy the PoCs determined for MonB5G Project. The VMs supported by the CNI are used as (worker/master) nodes to deploy Kubernetes clusters, while containers use different applications and microservices.

For the Experimental Scenarios of this deliverable and for the deployment of the partners' contribution, we have deployed two instances of the testbed shown in Figure 4. The second testbed instance has been deployed at Iquadrat (IQU), and consists of the 5G CORE (with Open5GS and Open Air Interface deployed, however Open5GS is used for the current functional validation), the 5G RAN (consisting of an Amarisoft Callbox gNB), and 5G UEs (consisting of multiple 5G phones and 5G-enabled embedded systems). The testbed deployed at IQU also has a 5G Edge Cloud domain attached to the 5G RAN. Both the 5G Cloud and the 5G Edge Cloud of IQU's testbed supports virtualization at different layers, which is a necessary feature to support VMs, containers and NFV/SDN functionalities.

The **deployments of the MS** (and in many cases, for the AE and DEs) in both testbed instances will depend on the capabilities of the CNI present in the 5G Core and 5G RAN domains. Many of the contributions

presented in this deliverable require the deployment of the MS in the Edge/RAN domain as well as in the 5G Core, which implies the presence of multiple MS deployments. Each of the MS deployments will require different directives on which information to sample from the monitored systems, in order to satisfy the control-loop constraints of the particular contribution.

In order to exercise control of the deployed 5G services under examination and the network slices, it is necessary to provide **Orchestration** capabilities in order to carry out the decisions issued by the DEs. In the cloud and edge cloud domain this is done by providing a customized interface with the Kubernetes scheduler in order to drive the resource allocation to VNFs/CNFs or to execute changes in other characteristics of the deployment of the 5G Services and/or network slices.

3.2.1.2 CONTRIBUTIONS FOR POC-1 AND EXPERIMENTAL SCENARIOS

The contributions of the partners to the MonB5G architecture for the PoC-1 address different improvements across the technological domains of a 5G mobile network. As was also mentioned in Section 3.2.1, the PoC-1 consists of two experimental scenarios (ESs) designed to test the partners' contributions in the corresponding experimental deployments. The experimental scenarios and their relationship to the contributions of the partners are explained Table 1. Here we see the different enablers that have been committed for deployment, and described the MonB5G component they were designed to instantiate. Notice that when it comes to the MS, the same instance is used in both ESs since this is a transversal component during the integration of the MonB5G architecture.

Table 1. Mapping of each partner solutions to functional components of MonB5G.

ES	MS	AE	DE	Actuator
ES#1	MonB5G MS	• Policy-based FL Predictor • Slice KPI Prediction with Interpretable Anomaly Detection • LSTM-based Anomaly Detection	• Fault Detector • Threshold based Scaling Decision	• Scaler: CPU Scaling of CNF • App relocator: Moving application container from cloud to edge
ES#2	MonB5G MS	• Enhanced Context-Aware Traffic Predictor (ECATP)	• Slice Admission control based on traffic prediction (TASAC) • A multi-agent learning for distribution	• BW allocator: A python script on Amarisoft BW update • Handover actuator: React to changes in RAN (move user from one to another)

			resource allocation in the RAN domain • Heuristically assisted DRL approach for network slice placement • Prediction-based Admission Control (PreBAC)	

In order to illustrate the relationship of the partners' contribution to the technological domains of the 5G architecture shown in Figure 4 and the partners contribution, Figure 5 shows where each contribution belongs in the technological domains, and to which scenarios of POC-1 they belong to. Notice that this figure also briefly illustrates the relationship between the contributions to the AE and DE instances, and the specifications of the MonB5G architectural components shown in Figure 1, Figure 2 and Figure 3. ECATP, for example, is specified as an instance of the AE in Table 1, and it is deployed at the 5G Edge Cloud Domain as a sub-component of the DMO at this respective domain. Another example is TASAC, which is an instance of a DE, and it is deployed in the 5G Cloud Domain as a sub-component of the respective DMO as well.

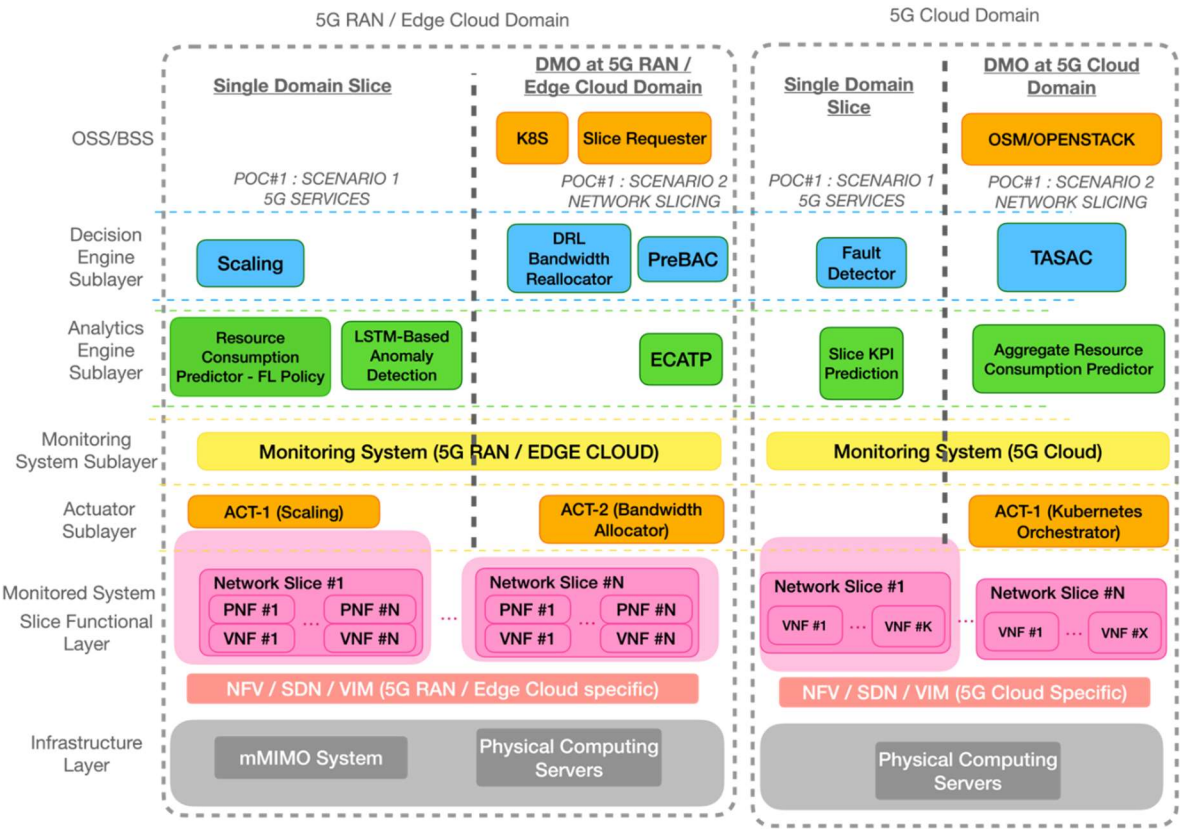

Figure 5. Summarizing the scenarios of PoC #1, and the way the enablers are deployed, in their respective domains and how they are integrated.

Figure 5 shows the domains displayed which are the 5G Cloud and the 5G RAN / Edge Cloud domain. Within each of these domains, we show how the DMO presented in the deployment for functional testing. The DMO has the MonB5G Layer, in which the MS/AE/DEs are deployed.

3.2.2 *PROOF-OF-CONCEPT #2*

The second Proof-of-Concept "AI-assisted policy-driven security monitoring & enforcement" consists of two experimental scenarios. The main purpose behind this use-case is to show both the efficiency of MonB5G when relying on AI to ensure new security threats detection in addition to their respective mitigation actions, and the proper enforcement of the AI-based techniques through novel trust-based evaluation mechanisms. By leveraging SDN, MANO and NFV technologies and through the usage of the platform built for MonB5G, the effectiveness of the security system will be demonstrated.

The first experimental scenario is about the attack identification and mitigation. This scenario applies Machine Learning algorithms (such as gradient boosting,), in order to identify attacks to slices and automatically apply actions to mitigate them.

The objective of this scenario is to demonstrate the robustness of MonB5G for identifying, detecting and then mitigating the in-slice and cross-slice attacks. Different malicious events would be detected thanks to various MonB5G's components including Monitoring System (MS), Analytics Engine (AE), and Decision Engine (DE). This experimental scenario will assist the efficiency of MonB5G for detecting attacks (such as DDOS and alter attacks).

The second experimental scenario is about the robustness of learning algorithms in the face of attacks, such as poisoning attack. First, this scenario will assess the vulnerability of the distributed learning algorithms to adversarial attacks and how their performance metrics (e.g., precision, accuracy, etc.) could be affected by those attacks. Then, the model robustness will be improved by MS, AE and DE components.

The objective of the second scenario is to demonstrate that even under significant numbers of misbehaving entities, distributed learning can be carried out in a robust way. This will be validated using standard metrics used in machine learning.

3.2.2.1 TESTBED FOR POC-2

EURECOM's 5G facility has been designed specifically to provide a vertical a high-level system to run a trial on top of a 5G infrastructure and collect KPI, without taking care about the complexity of the system, and low-level information. Figure 6 shows a high-level architecture of the adopted facility architecture. Three layers are distinguished: the vertical and users space, orchestration and management, and infrastructure. The user layer is where the vertical and the trial owner interact with the facility to define, run and monitor a trial. It is mainly composed by the Web Portal. The orchestration and management layer that is composed by all the entities that configure, instantiate, run the trial as a network slice, and monitor the KPI. Finally, the infrastructure layer, which is composed by the elements that runs the 5G components, such as RAN, CN, MEC applications, VNFs, and MEP.

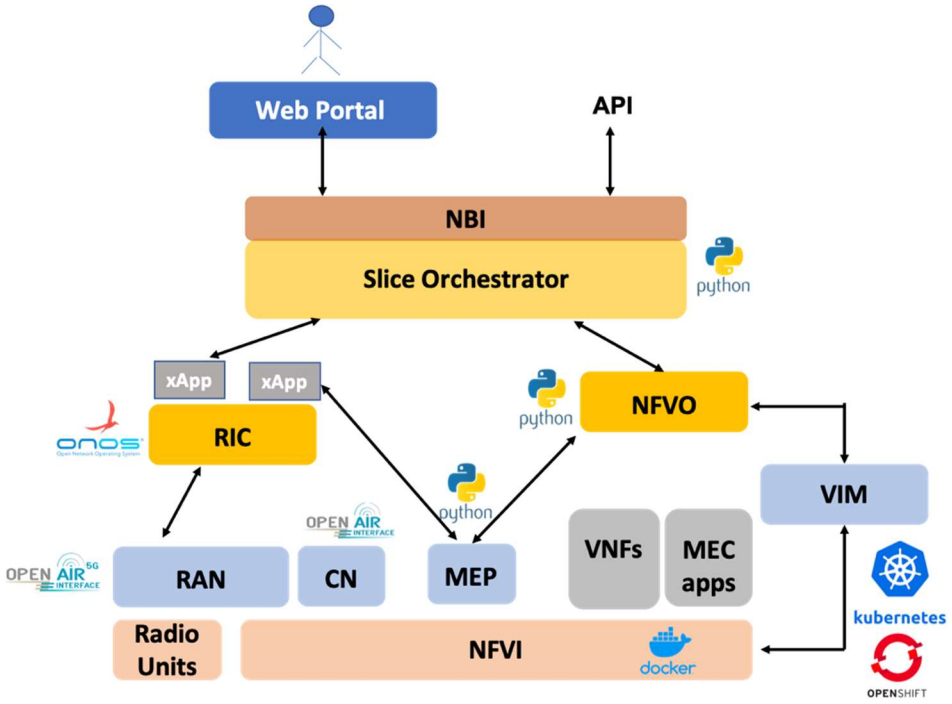

Figure 6. Global architecture of the 5G Facility.

The Web Portal is the key element of the facility, as it is the interface with the vertical and trial owner. The Web Portal should abstract the 5G components specific by providing a high-level of abstraction to the vertical to deploy and monitor a trial. Figure X illustrates the Web Portal architecture, which is composed of a front-end, trial enforcement, life-cycle management and KPI monitoring and presentation as well as two data bases (DB). All the components collaborate to ensure the life-cycle of a trial, which is composed of: definition and preparation, configuration and instantiation, run-time management, and deletion.

The orchestration and management layer both see the trial as a NS and is composed of the Slice Orchestrator (SO), which is in charge of the LCM of NS, RAN Orchestrator that manages the LCM of the RAN part of a NS, and the NFVO that manages the LCM of the MEC applications (MECapp) and VNF composing the service of the vertical. According to the 3GPP management architecture, the SO corresponds to the NSMF, the NFVO and RANO to NSSMF. The SO is the entry point of the 5G facility. It exposes a NBI to the Web Portal for the NS LCM and monitoring. It uses the NBI exposed by the NFVO and RANO to deploy a Network Slice on each domain and start the monitoring process. The SO of the EURECOM 5G facility implements the NS LCM as specified by 3GPP. The NFVO role is to deploys the service part of a Network Slice over the virtualization platform. The NFVO takes as inputs the NSD, including the list of AppD. At the slice creation it on-boards the virtual images, and at the instantiation request it creates the instances of all the applications. The NFVO of EURECOM facility is a home-made software, it can manage docker containers on top of VIM that run Kubernetes on bar-metal. It also supports OpenShift. Per the SO requests, the NFVO also creates a monitoring agent that collect data on the CPU, memory usages and networking information of the applications belonging to a specified slice. The NFVO interacts with the MEC Edge Platform (MEP) to ensure traffic redirection to the

MEC applications. The RANO aim is to manage the RAN resources dedicated to a Network Slice. The RANO checks the availability of the radio resources before deploying a network slice. RANO relies on the O-RAN based on ONOS (ONF) to handle the radio resources and radio slice configuration using xApp.

The **deployments of the MS** (and for the AE and DEs) will interact with the 5G Core and 5G RAN domains. In summary, partners' solutions are mapped to each of the MonB5G functional components as below Table 2.

Table 2. Mapping of each partner solutions to functional components of MonB5G PoC2.

		MS	AE	DE	ACT
scenario #1	mMTC attack	MonB5G MS	- Gradient boosting - Principal Component Analysis (PCA)	A python script reacts based on the detection value calculated by AE	An Element Manager (EM) implemented in the AMF, that adds the malicious UEs in a blacklist.
	aLTEr attack	MonB5G MS	- Machine Learning model of normal network activity	It generates a remediation action plan and requests its execution through the ACTUATOR	A Python program that translates action plans received from the MonB5G DE into API calls
scenario #2	FL attacks	MonB5G MS	- Dimensionality reduction algorithm	Clustering algorithm	A python script selects the trusted FL nodes.

4 Telco Cloud Faults

In this section, we investigate different telco cloud faults that are available in standardization domain to classify and use some of them in final PoC environments to demonstrate the benefits of MonB5G solutions. The 5G network is redesigned in accordance with a service-oriented architecture by breaking everything down into specific functions and sub-functions. "Slicing" is a method of provisioning a dedicated E2E network instance to end users, businesses, and MVNOs. One network can have multiple slices, each with unique characteristics that serve different use cases.

NFV, which is used by service providers, gives them a new way to design, implement, and manage network services and applications. NFV eliminates the need for specialized networking hardware by virtualizing the most essential network components and replacing them with software. This allows the configuration of entire network node functions as connectable components for the development of end-to-end communications networks.

Mobile network operators need to use a Management and Network Operations (MANO) system in order to design, implement, manage, and run a mobile network with the aforementioned features. Using MANO systems, a mobile network operator can build end-to-end network slices and manage them for their whole lifecycle as customers order them.

Figure 7 shows 3GPP network slicing management in NFV framework. The 3GPP 5G Management and Orchestration Framework's most important parts are outlined below:

- **Communication Service Management Function (CSMF)** is responsible for translating the communication service-related requirements into network slice related requirements.
- **Network Slice Management Function (NSMF)** is responsible for the management (including lifecycle) of NSIs. It derives the network slice subnet related requirements from the network slice related requirements.
- **Network Slice Subnet Management Function (NSSMF)** is responsible for the management (including lifecycle) of NSSIs.
- **NF Management Function (NFMF)** is responsible for application-level management of VNFs and PNFs and is a producer of the NF Provisioning service that includes Configuration Management, Fault Management and Performance Management.

These new capabilities correlate to the ETSI NFV architecture, which has enabled new levels of network slice management control, customization, and adaptability. All these new functionalities interact with the NFV Orchestrator (NFVO) in order to define, provision, configure, and manage the various network slices and their components. This means that the NSSMF and NFMF manage the VNFs and PNFs on behalf of the NFVO. Cloud computing services are realized through the mapping of the service layer network into the physical infrastructure. Multiple failures in the physical infrastructure could disrupt cloud network connectivity and cause cascading failures that affect customers.

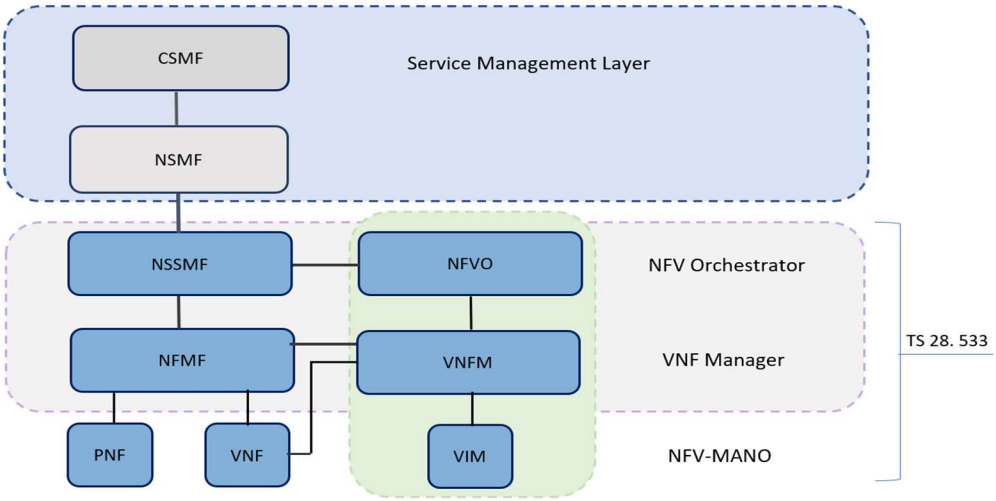

Figure 7. 3GPP network slicing management in NFV framework [ETSINFV2017].

The telco operators have a Network Slice Manager (NSM) on top of their NFV orchestration. The NSM is responsible for coordinating the resources between VNFs and PNFs, both at the slice level as well as at the service layer. NFMF manages network slice resources with slice aware FCAPS (Fault, Configuration, Accounting, Performance and Security management) support, including the ability to process and forward NE slicing alarms and counters to NSSMF as well as slice parameter configuration and provisioning towards NE.

As NFV systems are widely deployed to enable the flexible and robust provisioning of a variety of network services, including mobile telecommunications, they must be protected against large-scale failure scenarios.

The ETSI NFV industry specification group defines the NFV framework that is used by Telco Cloud as consisting of three levels.

1- Network Function Virtualization Infrastructure (NFVI) consists of hardware resources, virtualization layer, and virtual resources to deploy VNFs.
2- Virtual Network Functions (VNFs) block includes Network Functions (NFs) running in a virtual form on a shared virtual infrastructure.
3- NFV MANO contains a group of functions responsible for managing infrastructure resources, functions, and services

All network services should typically be protected at least against single failures in the corresponding service chains because they may depend on multiple network functions that are part of the corresponding service chain. A network service will therefore only be accessible while each of the associated functions is active and working properly. Therefore, a multi-layered approach should be employed to ensure maximum resilience. In order to increase the resilience and dependability of the services, MANO provides additional fault detection and repair capabilities.

The underlying NFV system must be able to reconfigure the service chains and perform the failover, in addition to the VNF redundancy and synchronization schemes. It means that the required monitoring and

management subsystems, as well as the related control channels and hardware providing the necessary resources, must be available at the time of failure.

NFV-MANO functional blocks identified by the NFV-MANO architectural framework are as follows:

• Virtualised Infrastructure Manager (VIM)

• NFV Orchestrator (NFVO)

• VNF Manager (VNFM)

An NFV deployment generally consists of a composition of individual elements that work together to provide a network service. These elements must be taken into account when estimating the reliability of the service. The elements that are considered in the reliability estimation are the presented in Table 3 [ETSIGS2016]. Elements of a network service that are not part of the NFV implementation are also included in order to complete the picture [ETSIGS2016].

VNFM performs lifecycle management, including fundamental operations (create, instantiate, scale in/out, terminate, delete, query VNF, and modify VNF) and advanced functions (such as scale up/down, healing, update/patching, upgrades, backup and restore) for applications. VNFM has a method for self-monitoring that checks operational services. VNFM also offers an integrated fault management system that can be accessible via the product's GUI (graphical user interface).

Table 3. Groups and types of network service elements for reliability estimation [ETSIGS2016].

Element group	Element type	Description
Hardware	Compute node	Hardware platforms, i.e. servers on which VNFCs (VMs or containers), controllers or NFV-MANO can run
	Storage node	Hardware platforms for storage solutions
	Network node	Hardware platforms for virtual networks, e.g. running software switches to manage networks between compute nodes and external networks
	Networking element	E.g. physical switches and routers, physical load balancers
	Physical link	LAN, MAN or WAN connections between physical networking elements, e.g. Ethernet
Virtualisation software	Hypervisor	Virtualisation environment running on a host
	Software switch	Switch implementations running on compute as well as networking nodes, e.g. vSwitch
	Network controller	Software responsible for the logically centralized network control if used for (virtual) network management, e.g. SDN controller
	NFVO	NFV-MANO part involved in all lifecycle operations of a NS
	VNFM	NFV-MANO part involved in all lifecycle operations of a VNF
	VIM	NFV-MANO part involved in all lifecycle operations of a virtualised resource
Virtualised element	VNFC	VNF implementations in VMs or container environment, VNF-internal load distributor
	Virtual link	Virtual connection between two VNFCs
Independent elements		Any element used which is independent from the NFV deployment, e.g. PaaS or PNFs

Different root causes (faults) of failure exist in the NFV system, and the most common are the following:

VNF resource faults

VNF resource faults are related to the abnormal functioning of VNFs and their respective resources. The type of resources can be compute, storage, or network.

VNF resource connectivity faults

A VNF resource connectivity alarm is triggered when VNFM cannot connect to one or more components of the virtualized infrastructure manager where a specific VNF is deployed.

Failures at the VNF/VNFC level

VNF lifecycle management (LCM) faults related to the LCM operation(s) of a specific VNF

VNF failures (abnormal functioning of VNFs and their respective resources) like overload situations, misconfiguration, a software bug, or failure of the supporting NFVI leading to VM failure.

Failures at the VNF Manager level

VNFM Resource faults

- Disk space low
- Memory low
- CPU usage high
- Disk I/O is overloaded

VNFM Infrastructure DNS faults

- DNS server is either unavailable or not configured.
- DNS error

VNFM Backup and Restore service faults

VNFM's full backup failed

VNFM incremental backup failed

VNFM catalog faults

- The catalog is not available.
- One or more VNF packages are missing from the external catalog.
- One or more VNF package ID have changed.

Faults in the VNFM Cluster

- Cluster service failed.
- Cluster service has been degraded.
- Cluster members are not accessible via the application service.
- VNFM node fault
- Cluster node outage

- VNF Lifecycle Change Notification delivery failure.

Types of failure for the NFV-MANO functional entity

- Failure of the whole NFV-MANO functional entity.
- Communication failures with other peering NFV-MANO functional entities.
- Failures that affect a specific interface produced by an NFV-MANO functional entity, such as the VNF lifecycle management interface made by a VNFM
- Malfunctioning of the NFV-MANO functional entity due to failures on resources supporting the execution of the entity, e.g., CPU, memory, is reported as an event type relevant to resources.

Fault management and troubleshooting are supported through the monitoring of software and hardware elements, including physical switches, host network interface cards, virtual switches, and multiple storage devices and services.

Fault information may be the result of several different sources of faults: physical infrastructure (i.e., physical NFVI compute, storage, and networking related faults); virtualised infrastructure (e.g. VM-related faults), and application logic (i.e. VNF instance related faults). When fault information related to the same primary cause is issued by some or all those multiple sources, it needs to be correlated. In the NFV-MANO architecture, such correlation could happen in multiple places: the NFVO, the VNF Manager, the EM and/or some OSS [ETSIGS2014].

A unified alarm database may be established for all faults by including VNFs, VIM, and VNFM into NFMF. The NF Management Function (NFMF), acting as the EM, communicates with the VNFM providing data on application aware VNF performance and faults. NFMF has capabilities like centralized threshold violation calculations, centralized alarm correlation, KPI calculations, and service impact analysis and all the alarms and performance data are combined in one system.

NFMF capabilities (Slice-Aware FCAPS)

- ✓ Alarm monitoring for all slicing alarms from managed NFs
- ✓ Alarm filtering for slicing related alarms
- ✓ PM report/KPI/threshold on radio slice resource level (SNSSAI), not slice/slice subnet level
- ✓ Translation from Slice SLA to CM (Configuration Management) parameter configuration plans
- ✓ Translation from Slice Tracking Area to impacted gNBs
- ✓ Association of network slice instance and corresponding network resources with working set
- ✓ Massive CM (Configuration Management) provisioning towards relevant NFs

With the process of "telco cloud alarm enrichment," VNF/VNFC alerts are supplemented with information about NFVI resources (such as compute node and VM IDs). More visibility at the OSS layer is achieved through enriched alarm sending to NBI (north bound interface).

Relationships between VNFs, Virtual Machines (VMs), and Compute Nodes are displayed in Alarm Monitor thanks to the telco cloud topology feature.

5 Integrated Deployment, Completion and Evaluation for PoC-1

5.1 Deployment of MonB5G MS at the PoC-1 Testbed

In our PoC deployment, we will be utilizing three instances of our Monitoring System across three distinct sites, designated as A, B, and C. Sites A and B are classified as edge sites, where in addition to the Monitoring System, we will also be running the VR/VoD APP server, federated learning clients, and an orchestrator. These edge sites will be close to their respective 5G RAN infrastructure, which are the equivalents of the 5G RAN Domain in Figure 4, while site C will serve as our 5G Cloud domain, also in shown in Figure 4. So, in addition to hosting the federated learning server, which will facilitate the exchange of weights between clients, it runs a *parent/root* instance of our Monitoring System that utilizes the monitoring systems at sites A and B as sampling functions. The overall deployment layout is illustrated in Figure 8.

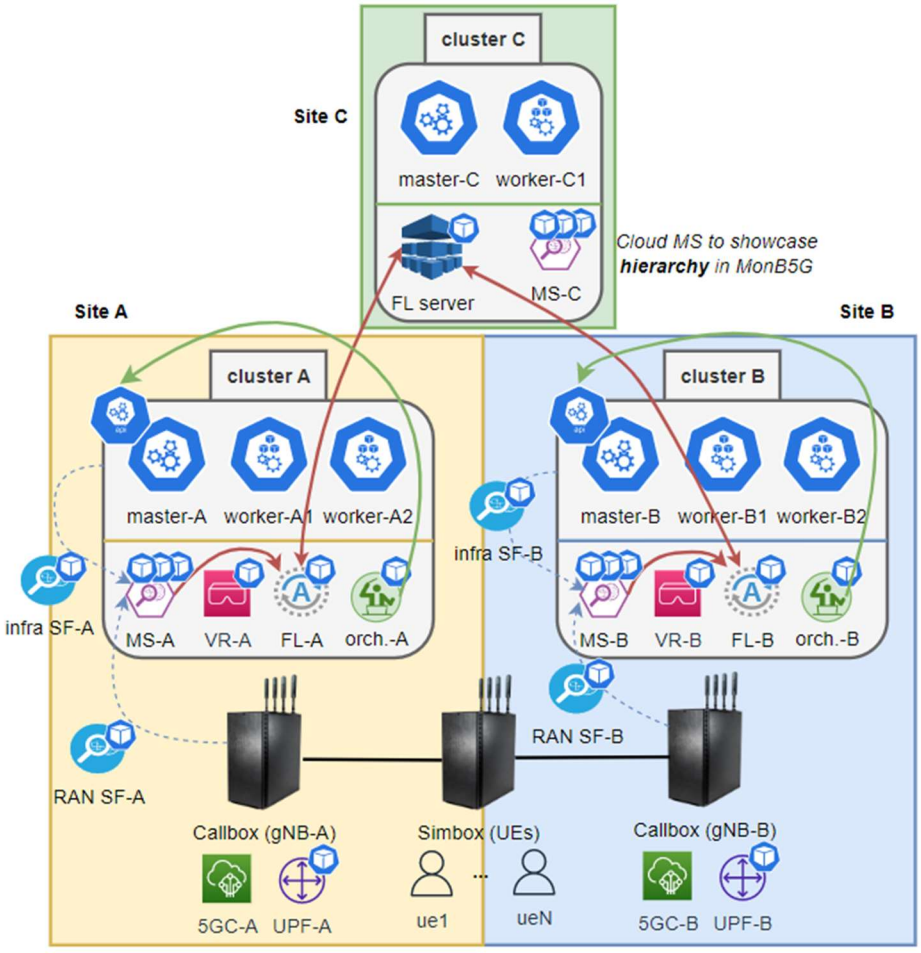

Figure 8. Architecture components for PoC1.1.

To ensure a smooth deployment process, we have outlined the various steps involved in deploying the various components of our Monitoring System. These steps will include preparing the necessary infrastructure, configuring, and installing the software, and testing and validation of the system prior to deployment.

5.1.1 DEPLOYMENT AND CONFIGURATION OF KUBERNETES CLUSTERS

In our implementation, we will be deploying and configuring Kubernetes clusters using Linux Containers (LXC) to instantiate VMs that will act as Kubernetes nodes. It is important to note that for the purpose of this PoC, we have implemented a minimal setup for each Monitoring system, consisting of only two Kubernetes nodes per cluster (one master and one worker). This is illustrated in Figure 9.

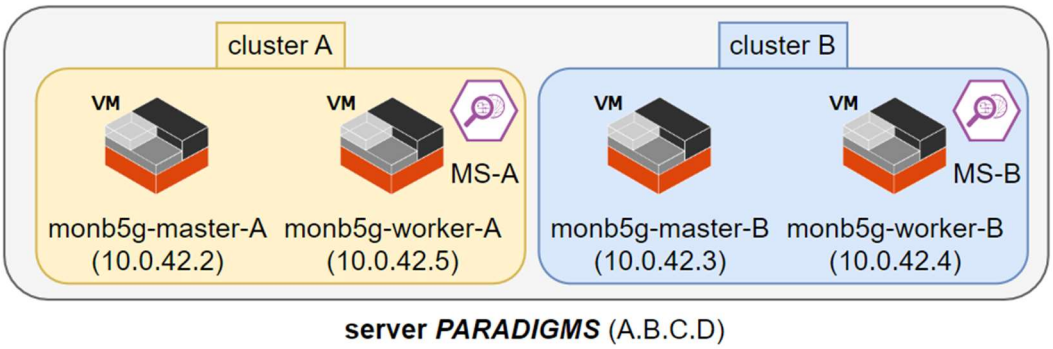

Figure 9. Deploying edge clusters in multi-domain sites.

5.1.2 DEPLOYING THE MS INSTANCES

In order to prepare the VMs for the deployment of our Monitoring System, we will first perform several initial setup tasks. One of these tasks is the deployment of a Kafka cluster, for which we will be using Strimzi. This latter is an open-source tool designed to simplify the deployment and management of Kafka clusters on Kubernetes. Once Strimzi is installed, we will proceed to create a Kafka cluster, which will then be used to create topics within the cluster. The Cluster Operator that is deployed when installing Strimzi will continuously monitor for new Kafka resources, ensuring that the cluster is running smoothly.

To ensure the persistence of data, we will be installing Persistent Volumes (PV) and Persistent Volume Claims (PVC) using openebs. This will allow us to store data on the cluster's storage resources, rather than on individual nodes, providing greater data durability and reliability. Next, we will be deploying the Time Series Database (TSDB) with InfluxDB, a powerful and flexible open-source TSDB that is well-suited for monitoring and analyzing time-series data.

Finally, we will be deploying the main components of the Monitoring System, such as the Manager, including Role-Based Access Control (RBAC) configuration and corresponding services. This will ensure that the system is properly configured and that only authorized users have access to the system's resources and functionality.

5.1.3 *OPERATE THE MS WITH SAMPLING FUNCTIONS*

Once the Monitoring System is deployed, we will proceed to install sampling functions that will feed the Kafka bus with metrics of interest coming from various endpoints and domains. In this PoC, we will be considering two main sampling functions: **amarisoft-gnb** and **pod-infra-metrics**. Both sampling functions are developed in Python and have been Dockerized, making them ready to be deployed as pods in the Kubernetes clusters.

The first sampling function, amarisoft-gnb, provides metrics from the 5G RAN, including information about UE and gNBs. In particular, we are interested in RAN metrics such as Channel Quality Indicator (CQI) and downlink bitrate, as these metrics can be used to estimate CPU usage at the APP server. These metrics are provided by a custom API that runs in Amarisoft Simbox and Callbox. However, we are still exploring other metrics of interest that may be useful for the system. Once the sampler functions have been Dockerized, they can be realized as full sampling functions (with the corresponding Kafka producer as a sidecar pod) and configured via the config loop. We will use the config loop to register the sampling function in the Monitoring System via the manager. Finally, the metrics of interest are grabbed and filtered by the Kafka consumer (e.g., the FL client) and only the relevant data is considered for the training an inference phases.

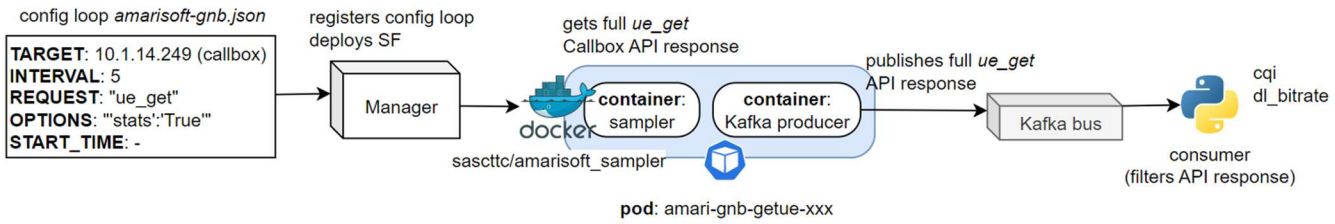

Figure 10 Flowchart of the amarisoft-gnb SF.

The second sampling function, pod-infra-metrics, follows a similar procedure, but this time we focus on the infrastructure metrics (e.g., CPU and memory usage) of pods of interest. In our scenario, the pods of interest are those related to Video-on-Demand (VoD) servers. The sampling function will feed the Kafka bus with metrics related to the CPU and memory of VoD servers. Then, the federated learning client will use this data (together with the RAN data) to train and infer CPU usage.

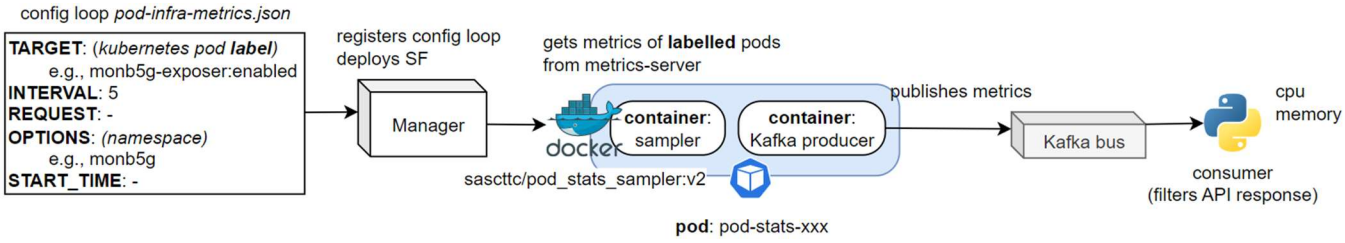

Figure 11. Flowchart of the pod-infra-metrics.json

Notice that this sampling functions uses the Kubernetes metrics server, so it must be enabled beforehand in the cluster. This is a crucial step as it allows for the collection of resource usage metrics, such as CPU and memory usage, for all pods in the cluster. Once the metrics server is enabled, we proceed to create the necessary Kubernetes objects to allow the sampling function to access the Kubernetes API and collect the desired metrics. Specifically, we will be creating a ServiceAccount, ClusterRole, and ClusterRoleBinding. The ServiceAccount will be used by the sampler pod to authenticate with the Kubernetes API and access the metrics. The ClusterRole and ClusterRoleBinding will be used to grant the ServiceAccount the necessary permissions to access all Namespaces within the cluster. Remarkably, we have upgraded the MS manager to let all this configuration be automized via specifying new parameters in the config loop.

5.1.4 *CONNECTING THE MS WITH GRAFANA*

In order to facilitate the visualization of the monitored metrics and other cluster metrics, we will be using Helm to install kube-prometheus. Helm is a package manager for Kubernetes that makes it easy to install, upgrade, and manage Kubernetes applications. However, it is important to note that we will be using a custom values file for the installation of kube-prometheus. This is because we want to instruct Prometheus to scrape additional services that have an annotation 'prometheus.io/scrape=true' and have a named port that ends with the word 'metrics'. This will allow us to collect and monitor more metrics from the cluster.

Once kube-prometheus is installed, we will proceed to configure Grafana, which is an open-source data visualization and monitoring tool. We will create a new dashboard and create charts with the queries to the metrics exposed by the Kafka consumer(s). See example in Figure XXX below.

Figure 12. Example of Grafana dashboard showing metrics of interest.

It is important to emphasize that kube-prometheus runs in parallel to the MonB5G Monitoring System and it acts only as an add-on to facilitate visualizing the monitored metrics by MonB5G Monitoring System and other cluster metrics in Grafana dashboards. This allows us to have a more comprehensive view of the system's performance and identify any potential issues more easily.

5.2 Deployment of MonB5G AEs at the PoC-1 Testbed

5.2.1 ENHANCED CONTEXT-AWARE TRAFFIC PREDICTOR

5.2.1.1 USE-CASE DESCRIPTION AND BASELINE

The Enhanced Context-Aware Traffic Predictor (ECATP) is designed to be a context-aware traffic predictor that enhances the ability of a Deep Neural Network (DNN) for forecasting through the embedding of problem-domain knowledge into its training procedure. The idea of this predictor is to provide a KPI prediction, specifically predicting the probability of SLA violations due to resource under-provisioning for a network slice. It was originally designed to predict SLA violations in a multi-slice setting depending on the resource allocation of each slice. Moreover, it was also designed to be used as a resource demand predictor at any Technological Domain of Figure 4. However, in its current instance it has been deployed in the 5G Edge Cloud domain, as close to the corresponding base stations at the RAN domain, providing traffic prediction values of the traffic profile of a slice running in a BS. The deployment of ECAPT took place in the Poc-1 testbed developed at Iquadrat's premises.

5.2.1.2 METHODOLOGY FOR INTEGRATION

In order to complete the validation of ECATP, a containerized version was developed to be executed in the 5G Edge Cloud domain using Kubernetes and Docker, alongside the deployment of a MS. The deployment of the MS in this case followed a similar procedure of deployment as it was detailed in Section 4.1. An extended software interface was coded for ECATP in order to successfully communicate with its containerized deployment. This extended interface allows ECATP to perform the following tasks:

1. Drive the periodic training of the ECATP based on baches of data sampled from the MS.
2. Perform inference from the current traffic values sampled by the MS and streamed directly to ECATP.

ECATP needs to be trained periodically with time-series data of the traffic profiles arriving to a 5G RAN Base Station. The extended software interface developed for ECATP in the containerized deployment has capabilities to sample sequential batches of data from the DB engine inside the MS, which consists of an InfluxDB. It is important to keep the sequential nature of the data in order to ensure the integrity of the training and the predictive capabilities. According to our experiments, it takes at least 48h hours of continues data sampling and periodic training every 10 minutes for ECATP to actually generate the expected predictions. However, there are more efficient ways of training

However, in order to make training more efficient, we also enabled ECATP to be trained offline with historical data gathered from the BS at the 5G RAN domain through previous iterations. Once the model is trained in this way, it can be loaded into the containerized deployment of ECATP, and then perform inference with the data sampled from the MS. In this way, the training can occur independently from the inference procedure

without interruption while the model trains on the background. The ability to load new models asynchronously was also added as part of the extended interface for ECATP in its containerized deployment.

In order to perform the inference procedure in the containerized version of ECATP, it is necessary to stream data directly from the MS through the Kafka messaging interface already implemented in the MS. The extended ECATP interface includes a subscriber to a corresponding MS Kafka topic to capture the latest traffic magnitude value, in order for ECATP to generate the predicted traffic value for the next time step. In this way, the inference can be performed as soon as new data becomes available through the Kafka bus, without requiring an additional access to the InfluxDB.

5.2.1.3 METHODOLOGY FOR VALIDATION

In order to validate the capabilities of the containerized version of ECATP, the resulting values of prediction and the corresponding current traffic values are logged into a separate table in the InfluxDB of the MS, with their corresponding timestamp values. Grafana is then deployed to read this table, to allow a comparison between the prediction values and the traffic values corresponding to that timestep. The Grafana dashboard for this case is updated in real time, and it is possible to assess the reliability of ECATP for traffic profile predictions.

5.2.2 POLICY-BASED FL PREDICTOR

5.2.2.1 USE-CASE DESCRIPTION AND BASELINE

As described in D3.3, this use case introduces a new method for ensuring efficient and scalable operation of AEs that use policy-based stochastic FL scheme in a non-IID environment. The method uses a cloud-native service-level agreement (SLA) to control resource provisioning at the edge of a radio access network. The method is developed in cloud native environment and is shown also to be more effective than other FL methods in terms of reducing SLA violations, shortening convergence time, and reducing computation costs, resulting in greater scalability in previous deliverables (e.g., D3.3, and D5.4).

Figure 13 below shows the proposed implementation of stochastic-based FL approach stochastic-based scalable FL and AE selection approach. All components of the FL are implemented as pods in K8s environment.

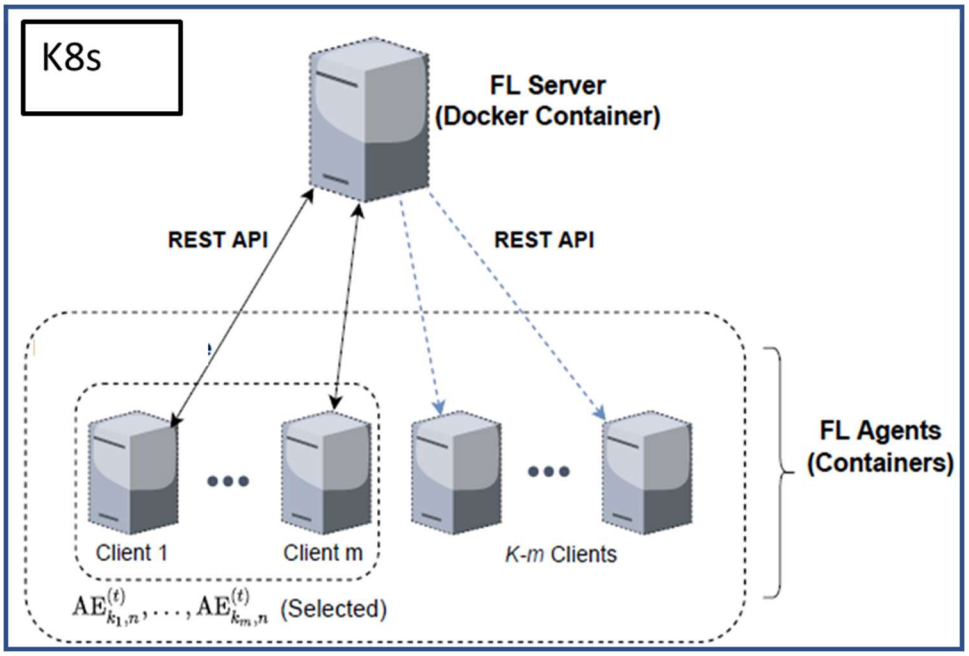

Figure 13. K8s implementation of stochastic-based scalable FL and AE selection approach.

5.2.2.2 METHODOLOGY OF TESTING AND VALIDATION

Code Overview

There are two main modules that are implemented: **the server module** and the **AE/client module.** Cloud implementation simulator used in our experiments is updated version of [Yanez2021]. Server module represents the aggregation server as its class diagram is given in Figure 14. Inside Server, several methods are implemented and each training client is interacting with one of those implemented methods during the FL training process.

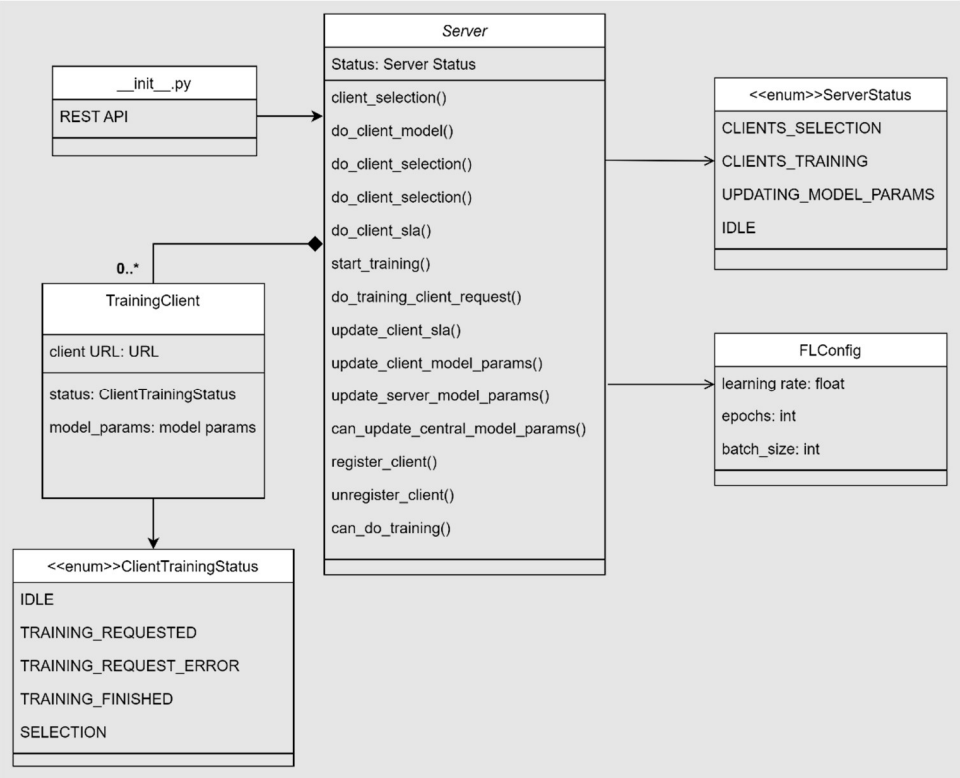

Figure 14. Class diagram of the server model.

In the server module, cloud native implementation is structured in two layers:

(1) **Web framework Layer:** This layer is responsible for REST service endpoints for HTTP and does not contain FL logic. It processes incoming requests extracts relevant information and calls a function in the Service Layer which performs FL logic. It packages the returned results appropriately and sends the response. It consists of the following files:

- main.py contains endpoint routes that map REST API to a function in the service layer. It also contains an HTTP server that starts the event loop and application. Some of the REST endpoints in this file are
    - `/select_client` (used to start the client selection process),
    - `/register_client` (used to add client from the network),
    - `/unregister_client` (used to delete client from the network),
    - `/prob_value` (used to send the SLA_based probability distribution),
    - `/model_params` (used to update the calculated parameters after the training)

(2) **Service Layer:** This layer contains only FL logic and does not contain HTTP and REST service endpoints. Communication protocols are stitched on top of it without interfering with FL logic.

- server.py contains all server related functions needed to implement the service endpoints.

Client module class diagram is given in Figure 15 below. In the client module, ML models under tensorflow Constrained Minimization Problem are implemented and the REST API calls the *Client* using a *ModelTrainer* implementation.

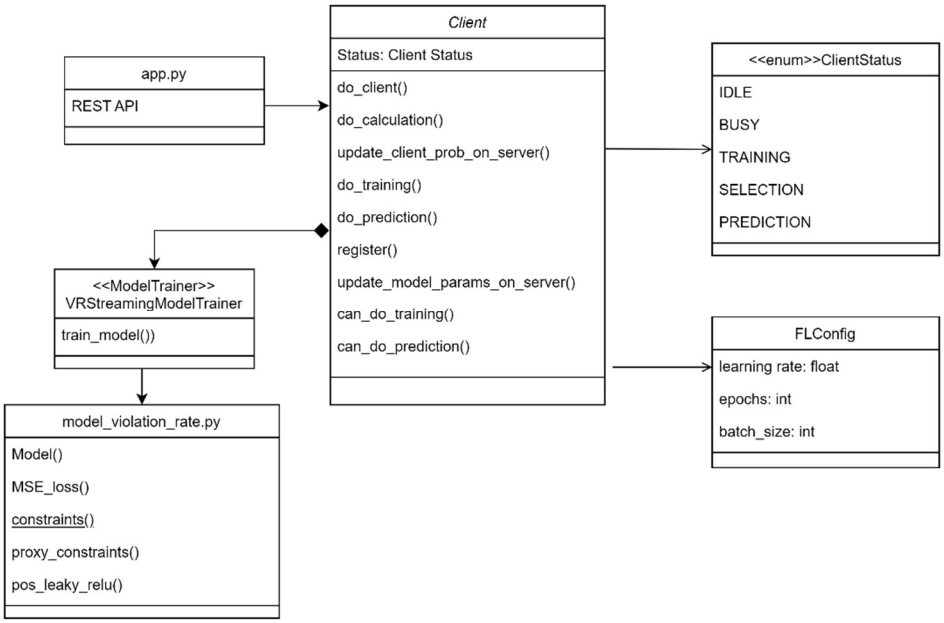

Figure 15. Class diagram of the client model.

In the client module, cloud native implementation is structured in two layers:

(1) **Web framework Layer:** This layer works similar to web framework layer in the server module. It consists of the following files:

- app.py contains endpoint routes that map REST API to a function in the service layer. It also contains an HTTP server that starts the event loop and application. Some of the REST endpoints in this file are
  - `/select_prediction` (used to request prediction to client),
  - `/training` (used to train the model),
  - `/SLA_calculation` (used to send request for SLA calculation),
  - `/worker_model` (used to update client model parameters)

(2) **Service Layer:** This layer works similar to web framework layer in the server module.

- client.py contains all client related functions needed to implement the service endpoints.

Communication Protocols:

HTTP is used through two different REST interfaces as communication protocol. One REST interface is implemented by the client nodes, and another is implemented by the aggregation server.

The aggregation server node has this set of basic REST operations:

- **POST/client:** Registering clients with the Server. (from Client to Server)
- **GET/select client:** Initiate policy for selecting clients and corresponding FL training. (from Admin to Server)
- **POST/SLA:** Clients send their SLA violation rate to the Server node. (from Client to Server)
- **PUT/model-weights:** Clients send calculated model parameters to the Server node. (from Client to Server)

In the client nodes we have the following operations:

- **PUT/SLA:** Server requests each of the clients to calculate their SLA violation rate. (from Server to Client)
- **POST/training:** Server requests the selected clients to start FL training with new model weights. (from Server to Client)
- **PUT/worker_model:** Update client initial model parameters. (from Server to Client)
- **GET/select_prediction**: Initiate prediction using the trained model in the docker container. (from Admin to Client)

Cloud native implementation of the above REST APIs has enabled easy configuration and installation of them in any device while allowing access to the aggregation server and intra-communication between clients from the Internet.

Infrastructure and MonB5G Components:

All MonB5G and infrastructure related components are below and will be running in K8s environment in final demo. In the infrastructure, a VR streaming client-server application will be used where we can monitor server and RAN related parameters via monitoring system for each FL AE client. Moreover, there are two independent sites with their own applications in different locations so that the data generated can be closer to independent identically distributed (i.i.d) in distribution.

- **VR Streaming Client:**
  - This is infrastructure related component
  - The UE is going to be emulated in Simbox.
  - To be implemented as a streaming client in Simbox.
  - VR Streaming client can also run in regular laptop
- **VR Streaming Server**:
  - This is infrastructure related component
  - To be deployed in a k8s cluster
  - This assumes that VR streaming server can be scaled up/down based on the commands coming from FL AE client and top orchestrator.
- **Centralized MANO**
  - This is used for benchmark comparisons. We will be building ML model using whole dataset collected from two MSs that are also inputting data to two different AEs
- **Monitoring System**

  - This is infrastructure related component.
  - MS to collect RAN and Server related parameters
  - Two MS instances to emulate local ones is suitable. Data will be collected from two different FL clients.
  - FL clients/AEs are connected to MS by fetching data from Kafka
- **Core Network**
  - This is infrastructure related component
  - This is Amarisoft Core and is used to represent the control elements. Assuming data many data plane elements can be created (UPFs) with a single control plane (Core NW control)
- **FL aggregation server**
  - This element is used to exchange weights between FL clients
  - This is implemented in Docker container and runs in K8s
- **FL AE/client**
  - This is used to build local models in training phase and make inferences in inference phase to predict CPU
  - This is implemented in Docker container and runs in K8s
  - It expects RAN related parameters and server related parameters detailed in Monitored KPIs part
- **K8s**
  - This the main orchestrator used in this scenario. It is used to handle manage the cluster and its containerized applications (which can include both MonB5G and Infrastructure related components)
- **Top Orchestrator (DE)**
  - This is a DE that has simple predefined rules (e.g., CPU> 80% scale up) and gives these commands to K8s orchestrator so that it can scale up/down the VR streaming server
  - FL AE clients are connected to top orchestrator by publishing data into Kafka to be used in top orchestrator

Considered features from MS used in FL are as follows (sampling functions):

- **Feature 1:** MIMO full rank (%) (RAN) (input)
- **Feature 2:** bit rate (RAN) (input)
- **Feature 3:** CQI cell-level (RAN) (input)
- **Feature 4:** CPU (VR Streaming Server) (output)

As mentioned above, we will use RAN related parameters as an input and CPU as output to build a ML model using federated learning approach.

Working Principles of the overall system:

The experiments will run in two phases: (1) Offline training and (2) Online Inference

(1) **Offline training phase:** Figure 16 shows the offline training phase of the experiment setup for policy-based FL approach

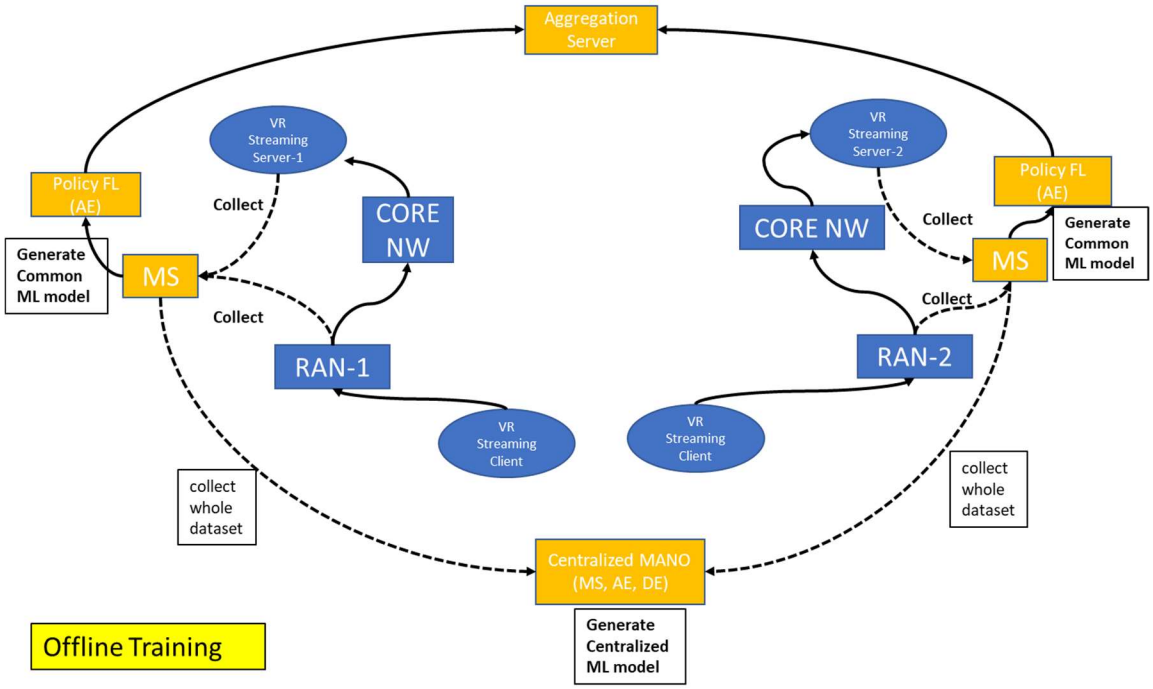

Figure 16. Experiment setup for Policy-based FL approach: Offline training.

STEP 1: REGISTRATION: All clients register with their IP address in the server node using **POST/client** request as shown in Figure 17 below.

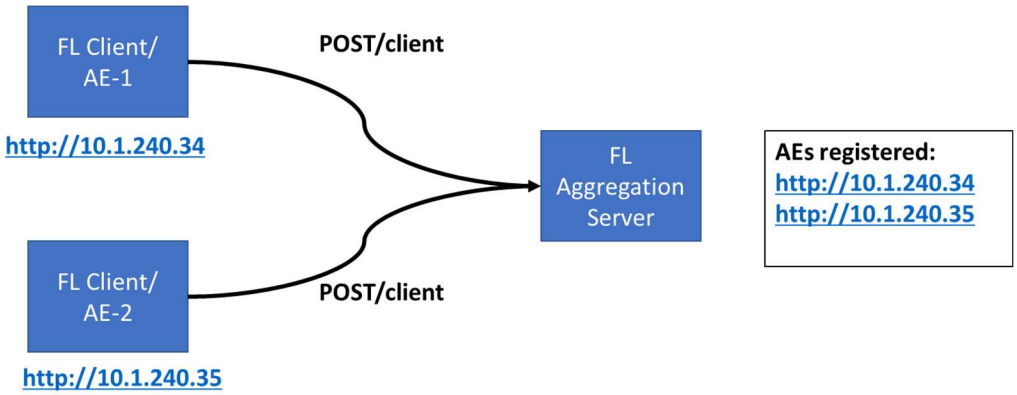

Figure 17. FL AEs/clients register into the network.

When the clients start, they will try to register in the FL network, and from there it will be available for the aggregation server to request for local trainings. There are many clients registered to participate in the FL model training process. Therefore, clients need to know the IP address of the aggregation server. When

nodes are joined into the network, they send a **POST/client** request with their IP address so that aggregation server registers clients in the list of available clients for training the FL model.

STEP 2: START: After registration, server sends a request to all the registered clients to start the client selection process through **POST/select-client** request (as given in Figure 18). Each local MS collects the relevant measured KPIs from RAN (CQI, MIMO Full rank, bitrate) and VR streaming server (CPU). Each local FL client grasps those data.

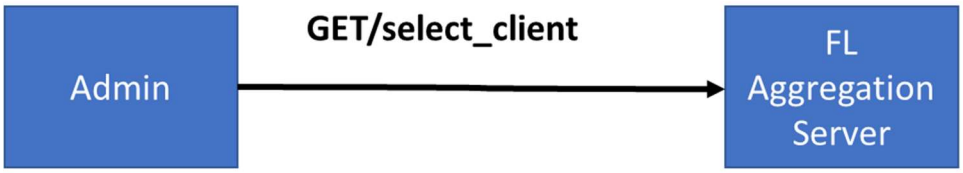

Figure 18. Admin starts the select client process by sending request to FL Aggregation server.

STEP 3: COMPUTE SLAs: All clients compute and send their SLA violation rate to the server through **PUT/SLA & POST/SLA**. First, FL aggregation server requests FL AEs to calculate their SLA violation rates through **PUT/SLA** as given in Figure 19. Later, FL AEs return their SLA violation rates to FL aggregation server through **POST/SLA.**

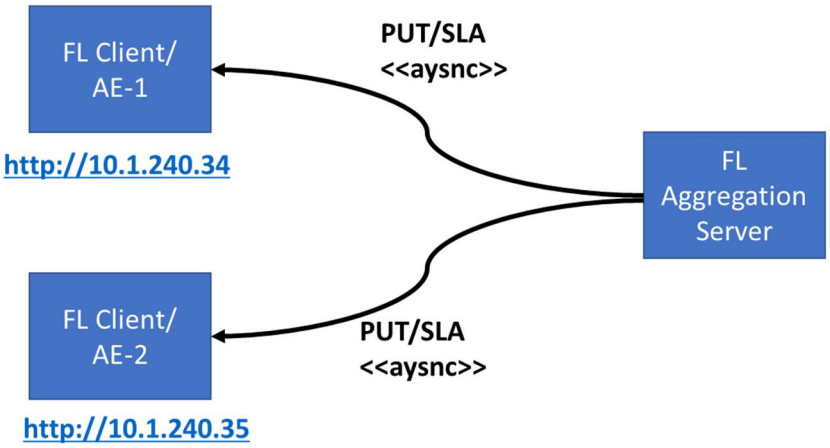

Figure 19. FL aggregation server requests FL AEs to calculate their SLA violation rates.

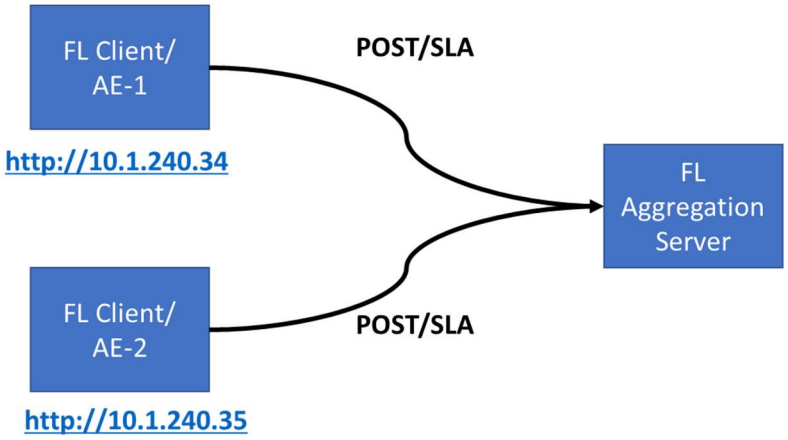

Figure 20. FL AEs return their SLA violation rates to FL Aggregation Server.

STEP 4: COMPUTE PROB. DISTRIBUTION: Server generates the probability distribution of the clients using the softmin function and selected clients using **numpy.random.choice.**

STEPS 5-6: TRAINING: Server sends **POST/training** requests to the selected and registered clients and start FL training as shown in Figure 21 below. This request is made asynchronously through the POST /training endpoint of the FL AEs. Each local FL client builds local models to predict CPU based on local observations from each local MS and server Each using a number of epochs of training predefined by the FL aggregation server. In this request, the averaged model parameters, if any and the type of training to perform (because a FL AE can perform several different trainings depending on availability of different datasets), and the hyperparameters for the training (e.g., the number of epochs, batch size, etc.).

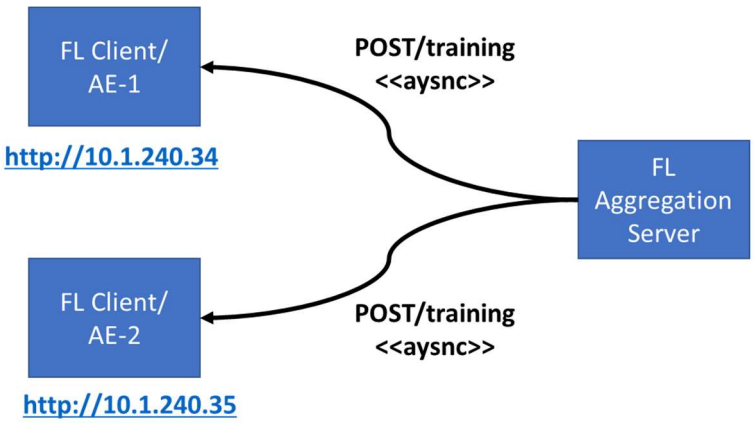

Figure 21. FL aggregation server requests training.

STEPS 7-8-9: COMPUTE & UPDATE WEIGHTS: Model weights of each selected client are sent to the server through **PUT/model-weights (**as shown in Figure 22) when each FL AE finishes the training, and then Server averages the weighs and update the weights of the clients using **PUT/worker-model (**as shown in Figure 23) & repeat same procedure next FL rounds (GO TO STEP 3).

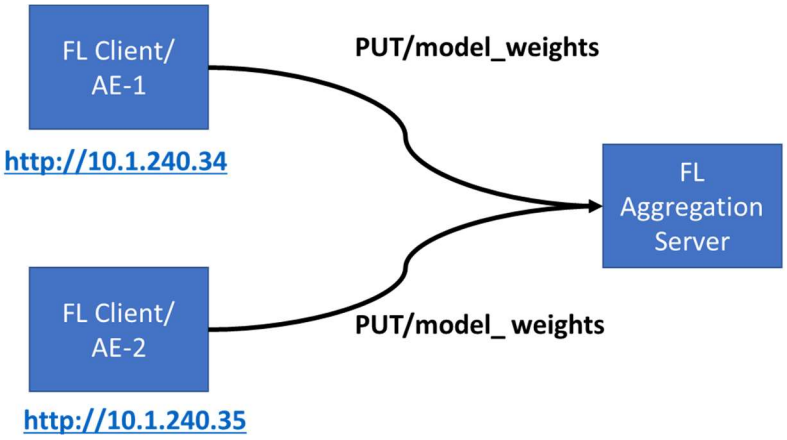

Figure 22. FL AEs send model weights to FL aggregation server.

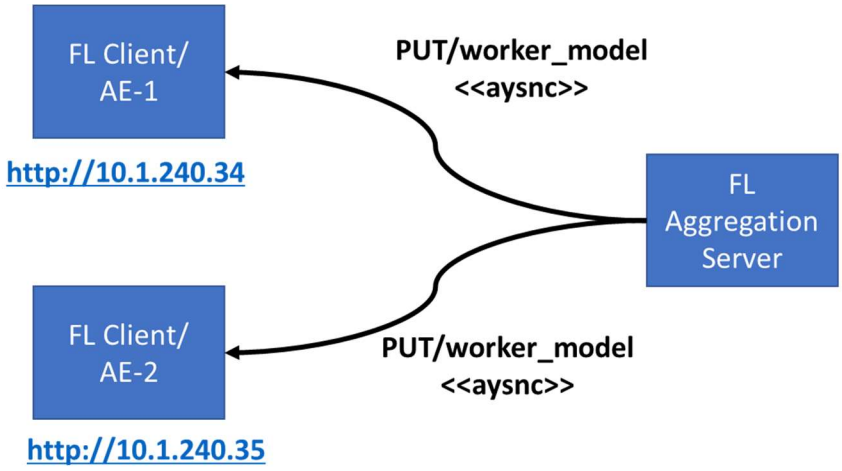

Figure 23. FL aggregation server sends averaged weights to the FL AEs.

Finally, all those rounds are visualized via Kibana dashboard [MonB5GD3.3] and final models are stored in both local nodes inside docker containers.

(2) **Online Inference phase:** Figure 24 shows the offline training phase of the experimental setup for policy-based FL approach

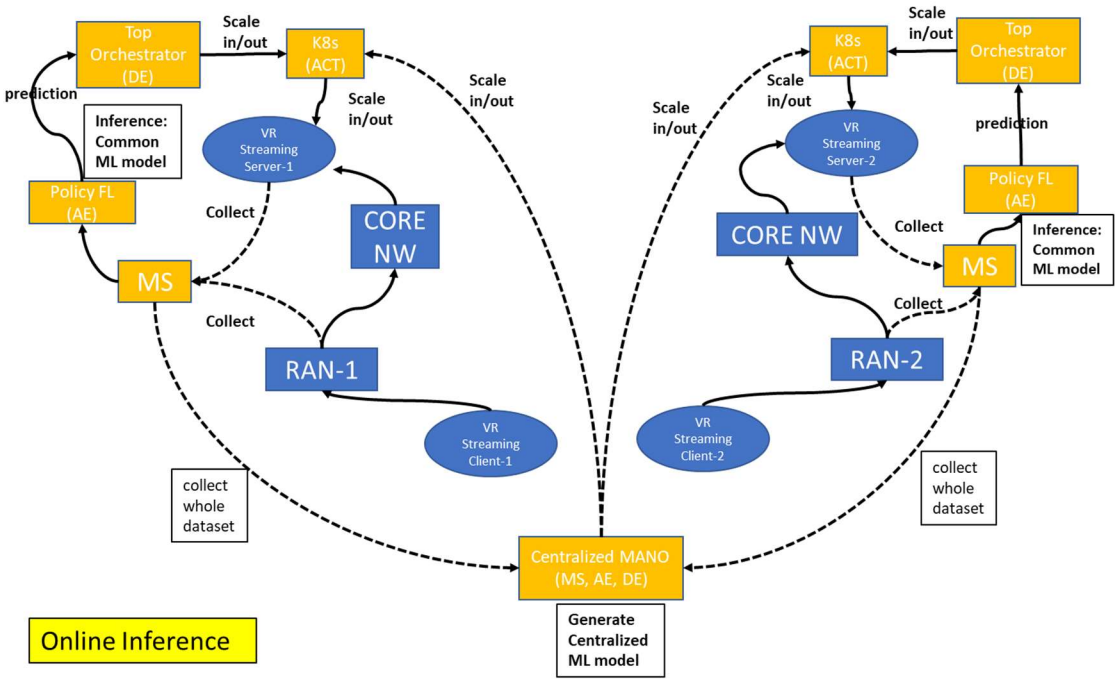

Figure 24. Experiment setup for Policy-based FL approach: Online Inference.

During online inference phase, the process will be initiated by **GET/select_prediction** command by the admin (as show in Figure 25)**.**

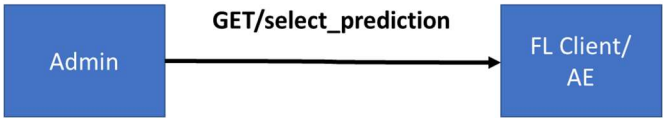

Figure 25. Admin starts the prediction process by sending request to FL AE.

DE based-on threshold-based scaling: Note that in this phase, MS constantly streams relevant data (related to VR streaming and RAN parameters) into each FL client. Then, FL client (which already has the model inside the docker container) makes inferences (e.g., VR streaming pod CPU prediction using the observed real-time RAN KPIs as input parameters). Later, the predicted CPU is transmitted into top orchestrator (DE) which makes the decision to scale-up/down of the VR streaming server based on the predicted CPU. This decision is based on threshold-based scaling (e.g., if the predicted CPU level at next time is above or below a certain threshold, DE will take the decision to scale out/up or scale in/down respectively) K8s orchestrator as an actuator scales up/down the VR streaming server.

5.2.3 *SLICE KPI PREDICTION WITH INTERPRETABLE ANOMALY DETECTION*

We propose a Graph-based Interpretable Anomaly Detection (G5IAD) reference architecture which comes with the following list of contributions. (1) The importance of slice level KPIs predicted by graph-based representation learning method applied in conjunction with recurrent neural networks (RNN). (2) This solution coupled with an interpretable Artificial Intelligence (XAI) - based solution enables a wider adoption of automated management by telecom operators. The proposed framework G5IAD provides explanations (e.g. what combination of KPI values impacted the slice latency KPI) that can be used by the experts and/or by a flexible DE for maintaining slice SLAs, by using direct information from the interpretable method, compared to methods that compute and compare outcomes for all possible actions.

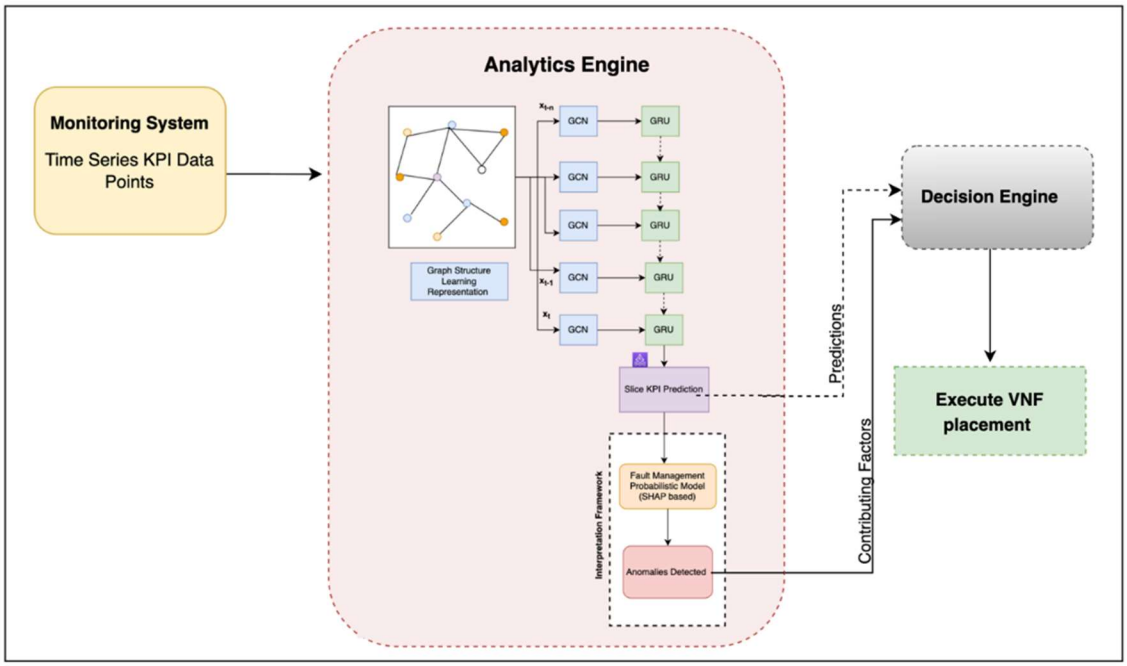

Figure 26. Analytics Engine reference Architecture.

Fault Management has been a fundamental part of network management, included in the FCAPS operations (Fault, Configuration, Accounting, Performance and Security). It aims to detect and eliminate any malfunctions that have occurred in the monitored systems to prevent the degradation of the provided services. In 5G/B5G networks, because of the high degree of flexibility and change in the network, faults must be carefully considered within the particular context in which they appear.

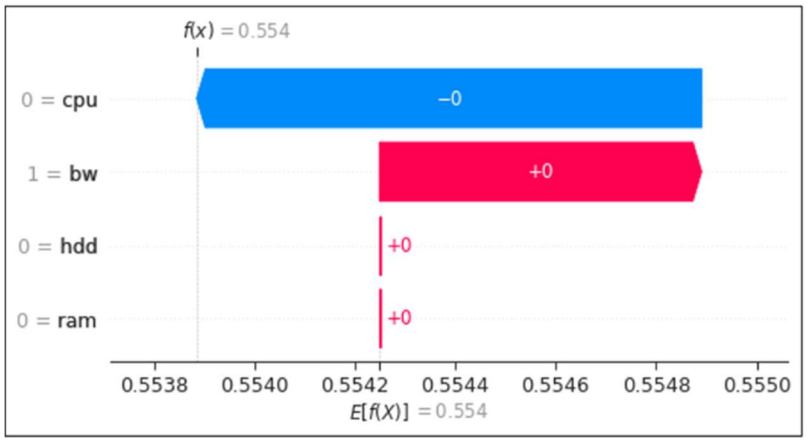

Figure 27. Interpretable Anomaly Detection depicting contributing Factors.

To get a representation of the relationships between features for these anomalous KPI sequences, we obtained the most contributing feature(s) for slice Latency KPI (in our case, the top impacting feature was bandwidth). This information is taken as input by the operators and/or the DE.

The Analytics Engine has been deployed in a container as a Python script. The Analytics Engine can be deployed quickly as a stateless container at any node of the network, eliminating the single point of failure. The Decision Engine is instantiated as a docker container in the cloud, and it is responsible for the VNF orchestration of the main slice. The two modules communicate through a Kafka bus as depicted in Figure 28. The Analytics Engine receives a row of dataset at the predefined time interval of 60 seconds. The Analytics Engine responds to the Decision Engine with a slice KPI prediction at the next interval.

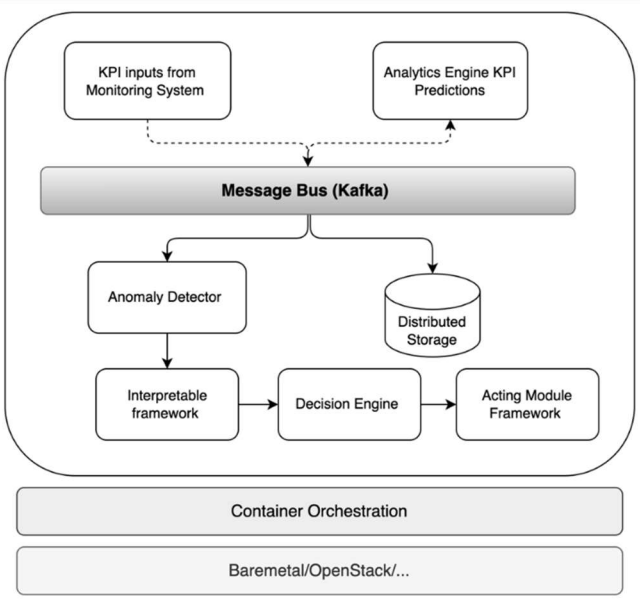

Figure 28. Cloud Implementation of the AE deployment.

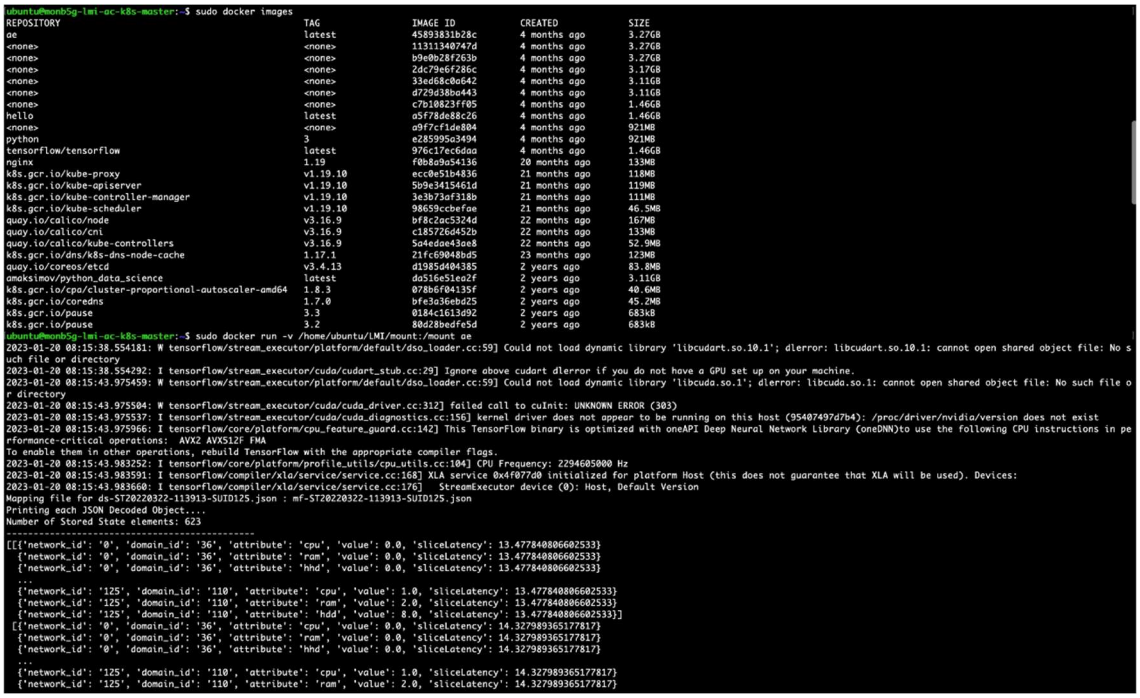

Figure 29. AE deployed and running in the testbed (CTTC).

5.2.4 *LSTM-BASED ANOMALY DETECTION*

Anomaly Detection (AD) enabler uses LSTM to predict the next value of a sample based on the past sequence of data. If the absolute error between the predicted and real sample crosses the threshold value, the latest sample is marked as anomalous. The threshold is adaptive and largely depends on the average error value of a short time window containing latest non-anomalous samples. If the threshold becomes too high, the model is recommended to be updated. This enabler works best at identifying anomalies in data that has a characteristic repeating pattern, such as aggregated slice bandwidth during the day. Anomalies detected in bandwidth data could indicate a fault, an outage, or be used as a forecast of an overall increase in BW usage that day due to a special event (festivities, sports match, etc.).

Anomaly Detection has been deployed in the Kubernetes [Kubernetes] instance of the testbed. Integration was tested with bandwidth sampling function (BWSF) as a data source. Anomaly Detection provides output onto the Kafka message bus and into InfluxDB [InfluxDB] database via Telegraf [Telegraf].

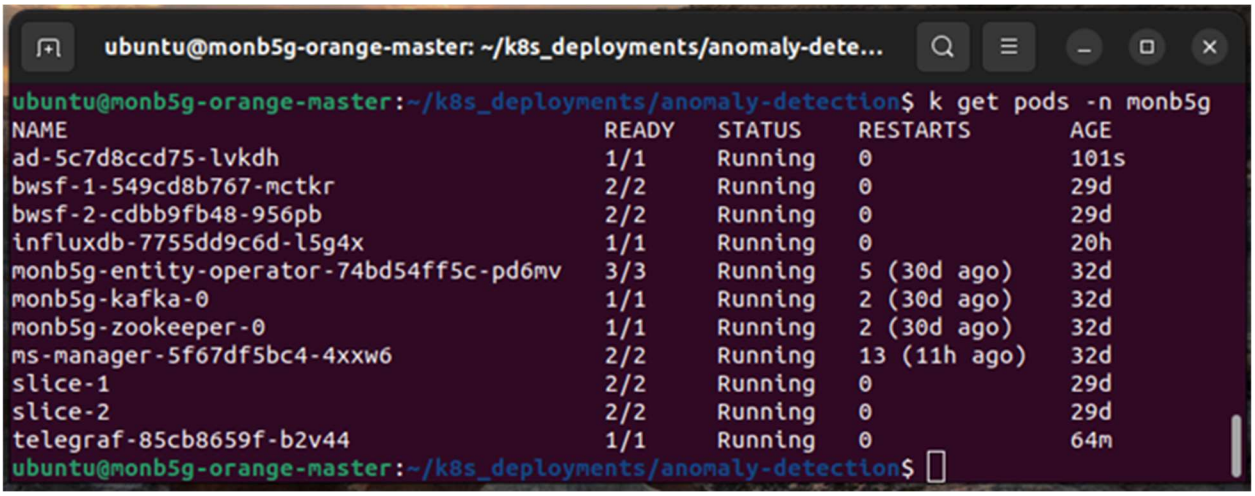

Figure 30. Pods running in the testbed cluster.

Components listed in Figure 30 communicate with each other in accordance with Figure 31.

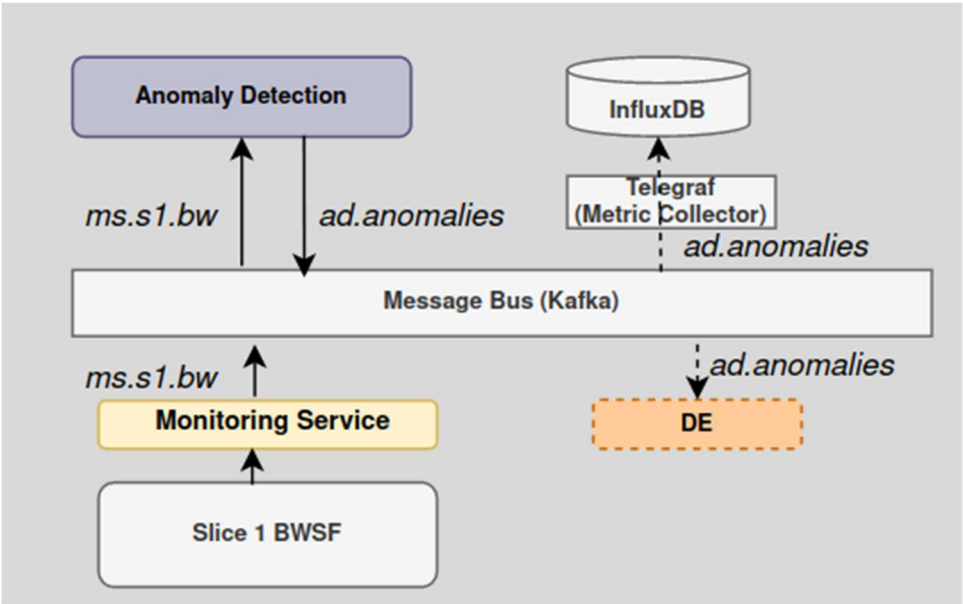

Figure 31. Integration of Anomaly Detection enabler with testbed components.

Messaging between components is performed over message bus (Kafka) via topics shown in Table 4.

Table 4. Kafka Topics Between Components

Message Key	Producer	Consumer	Message content
ms.s1.bw	Monitoring Service	AD	Current aggregate bandwidth consumption in the slice.
ad.anomalies	AD	Telegraf, DE	Metrics related to detected anomalies, i.e. how many anomalous samples in a row, anomaly type

For the purpose of testing component integration, BWSF provided artificially generated bandwidth data, based on which AD's machine learning model was created. AD was tested using undisrupted (normal) data (Figure 32), as well as disrupted data (Figure 33).

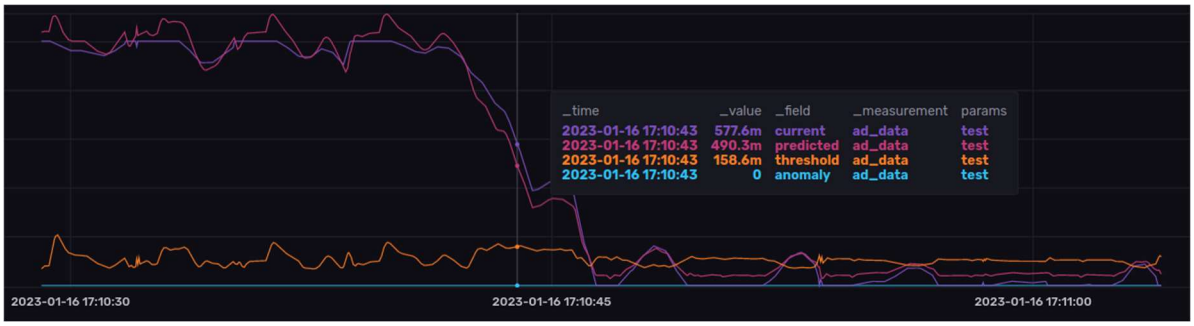

Figure 32. Anomaly Detection output based on undisrupted data and without anomalies.

Figure 33. Anomaly Detection output with injected disrupted data and detected anomalies.

Above pictures show AD's output in InfluxDB data explorer, which indicates that all components are integrated and ready for deployment in more specific scenarios. AD can be easily adapted to use real traffic or any other single-variate metric with the characteristic pattern as a data source.

5.3 Deployment of MonB5G DEs at the Testbed

5.3.1 SLICE ADMISSION CONTROL BASED ON TRAFFIC PREDICTION

The main goal of the Time Aware Slice Admission Control (TASAC) component is to provide the optimal slice admission policy that maximizes resource consumption while taking into account the periodic traffic patterns characteristic for specific groups of slices. In a simplified case, two categories of slices can be distinguished with regard to the daily resource consumption changes, which include Time of Day dependent (such as eMBB slices) and ToD-independent (e.g., mMTC, HMTC). As typically the network users' activity changes significantly during the day, they result in high peak-to-peak traffic values, with maxima and minima occurring in a nearly periodic manner. The TASAC algorithm leverages this tendency by incorporating resource consumption prediction mechanisms in order to maximize the actual resource utilization (corresponding to increased slice requests acceptance rate and resulting operator's profit) while minimizing potential SLA violations [MonB5GD4.2, MonB5GD4.3]. While being able to provide slice admission decisions in real-time, it has to be emphasized that the benefits of the TASAC algorithm can be observed while applied to long-lived slices and a relatively long-time scale (week/month) to enable slice admission policy adjustments of the in-built DRL agent (from the real production network data used during pre-training to the new environments' data). Also, as the time needed for both the agent's training and re-training phases is dependent on the environment's complexity, slice tenants' behaviour and the commonalities of training and production environment, making the agent effective can consume very long periods of time (a few weeks).

To address the above-mentioned issue, a specific approach to integration and deployment has to be taken, as depicted in Figure 34. First, to facilitate and speed up the training phase of the DRL agent, the event-driven simulator implementing the environment (slice tenants issuing slice requests, slice admission/rejection, actual and predicted resources calculations) has been developed. The TASAC DE supports the communication with both the simulated environment as well MonB5G Testbed Facilities. The latter is realised by using the specific connector to the Kafka message bus (used as the basis of Testbed Monitoring Service) and a unified messaging scheme. The ability to operate with both the simulated environment and the MonB5G Testbed allows for pre-training of the ML model, and faster adaptation of the agent's operation to the real-network environment.

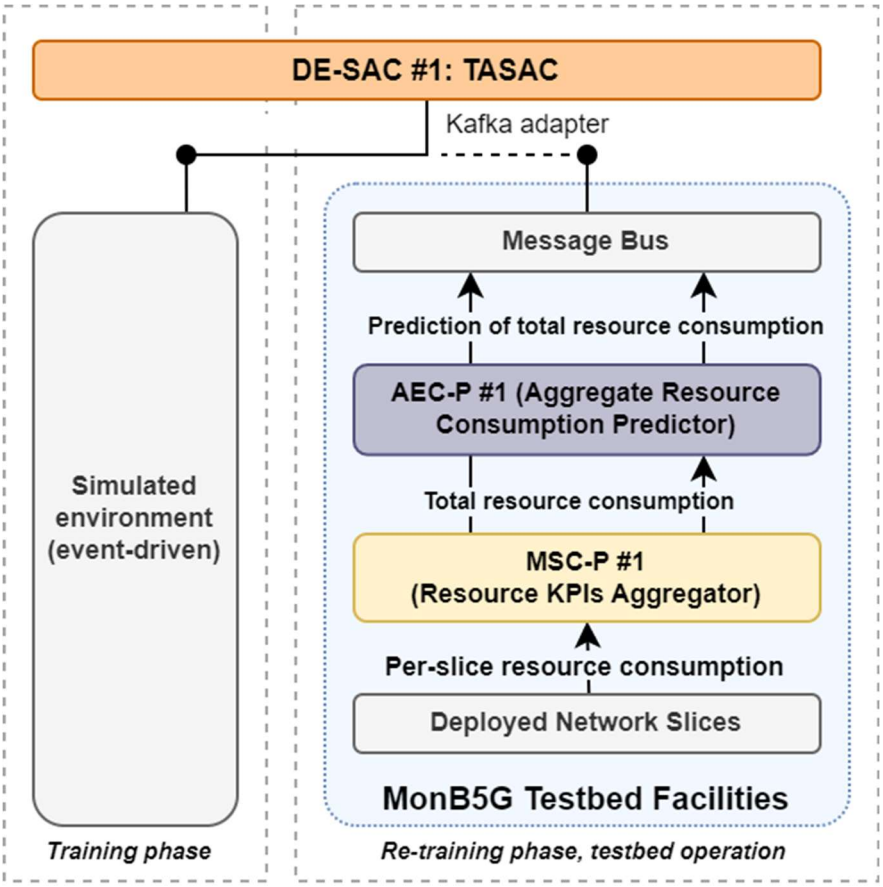

Figure 34. Integration approach of TASAC DE with MonB5G testbed and simulation environment.

As the TASAC enabler can be used as a part of MonB5G layer of DMO/IDMO, the components' integration considers both message exchange over the message bus (MonB5G layer) as well as the internal DMO/IDMO components. The integration for DMO case is depicted in Figure 35 (the IDMO case follows the same principles and involves delegation of slice admission decision to the relevant DMOs selected by IDMO during initial contract negotiation phases). The slice admission process is performed in the following steps:

- The slice tenants issue the slice admission requests, which are transferred from the upper layers to the DMO Slice Admission block over the Iid web-based interface. The generation and issuing slice admission requests are performed by the Slice Requester pod (simulation of tenants behaviour).
- The request is analysed in the Slice Admission block of OSS/BSS part of DMO. In order to admit the slice, several admission engines deployed in MonB5G layer can be used, such as TASAC DE.
- The Slice Admission block issues REST-based request to the TASAC DE. The TASAC DE evaluates the request as described in Deliverable D4.2 [MonB5GD4.2], using the actual resource consumption provided by the Metric Streamer (MS) module, and the resource consumption forecast provided by Resource Consumption Predictor (RCP). The messages exchanged between the modules are described in Table 5. It is assumed that per-slice resource consumption is provided directly to the

message bus by the Functional Layer of DMO (the orchestrator monitors the resources usage of each network slice deployed in the Slice Orchestration Domain) and the MS performs the aggregation in every measurement step.

- The taken admission decision is published to the message bus as well as to the endpoint specified in the slice admission requests (see Deliverable D4.3 [MonB5GD4.3])
- If accepted, the OSS/BSS part of DMO communicates with the Functional Layer via Sc-Or interface and performs the deployment of a slice. The slice metadata is afterwards added to the Deployed Slices Database. If rejected, the respective notification is sent to the tenant via Iid interface.

It has to be emphasized that the MS can also operate in the simulation mode and derive the aggregate resource consumption on the basis of the entries in the Deployed Slices Database. This functionality enables the TASAC concept testing under limited infrastructural resources (without an orchestrator) before its application in real-life network environments.

Table 5. Messages exchanged over the message bus between components implementing the TASAC concept.

Message Key	**Producer**	**Consumer**	**Message content**
ms.ms.total_bw	MS	RCP, TASAC	Current aggregate bandwidth consumption in the domain (for all deployed slices)
ae.rcp.total_bw_pred	RCP	TASAC	The total bandwidth prediction for the next sample (based on the predefined interval)
ae.rcp.api.prediction.req	TASAC	RCP	Prediction requests containing the time range of prediction, unique request id
ae.rcp.api.prediction.res	RCP	TASAC	Prediction response containing the predicted values of the resource consumption in the requested time period together with the unique request id
de.sac.new_slice	TASAC	RCP, MS	The message generated in case of slice admission request acceptance (used by RCP to monitor the list of all accepted slices for the purpose of prediction)

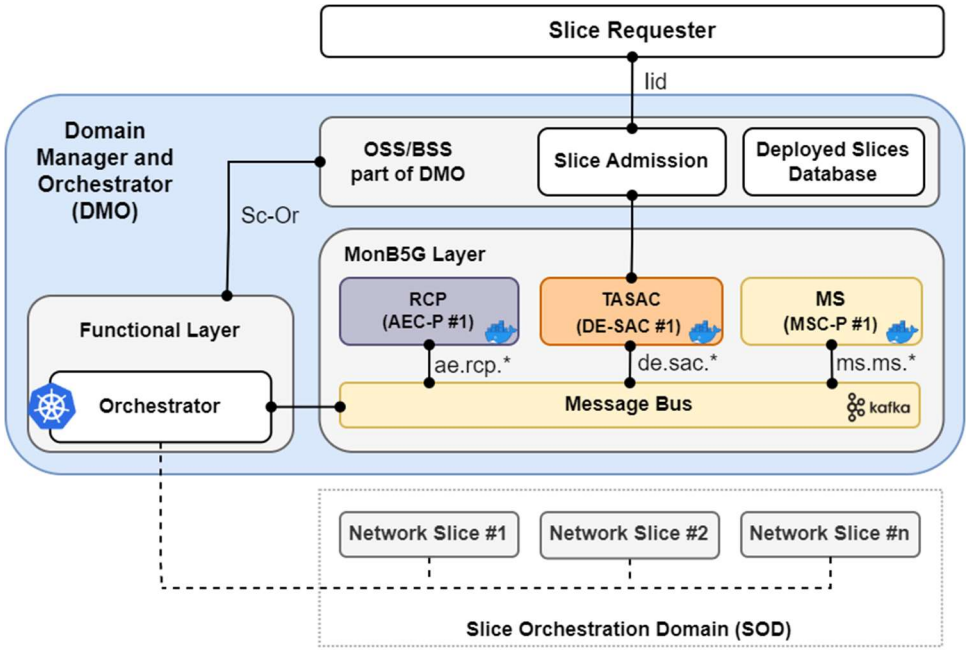

Figure 35. Deployment of TASAC on the MonB5G Testbed.

While the TASAC enabler can be smoothly connected to the other testbed components, the presentation of the benefits of its operation remains problematic. In addition to the long periods of time needed for the agent's adaptation, the concept assumes a large number of arriving network slice requests. Orchestration of vast numbers of slices implies the requirement for the availability of a tremendous amount of resources in the testbed. Therefore, for the purpose of the tests, the hybrid approach in which TASAC DE interacts with a simulated environment deployed in the testbed (pod simulating the actual resources consumption information per slice and pushing the data to Kafka bus). Some of the results of the TASAC DE operation in the testbed are presented in Figure 36 and Figure 37.

Figure 36. Slice Admission Decisions of TASAC DE.

Figure 37. Histogram of admission decisions made by the TASAC DE.

5.3.2 *A MULTI-AGENT LEARNING FOR DISTRIBUTION RESOURCE ALLOCATION IN THE RAN DOMAIN*

We propose a distributed architecture for RAN slice resource orchestration based on DRL, consisting of multiple AI-enabled decision agents that independently take local radio allocation decisions without the need for a centralized control entity. In particular, we deploy the DE described in [MonB5GD4.2], namely multi-agent, which is comprised of multiple independent agents (one per each deployed slice) taking care of the radio resource allocation, i.e., each agent selects the amount of PRB to allocate for accommodating the traffic demand of a network slice without violating the service level agreement. Agents comprising the multi-agent coordinate for preventing the selection of unfeasible actions exceeding the available resources of the RAN node. We design and implement the overall framework as shown in Figure 38.

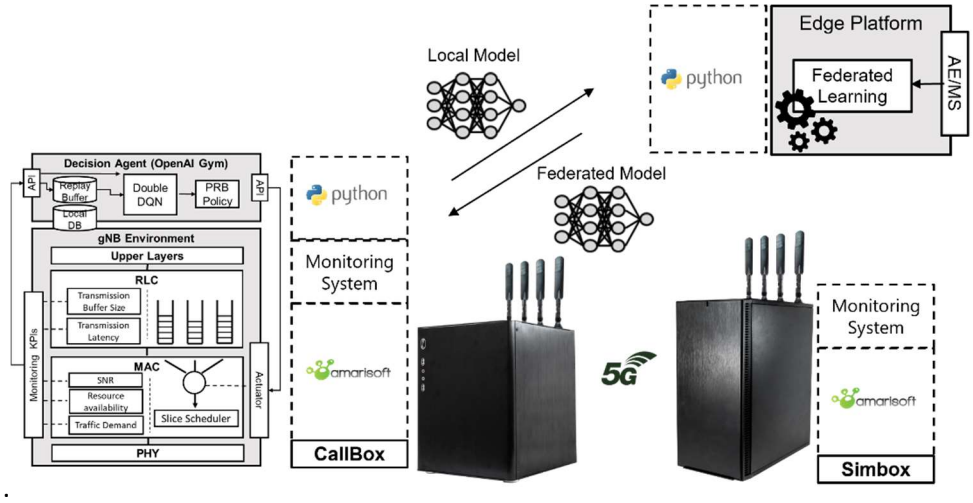

Figure 38. Federated RAN resource allocation architecture overview.

The testbed includes:

AMARI UE Simbox: it is capable of simulating tens of UEs sharing the same spectrum with different types of traffic within multiple cells. Each UE can be independently configured as a 5G NR device, connect to the EPC/5GC, and generate (uplink) or receive (downlink) traffic. In our testbed, once the UEs are configured and connected to the desired gNodeB cell, we generate synthetic downlink traffic traces by means of the Multi-Generator (MGEN) traffic generation tool (https://github.com/USNavalResearchLaboratory/mgen). MGEN allows to modulate several traffic generation parameters, e.g., packet size, packet transmission rate, pattern type, etc., allowing us to obtain realistic traffic patterns to test our solution. An mgen command accounts for a destination IP, destination port, and traffic characteristics like in the following example:

```
./mgen event \"0.0 ON 1 UDP DST 192.168.2.2/5000 BURST [REGULAR 10.0 PERIODIC
[1000 1500] FIXED 5]
```

In this case, the tool generates a downlink traffic towards the IP 192.168.2.2 over port 5000. The traffic accounts for regularly spaced bursts of duration 5 seconds, regularly spaced every 10 seconds. The burst generates 1000 packets/s with each packet of size 1500 bytes.

AMARI Callbox: it provides Enhanced Packet Core (EPC)/5G Core (5GC) functionalities, including authentication of UEs. Moreover, it implements up to 3 gNodeBs cells enabling functional and performance testing. Thanks to its multi-cell configuration, it is also suitable for handover and reselection tests. The technical specifications of the gNB and the core can be found on the vendor's website [AmariCallBox].

Monitoring System: Exploiting a dedicated API hosted by the gNB Callbox instance, it enables real-time monitoring of multiple KPIs, including the number of connected devices, bandwidth utilization, channel quality, etc. The developed DE is integrated to this Kafka cluster MonB5G-MS, and periodically queries it to gather information on the status of the gNB and the UEs. Available information is used to build the status of the local DEs, which will perform radio resource allocation accordingly.

As shown in Figure 39, we also provide an API client that is able to connect to Kafka cluster and obtain the different monitoring parameters of the Amari Callbox platform.

```
BOOTSTRAP_IP   = "10.142.30.189"
BOOTSTRAP_PORT = "31884"

#BOOTSTRAP_IP   = "10.64.116.10"
#BOOTSTRAP_PORT = "31308"
BOOTSTRAP = BOOTSTRAP_IP+":"+BOOTSTRAP_PORT

#This is the list of all Sampler Functions that are deployed in the MonB5G-MS
all_sfs = {"mme.config", "mme.log", "mme.stats", "mme.ue", "mme.erab", "mme.qos", "mme.iq",
                "enb.config", "enb.log", "enb.stats", "enb.ue", "enb.erab", "enb.qos", "enb.iq",
                "ue.config", "ue.log", "ue.stats", "ue.ue",
                "netdata.cpu", "netdata.ram", "netdata.bw"
                }


#If you are not interested in some of those sampled data, add them to the following list
unused_sfs = {"mme.config", "mme.log", "mme.qos", "enb.config", "enb.log", "mme.iq", "enb.iq", "mme.erab", "ue.config", "ue.log"}

exported_sfs = {i: True for i in all_sfs}
for i in unused_sfs:
    exported_sfs[i] = False
```

Figure 39. API client provides the information from Kafka cluster and Callbox.

In order to monitor the interaction of the DE with the network and validate the proposed framework, we design an online dashboard that prompts the temporal evolution of the experiment. In particular, from left to right and from top to bottom, it keeps track of the communication overhead savings, the per-slice traffic demand, the DE reward, the downlink capacity, the PRB allocation decision process, and the resource allocation gap. The dashboard also provides auxiliary information such as the federation episode counter, the number of users connected per slice, the current MCS selected for the communication and e platform CPU utilization, etc. The dashboard is updated in real-time. An example of the designed dashboard monitoring the behaviour of 2 agents during test scenarios is shown in Figure 40.

Figure 40. Monitoring dashboard based on Grafana.

Local Decision Agents: collect and consume local monitoring information from the monitoring system to adjust radio resource allocation policies according to real-time traffic variations. Decision-making is supported by AI algorithms and DRL approaches as detailed in MonB5G WP4 deliverables. In Figure 41 we provide an example of interaction between the DEs and the monitoring system, while in Figure 42 we depict a test interaction of the local DEs with the while performing radio resource allocation decisions over the Amari Callbox platform.

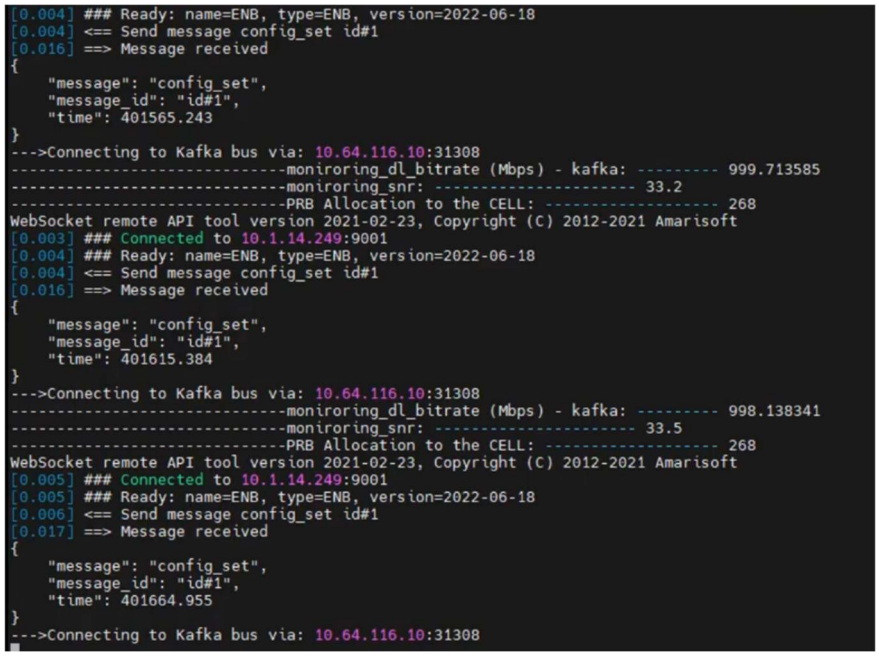

Figure 41. Interaction between local DE and Monitoring System.

In Figure 42 we depict a screenshot taken from the running multi-agent setup.

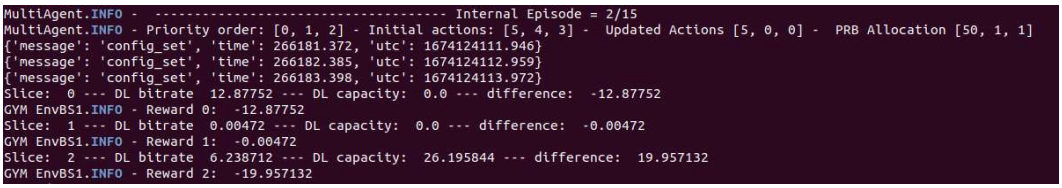

Figure 42. Interaction between local DE and AmariCallbox.

The radio resource allocation decisions performed by the agents are translated into API calls like the following example:

```
./amari_api.js 10.1.14.249:9001 '{"message":"config_set",
"cells": {"1":
                {"pdsch_fixed_rb_alloc":true,
                "pdsch_fixed_rb_start":10,
                "pdsch_fixed_l_crb":30}
                }
            }'
```

The command includes the considered cell id, an allocation flag, and the boundaries of the radio resource allocation bandwidth, including the index of the starting PRB in the PDSCH channel and the number of continuous resource blocks to be allocated. In this example, 30 PRBs are allocated, starting from the PRB number 10 of the radio resource grid of cell 1. The Figure 43 shows the cloud-native deployment of DE and traffic generator, which are deployed and automatically executed exploiting a docker-compose configuration.

```
ubuntu@monb5g-nec-lz-mec-k8s-node:~/monb5g_federated_demo$ docker-compose -f docker-compose.yml up
Starting traffic_generator_3 ... done
Starting traffic_generator_1 ... done
Starting listener            ... done
Starting traffic_generator_2 ... done
Attaching to traffic_generator_3, listener, traffic_generator_2, traffic_generator_1
traffic_generator_1    | WARNING: no logs are available with the 'none' log driver
traffic_generator_2    | WARNING: no logs are available with the 'none' log driver
traffic_generator_3    | WARNING: no logs are available with the 'none' log driver
listener               | {'message': 'ready', 'type': 'ENB', 'name': 'ENB', 'version': '2022-12-16', 'time': 716307.593,
'utc': 1674574238.258}
listener               | ue_get {"message":"ue_get" ,"stats"=true}
listener               | bs_get {"message":"stats" ,"stats"=true}
listener               | 'global'
```

Figure 43. The cloud native deployment of DE API with docker-compose.

Federated Learning Layer: It acts as an aggregation point for the local decision engines deployed at RAN. It collects locally trained (and therefore heterogeneous) decision models and combines them to gain global knowledge about the underlying infrastructure behaviour to improve the generalization of the decision process in the agents.

The overall testbed is depicted in Figure 44.

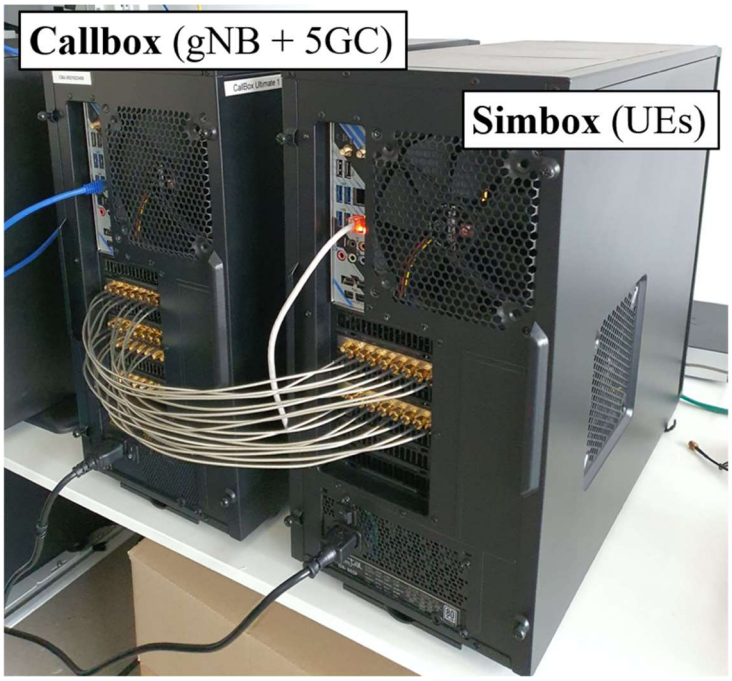

Figure 44. Testbed deployment.

The preliminary integration of the software components developed in the context of WP4 in the MonB5G testbed has been already started. Exploiting the monitoring system API, the RAN agents are able to retrieve real-time monitoring information from the Amarisoft RAN platform, which in turn are used to perform training activities pursuing RAN resource allocation.

5.3.3 RL-BASED SLICE ADMISSION CONTROL

The admission control mechanism in this case relies on the predictions provided by the ECATP module in Section 4.2.1. This mechanism, formally called “Prediction-Based Admission Control” (PreBAC), is an instance of a DE also deployed as a container using Kubernetes/Docker. It is based on RL, and its code has been implemented in Tensorflow 2.1 and Python 3.8. PreBAC is intended to be deployed in the 5G Edge Cloud domain alongside the MS and the containerized deployment of the MS.

PreBAC is originally designed to perform orchestration at the Base Stations of the 5G RAN, since it performs bandwidth reallocation in order to determine the User Service Requests (USRs) that are admitted into service depending on the amount of resources (bandwidth) available to the slice. The amount of bandwidth a slice can vary dynamically, and if the slice does not have enough bandwidth resources to admit the USRs attempting transmission into the BS, the excedent traffic will be rejected, only admitting the traffic that can be serviced with the current resources the slice has. PreBAC determines the strategy used to make bandwidth available to the slices, in order to improve the overall admission rate of the incoming traffic

5.3.3.1 METHODOLOGY FOR INTEGRATION

The containerized version of PreBAC requires only to stream data directly from the MS and the AE. It is not necessary to access the Common Storage Element inside the MS, since the only requirement are to stream current traffic magnitude of the slices towards the DE instance running PreBAC, and the prediction values for the next control cycle from ECATP. Note that ECATP is an instance of an AE, as previously state in Section 4.2.1.

In order to stream the data to PreBAC as it arrives, the containerized deployment of PreBAC has an extended interface that has two subscriptors and an InfluxDB interface to the MS. The extensions to the containerized version of PreBAC can be summarized as follow:

1. A subscriptor to a Kafka topic into which the MS publishes the current traffic values of a slice
2. A subscriptor to a Kafka topic into which ECATP (i.e. an AE instance) publishes the predicted traffic values of a slice for the next control-cycle
3. An InfluxDB interface to gather data for training

The introduction of these subscriptors into PreBAC's extended interface in fact increases the reach of the Kafka bus of the MS all the way to the corresponding DE instance. This is feasible to do in this case, since all three components (MS, ECATP and PreBAC) are deployed in each other's vicinity, all of them in the same 5G Edge Cloud domain. The decisions taken by PreBAC are also enforced locally, and does not require intervention from other DEs in other technological domains, or in the 5G Edge Cloud domains. PreBAC, together with the MS and ECATP, were deployed at the testbed instance deployed at Iquadrat.

In this current integration step, the training of PreBAC is done online, for which the InfluxDB interface of PreBAC is required. It is possible to do the training online in this case, because the traffic load generated from the slices can be controlled for the purposes of functional testing. The containerized version of PreBAC thus includes an InfluxDB interface that it uses to query data for its training, basically using a separate table of the InfluxDB as buffer for state of the BS in the 5G RAN domain, the traffic load, the predicted traffic, the KPIs measured, and the resulting actions of PreBAC. The training is done periodically at 48 control-cycles apart. It is important to note that even though the InfluxDB of the MS' COMS was used as this data store, the training procedure of PreBAC does not impose any temporal constraint on the data used for training.

In many cases, it is not preferrable to train RL agents online for admission control problems, but it is feasible to pursue this methodology for this functional integration of MonB5G. However, for real deployments, it is preferrable to perform the bulk of the training offline once a sound policy has been learned by the RL agent, and very little exploration is needed to achieve near-optimal functionality.

5.3.3.2 METHODOLOGY FOR VALIDATION

In order to validate the effectiveness of PreBAC, the rejection rate and the admission rate of the incoming USRs was evaluated. These two amounts are related to each other, since once is the complement of the other. Thus, it is enough to evaluate just one of them in order to establish numerically the performance of PreBAC. This comparison is made possible since the traffic profiles used for the experiments are known, however PreBAC and ECATP are totally agnostic to them during run-time. By looking at the amount of traffic

a slice can process in a given control-cycle and comparing it to the amount of traffic that the aggregate of USRs have generated in the same control-cycle, it is possible to establish a precise estimation of the traffic that gets admitted and rejected by the BS in the 5G RAN domain.

The information of traffic generated by the slices and the traffic admitted into service due to PreBAC's orchestration decisions are logged in separate files, and analyzed after the experiments have been done. It is possible to analyze the data also in real-time, but this can only be done locally in the 5G RAN domain. In order to visualize the data after the experiments, the logs still remain persistent and they can be visualized at any technological domain of the 5G testbed. Grafana was again used for visualization, both in real-time and after the experiments were done running.

5.3.4 HEURISTICALLY ASSISTED DRL APPROACH FOR NETWORK SLICE PLACEMENT

The Heuristically Assisted DRL (HA-DRL) framework presented in Figure 45, is applied to accelerate and stabilize the convergence of DRL techniques when applied to the Network Slice Placement. The proposed contributions combine the Advantage Actor Critic and a Graph Convolutional Network (GCN) to solve Network Slice Placement optimization problem. We reinforce the DRL learning process by using the P2C based heuristic [10] to control the DRL convergence.

The proposed framework is divided into two main components: the analytics and decision engines. The AE contains the Physical Substrate Network (PSN) database that stores the updated data about the available resources of the PSN. It also contains the Network Slice Placement Request (NSPR) generator that is used to generate NSPR arrivals according to a specific network load regime. It generates slice requests arrivals considering three different network load scenarios: stationary, cycle-stationary, and non-stationary network load scenario. The stationary network load scenario static while the cycle-stationary and non-stationary loads vary in time. The former varies with a predictable periodic load while the latter in a non-predictable change. The NSPR requirements and PSN available resources data are used as inputs to the DRL algorithm by the placement module.

The analytics engine implements the DRL-based algorithms and the Power of two choice (P2C) heuristic algorithm referred to in the following as HEU used by HA-DRL and HA-eDRL algorithms to accelerate convergence. Once the calculation of the Placement decision is done for one NSPR, an update of the available resources in the PSN is made and some key performance metrics are registered in the form of data series; the key metrics are the acceptance ratio of network slices and the resource usage.

Each of the engines is deployed in containers and expose APIs to exchange (Figure 46). The proposed solution can be deployed in a docker-compose or a Kubernetes environment to enable more flexibility.

The DE and AE communicates through APIs. The DE calls the AE API in the following address (http://10.5.0.6:5000/PSN or /NSPR) to get the updated version of the PSN and the NSPR requests. The DE API can be called externally in the (http://10.5.0.6:5000/decision) address to start the learning process or get the results. The data is stored in the dockers volumes. Figure 47 illustrates the results from calling the decision API to start the learning process.

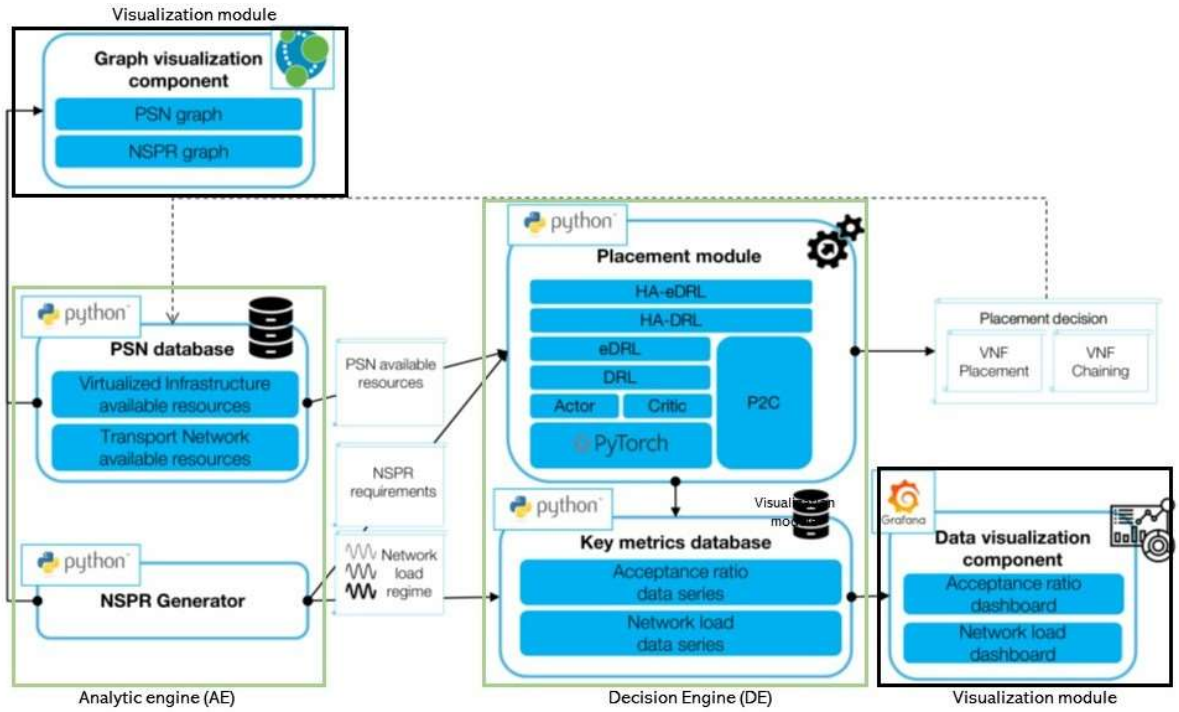

Figure 45. The DRL-HA deployment frameworks.

```
Toutes les 2,0s: kubectl get all                          cherrared-XPS-9320: Mon Jan 30

NAME                                        READY   STATUS    RESTARTS        AGE
pod/analytic-engine-65758d75b4-qv6h4        1/1     Running   1 (8m23s ago)   25h
pod/decision-engine-686fcc8b7d-bc49d        1/1     Running   1 (8m23s ago)   25h

NAME                         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/analytic-service     ClusterIP   10.101.91.209    <none>        80/TCP     25h
service/decision-service     ClusterIP   10.106.254.225   <none>        80/TCP     25h

NAME                                   READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/analytic-engine        1/1     1            1           25h
deployment.apps/decision-engine        1/1     1            1           25h

NAME                                              DESIRED   CURRENT   READY   AGE
replicaset.apps/analytic-engine-65758d75b4        1         1         1       25h
replicaset.apps/decision-engine-686fcc8b7d        1         1         1       25h
```

Figure 46. The cloud native deployment of analytic and decision engines API with Kubernetes.

```
==================================== Request  3
Arrival of Request  3  of type  0  with  5  nodes
--|Time Step:   10      |Request id:    3       |Request type:  volatile        |Current VNF:   0    |--
Placing VNF  0
======================================== Current Action tensor(75)
======================================== Current Action success True
======================================== Testing placement in node id: 75
======================================== Testing placement of VNF 0
======================================== Current Action path []
======================================== Current Action used bandwith 0
--|Time Step:   11      |Request id:    3       |Request type:  volatile        |Current VNF:   1    |--
Placing VNF  1
======================================== Current Action tensor(29)
INFO:root:======================================== Current Action success: True
INFO:root:======================================== Testing placement in node id: 29
INFO:root:======================================== Testing placement of VNF: 0
INFO:root:======================================== Current Action path: [(75, 2), (2, 22), (22, 0), (0, 21), (
```

Figure 47. The results of calling the DE API to train the model.

5.4 Deployment of MonB5G Actuators

Some of the MonB5G ACT that are available to be used in the above MonB5G enablers are as follows:

5.4.1 CPU AUTOSCALING AT POD LEVEL

The first actuator, as a result of the interaction between the orchestrator and the Kubernetes scheduler, is the scaling of the CPU resources allocated to the pod hosting the Video-on-Demand (VoD) server. The orchestrator, through the federated client, continuously monitors the CPU usage predictions of the VoD server. If the predicted CPU usage exceeds a predefined safety threshold, the orchestrator will send a signal to the Kubernetes scheduler to increase the CPU resources allocated to the pod hosting the VoD server. This is known as scaling up the resources. In contrast, if the predicted CPU usage falls below a certain threshold, the orchestrator will signal the Kubernetes scheduler to decrease the CPU resources allocated to the pod, this is known as scaling down the resources.

This mechanism of scaling up and down the CPU resources of the pod hosting the VoD server is known as autoscaling. Autoscaling allows the system to automatically adjust the resources allocated to the pod based on the current workload, ensuring that the system is always operating at optimal performance while avoiding unnecessary resource wastage.

5.4.2 APPLICATION/POD MIGRATION

The second actuator, as a result of the interaction between the orchestrator and the Kubernetes scheduler, is the relocation of the Video-on-Demand (VoD) server pod to a Kubernetes node that could have lower overall CPU usage or improves the latency between the user and the VoD server. The orchestrator, through the federated client, may continuously monitor the CPU usage and latency of all the nodes in the cluster. If the orchestrator detects that a node has lower overall CPU usage or a better network connection to the user, it will signal the Kubernetes scheduler to relocate the VoD server pod to that node.

This mechanism of relocating the pod to a more suitable node is known as pod migration. Pod migration allows the system to improve the performance of the VoD server by moving it to a node with lower CPU usage or better network connectivity, which can lead to a better user experience and reduced latency.

5.4.3 BANDWIDTH MANAGEMENT

The third actuator, as a result of the interaction between the orchestrator and the 5G RAN, is the modification of the bandwidth allocation to one or multiple cells in the 5G RAN. The orchestrator, through the federated client, uses Amarisoft Callbox API to continuously monitor the bandwidth usage and signal quality of all the cells in the RAN in real-time. If the orchestrator detects that a particular cell is experiencing high traffic or poor signal quality, it will signal the 5G RAN, using the Amarisoft Callbox API, to adjust the bandwidth allocation to that cell by modifying the Physical Resource Blocks (PRBs) in real-time. This can be done by increasing or decreasing the amount of bandwidth allocated to that cell or by redistributing the bandwidth among multiple cells in the RAN.

This mechanism of modifying the bandwidth allocation to cells in the 5G RAN is known as bandwidth management. Bandwidth management allows the system to improve the performance of the RAN by adjusting the allocation of bandwidth to cells based on their current usage and signal quality, which can lead to a better user experience and reduced congestion.

5.4.4 HANDOVER ACTUATOR

Lastly, the handover actuator is a tool that targets efficient management of resources for providing a high-quality service to users by automating the process of handover between cells. The actuator, written in Python, functions as an intermediary between the gNBs and the decision engine. It uses the Amarisoft API to monitor the signal strength and quality of the connection between a UE and its current serving gNB.

When the signal strength or quality drops below a predefined threshold, the actuator may trigger a handover to a neighbouring cell with a stronger signal, either in the same or different gNBs. This ensures that the UE maintains a stable and high-quality connection and allows for a more efficient use of resources by avoiding overburdening any one gNBs. The Amarisoft API also provides means to control handover's parameters, such as triggering conditions and measurement reports, which can be fine-tuned by the actuator to optimize the handover process.

6 Integrated Deployment, Completion and Evaluation at the Testbeds for PoC-2

6.1 Deployment of MonB5G MS at the PoC-2

6.1.1 SCENARIO MMTC ATTACK

We have used a 5G testbed deployed at EURECOM. The testbed has been developed and used in many 5G European projects such as 5GEve[1] and 5G!Drones[2]. We have implemented the closed-control loop components (i.e., MS, AE, and DE) and an Element Manager (EM) on top of the AMF. The latter exposes API to 1/ MS to monitor the Attach Request message; 2/ DE to detach and blacklist UEs involved in an attack.

Figure 48 and Figure 49 highlight the interaction among the different actors involved in detecting and mitigating DDoS attacks: the mMTC network slice components (UEs and 5G CN) and the closed-control loop elements (MS, AE, and DE). It is worth noting that the closed-control loop runs in parallel to the mMTC network slice elements and only monitors Attach Requests to detect and mitigate attacks.

In the considered scenario, the MTC devices (or UEs), when detecting an event or participating in an attack, first send an Attach Request to AMF. The latter must first authenticate the devices and then give them access to the network resources (register the device), mainly to the data plane, to send data to the remote application. During the authentication process, the AMF checks with the UDM if a device is blacklisted or not. To recall, UDM is the 5G CN function, which stores subscribers' information (Subscriber Permanent Identifier -SUPI - Quality of Service -QoS- Policy, the key k, Operator key, etc.). A device is blacklisted if it has participated in an attack.

[1] https://www.5g-eve.eu/, Available Online: February 2023

[2] https://5gdrones.eu/, Available Online: February 2023

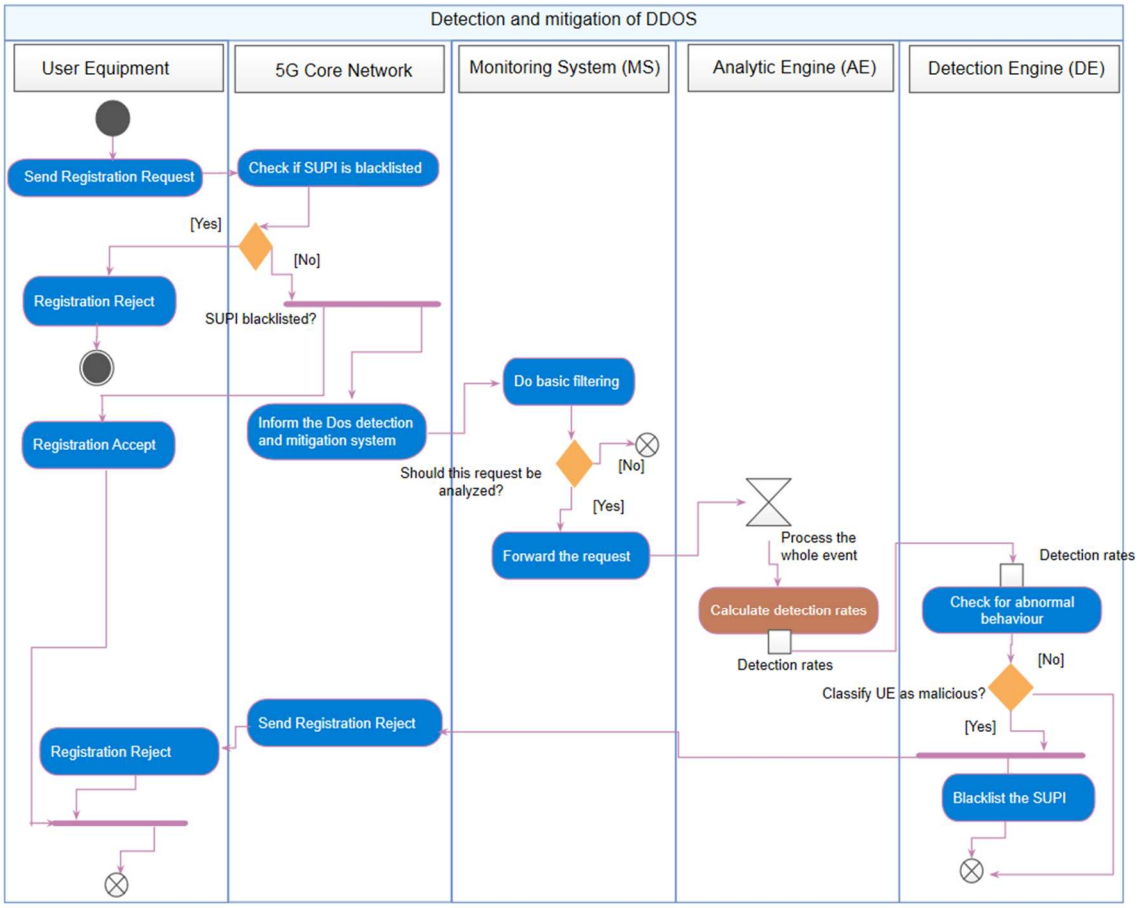

Figure 48. Interaction of the mMTC network slice and the closed-control loop to detect and mitigate attacks.

Meanwhile, MS, via the EM/AMF API, monitors the Attach Requests received by AMF. MS filters the data to extract the needed information, such as timestamps and SUPI. This information is communicated to AE that processes the whole event (attach period that may correspond to an attack) in order to classify if the event corresponds to an attack or not. When the event finishes, AE communicates the list of involved UEs; for each UE, a probability of being part of the attack is included. DE then mitigates the attack by requesting AMF to send a Registration Reject message to UEs having a high probability of being in the attack while adding the concerned UEs to a blacklist maintained by the UDM.

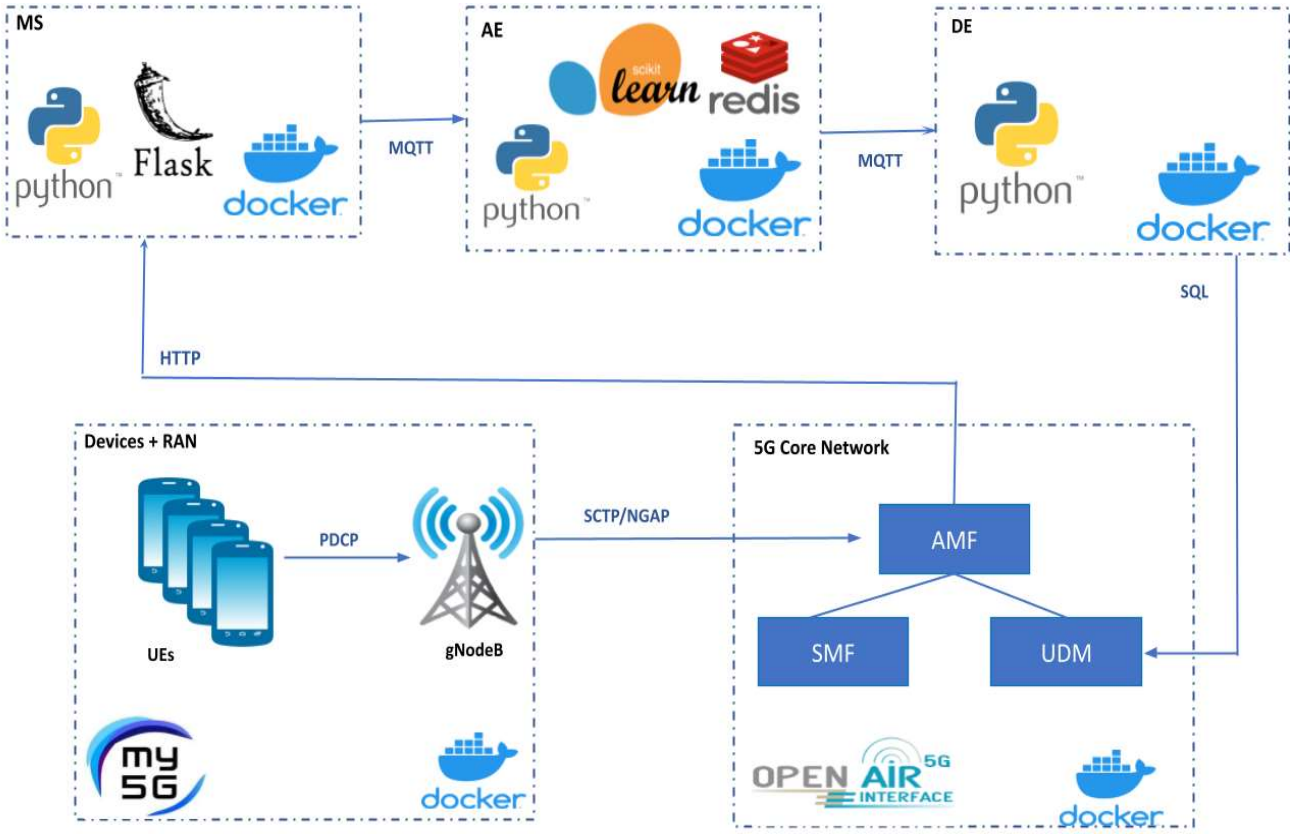

Figure 49. Test platform and technological components.

MS collects data from AMF on every Attach Request received from the MTC devices (Figure 50). This data is accessible through the API exposed by EM of AMF. For each Attach Request, MS extracts the device identifier SUPI and a precise timestamp. Indeed, in the 5G protocol, each UE is identified with a unique identifier called SUPI. The latter is encrypted to provide better privacy and prevent the IMSI catcher attacks that were popular on previous-generation telecommunication protocols. The SUPI should not be transferred in clear text over the RAN except routing information, e.g., Mobile Country Code (MCC) and Mobile Network Code (MNC). For this reason, it is challenging to identify devices from the traffic received on the radio layer; hence our solution has to intervene at the 5G CN (i.e., AMF), which can decrypt the SUPI information. The extracted information is then forwarded to AE via a communication bus based on the Publish/Subscribe concept.

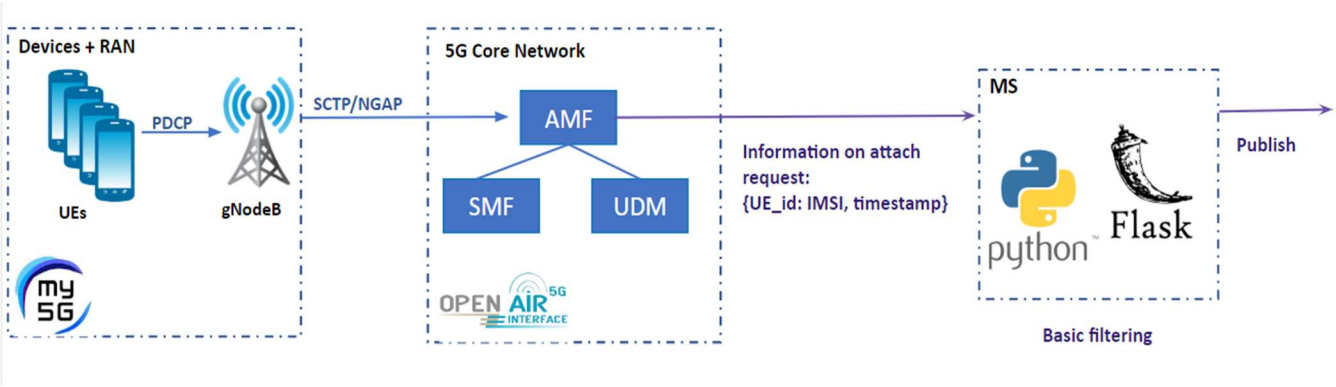

Figure 50. MS's components.

6.1.2 *SCENARIO FL ATTACK*

As depicted in Figure 51, we consider n running network slices that may be initiated by different vertical industries, such as intelligent transportation, Industrial IoT, and eHealth verticals. The running network slices are interconnected to an IDSM, which is in charge for the management and orchestration of network slices. To enable ZSM, the IDSM side includes an AE for building learning models and a DE, to make suitable decisions based on AE's outputs. On the other side, each running network slice is managed locally by a DSM, which also includes a MS for monitoring data and in-slice traffic, and an AE for building learning models.

The proposed framework enables to secure federated learning in B5G networks, against poisoning attacks, named TQFL for "Trust deep Q-learning Federated Learning". The design of our framework comprises four main steps, starting from generating a realistic dataset to designing a detection scheme of poisoning attacks: (i) The generation of a realistic dataset about the AMF function's latency of running network slices and its (latency) related parameters. (ii) Building a deep learning model to predict the AMF function's latency of each running network slice in a federated way in order to prevent any latency-related SLA violation. (iii) Building an online DRL model that dynamically selects a network slice as a trusted participant (see step 1). (iv) After the first FL rounds (see steps 2, 3), the trusted participant applies a dimensionality reduction scheme and unsupervised machine/deep learning to detect the malicious participant (s) (see steps 4, 5, 6,7).

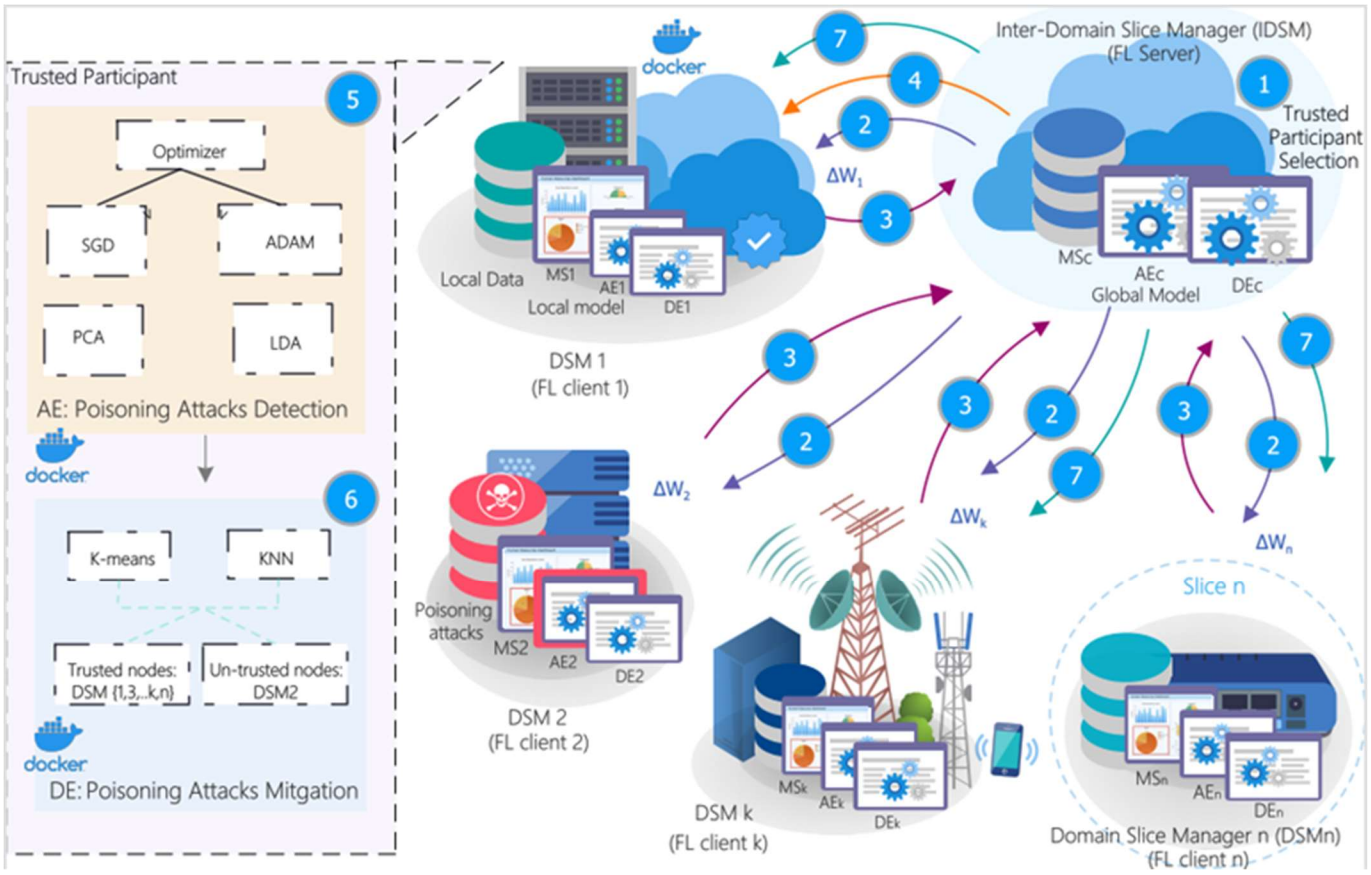

Figure 51. Overview of TQFL Framework.

Each FL rounds, the MS receives the n model updates of network slices from the central IDSM. The updates of the FL clients are weight matrix with n dimensions.

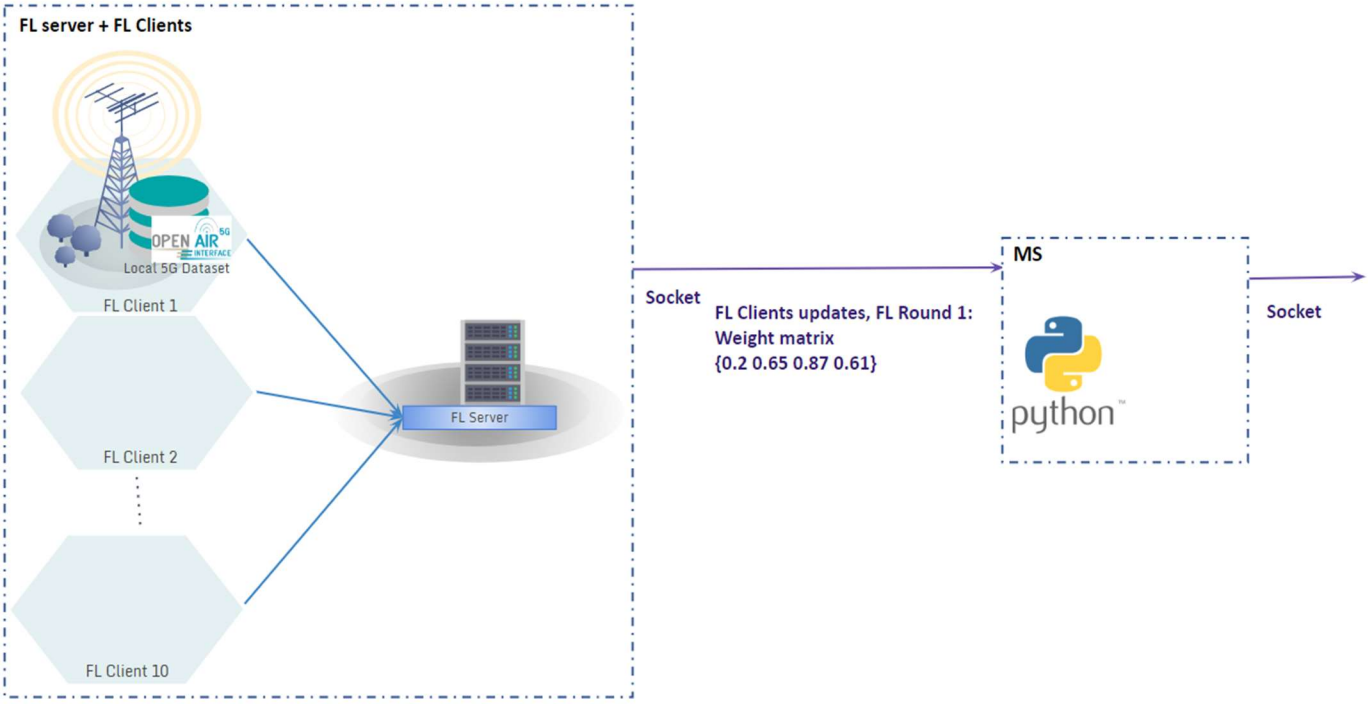

Figure 52. MS's components.

6.1.3 SCENARIO ALTER ATTACK

As described in D5.4, a subset of the functions defined in the incident handling guidelines from (NIST.SP.800-61-r2) have been implemented to detect the security incident of an aLTEr attack and minimize its impacts. The implementation leverages the MonB5G components MS, AE, DE, ACT and its architecture proposed in D2.4 to orchestrate security services. For this scenario, we have to observe IP flows of PDU session, therefore the MS includes the three below sub functions:

- The extraction based on port mirroring, also known as SPAN, is a method of monitoring network traffic by transmitting a copy of each packet entering and/or leaving the N6 interface of an UPF instance to a destination port where raw data are processed.
- The mirrored data are loaded and made available on a VXLAN interface for the transformation
- The transformation module based on the COTS Zeek to translate raw IP flows into logs of high-level protocols (HTTP, DNS …). The outcome logs contained the synthetic information required for the core security function Threat Detection

As one of the steps of the attack is to modify the packet containing a domain name resolution request, a control element is added to the extraction to filter the data to be copied and sent. For this scenario, only DNS packets are extracted from the data plane, thus the transformation is not burdened by any unnecessary data.

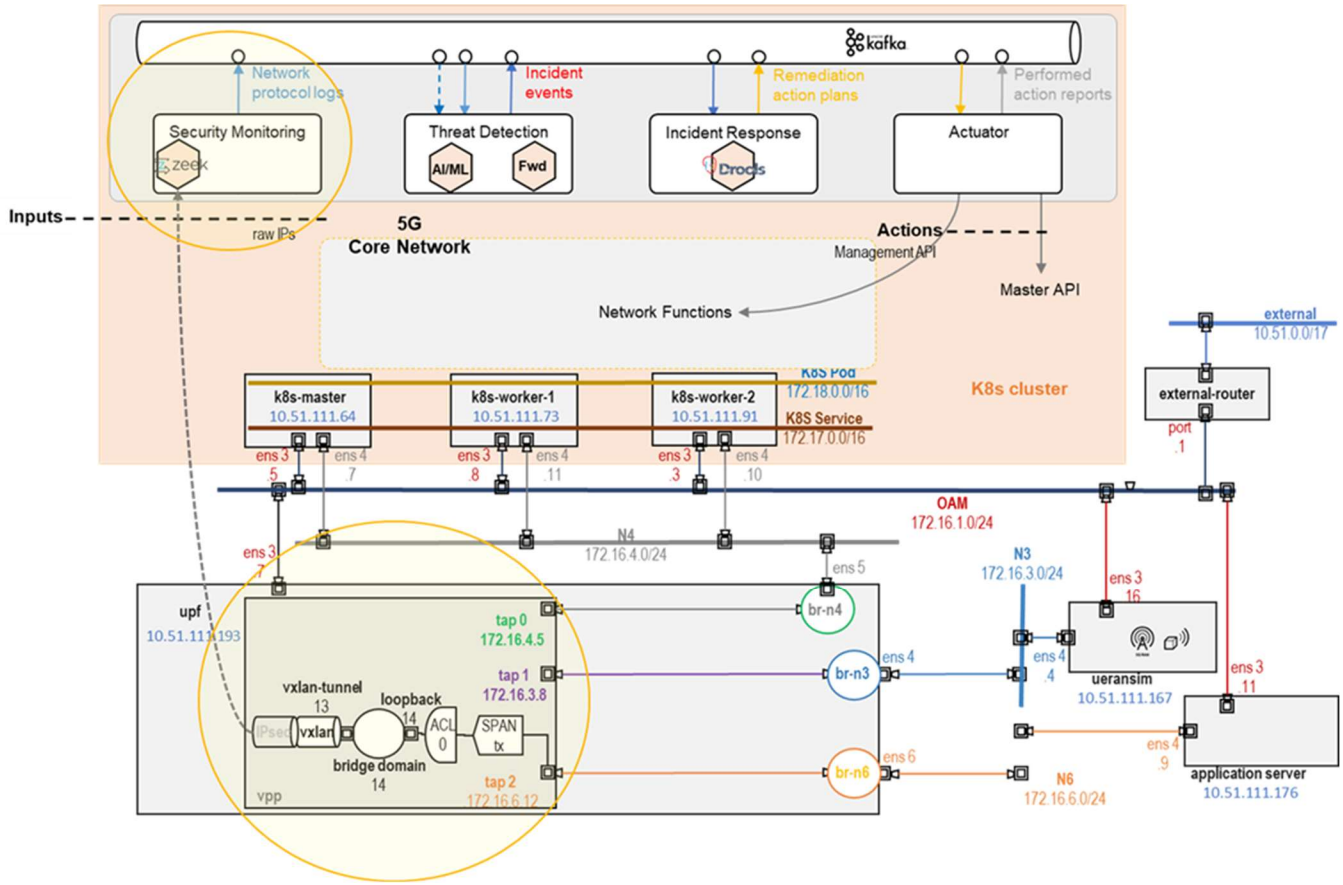

Figure 53. The integration of the MS for the scenario aLTEr.

Figure 53 presents the integration of the MS components to the 5G core network to observe user data at the N6 interface. The collected data transmitted over the VXLAN tunnel can be secured by means of IPsec or MACsec security protocols.

6.2 Deployment of MonB5G AEs at the PoC-2

6.2.1 *GRADIENT BOOSTING INTERVAL REGRESSION*

Figure 54 illustrates a detailed vision of the AE components, which are: Sampler, Activity Detector, DataBase (DB), Event Detector, and Analysis components. They interact together to: (1) detect when an event starts and ends. It can correspond to MTC devices report (normal traffic) or attacks; (2) analyse the event to detect if it is normal or abnormal traffic; (3) compute the detection rate for each device (probability that a device has participated in the attack) and send a report to DE. We decided to separate the event detection from event analysis to improve performances and consider all the relevant data when running the overall attack detection algorithm. Indeed, we decided to detect activity periods (i.e., events) in the network traffic and

only feed data to the ML algorithm at the end of an event, which provides the advantage that the resource-intensive component (detection analysis) runs once every event. We also consider two corner cases: (1) after a duration clearly greater than the maximum length of an event; (2) when peak traffic exceeds a limit indicating that it is definitely an attack. For both cases, we tag the devices as malicious.

The only downside of separating event detection from the analysis is that UEs participating in an attack will not get banned right away when the attack starts. However, since the damage in DDoS attacks stems from their duration in time, the devices will get disconnected and blacklisted a few seconds or minutes after the event starts by DE. The detection algorithm does not need to run in real-time, and it can look at data of the whole event.

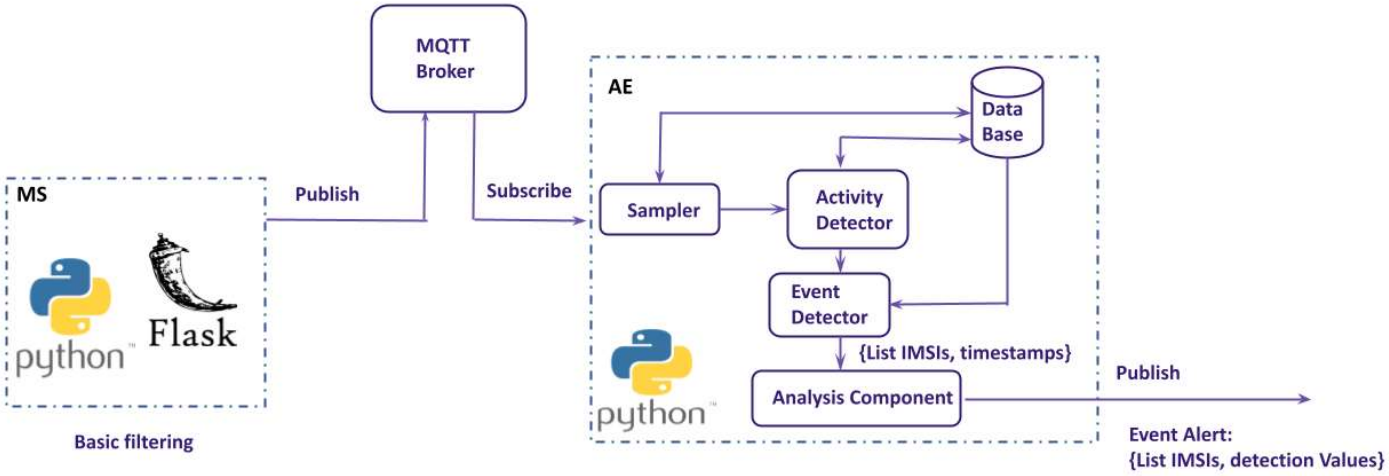

Figure 54. AE's components.

a. Event detection

To detect activity periods, we first calculate the rate of Attach Request. To this end, we devise a new component called the Sampler, which receives data that reaches AE from MS and emits data periodically by grouping the Attach Requests in time intervals of a fixed length. Figure 55 illustrates how the Sampler works.

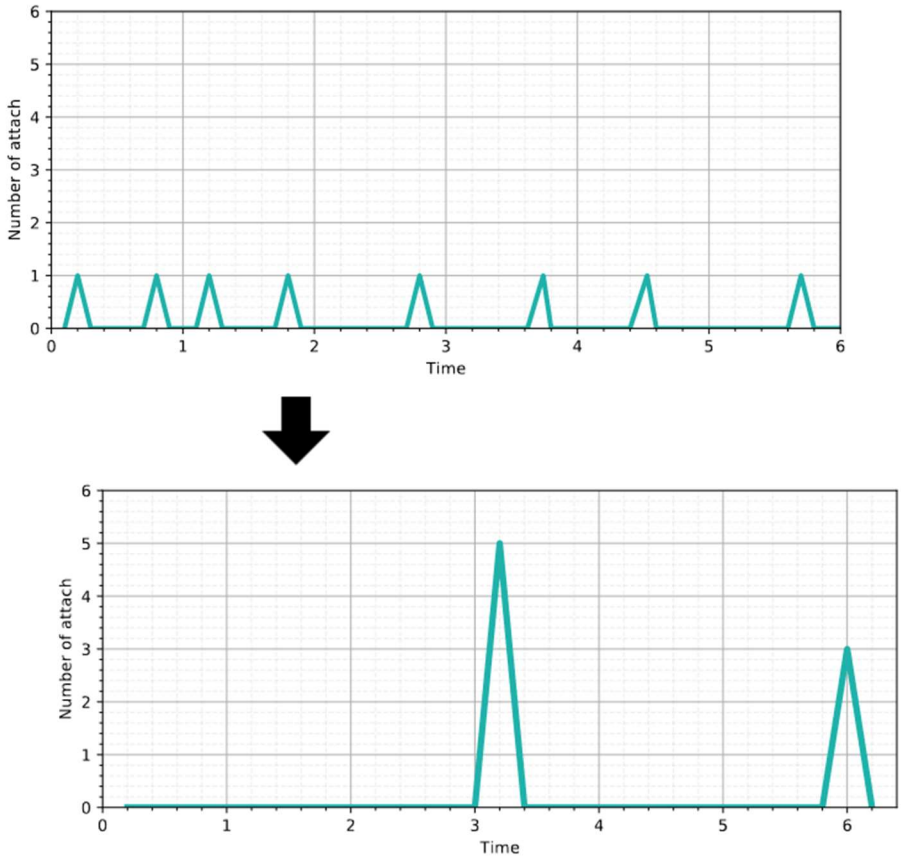

Figure 55. Sampler: attach requests on time intervals of a fixed length.

The upper part of the figure shows the real Attach Request timestamp reception, while the down part of the figure shows the output of the sampler that groups the Attach Request every 3 seconds. The Sampler outputs allow detecting the start and end of an event. If there is traffic, and we are not already in an event, we consider that an event has started. If we are in an event and detect that the system has not received traffic for a given duration, we assume that the event has ended. We use a DB for storing all the data relevant to the event.

To have a modular system, we separate the Activity Detector (the part that detects whether there is traffic or not) and the Event Detector (the part that delimits an event's start and end timestamps).

As stated earlier, a DB is used to store information about the event, namely the number of Attach Requests received for each event, a list of SUPI values that identify the devices that emitted each Attach Request, and timestamps. The DB is accessible and used by: (1) the Sampler to store pairs (timestamp, supi), retrieve the largest timestamp stored, retrieve all the data stored, and delete all the data stored; (2) Activity and Event Detectors to store and edit a boolean value *is_event*. It is worth noting that a relational database is not an

optimal choice in terms of the database model. A better choice is a Time Series DB, as we will store time events. After many considerations, we finally opted for a key-value database. Indeed, our processing of timestamps will still be efficient, as we will be using a sorted set, and we will be able to store the boolean value needed by the Activity Detector. This avoids the need for multiple DBs inside AE.

The design of the database is highlighted in Table 6.

Table 6. The design of our key-value database.

Key	Type	Used By
devices	**sorted set**	Sampler
is_event	**string**	Activity and Event Detector
last_attach_requests	**sorted set**	Activity and Event Detector

b. Event analysis: ML Algorithm

The core component of AE is the Analysis Component, which receives data about one event, including the SUPI identifiers of all the devices that sent an Attach Request; then, it calculates a percentage for each device being part of an attack. The Analysis Component output (i.e., a percentage) should be zero ideally for normal traffic, as no abnormal behavior is present. When some devices misbehave and send Attach Requests that clearly do not correspond to normal traffic, the Analysis Component output should be higher than 0. The detection rate should increase as the traffic rate increases above the normal level. We also want high detection rates (preferably 100%) for traffic likely to cause a DDoS attack.

Many algorithms can be applied to solve this problem. However, not all of them can be used as we have two important considerations:

- We already have a model for our data, the b(3,4) probability distribution, but the goal of ML algorithms is usually to estimate a function we do not know from its inputs and sometimes its outputs (in the case of supervised learning).
- New equipment can be introduced so that the event lengths can vary. Hence, the envisioned algorithm should not detect these changes as anomalies.

While malicious traffic that triggers DDoS is easily discernible from normal traffic, it is hard to evaluate the detection rates for anomalies that are not blatant attacks (i.e., when the traffic rate is just a bit above the prediction interval bound, for example).

We discarded Deep Learning based algorithms as learning from data is not something we want. It would break our second consideration, and the change in the number of connected devices could trigger wrong predictions. Further, we believe that a neural network trained on normal traffic will not give a gradually higher detection probability on outliers because it does not consider outliers as elements that are more

distant from normal data. It instead works with the combination of linear and non-linear mathematical operations.

We considered algorithms that can calculate prediction intervals, which are intervals that likely include our data. Indeed, if we can get an interval that includes our data, we can use the distance from its upper bound to calculate a prediction percentage. The most popular ML algorithm for calculating prediction intervals is Gradient Boosting. It is an extension of Boosting, where the additive generation of weak models is based on the gradient descent algorithm over an objective function.

The Gradient Boosting decision tree algorithm has demonstrated great performances on many machine learning tasks, including global contests. It produces a prediction model in the form of an ensemble of weak prediction models, often decision or regression trees. It combines weak "learners" into a single strong learner in an iterative fashion, lowering the error estimated with the chosen loss function at each stage.

c. Event analysis - Data generation and model training

We generated data for training the ML algorithm in the same format as the data we run our detection on, i.e., timestamps of Attach Requests in a simulated event. We used a 5G testbed with emulated UE to emulate an event and generate data.

Algorithm 2 Data Generation

```
1: procedure GENERATE_EVEN_DATA (int duration, int sample duration )
   ▷ The average number of equipments that would connect during an event of
   length duration
2:     num_equipments = floor((duration×3)/7)
3:     timestamps is an array of num_equipments reals
4:     for i ← 1 To num_equipments do
5:         timestamps[i] ← random_β3,4() × duration
6:     end for
7:     Sort the timestamps array
   ▷ At this point, timestamps has real values in the range [0,duration] sampled
   is a list of integers
8:     k ← 0.0
9:     i ← 0
10:    sampled_index ← 0
11:    while k <= duration do
12:        j ← 1                          ▷ Count the number of samples in the range
   [k, k + sample_duration]
13:        count ← 0
14:        while j < num_equipments and timstamps[j] < k + sample_duration do
15:            count ← count + 1
16:            j ← j + 1
17:        end while
18:        i ← j
19:        sampled[sampled_index] ← (k, count)
20:        sampled_index ← sampled_index + 1
21:        k ← k + sample_duration
22:    end while
23:    Output sampled
24: end procedure
```

Algorithm 1. Used for generating data. We consider that the average number of equipments that would.

take part in an event is $\frac{duration \times 3}{7}$. We argue this by the fact that the mean of the β(a, b) distribution is $\frac{a}{a+b}$, where a=3 and b=4. We generate random numbers following the β(3, 4) probability distribution. These numbers represent the expected timestamps of Attach Requests. We send Attach Requests and sleep for a duration equal to the difference between the random numbers generated to simulate the event. When this step ends, the data is stored in a file, where each entry corresponds to a sample period.

Let us note by **S** the sample vector defined as {s_1, s_2, s_3, …., s_n}, where s_1 corresponds to the sample period 1, and noted in an entry of the file as (t_1 - 0) (t_1= Δ); s_i is (t_{i+1}-0) (t_{i+1}=i+1* Δ). The duration of the sample period is identical and obtained as follows: $\Delta = \frac{Total\_Duration\_Event}{n}$. For the training phase, we will use the data generated earlier to train a Gradient Boosting algorithm to predict for each sample s_i, the

upper bound of the number of Attach Request expected during that sample period; we note it by *Predict_i*. Here we are interested in the upper bound value as it corresponds to the maximum intensity of normal traffic; hence exceeding this value means that we are most probably facing a DDoS attack. Once the training is done, we obtain a vector P that contains n predicted values corresponding to the maximum expected Attach Requests per sample period. The *Predicted_i* values are also stored in the file with each corresponding entry. The file will be used later as an input to analyze an event and calculate the detection rate.

d. Event analysis – Detection

During the detection step, the event detected by the Sampler is stored in the DB. The event is organized in sample periods equal to **n** with a duration Δ (the same value used for the training phase). This will allow us to normalize the number of samples of an event since each event has a different duration period, and the β(3,4) intensity depends on the duration. Thus, we do not need to train the model using different durations, as the normalization step will allow training on a single duration corresponding to β(3,4) distribution. Since the event has been organized by sample periods with the total number of Attach received during the sample period as well as the SUPI of the UE, we use the ML model (mainly the file obtained in the precedent step) to extract the *Predict_i* values for each sample period. Then, we derive another bound for each sample period as follows: *Predict_i* × *(AE_DETECTION_THRESHOLD - 1)*; a limit above which any traffic gets a 100% detection rate and gets classified as malicious. For each sample period, we calculate the detection rate as follows:

For x_i, the number of Attach Requests in the sample period [s_i] :

$$\text{If } x_i \leq Predict_i, \text{ then } detection_i = 0.$$
$$\text{Else,}$$
$$detection_i = min(1.0, \frac{x - Predict_i}{Predict_i \times (AE\_DETECTION\_THRESHOLD - 1)})$$
$$\text{where } 0 < detection_i < 1.$$

Let us suppose that during an interval (Δ), the number of Attach Requests is greater than the predicted one, meaning abnormal traffic. In this context, to estimate the other bounds, from which detection values are 100%, we use $predicted_i \times (AE\_DETECTION\_THRESHOLD - 1)$; the predicted value is multiplied by a constant, corresponding to the rate between the distance of x from the *Predict_i.* This is needed to reduce the ML errors impact and hence reduce the False Positives. Note that if *detection_i* is higher than zero, all the involved UEs during that sample period are considered as part of a DDoS attack with rate *detection_i.*

6.2.2 *DIMENSIONALITY REDUCTION*

Trust Participant Selection using Deep Reinforcement learning:

In each FL round, the central IDSM of running network slices selects a running network slice (DSM), as a trusted node. To do so, we design a new deep reinforcement learning-based model, to derive an optimal policy about trust node selection, while considering several criteria related to such nodes, such as their reputation, detection rate of malicious nodes, and its accuracy in building learning models. Deep

Reinforcement learning is a process that enables one or set of agents to learn how to make suitable decisions, through error and trial, and based on their (agents) previous experiences. Specifically, each agent interacts with the environment to receive either penalties or rewards, for the actions it makes. Hence, the main objective of deep reinforcement learning is to derive an optimal policy about agents' actions, that maximizes agents' cumulative reward.

Dimensionality Reduction of model updates:

Once a trust client is selected, it will be in charge of detecting whether the received updates include a malicious model or not. First of all, the AE of the selected trust client receives the n model updates of network slices from the central IDSM, and then applies a dimensionality reduction technique to be able in presenting the model updates in 2D dimensions. Indeed, the dimension reduction is the transformation of dataset from a high-dimensional space into a low-dimensional space, in such a way, the low-dimensional representation retains the meaningful properties of the original data.

Figure 56. AE's components.

Figure 57 shows a dimensionality reduction of our FL model updates in 3D, when applying linear discriminant analysis reduction technique. The different colour of the points, which are defined by 3 axes (x:0, y:1, z:2), present the updates of different nodes (10 clients). However, as we can notice, the reduction to 3D will not provide a clear visualization, to interrupt and classify the model updates. That is why we decided to apply dimensionality reduction to 2dimensions (2D) (Figure 58). In our study, to design an efficient detection scheme, we are based on two different techniques: Principal Component Analysis (PCA) as unsupervised technique (ignores class labels), and Linear Discriminant Analysis (LDA) as a supervised technique.

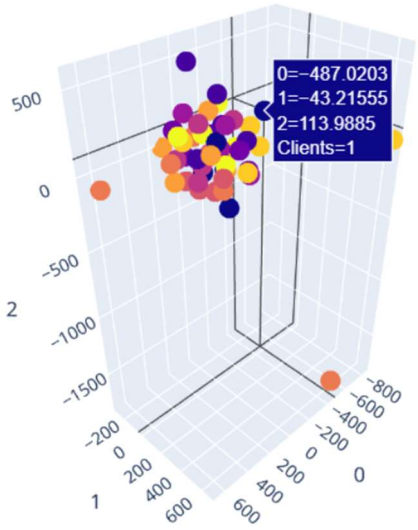

Figure 57. LDA visualization with 3 dimensional applied on the nodes update.

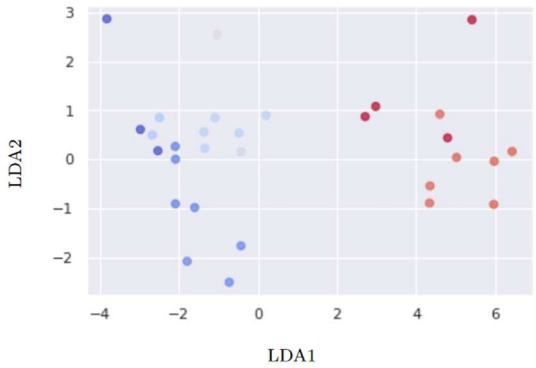

Figure 58. LDA visualization with 2 dimensional applied on the nodes updates.

6.2.3 *PRINCIPAL COMPONENT ANALYSIS*

The purpose of the dimensionality reduction is to reduce the number of features under consideration, where each feature is a dimension that partly represents the objects. Dimensionality reduction can be executed using two different methods:

- Selecting from the existing features (feature selection)
- Extracting new features by combining the existing features (feature extraction)

The main technique for feature extraction is the PCA. PCA guarantees finding the best linear transformation that reduces the number of dimensions with a minimum loss of information. Sometimes the information that was lost is regarded as noise – information that does not represent the phenomena we are trying to model but is rather a side effect of some usually unknown processes. PCA process can be visualized on Figure 59:

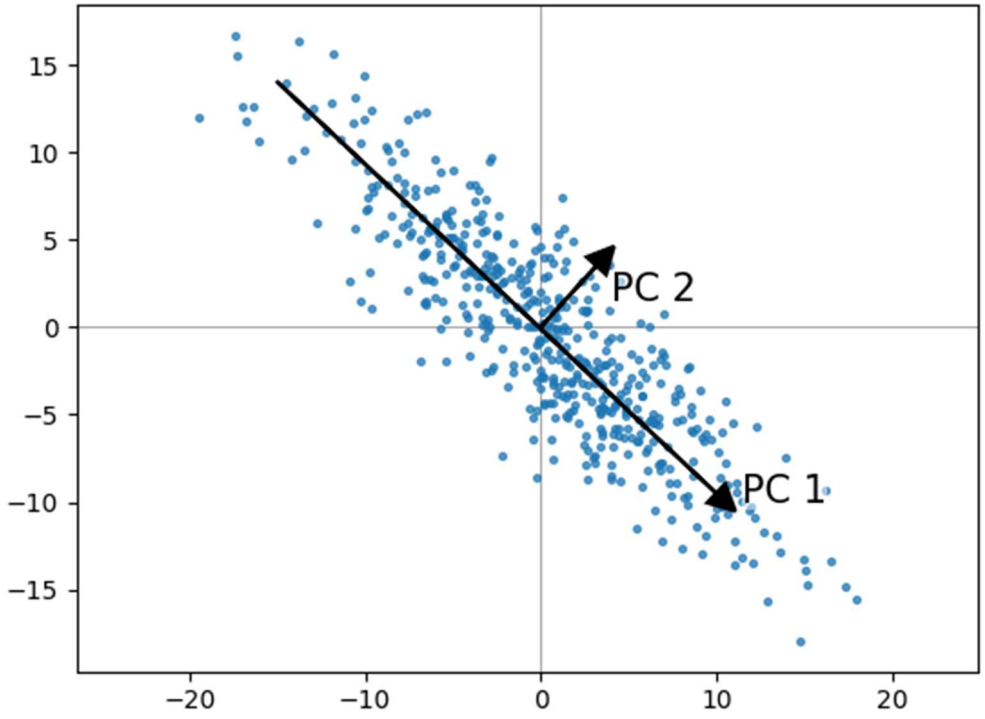

Figure 59: Principal Component Analysis

Theoretically PCA dimensional analysis (the first K dimension retaining say the 90% of variance does not need to have direct relationship with K Means cluster), however the value of using PCA came from:

a) Practical consideration given the nature of objects that we analyse tends to naturally cluster around/evolve from (a certain segment of) their principal components

b) PCA eliminates those low variance dimension (noise), so itself adds value (and form a sense similar to clustering) by focusing on those key dimension in simple terms, it is just like X-Y axis is what help us master any abstract mathematical concept but in a more advance manner

For a set of objects with N dimension parameters, by default similar objects Will have MOST parameters “similar” except a few key differences will have some highly similar features (low variance) but a few key features still quite diverse and capturing those "key principal components" essentially capture the majority of variance. Hence low distortion if we neglect those features of minor differences, or the conversion to lower principal components will not loss much information.

It is thus “very likely” and “very natural” that grouping them together to look at the differences (variations) make sense for data evaluation.

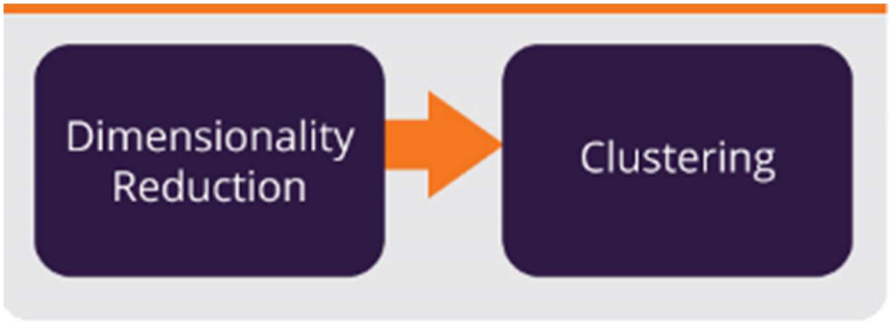

Figure 60. Illustrating dimensionality reduction towards clustering.

After the dimensionality reduction (PCA) the clustering algorithms were performed that included the K-means, one class SVM, isolation forest algorithm, kernel density algorithm, gaussian mixture algorithm and elliptic envelope algorithm.

K-means Algorithm

Under K-means technique, we try to establish a fair number of clusters so that those group elements (in a cluster) would have overall smallest distance (minimized) between Centroid and whilst the cost to establish and running the K clusters is optimal.

The dimensionality reduction along with the k means is applied for the data set that has 2 means as well as 3 means for the data set. The 2d plots as well as the 3d plots are shown in the below mentioned figure. It can be seen from the figure that the k means was successful in portioning the two different types of the data sets.

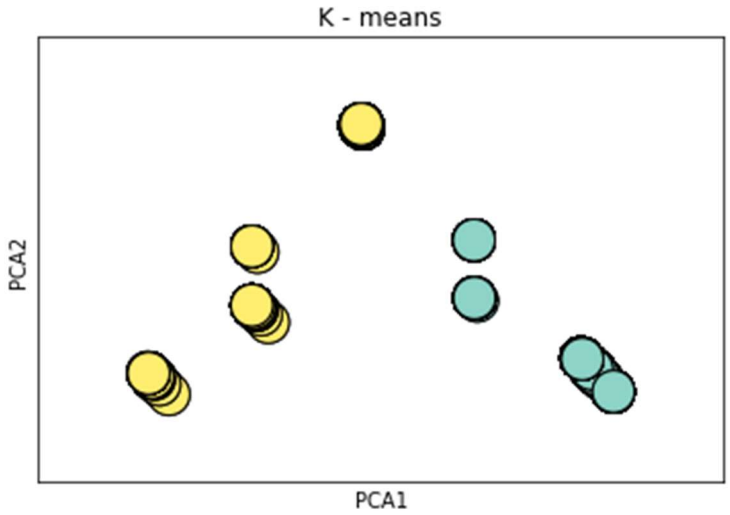

Figure 61. Relationship between PCA and K-Means in 2D.

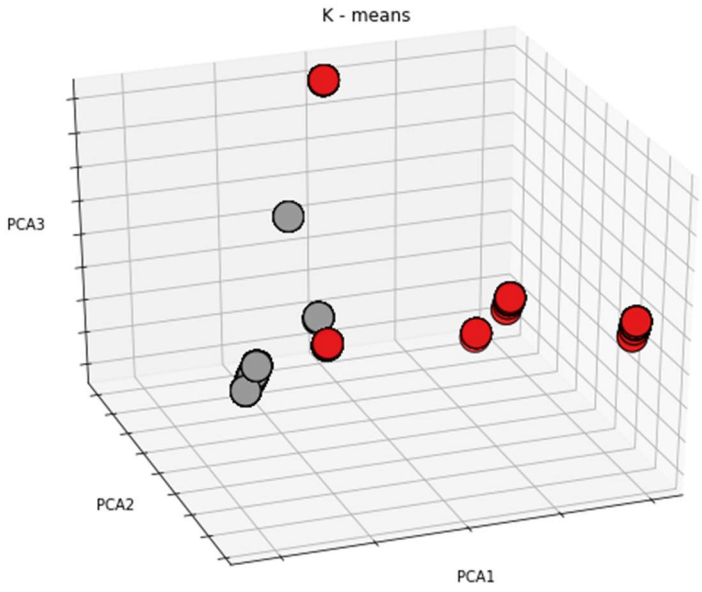

Figure 62. Relationship between PCA and K-means in 3D.

K Means grouping could be easily "visually inspected" to be optimal, if such K is along the Principal Components, that tend to share similar characteristics so if you cluster those based on those PCs, then that achieve the minimization goal. Also, those PCs quite often are orthogonal, hence visually distinct by viewing the PCA as shown above by the given figures.

The PCA having two means and three means display our K clustering results to be orthogonal or close to, which is a sign that our clustering is sound, each of which exhibit unique characteristics since by definition PCA find out / display those major dimensions (2D or 3D) such that say K (PCA) will capture probably over a vast majority of the variance.

One-Class SVM

The one class SVM clustering was done for the two means as well as the three means PCA and the results were plotted for the clustering result. The parameters that were used for the simulation were kept as following;

(kernel='sigmoid', gamma='auto', nu=0.03, tol=1e-4)

Figure 63. Comparison between PCA and SVM in 2D.

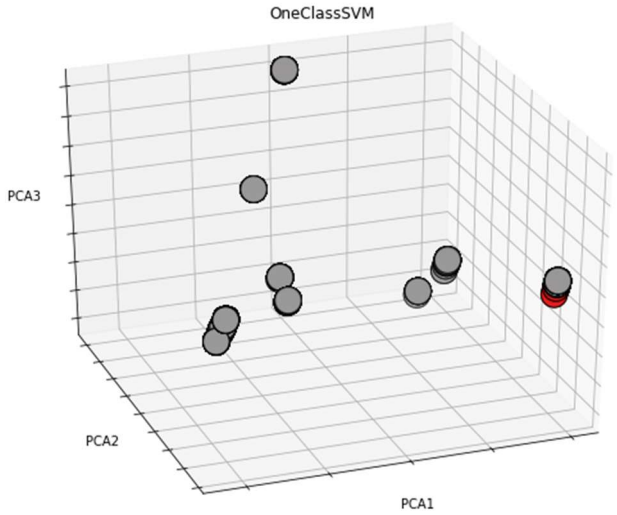

Figure 64. Comparison between PCA and SVM in 3D.

Isolation Forest Algorithm

The isolation forest was repeated using the same two means as well the three means of the PCA. The results of the isolation forest are shown below. For the isolation forest the (n_estimators=100) was kept as the minimum number of the samples for the clustering purposes.

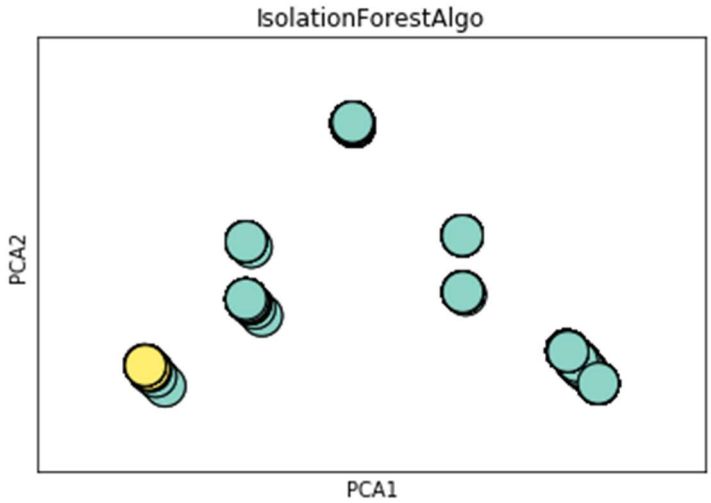

Figure 65. Comparison between PCA and Isolation Forest in 2D.

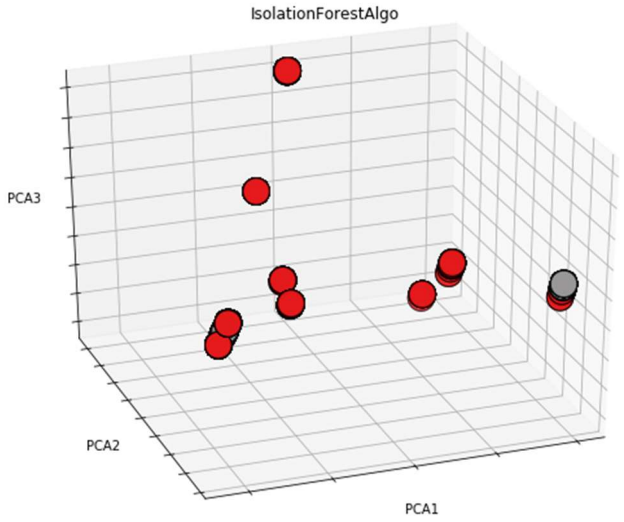

Figure 66. Comparison between PCA and Isolation Forest in 3D.

Kernel Density Algorithm

The kernel density algorithm was used by using two means as well the three means of the PCA. The settings for the results of the kernel density algorithm are shown below.

(kernel='gaussian', bandwidth=0.29, algorithm='auto', metric='euclidean', atol=0.1, rtol=0.1, breadth first=True, leaf_size=60)

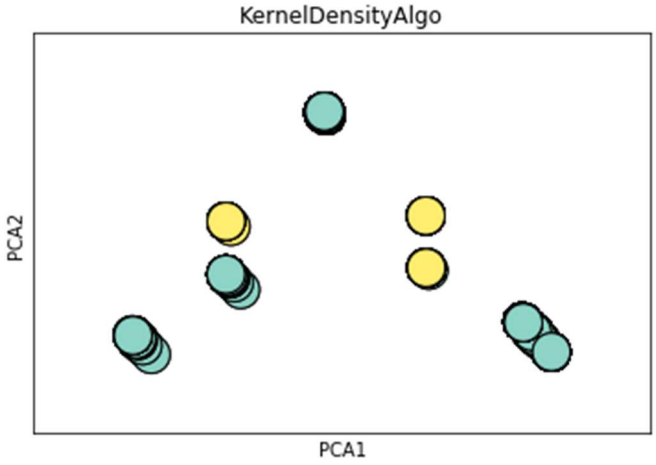

Figure 67. Comparison between PCA vs. Kernel Density in 2D.

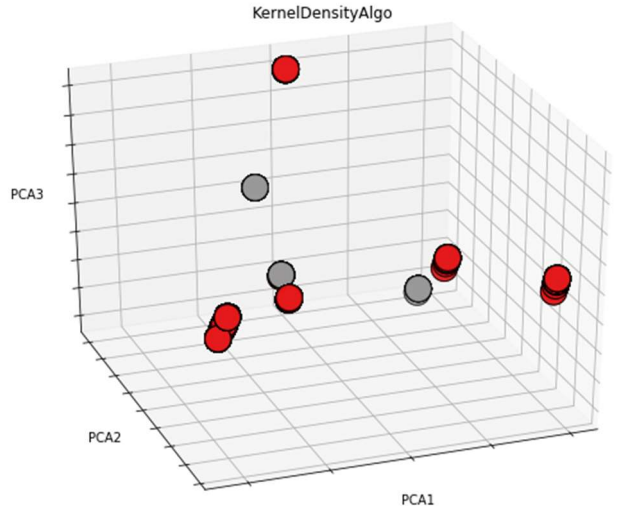

Figure 68. Comparison between PCA vs. Kernel Density in 3D.

Gaussian Mixture Algorithm

The gaussian mixture algorithm was used by using two means as well the three means of the PCA. The settings for the results of the gaussian mixture are shown below.

(n_components=2,verbose_interval=50,max_iter=200, covariance_type='spherical' ,n_init=1,init_params='kmeans',random_state=0)

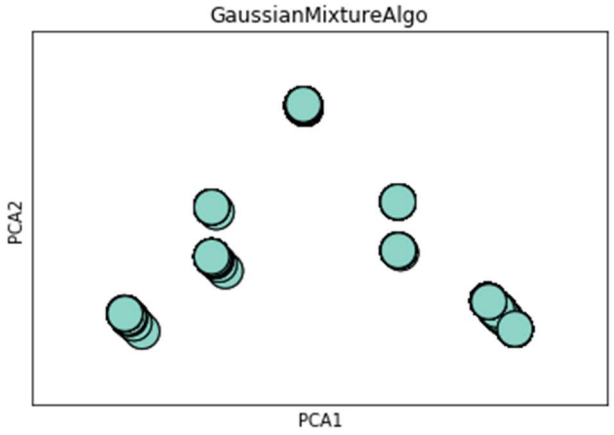

Figure 69. Comparison between PCA vs. Gaussian Mixture in 2D.

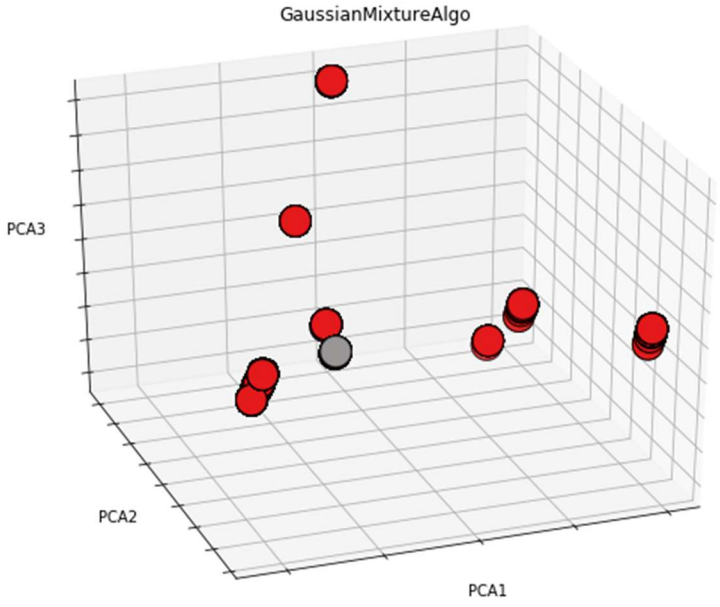

Figure 70. Comparison between PCA vs. Gaussian Mixture in 3D.

Elliptic Envelope Algorithm

The elliptic envelope algorithm was used by using two means as well the three means of the PCA. The settings for the results are shown below.

(store_precision=True,assume_centered=False,contamination=0.015)

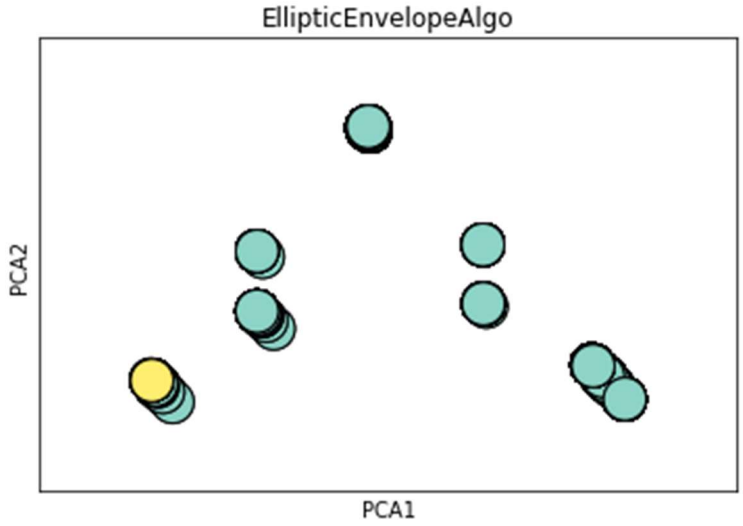

Figure 71. Comparison between PCA vs. Elliptic Envelope in 2D.

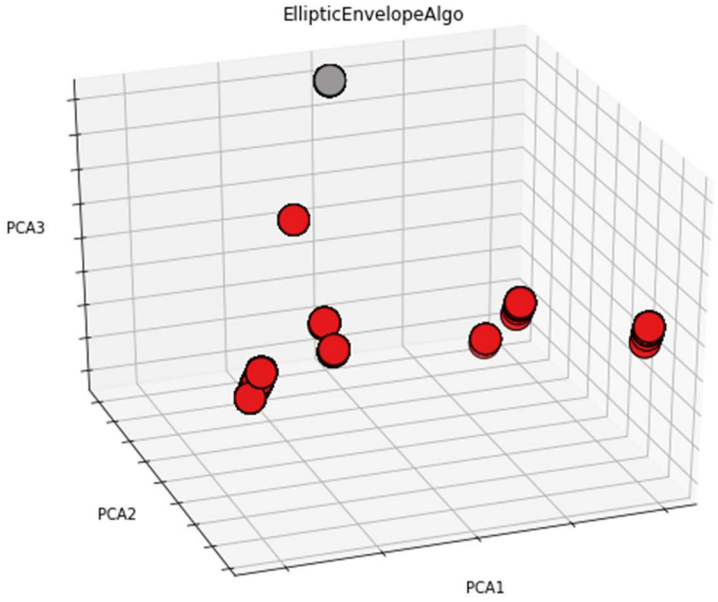

Figure 72. Comparison between PCA vs. Elliptic Envelope in 3D.

In conclusion the K means clustering is outperforming the other clustering algorithms and tends to cluster the data set having similar characteristics more efficiently.

6.2.4 *ANOMALY DETECTION OF ATTACKS*

As described in [MonB5GD5.2], our approach to uncovering this attack is to look for changes in the use of network protocols where the attacker takes advantage of the vulnerability of not protecting the integrity of the PDU session to change the recipient of the DNS packet. For the aLTEr scenario, the AE will focus only on the detection task of the incident handling guidelines by leveraging AI/ML techniques.

From the integration point of view, Apache Kafka is used for the communication between MonB5G components. AE subscribes to the Kafka topic dedicated to receive high level protocol logs from the MS as the transformation outcome of raw capture data.

If an anomaly is found in the parameters of a DNS protocol session, the RA generates an event to indicate that an aLTEr attack is in progress and posts the event in a Kafka topic dedicated to security incidents as shown in Figure 73.

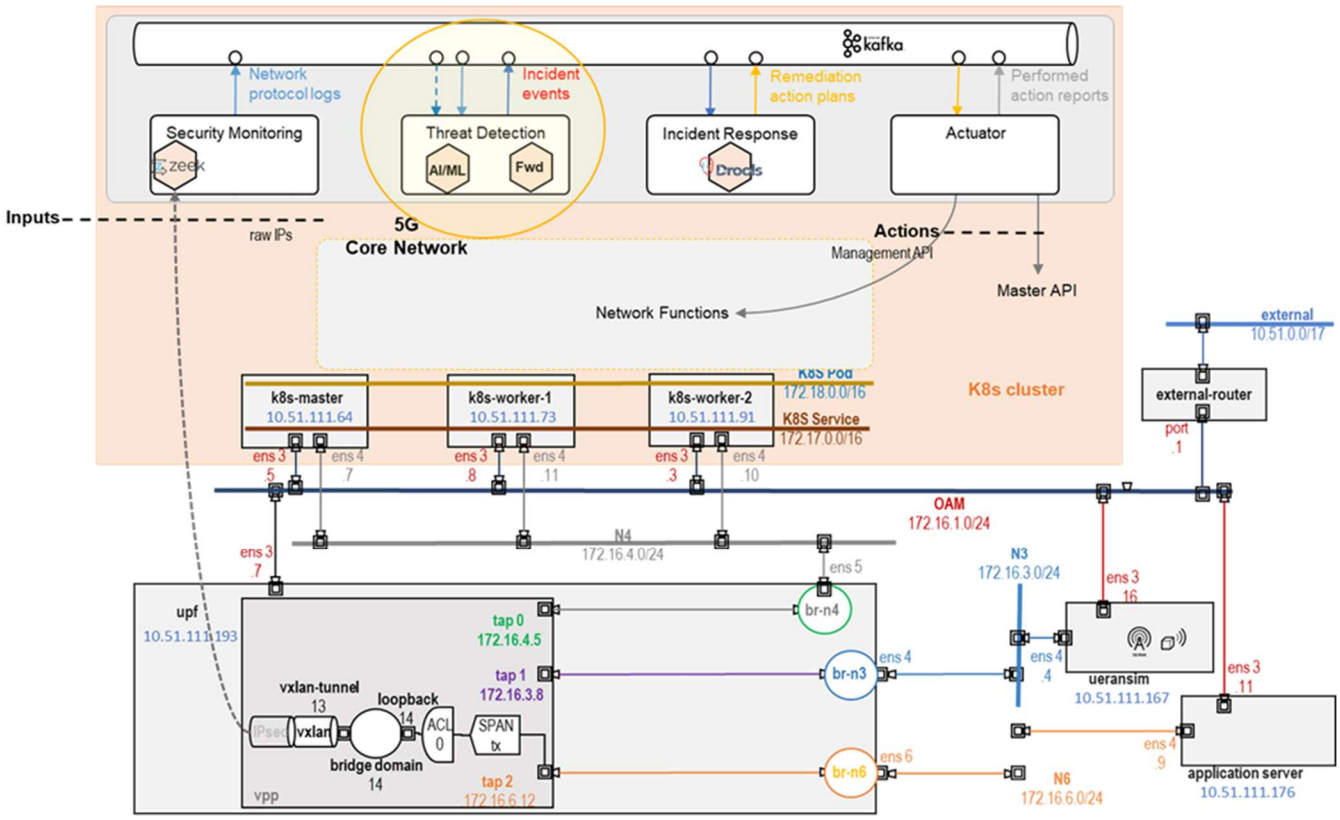

Figure 73. The integration of the AE component in the 5G core network for the scenario aLTEr.

6.3 Deployment of MonB5G DEs

6.3.1 *MMTC ATTACK*

DE component is the decision-maker of the closed-control loop system. It receives data from AE and decides the actions to take for UEs that emit abnormal traffic. DE gets as input a list including the suspected UEs (SUPI) and their corresponding detection rate values belonging to the attacks.

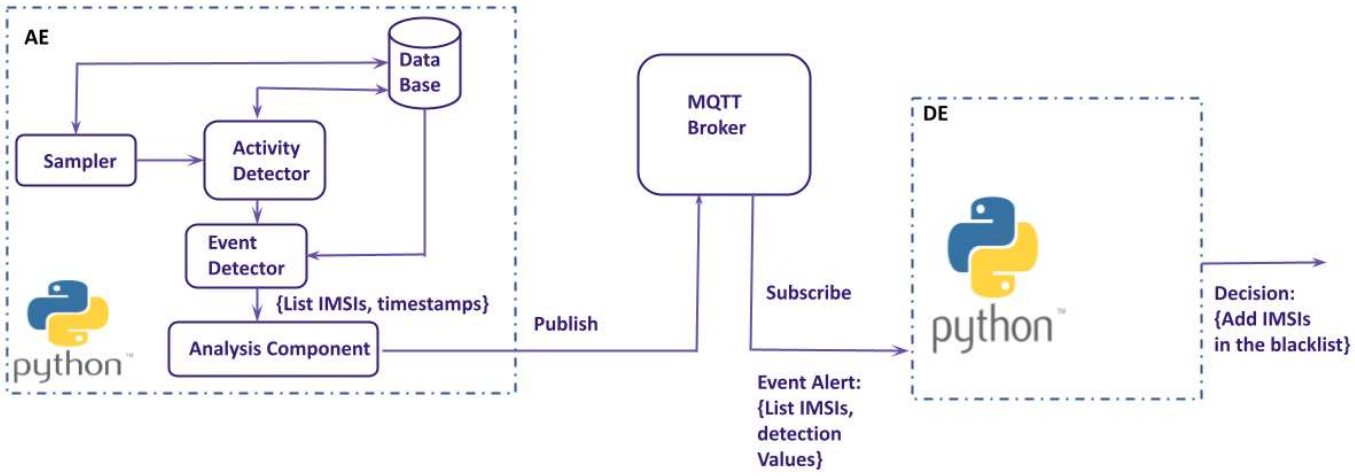

Figure 74. mMTC attack detection and blacklisting.

We devise two versions of DE. The first solution blacklists devices if their calculated prediction is higher than "DE DETECTION THRESHOLD" (i.e, a configurable parameter). The lower the threshold value, the higher the probability that devices are blacklisted. Therefore, the network operator would use lower values in order to be more strict, but in turn, it may increase the false positive. To avoid having high false-positive results, we introduce a second solution that relies on three thresholds. This solution considers the whole event and classifies the received list of UEs into three categories: F_1, F_2 and F_3 (Figure 75). DE calculates how many UEs have obtained a detection rate (detection_i) higher than 0.8 and assigns it to the first category, namely F_1. The second category includes UE having a detection rate between 0.3 and 0.8. This category corresponds to F_2. The remaining UEs are included in F_3. Then, DE checks if, in the event, a significant part of the devices had higher than usual detection rates. As a result, different decisions are to be taken:

- First, if F_1 > F_2 and F_1 > F_3, DE blacklists all the devices by adding their SUPI values to a table of blacklisted values, and disconnect them from the network.
- Second, if F_2 > F_1 and F_2 > F_3, DE adds the SUPI values to a table named ``non-trusted devices". Each UE belonging to this table has a counter named T_imsi. The counter is increased by 1 each time the UE is involved in abnormal traffic that is not blatantly an attack. The counter is incremented until it reaches a value that yields to blacklist the device.
- Third, DE ignores the alert sent by AE, and it will do nothing.

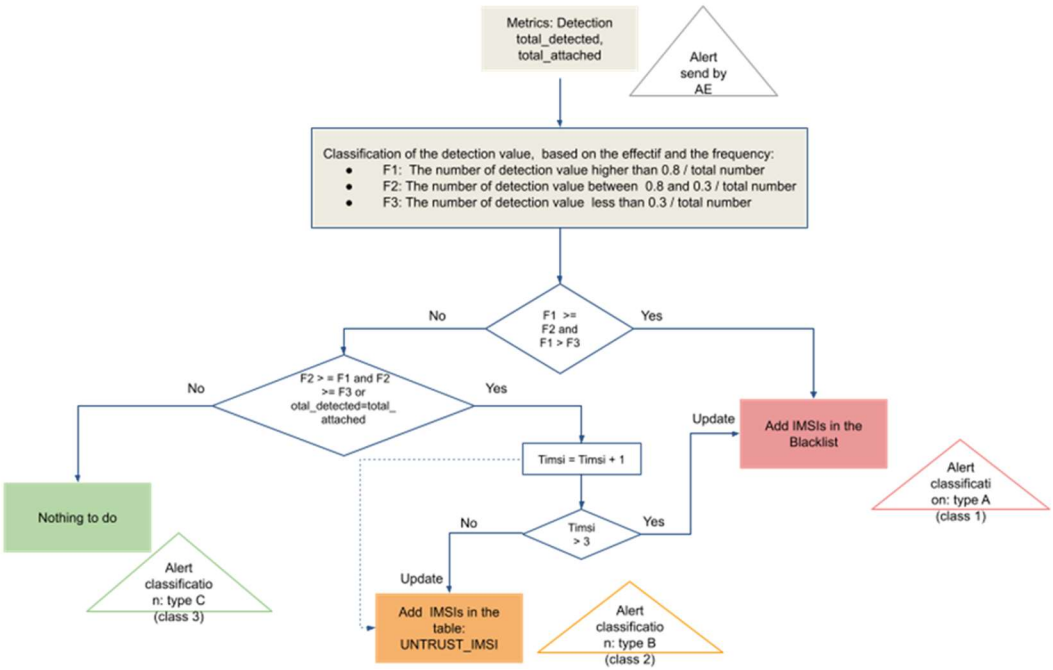

Figure 75. Flowchart of the DE's components.

The reason behind using 3 categories is based on the AE analysis and the results accuracy of the employed ML algorithm. Indeed, we notice that when the detection values associated with the connected devices sent by AE are high, the devices' generated traffic does not follow the Beta distribution. Hence, they should be immediately blacklisted as they present abnormal behaviour, justifying the need for the first category. The latter is considered a red alert, and the associated devices SUPI are blacklisted.

However, when the values are neither too high nor too low, it means that the traffic is almost close to the Beta distribution. In this case, the detected devices have malicious behaviour, or the values are due to technical failure. So, we introduced the second category, which means that the devices are not blacklisted, but DE will memorize the associated SUPI for future events. If the devices are classified in the second category more than 2 times, they will be considered malicious and moved to the first category to be blacklisted. The last category corresponds to the detected traffic being very close to the Beta distribution. This case can be either an error in the ML calculation or a delay in sending the Attach Request.

6.3.2 *ROBUSTNESS OF LEARNING ALGORITHMS IN THE FACE OF ATTACKS (EUR – POC2)*

Once applying dimensionality reduction techniques and depicting network slices' updates in a 2D plan, the last step consists of grouping the received updates into several clusters, in order to determine malicious updates/models.

Figure 76. The DE's components.

To ensure an effective detection of malicious updates, two different clustering algorithms (unsupervised and supervised) are selected: 1/ k-means which is an unsupervised learning algorithm, that considers no labelled update models' data, and 2/ k-nearest neighbours (KNN), as supervised learning algorithm which considers labelled data about model updates. Both algorithms aim to divide the received update models, at each FL round, into k clusters that share similarities and are dissimilar to the model updates belonging to another cluster. We note that we leverage the model update of the trust participant as a reference that supports to make the difference between malicious and trust models.

6.3.3 AE FOR MMTC ATTACK AND ALTER ATTACK

As part of the Zero-touch network and Service Management architecture of MonB5G is the AE. One of the AE's aims is to detect security threats to be sent to the DE for mitigation. Our solutions provided for the implementation of the AE are using AI/ML algorithms, for detecting attacks on the network and needed to be integrated on the different testbeds of the project, provided by BCOM and EURECOM. Each partner provided a sample dataset, with the same data structure as in their testbeds, to be used for implementing and evaluating our solutions. The data received for each testbed differs in structure and size, so different AI/ML algorithms were implemented for each testbed. As part of the AE engine, in addition to the algorithms that detect the security threats in the network, are also the processes for retrieving the information from the MS and for data preparation. Data preparation consists of all the transformations and feature engineering that we perform to the raw data to generate additional features and bring them to the right format that the ML models required. For deploying our solution, each testbed followed a different integration approach. A brief description of our solutions and the steps for integrating them into the testbeds are given below. For a detailed description of the machine learning algorithms used please see deliverable D5.2 of MonB5G [MonB5GD52].

BCOM provided the dataset and the testbed for the aLTEr attacks. The sample dataset provided to us contained around 800000 observations and was used for training and evaluating the different machine learning algorithms. The dataset, as well as the data received from the MS in the testbed, are not labelled, meaning that there is no flag if an observation was an attached or normal traffic. Hence, unsupervised learning techniques were used. Our solution processes the provided dataset using simple processing

techniques such as cleaning null values and removing unnecessary columns and rows. Then a transformation procedure is used to encode the ID columns into logical integer numbers and reduce dimensionality.
For integrating the solution to the testbed, we followed a template script in Python provided to us by BCOM. The template script provides the code needed to stream the data in the AE from the MS, using Kafka. All the Kafka configurations and helper functions were provided. In addition, the code required to send the anomalous records to DE, using Kafka, was also provided in the template script. The following figure shows the details.

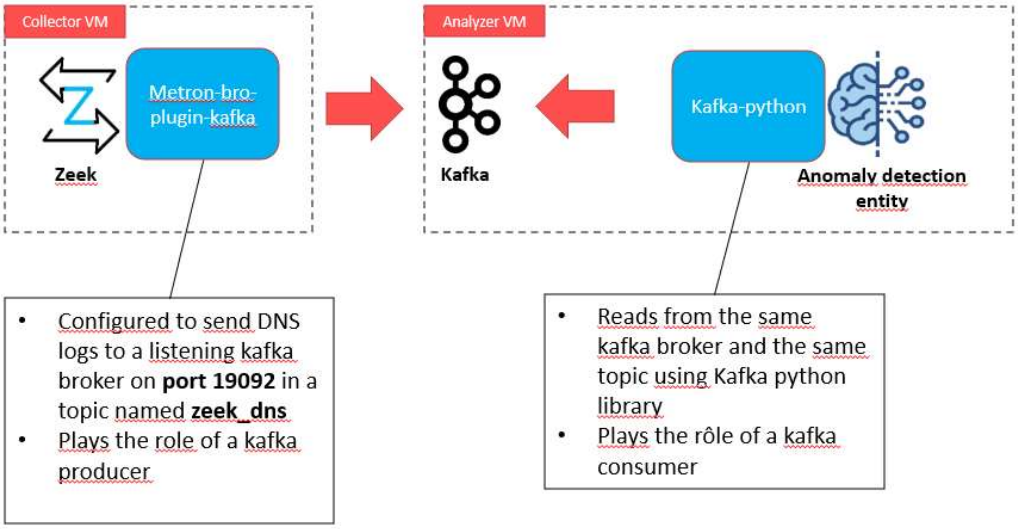

Figure 77. AE for mmTC and aLTEr attack.

The metron bro plugin was configured to the following configuration lines to the local .zeek file.

```
@load policy/tuning/json-logs.zeek
redef ignore_checksums = T;
@load /usr/local/zeek/lib/zeek/plugins/APACHE_KAFKA/scripts/Apache/Kafka
redef Kafka::logs_to_send = set(DNS::LOG);
redef Kafka::topic_name = "zeek_dns";
redef Kafka::kafka_conf = table(
    ["metadata.broker.list"] = "172.16.0.13:19092"
);
```

The following configurations were used;

```
##########Kafka configs##########
ZEEK_DNS_TOPIC = os.getenv("ZEEK_DNS_TOPIC", "ndr.zeek.logs.dns")
DROOLS_OUTPUT_TOPIC = os.getenv("DROOLS_OUTPUT_TOPIC", "ndr.ae.alert.alter")
KAFKA_HOST = os.getenv(
    "KAFKA_HOST", "ndr-kafka-bootstrap.ndr.svc.cluster.local")
KAFKA_PORT = os.getenv("KAFKA_PORT", "19092")
kafka_bootstrap_server = f'{KAFKA_HOST}:{KAFKA_PORT}'

########## Kafka Helper Functions ##########

def to_kafka(message, producer):
    message_domain_name = message['query']
    message_src_ip = message['id.orig_h']
    drools_format_message = {"data": {"event": "aLTEr_attack",
                                      "domain_name": message_domain_name, "src_ip": message_src_ip}}
    producer.send(DROOLS_OUTPUT_TOPIC, json.dumps(
        drools_format_message).encode('utf-8')).get(timeout=30)
    return 0
```

Figure 78. Configurations for the AE.

The function calls the send method on the producer instance, passing in the DROOLS_OUTPUT_TOPIC and the serialized version of the drools_format_message dictionary. The send method publishes the message to the specified topic. The get method is called with a timeout of 30 seconds to wait for the send operation to complete. If the operation completes within the timeout, the function returns 0, indicating that the message was successfully sent to the topic.

Finally, ML models have been implemented to identify data points that differ significantly from other observations. The pipeline of our solution and the ML algorithms used are given in the figure below.

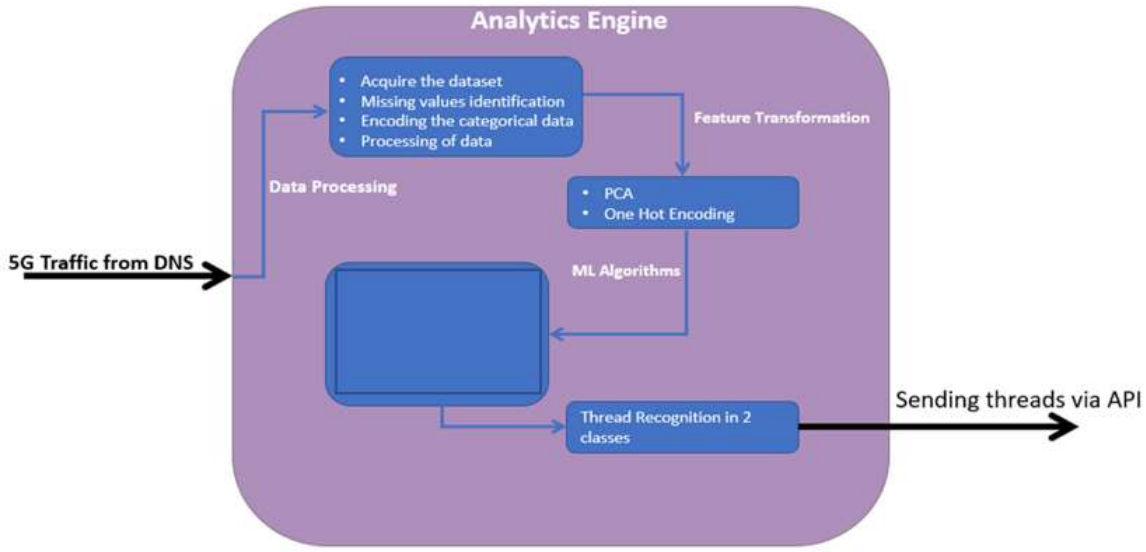

Figure 79. Diagram for the AE workflow.

For each of the proposed ML algorithms we had to modify the template script such that the correct data preparation and feature engineering process is used. The pipeline implements the ML algorithms and the result is the mode of the voting for the specific message. The PCA and scalar models are also used as helper functions during the procedure.

```
def main():
    model_kmeans = pickle.load(open(model_kmeans_path , 'rb')) ### k means model that is trained
    model_iforest = pickle.load(open(model_iforest_path , 'rb')) ### iforest model that is trained
    model_elenv = pickle.load(open(model_elenv_path, 'rb')) ### elenv model that is trained
    pca_model = pickle.load(open(pca_path, 'rb'))
    scaler_model = pickle.load(open(scaler_path, 'rb'))

    # retrieve data
    consumer = KafkaConsumer(bootstrap_servers=kafka_bootstrap_server,
                             value_deserializer=lambda x: x.decode('utf-8'))
    consumer.subscribe([ZEEK_DNS_TOPIC])
    producer = KafkaProducer(bootstrap_servers=kafka_bootstrap_server)
```

Figure 80. Code showcasing the AE main process.

The process reads the trained ML models that were trained on the sample dataset and makes predictions if a new observation is anomalous. Hence, the trained models for each of the ML algorithms used are stored

and passed as volume to the deployment. For the integration to the testbed, we shared with BCOM our python scripts and our trained models, so that they build the docker image and include it as part of the docker compose file of the testbed.

The dataset obtained from EURECOM contains connections between sources and destinations. Each connection requires a protocol, and it logs the source/destination IPs, the execution time, the length of the connection and some basic information. Similar with BCOM, the observations are not labelled, and an unsupervised learning approach was utilised. The methodology relies on a distance metric between the execution time and the length of the connection, described by the 'Mahalanobis' function. This function is applied in multi-dimensional datasets, and it uses the variance and covariance between the variables to extract a 'distance' metric. The derived metric is then modelled via an anomaly detection algorithm to account for any outliers. The solution also consists of the data preparation process for transforming each observation in the required format. Retrieving new data in the testbed the solution needs to trigger anomalous traffic using an API call.

For deploying our solution, EURECOM gave us access to their testbed and to the docker images of MS, AE, and DE. For accessing the testbed, running on a virtual machine on EURECOM's premises, we provided access to EURECOM's network via VPN. FortiClient – Zero Trust Fabric Agent were used to connect to EURECOM, together with Duo Security for setting a 2FA code.

The ML algorithm that was used for this purpose was Random Forest. Random forest is a technique used in modelling predictions and behaviour analysis and is built on decision trees. It contains many decision trees representing a distinct instance of the classification of data input into the random forest. The random forest technique considers the instances individually, taking the one with the majority of votes as the selected prediction.

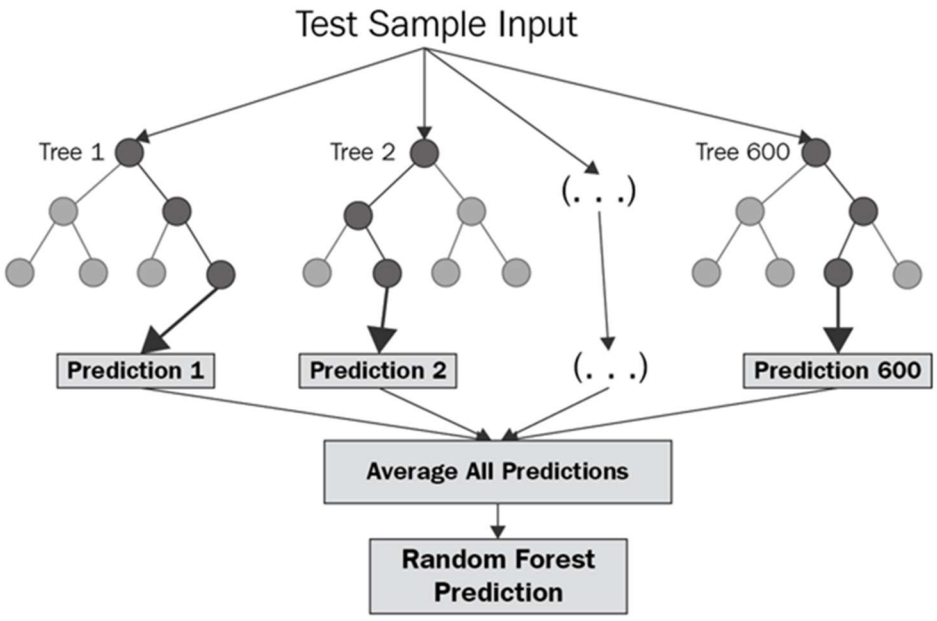

Figure 81. Random Forest Technique.

Each tree in the classifications take input from samples in the initial dataset. Features are then randomly selected, which are used in growing the tree at each node. Every tree in the forest should not be pruned until the end of the exercise when the prediction is reached decisively. In such a way, the random forest enables any classifiers with weak correlations to create a strong classifier. The model was trained based on the data set provided by EURECOM as described in the algorithm data generation flowchart. The parameters that were kept during the model training are given below;

```
clf = RandomForestClassifier(n_estimators=80, max_features = None, random_state=10, oob_score=False)
clf.fit(X_trainR,y_trainR)
predictionRandom=clf.predict(X_testR)
Randomscore = clf.score(X_testR, y_testR)
#print(Randomscore)
```

The trained model was then saved and shared that could be used for the prediction purposes.

```
import pickle
import joblib
filename = 'random_forest.sav'        #### saving the model for the trained
pickle.dump(clf, open(filename, 'wb'))
```

The trained random forest model can be used to extract the *Predict_i* values for each sample period.

```
loaded_model.predict_proba(X_testR)
```

```
array([[0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       ...,
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.]])
```

```
np.max(loaded_model.predict_proba(X_testR), axis=1)
```

```
array([1., 1., 1., ..., 1., 1., 1.])
```

6.3.4 *DE FOR MMTC ATTACK AND ALTER ATTACK*

After the threat detection procedure, the next major one is to minimize the impacts of the incident. The response to the incident activities is handled by the DE and the ACT. Based on the security rules and the incident event, the DE generates the action plan remediate the attack. As described in the D5.4, the action plan prepared for the scenario aLTEr is to use TLS to transport DNS packets, thus it prevents the packets from being modified between the UE (DNS client) and the server. In our implementation, the DE generates an action plan indicating the modification of protocol from DNS to DNS over TLS as a desired state. As for the ACT, it will then manage to make the network configuration update on the UE and deploy on demand a DNS over TLS (DoT) server that will proxy the original DNS server. The DE communicates with the ACT using a Kafka topic, and the ACT calls master API of Kubernetes to deploy security functions and a dedicated function to update the DNS configuration on the UE. Figure 82 depicts the integration of the DE and the ACT to the 5G network where they leverage network function management API and network function lifecycle management API to carry out incident response actions.

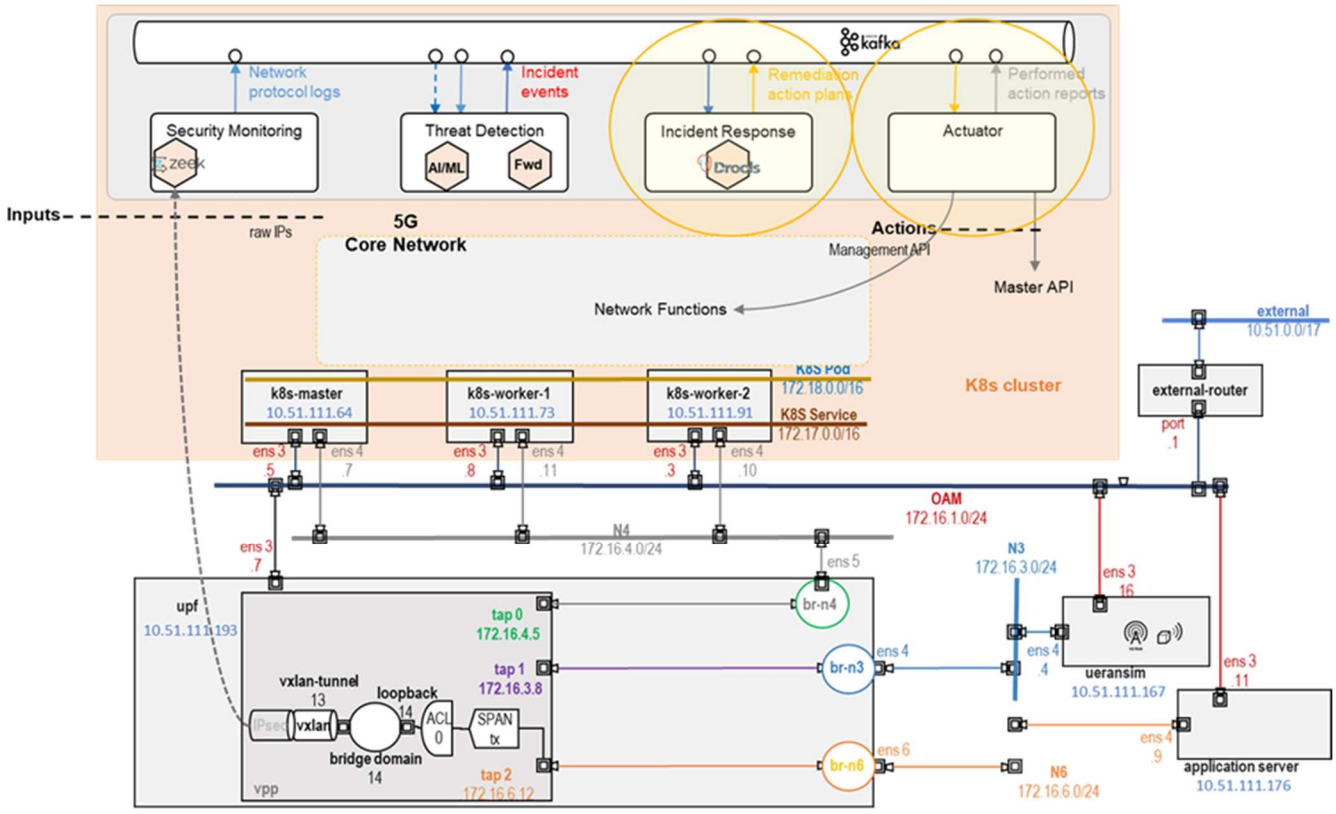

Figure 82. The integration of the DE and the ACT in the 5G core network to respond to incidents for the scenario aLTEr.

6.4 Deployment of MonB5G Actuators for PoC2

6.4.1 *RESPONSE AND MITIGATION ALTER ATTACK*

The four MonB5G components MS, AE, DE and ACT together constitute the security orchestrator as defined in D2.4. They act as a reactive line of defence for the 5G network by detecting threats that violate the preventive measures in place, then remediating them through management and orchestration of security and network functions. In the aLTEr scenario, the remediation actions are twofold and consist of:

- Requesting the 5G control plane using the exposed API to update the user plane security policy for the PDU session and to inform the UE of the new DNS configuration,
- Requesting the K8S orchestrator using the master API to create a service relaying DNS over TLS flows between the UE and the legitimate DNS server.

The information of the secured DNS is transmitted to the UE via the messages PDU Session Establish/Modification Command, which contains the extended configuration options information element (IE) "0031H DNS server security information" as referenced in the [3GPP TS 24.008]. This IE indicates DNS security information such as:

- The security protocol type (TLS or DTLS)
- The port number
- The authentication domain name
- The root certificate

Regarding the DoT service, as the legitimate DNS server may not support the DoT protocol, ACT will deploy a new network function in the 5G core network, which acts as a proxy between the UE and the legitimate DNS server and secures DNS messages in the PDU session without the need to enable integrity protection for the PDU session.

We chose to use the COTS CoreDNS tool that supports DNS over TLS and can handle TLS handshake and forward raw DNS requests to the data network DNS server. The CoreDNS is deployed as a K8S service of NodePort type exposing a reachable port from a network external to the cluster. The figure below shows the placement of the DoT instantiated by CoreDNS between the UE and the legitimate DNS server.

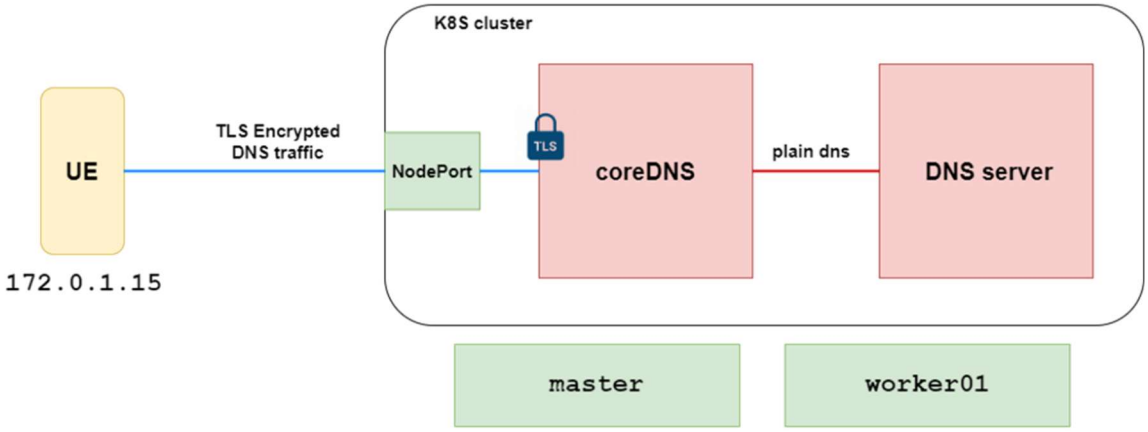

Figure 83. CoreDNS as a DoT to secure DNS communication between the UE and the legitimate DNS server.

7 Conclusions

This deliverable (D6.1) represents a significant milestone in the MonB5G project's journey towards realizing a decentralized control solution for Beyond-5G communication networks. It focuses on the validation of inter-operability and control-loop capabilities envisioned by the MonB5G Architecture specified in Work Package 2 (WP2). This deliverable joins up the work presented in Work Packages 3, 4, and 5 to materialize the project's mission of providing autonomic management with the features envisaged in the early stages of the MonB5G project.

The key achievements covered by this deliverable include the demonstration of the integration of the MS/AE/DE/ACTs for the materialization of control loops across the 5G network, the fulfilment of the architectural vision of MonB5G, and the functional validation of partners' enablers and their compliance with MonB5G specifications. This deliverable also presents, from a high-level architectural perspective, the experimental platforms in which the MonB5G components have been integrated and functionally tested, providing context for the testing procedure and the properties of the environments in which MonB5G has been demonstrated to thrive. Overall, this deliverable represents an essential step forward in the MonB5G project's mission and serves as a foundation for future work towards the project's ultimate goals to be demonstrated in D6.2.

References

[Yanez2021] Yáñez Parareda, Eduardo. "Federated learning network: Training distributed machine learning models with the federated learning paradigm." Máster en Ingeniería Informática, (2021).

[ETSINFV2017] ETSI NFV ISG, "GR NFV-EVE 012, "Network Functions Virtualisation (NFV) Release 3; Evolution and Ecosystem; Report on Network Slicing Support with ETSI NFV Architectural Framework", v3.1.1," 2017

[ETSIISGNFV2021] ETSI ISG NFV, "ETSI ISG NFV: Work Program and Releases Overview", Available Online: https://docbox.etsi.org/ISG/NFV/Open/Other/ReleaseDocumentation/NFVTSC(21)000004r1_Summary_of_NFV_Releases_by_Jan_2021.pdf

[NFVETSI] ETSI Network Functions Virtualisation (NFV), Available Online: https://www.etsi.org/technologies/689-network-functions-virtualisation

[MonB5GD4.2] MonB5G consortium, D4.2: Final release of the MonB5G architecture (including security), https://www.monb5g.eu/wp-content/uploads/2021/11/D2.4_MonB5G-Architecture.pdf

[MonB5GD4.3] MonB5G consortium, D4.3: Report on Integration and testing of the MonB5G DE, https://www.monb5g.eu/wp-content/uploads/2022/12/D4.3-Report-on-Integration-and-testing-of-the-MonB5G-DE-FINAL-FULL.pdf

[ETSIGS2016] ETSI GS NFV-REL 003 V1.1.2 (2016-07) Network Functions Virtualisation (NFV); Reliability; Report on Models and Features for End-to-End Reliability

[ETSIGS2014] ETSI GS NFV-MAN 001 V1.1.1 (2014-12) Network Functions Virtualisation (NFV); Management and Orchestration

[Kubernetes] Kubernetes, open-source container orchestration engine, [Online: https://kubernetes.io/, Available: February 2023

[MonB5GD52] MonB5G consortium, D5.2: Final report on AI-driven MonB5G security techniques. [Online Available]: https://www.monb5g.eu/wp-content/uploads/2022/09/MonB5G_D5.2_submitted.pdf

[InfluxData] InfluxDB Times Series Data Platform, [Online: https://www.influxdata.com/, Available: February 2023]

[Telegraf] Open-Source Server Agent of InfluxDB, [Online: https://www.influxdata.com/time-series-platform/telegraf/, Available: February 2023]

[3GPP TS 24.008] 3rd Generation Partnership Project: Mobile radio interface Layer 3 specification; Core network protocols; Stage 3, (Release 17) V17.8.0 (2022-09)

[AmariCallBox] Amari Callbox Series, Amarisoft [Online: https://www.amarisoft.com/products/test-measurements/amari-lte-callbox/, Available: February 2023]

[MonB5GD3.3] MonB5G consortium, Report on Integration and testing of the MonB5G AE and MS, [Online: https://www.monb5g.eu/wp-content/uploads/2022/11/MonB5G_D3.3_Submitted.pdf, Available: February 2023]